I. CAPÍTULO : PROYECTO

I.1 PRESENTACIÓN DEL PROYECTO

I.1.1. Titulo

Mejoramiento de la Administración en los Colegios de Profesionales con el uso de herramientas Web 2.0

I.1.2. Área del Proyecto

Tecnologías de Información y Comunicación

I.1.3. Responsable del Proyecto

Carrera de Ingeniería Informática – Taller III – Grupo 3

I.1.4. Entidades Asociadas(s)

Colegios de Profesionales

I.1.5. Compromiso del Director del Proyecto

Yo, **Marianela Alex Soliz** Directora del proyecto acepto las bases y condiciones del concurso, asimismo asumo la responsabilidad de cumplir los compromisos de ejecución del proyecto **"Mejoramiento de la Administración en los Colegios de Profesionales con el uso de herramientas Web 2.0"** en caso de aprobarse. Univ. Marianela Alex Soliz	
Nombre de la Directora	**Firma de la Directora**

Tabla 1. Compromiso de la Directora del Proyecto

I.1.6. Duración

La duración del proyecto será de 8 meses

I.1.7. Resumen del Proyecto

Desde hace algunos años existe una nueva filosofía o actitud en el desarrollo de sitios Web, que permite conectar personas, mejorando su interacción y promoviendo su participación como co-autores de la Web, de este modo se genera la inteligencia colectiva. Estos sitos Web son más conocidos como herramientas sociales, porque permiten o ayudan a las personas a ganar vínculos sociales alrededor del mundo. Algunos ejemplos de estas herramientas son los gestores de blogs (blogger, wordpress, movableType, bitácoras, la coctelera, etc), las redes sociales (facebook, hi5, twiter, likedIn, Sonico, myspace, etc) y otras aplicaciones.

En nuestra ciudad existe poco conocimiento de estas herramientas sociales en las organizaciones por lo que no se ven beneficiados con las utilidades y facilidades que estas ofrecen. Como por ejemplo los Colegios de Profesionales, organizaciones importantes que asocian a muchos profesionales de acuerdo a su área.

El presente proyecto contribuirá a los Colegios de Profesionales para que puedan cumplir adecuadamente su rol representativo en la sociedad, específicamente el propósito del proyecto es mejorar su administración. Para conseguir el propósito del proyecto se desarrollará:

- **Blogprof,** Herramienta social Web 2.0 que mejorara la administración de estas organizaciones y facilitara el intercambio de experiencia, materiales e informaciones entre los colegiados/as, de esta forma también la sociedad se verá beneficiada porque tendrán una mejor interacción y conocimiento de estas organizaciones.

- **Socialización y Capacitación** Talleres de capacitación sobre la temática del uso de herramientas de sociales Web 2.0 y BLOGPROF a los miembros de los Colegios, y profesionales en general.

La herramienta social **BLOGPROF** permitirá a cualquier Colegio de Profesionales crear su blog y mostrar su información, el administrador además podrá registrar a los demás miembros de la organización de esta forma se crearan los grupos, todos los miembros registrados podrán colaborar en el blog que se encuentra registrado publicando su conocimiento a través de artículos que podrán ser comentados por los lectores que dependiendo del autor, este podrá responder o no, además también se podrá publicar fotos, links, archivos, encuestas que promueven la participación de los usuarios registrados y no registrados.

I.1.8. Síntesis Vinculación entre objetivos, macro actividades y recursos

OBJETIVOS	MACROACTIVIDADES	RECURSOS
El uso de herramientas Web 2.0 para una mejor Administración en los Colegios de Profesionales	➢ Desarrollar la herramienta social para los Colegios de Profesionales ➢ Socializar a los miembros de los Colegios de Profesionales sobre el uso de herramientas Web 2.0 y BLOGPROF.	

Tabla 2. Síntesis vinculación entre objetivos, macro actividades y recursos

I.2 PLAN ESTRATÉGICO DE LA UNIDAD

I.2.1. Misión

La misión del Colegio Médico de Tarija es: *"Desarrollar estratégicamente acciones efectivas que promuevan el desarrollo integral de sus colegiados, así como impulsar la profesión a cumplir su rol social y promotor de políticas públicas que contribuyan al desarrollo del país."*

I.2.2. Análisis FODA (de los factores externos e internos)

Análisis del Contexto Externo

Oportunidades

1. Crecimiento poblacional que nos permite tener una alta demanda potencial
2. Dialogo permanente con el sector salud.
3. Capacitación de nuestros afiliados a través de convenios interinstitucionales.
4. Normatividad vigente que respalda las acciones en el desarrollo de nuestra profesión.
5. Buenas relaciones con instituciones públicas y privadas de nuestra país

Amenazas

1. Mucha oferta de egresados de las universidades y pocas fuentes de empleo, que ocasionan que gran parte de ellos emigren o laboren en áreas distintas a la medicina.
2. Incremento en la oferta de cursos de capacitación por parte de instituciones educativas.
3. Crecimiento poblacional en zonas de pobreza que agudiza su condición y limita su capacidad de acceso a nuestro servicio.

Análisis del Contexto Interno

Fortalezas

1. Profesionales comprometidos e involucrados, con alto sentido de desprendimiento personal, en los procesos de gestión y desarrollo de las actividades de la institución.
2. Nuestros socios son profesionistas de alto prestigio.
3. Tenemos una buena comunicación con las universidades de la localidad.

Debilidades

1. Existe una deficiente comunicación entre la institución, sus afiliados y la sociedad.
2. Débil cultura organizacional.
3. Limitada participación en la marcha institucional.
4. Indiferencia frente a la problemática corporativa.
5. Deficiente ética profesional.
6. Ausencia de especialidades.
7. Deficiente vinculación con el entorno social y organizacional de nuestro Colegio

I.2.3. Conclusiones del análisis FODA

Según el presente análisis de las Fortalezas, Oportunidades, Debilidades y Amenazas se puede indicar como la mejor oportunidad de esta institución en el ámbito externo es la capacitación de sus afiliados a través de convenios interinstitucionales que permite fortalecer la actualización de conocimientos.

Se puede deducir como una gran amenaza la poca existencia de empleo ante la creciente oferta de egresados de nuestras universidades lo que ocasiona que gran parte de estos emigren a otras ciudades.

En el ámbito interno, la mejor fortaleza de esta institución es la buena vinculación con el entorno social, empresarial de nuestro departamento y su mayor debilidad es que existe una deficiente comunicación entre la institución, sus afiliados y la sociedad.

Una de sus grandes debilidades es el deficiente relación entre sus afiliados dentro de la misma institución y también el carente vínculo social con su comunidad, que no permite a la sociedad saber el rol que cumple esta institución.

I.2.4. Objetivos de la organización

Objetivo Principal

Representar activamente a la totalidad de sus asociados a nivel departamental, nacional e internacional, promoviendo permanentemente capacitación, mejores prácticas profesionales, en cumplimiento de la ética, asegurando calidad y excelencia en el ejercicio profesional, con el fin de gestar una sociedad más correcta, transparente y respetuosa de la ley.

Objetivos Secundarios

- Normar y controlar el ejercicio de los profesionales en Informática.
- Velar por el fiel cumplimiento de la ética profesional.
- Formación continua.

I.2.5. Estrategias y planes de acción

Las estrategias y su respectivo plan de acción que propone el proyecto para el logro de los objetivos del plan estratégico del Colegio Médico de Tarija y de cualquier otro Colegio que tenga objetivos similares son los siguientes:

Estrategia 1

Fomentar el intercambio de conocimientos entre los colegiados/as.

Plan de Acción

Desarrollo del gestor de Blogs para Colegios de Profesionales (BLOGPROF)

Actividades a desarrollar:

- ✓ Ingeniería de Requerimientos
- ✓ Análisis y Diseño de la plataforma de Blogs
- ✓ Instrumentación
- ✓ Pruebas y Despliegue

Estrategia 2

Capacitar y socializar a los afiliados sobre la temática del uso de herramientas de sociales.

Plan de Acción

Socialización y Capacitación de BLOGPROF y otras herramientas Web 2.0.

Actividades a desarrollar:

- ✓ Planificar y realizar el proceso de socialización
- ✓ Planificación y realizar el proceso de capacitación
- ✓ Elaboración del informe final

I.3 VINCULACIONES DEL PLAN ESTRATÉGICO Y DEL PROYECTO

I.3.1. Coherencia del Proyecto con el Contexto

La ejecución del proyecto permitirá a los Colegios de Profesionales mejorar su administración permitiendo una comunicación más fluida de la información institucional, científica y gremial, para este cometido es necesario el desarrollo de una plataforma o gestor de Blogs para estas instituciones.

El desarrollo de esta herramienta social se detalla en las actividades señaladas para este componente. La plataforma permitirá la administración y el mantenimiento de los blogs creados por estas instituciones.

De esta manera estas instituciones tendrán más representatividad en la sociedad porque podrán contar con una herramienta que les permita interactuar con sus afiliados e informar a la población en general sobre todas las actividades que realizan en favor de ellos.

Para que estas instituciones conozcan esta nueva herramienta de información así como también su administración se realizara un taller de capacitación a los miembros de los Colegios de Profesionales al finalizar el desarrollo de la herramienta.

I.3.2. Vinculaciones de los problemas que reconoce la entidad y los problemas que busca resolver el proyecto

PROBLEMAS DEFINIDOS COMO PRIORITARIOS	PROBLEMAS QUE ABORDA EL PROYECTO
Existe una deficiente comunicación entre la institución, sus afiliados y la sociedad.	Los Colegios de Profesionales no cumple adecuadamente su rol
Reducida participación de los afiliados en las diferentes actividades que realiza la institución.	Deficiente administración en los Colegios de Profesionales
Los canales de difusión cumplen en forma insatisfactoria	La difusión de la información por parte de los Colegios de Profesionales es inadecuada
Los afiliados a estas instituciones tienen pocos conocimientos de los avances tecnológicos en informática.	Poco conocimiento sobre herramientas tecnológicas de información y comunicación

Tabla 3. Cuadro de los problemas que reconoce la entidad y los problemas que busca resolver el proyecto

I.3.3. Vinculaciones estrategias prioritarias y estrategias (macro actividades) del proyecto

ESTRATEGIAS	ESTRATEGIAS (MACROACTIVIDADES) PROYECTO
- Comunicados por medio de la radio y/o TV - Comunicados por medio de prensa escrita - Informar por medio de Circulares - Comunicar vía telefónica - Enviar mensajes por medio de correo electrónico - Realizar Reuniones	Desarrollo de una plataforma de Blogs para Colegios de Profesionales
	Capacitación realizada a los Colegios de Profesionales sobre la temática de "Uso de herramientas Web 2.0 y BLOGPROF".

Tabla4. Cuadro de estrategias prioritarias y estrategias del proyecto

I.3.4. Justificación

Muchas personas de nuestra población no tienen conocimiento de los objetivos y finalidades de los diferentes Colegios que agrupan profesionales de una misma área.

Generalmente estas instituciones tienen actividades o reuniones importantes que deben desarrollarse en el cual deben participar todos los afiliados pero en la mayoría de los casos no participa la mayoría lo que debilita el fortalecimiento institucional.

Estas instituciones son entes que agrupan o afilian profesionales, para dar a conocer comunicados o temas de interés a sus afiliados o al público en general, utilizan los medios de comunicación más conocidos como la radio, televisión, prensa escrita, vía telefónica, trípticos, circulares, etc., pero generalmente la información no llega a todos sus destinatarios, además esta información se publica de manera muy resumida. Esto demuestra que los canales de difusión utilizados no cumplen de manera satisfactoria.

Los Colegios deben promover la formación permanente de los colegiados y esto puede hacerse utilizando herramientas tecnológicas que permitan el intercambio de conocimientos.

I.4 EL PROYECTO

I.4.1. Objetivo general, específicos e indicadores de resultados

I.4.1.1. Cuadro de Involucrados

Grupos	Intereses	Problemas Percibidos	Recursos y Mandatos
Directivas de los Colegios de Profesionales	- Contar con una comunicación más fluida entre la institución y sus afiliados. - Desempeñar nuestro rol como institución de una manera más eficiente en la sociedad.	- Existe una deficiente comunicación entre la institución, sus afiliados y la sociedad. - Reducida participación en las actividades que organiza la institución.	- **R:** Circulares, comunicados radiales, televisivos, correos electrónicos. - **M:** Todos los Colegios de Profesionales tienen una misión.
Afiliados de los Colegios de Profesionales	- Difusión de la información por parte de los Colegios de Profesionales más adecuada	- La población no se entera de las actividades que se desarrollan a favor de ellos. - Los canales de difusión cumplen en forma insatisfactoria. - Los afiliados a estas instituciones tienen pocos conocimientos de los avances tecnológicos en informática.	- **M:** Asistir a las reuniones y actividades programadas por la directiva - **R:** Participación en las diferentes actividades de la institución
Población en Gral.	- Tener conocimiento sobre los beneficios que puedan ofrecernos estas instituciones. - Contar con información al alcance de los profesionales de nuestra ciudad.	- No se conoce este tipo de instituciones - Muchas veces los profesionales no cumplen eficientemente con su deber.	- M: Beneficiarse de las actividades que organizan estas instituciones a favor de la población - R: Escuchar o ver spots publicitarios que comunican estas actividades.

Tabla 5. Cuadro de Involucrados

I.4.1.2. Árbol de Problemas

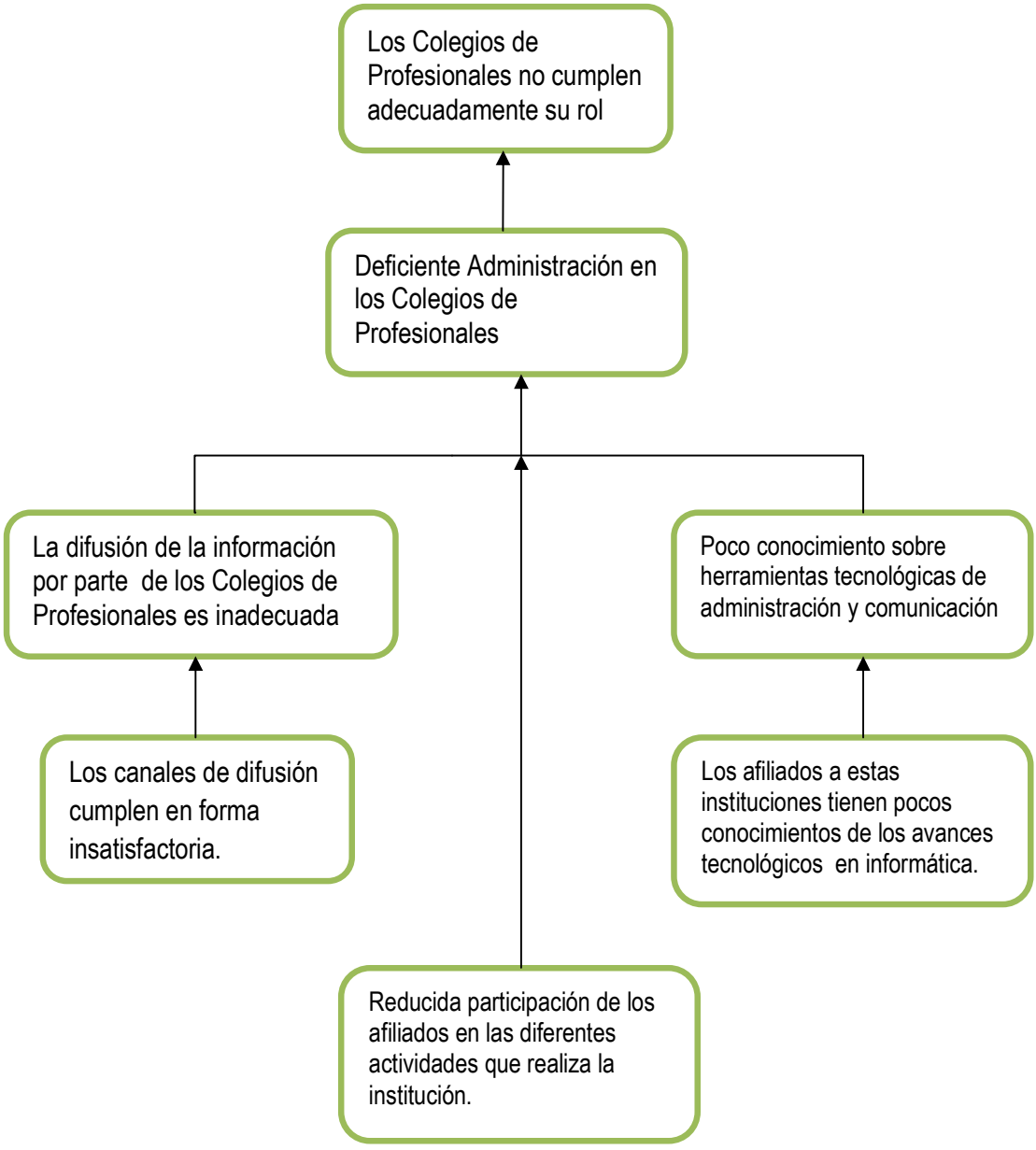

Figura 1. Árbol de Problemas

I.4.1.3. Árbol de problema-objetivo

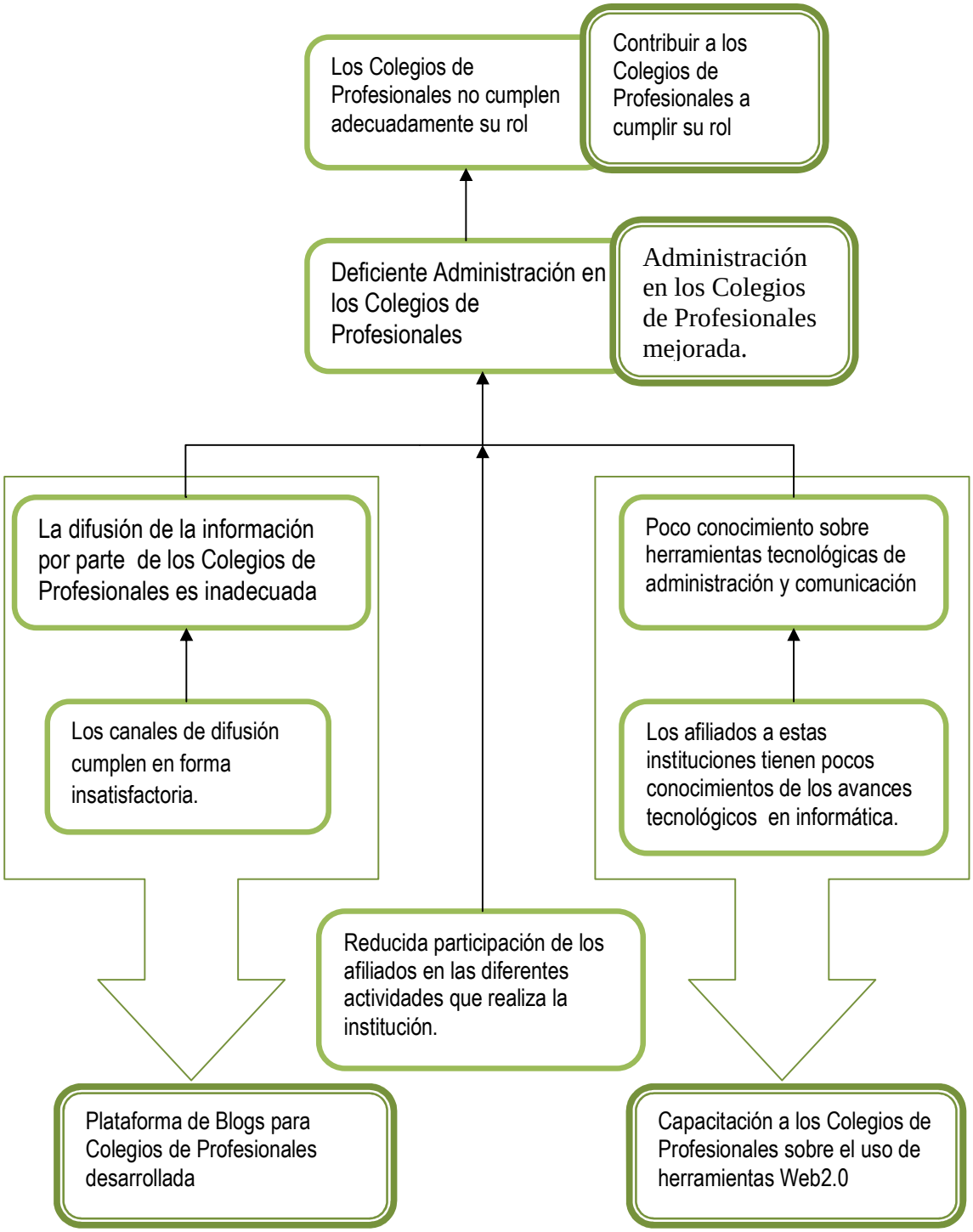

Figura 2. Árbol de Problemas-Objetivos

I.4.1.4. Árbol de objetivos

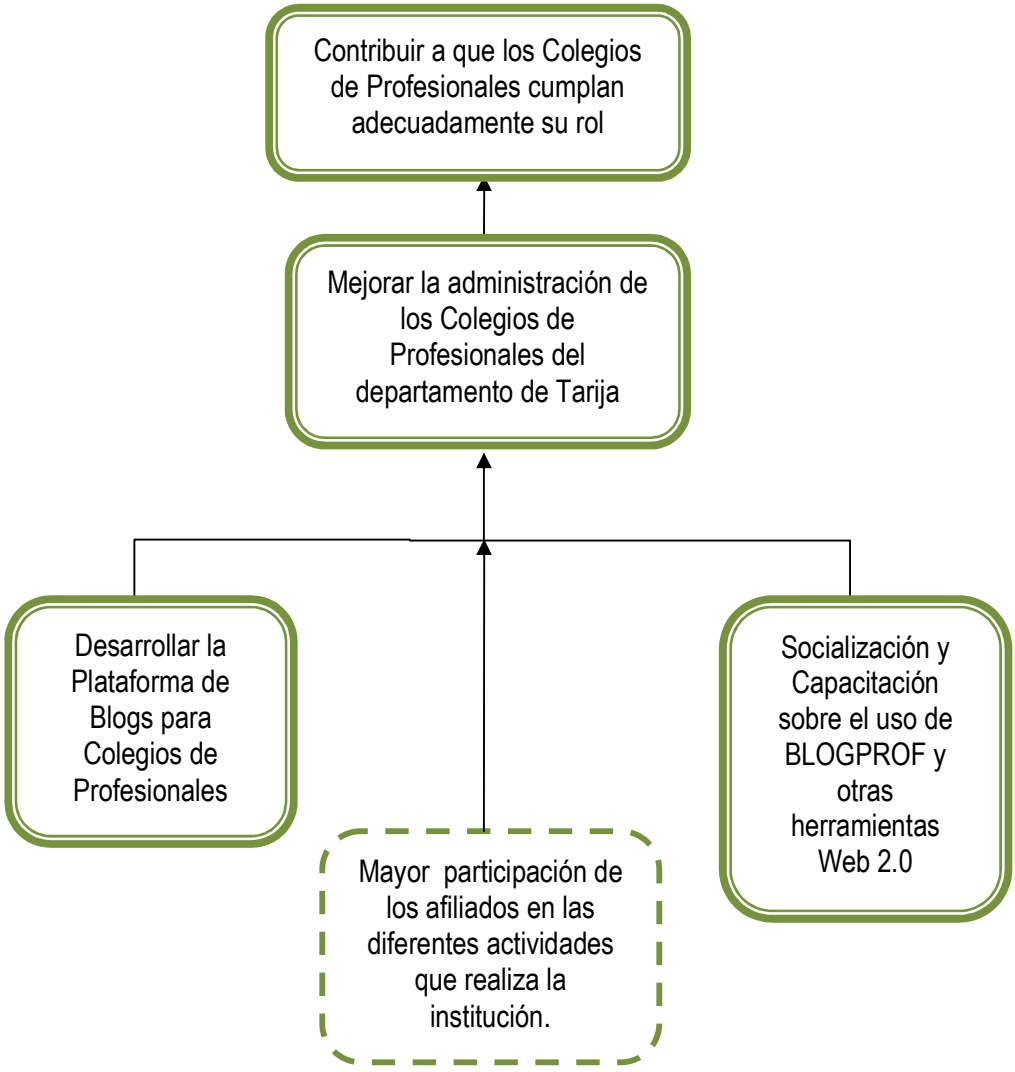

Figura 3. Árbol de Objetivos

I.4.1.5. Matriz de Marco Lógico

RESUMEN NARRATIVO	INDICADORES	MEDIOS DE VERIFICACION	SUPUESTOS
Fin Contribuir a que los Colegios de Profesionales cumplan adecuadamente su rol representativo ante la sociedad.	A partir del primer año de ejecutado el proyecto, más del 50% de los colegios departamentales de profesionales de Tarija, han creado su blog específico.	Reporte del Administrador de Blogs validado por los Colegios que han creado su blog en la plataforma.	La conectividad es adecuada y los colegios muestran interés en la continuidad del proyecto
Propósito Mejorar la administración en los Colegios de Profesionales de la ciudad de Tarija	Al finalizar la ejecución del proyecto se encuentra disponible un medio que permitirá mejorar la administración en los Colegios de Profesionales.	Informe del director, avalado por el director del departamento de informática y sistemas de la UAJMS	La directiva de cada uno de los Colegios cuenta con una conexión de Internet. Los presidentes de los Colegios socializan el sitio Web a sus afiliados.
Componentes /Resultados 1. Desarrollar la plataforma de blogs para los Colegios de Profesionales 2. Socializar y capacitar a los Colegios de Profesionales sobre la usabilidad de BLOGPROF y otras herramientas Web 2.0	Al finalizar el proyecto, se ha desarrollado la plataforma de blogs de acuerdo a la ingeniería de requerimientos, orientado a los Colegios de profesionales. Al finalizar el proyecto, se realizo la socialización y se ha realizado al menos un taller de capacitación con miembros de los colegios de profesionales.	Informe de Conformidad por parte de los Colegios de Profesionales con los cuales se realizaron las pruebas, avalado por el Departamento de Informática y Sistemas Informe del director del proyecto, avalado por el director del departamento de informática y sistemas.	Los colegios de profesionales participan activamente en la ejecución del proyecto Los colegiados participan de los talleres.

ACTIVIDADES 1.1. Desarrollar la plataforma de blogs para los Colegios de Profesionales 1.1. Ingeniería de Requerimientos 1.2. Análisis y Diseño de la plataforma de Blogs 1.3. Instrumentación 1.4. Pruebas 2. Socializar y capacitar a los Colegios de Profesionales sobre la temática de administración y usabilidad de los sitios Blog. 2.1. Planificar el proceso de socialización 2.2. Realizar el proceso de socialización 2.3. Planificación 2.4. Ejecución de los talleres 2.5. Elaboración del informe final	Presupuesto por partidas	Informe del presupuesto ejecutado.	Los desembolsos se realizan de acuerdo al cronograma

Tabla 6. Matriz de Marco Lógico

I.4.1.6. Objetivo general

Mejorar la administración de los Colegios de Profesionales de la ciudad de Tarija para que estos puedan cumplir adecuadamente su rol como institución, con sus afiliados y la sociedad.

I.4.1.7. Objetivos Específicos

Los objetivos específicos que se conseguirán finalizado el proyecto son los siguientes:

1. Desarrollar una Plataforma de Blogs para Colegio de Profesionales.
2. Socializar y Capacitar a los Colegios de Profesionales sobre la temática de "Uso de herramientas Web 2.0 y BLOGPROF".

I.4.1.8. Vinculaciones de Objetivos Específicos e Indicadores de Resultados

OBJETIVOS ESPECIFICOS	INDICADORES DE RESULTADOS
Plataforma de Administración de Blogs para Colegio de Profesionales instalado y en funcionamiento.	Al finalizar el proyecto, los Colegios de Profesionales con los cuales de realizaron las pruebas muestran su satisfacción con el funcionamiento de la plataforma desarrollada.
Capacitación realizada a los Colegios de Profesionales sobre la temática de administración y usabilidad de los sitios Blog.	Al finalizar el proyecto, se capacita al menos al 60 % de los miembros de cada Colegio incorporado en el proyecto

Tabla 7. Cuadro de vinculaciones de objetivos específicos e indicadores de resultados

I.4.2. Equipo del Proyecto, Actividades y Recursos

I.4.2.1. Equipo del proyecto

Marianela Alex Soliz

Unidades de gestión

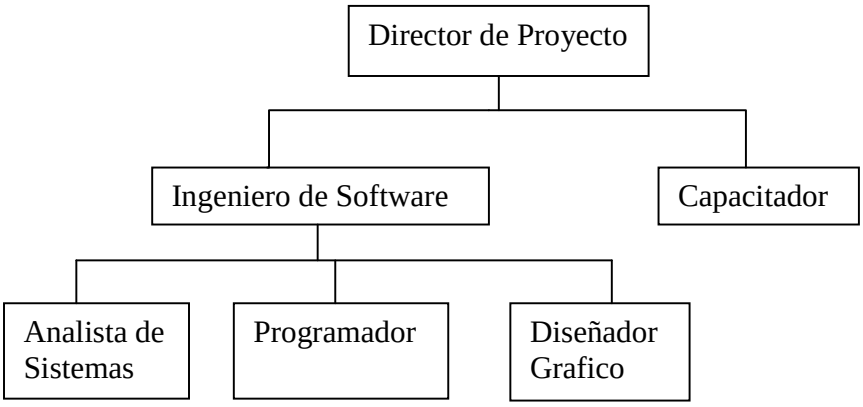

Figura 4. Organigrama de la estructura organizativa del proyecto

I.4.2.2. Actividades

Nombre del Cargo: Director
Funciones - Elaborar el manual de funciones - Preparar el ambiente de trabajo del equipo de proyecto - Analizar la situación problemática de las instituciones a las que favorecerá el proyecto - Identificar el propósito del proyecto en la elaboración del perfil de proyecto - Planificar las actividades a realizar en el transcurso del proyecto - Realizar un seguimiento de las actividades planificadas al inicio del proyecto - Gestionar el presupuesto del proyecto asignado para el desarrollo del mismo - Realizar gestiones para conseguir financiamiento externo del proyecto - Coordinar las interacciones con las instituciones beneficiadas - Asegurar la entrega de los resultados esperados
Relaciones **Supervisa** a Ingeniero de Software y Capacitador

Tabla 8. Cuadro de Actividades de Director

Nombre del Cargo: Ingeniero de Software
Funciones - Hacer el seguimiento correspondiente en el desarrollo del software (Plataforma de Blogs) - Realiza el estudio de viabilidad del software - Elaborar la documentación del software - Definir la arquitectura del sistema o software - Gestionar y priorizar los requerimientos del software - Gestionar los riesgos en el desarrollo del software - Gestionar los cambios y la configuración en el desarrollo del software - Elaborar el modelo de datos - Preparar las pruebas funcionales al software - Elaborar modelos de implementación y de despliegue
Relaciones **Reporta a:** Director de Proyecto **Supervisa a:** Analista de Sistemas, Programador y Diseñador Grafico

Tabla 9. Cuadro de Actividades de Ingeniero de Software

Nombre del Cargo: Analista de Sistemas
Funciones - Captura y selección de Requerimientos - Especificación y validación de los Requerimientos - Identificar las funcionalidades del software según requerimientos - Transformar los requerimientos en el Modelo de Análisis y Diseño - Colaborar en la elaboración del modelo de datos y pruebas funcionales - Elaborar el glosario de términos utilizados en la documentación del software
Relaciones **Reporta a:** Ingeniero de Software **Se relaciona con:** Programador

Tabla 10. Cuadro de Actividades de Analista de Sistemas

Nombre del Cargo: Diseñador Grafico
Funciones - Diseñar el entorno grafico de las interfaces de usuario - Diseñar las imágenes, logos del software
Relaciones **Reporta a:** Ingeniero de Software **Se relaciona con:** Programador

Tabla 11. Cuadro de Actividades de Diseñador Grafico

Nombre del Cargo: Programador
Funciones - Construcción del software según especificaciones del análisis y diseño - Implementar el modelo de datos en un esquema físico y determina el SGBD a utilizar - Integrar la base de datos con el software desarrollado - Adapta las funcionalidades del software en las interfaces - Colaborar en la elaboración del modelo de datos y pruebas funcionales - Colabora en el modelo de implementación y despliegue
Relaciones **Reporta a:** Ingeniero de Software **Se relaciona con:** Analista de Sistemas y Diseñador Grafico

Tabla 12. Cuadro de Actividades de Programador

Nombre del Cargo: Capacitador
Funciones - Definir los grupos que serán capacitados - Establecer los indicadores iníciales del proceso. - Determinar el titulo del taller, los objetivos, el contenido temático y el tiempo de ejecución. - Definir la profundidad requerida con la que se abordara el contenido temático. - Desarrollar los talleres de capacitación. - Evaluar el impacto y los indicadores iníciales para después realizar el informe final. -
Relaciones **Reporta a:** Director de Proyecto

Tabla 13. Cuadro de Actividades de Capacitador

I.4.2.3. Recursos

OBJETIVOS ACADEMICOS	RECURSOS (VALOR Y JUSTIFICACION)
Plataforma de Blogs par Colegios de Profesionales	5060 Bs.
Capacitación sobre el uso de herramientas Web 2.0 y BLOGPROF	2800 Bs.

Tabla 14. Recursos

I.4.2.4. Cronograma de Actividades

Nombre de la tarea	Duración	Comienzo	Fin	Costo
Desarrollo de la Plataforma de Blogs	**214 días**			
- Ingeniería de Requerimientos	30 días	01/04/09	30/04/09	900.00 Bs
- Análisis y Diseño	31 días	01/05/09	31/05/09	1330.00 Bs
- Instrumentación	92 días	01/06/09	31/09/09	1800.00 Bs
- Instalación	61 días	01/10/09	31/11/09	1030.00 Bs
Socialización y Capacitación	**28 días**			
✓ Socialización de BLOGPROF	10 días	01/12/09	10/12/09	
✓ Planificación de la Capacitación	5 días	11/12/09	15/12/09	2180.00 Bs
✓ Ejecución de la Capacitación	3 días	16/12/09	18/12/09	
✓ Elaboración del Informe Final	10 días	19/12/09	29/12/09	
Total	**242 días**			**7240 Bs.**

Tabla 15. Cronograma de Actividades

II. CAPITULO : MARCO TEORICO

II.1 Organización

Figura 5: Organización de Personas

Una organización es el establecimiento de la estructura necesaria para la sistematización racional de los recursos, mediante la determinación de jerarquías, disposición, correlación y agrupación de actividades, con el fin de poder realizar y simplificar las funciones del grupo social. Es un sistema de actividades conscientemente coordinadas formado por dos o más personas. La cooperación entre ellas es esencial para la existencia de la organización.

La sociedad moderna está conformada por organizaciones de las cuales tienen sistemas muy complejos y diferentes. Ejemplos: industrias, empresas comerciales y de servicios, organizaciones militares y gubernamentales, instituciones públicas y privadas, iglesias, entre otros. En resumen, las organizaciones se conforman para que las personas que trabajan allí, alcancen objetivos que no podrían lograr de manera aislada, debido a las limitaciones individuales.

II.2 Administración

La administración es un instrumento para llegar a un fin, ya que su finalidad es eminentemente práctica y mediante ésta se busca obtener resultados determinados previamente establecidos, por medio de la planeación, organización, delegación de funciones, integración de personal, dirección y control de otras personas, creando y manteniendo un ambiente en el cual la persona se pueda desempeñar entusiastamente en conjunto con otras, sacando a relucir su potencial.

La administración es intangible, su presencia queda evidenciada por el resultado de los esfuerzos, se aplica a todo tipo de organismos sociales (estado, ejército, empresas, iglesias, familia, etc.), porque en él siempre tiene que existir coordinación sistemática de medios y debe ser aplicada de manera eficaz y eficiente.

II.3 Colegios de Profesionales

Origen

El origen de las actuales estructuras colegiales se remonta a mediados del siglo XV, cuando los profesionales de los llamados artes mayores se agruparon con dos finalidades: formar a las siguientes generaciones, normalmente los hijos de los propios miembros del Colegio, y participar en las instituciones del estado medieval, nombrando, junto con gremios, los representantes del tercer brazo de las cortes, tras los de la iglesia y la nobleza.

Definición

Los Colegios Profesionales son Corporaciones de derecho público, amparadas por la ley y reconocidas por el Estado, con personalidad jurídica propia y plena capacidad para el cumplimiento de sus fines y que son fines esenciales de estas Corporaciones la ordenación del ejercicio de las profesiones, la representación exclusiva de las mismas y la defensa de los intereses profesionales de los colegiados, todo ello sin perjuicio de la competencia de la Administración Pública por razón de la relación funcionarial y de las específicas de la Organización Sindical en materia de relaciones laborales.

El mejor control del ejercicio profesional exige de la colegiación obligatoria, que se manifiesta como una medida de eficacia, pero también como una garantía para los ciudadanos que pueden verse afectados por la actividad profesional y que lo normal, por ser de interés público, es que seamos todos los ciudadanos.

Funciones

Las funciones más principales en los Colegios de Profesionales se pueden resumir en los siguientes puntos:

- Representar a la profesión y el fortalecimiento Institucional
- Cumplir y hacer cumplir a los colegiados/as las leyes generales y especiales.
- La función primordial que les queda a los colegios, entendidos como corporación de derecho público, es asegurar que los profesionales que la Sociedad ha formado en la Universidad le den realmente el servicio que la Sociedad espera.
- Velar por las condiciones de trabajo de sus integrantes, naturalmente, sin interferir con las de los demás profesionales y sin entrar a ejercer competencias que correspondan a otras organizaciones como los sindicatos.
- Ofrecer los medios técnicos, bibliotecas, recursos necesarios para determinadas tareas, acceso a instalaciones especiales, etcétera, son servicios que los Colegios deben prestar para que sus miembros puedan ejercer su profesión en las mejores condiciones posibles.

II.4 Internet

Internet es un conjunto descentralizado de redes de comunicación interconectadas, que utilizan la familia de protocolos TCP/IP, garantizando que las redes físicas heterogéneas que la componen funcionen como una red lógica única, de alcance mundial. Sus orígenes se remontan a 1969, cuando se estableció la primera conexión de computadoras, conocida como ARPANET, entre tres universidades en California y una en Utah, Estados Unidos.

El internet ha sido creado con la intención de que los individuos y los grupos se relacionen a través de una serie de medios de comunicación más o menos nuevos (correo electrónico, news, listas de distribución, videoconferencia, chats...) o más o menos viejos (como una conversación telefónica, poner un fax, etc.,) y también incluye dentro de sí a los denominados medios de comunicación de masas (radio, televisión, periódicos y revistas "on-line", cine, la omnipresente publicidad, etc.).

II.5 Web 2.0

Figura 6: Web 2.0

El concepto original de la Web, llamado *Web 1.0* era páginas estáticas HTML que no eran actualizadas frecuentemente. El éxito de las punto-com dependía de webs más dinámicas (a veces llamadas *Web 1.5*) donde los CMS Sistema de gestión de contenidos (Content Management System en inglés, abreviado CMS) servían páginas HTML dinámicas creadas al vuelo desde una actualizada base de datos. En ambos sentidos, el conseguir *hits* (visitas) y la estética visual eran considerados como factores importantes.

La Web 2.0 es la representación de la evolución de las aplicaciones tradicionales hacia aplicaciones web enfocadas al usuario final. Esta nueva filosofía o actitud en el desarrollo de sitios Web surge gracias al desarrollo tecnológico, no sólo de nuevas arquitecturas software, sino también en la evolución tecnológica de las infraestructuras de telecomunicaciones.

La Web 2.0 posibilita la participación colectiva, es decir el internauta puede compartir, conversar, generar artículos o contenidos, comentar artículos creados por otros usuarios, de esta manera el usuario de internet ya no es solamente un consumidor sino que pasa a convertirse en colaborador utilizando las herramientas Web 2.0. A continuación se explican sus características más esenciales.

II.5.1. Tecnología Web 2.0

Existe una gran cantidad de utilidades, herramientas, framework para el desarrollo web con el uso de las siguientes técnicas: CSS, marcado XHTML validado semánticamente, técnicas de aplicación como AJAX, soporte para postear en un blog, la redifusión del contenido de una Web, usando protocolos estandarizados como RSS y ATOM que permitan a los usuarios finales usar el contenido de la web en otro contexto. La arquitectura central de la Web no varía en lo absoluto los servidores, servicios y protocolos son los mismos. El incremento en la capacidad de almacenamiento y procesamiento de los datos han hecho posible el desarrollo de proyectos de esta naturaleza.

II.5.2. Usabilidad Web 2.0

En los proyectos de esta categoría, la usabilidad pasa a ser un requisito indispensable, la usabilidad es una característica común a todos estos sistemas y es un factor clave en la velocidad de su difusión. La Usabilidad ha sido instrumental para que las personas participen cada vez más activamente de la Web.

II.5.3. Participación Web 2.0

La participación es un elemento central en la mayoría de los proyectos Web 2.0. Los modelos de negocio de estos sitios están centrados en los usuarios. Muchos de estos sitios forman espacios de participación entre personas, dan la estructura para la creación de comunidades electrónicas y esperan a que los usuarios finales sean quienes dan vida al sistema.

II.6 Herramientas Sociales

Programas que apoyan, extienden y agregan valor al comportamiento social del ser humano (Coates, 2006). No son productos sino soluciones, presenta soluciones simples y atienden necesidades de las personas. Sitios revolucionarios que rompen con las normas de lo establecido para inventar nuevas formas de operar. Actualmente, este tipo de sitios ha reunido una masa crítica importante, con ejemplos que han tenido gran influencia más allá de la misma Web.

Los programas se ofrecen como un servicio a través de una plataforma web y no como un producto cerrado para un sistema operativo determinado. Por tanto, la capacidad de actualizarlo se aprovecha para mejorar la aplicación (y con ello el servicio) constantemente. De allí que los ejemplos más claros de herramientas sociales estén en beta constante

II.6.1. Blogs

Figura 7: Servicios de Blogs más conocidas

Antes de definir una plataforma de blogs necesitamos saber que es un **blog.** Los blogs son uno de los sistemas representativos de la Web 2.0. Este término inglés *blog* o *weblog* proviene de las palabras *web* y *log* ('log' en inglés = *diario*). Estos sistemas de publicación personal comenzaron como una herramienta de unos pocos usuarios avanzados, actualmente son páginas en el Internet muy fáciles de crear y actualizar.

Usualmente, en cada artículo de un blog, los lectores pueden escribir sus comentarios y el autor darles respuesta, de forma que es posible establecer un diálogo. Los blogs permiten a todo el mundo ser leído hasta en los más remotos lugares, pero antes que nada hay que aprender a publicarlos. Para ello existen diferentes plataformas, y el usuario deberá escoger el que mejor se adapte a sus características.

Existen dos formas de crear un blog:

- Usar **plataformas gratuitas** como la coctelera, blogger, bitacoras.com, 1blogs, nireblog, vox, TypePad, zeppia que tienen las funcionalidades básicas. Estos servicios nos permiten alojar nuestro blog gratuitamente, lo que significa que ofrecen al usuario su propio servidor de 'hosting' (almacenaje), generalmente bajo su propio dominio. La dirección url del blog es una combinación del nombre elegido por el usuario y el nombre del servicio.

- Instalar **CMS** (sistemas de gestión de contenido) en servidor, tal como b2 evolution, WordPress, Movable Type, etc. Requieren del usuario que disponga de un servidor propio donde instalar el software correspondiente. Para hacerlo, hay que descargar el software y ponerlo en marcha sobre el servidor, un proceso que requiere algo de maña, tiempo y un cliente FTP.

La capacidad de las plataformas depende de muchos factores pero, a medida que avanzan los años, cada vez está más claro que el factor más importante es la comunidad.

II.6.2. Redes Sociales

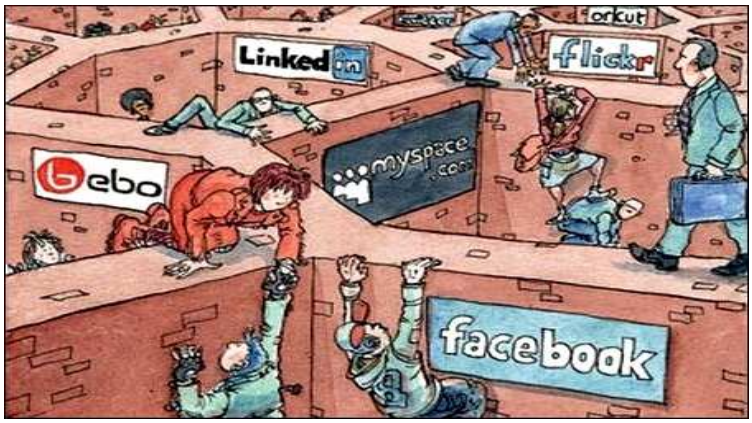

Figura 8: Redes Sociales

Es una estructura social que se puede representar en forma de uno o varios grafos en el cual los nodos representan individuos y las aristas relaciones entre ellos. Las relaciones pueden ser de distinto tipo, como intercambios financieros, amistad, o rutas aéreas. También es el medio de interacción de distintas personas como por ejemplo juegos en línea, chats, foros, espacios, etc.

El origen de las redes sociales se remonta, al menos, a 1995, cuando Randy Conrads crea el sitio web classmates.com. En 2002 comienzan a aparecer sitios web promocionando las redes de *círculos de amigos* en línea cuando el término se empleaba para describir las relaciones en las comunidades virtuales, y se hizo popular en 2003 con la llegada de sitios tales como MySpace o Xing.

En general, los servicios de redes sociales permiten a los usuarios crear un perfil para ellos mismos, y se pueden dividir en dos grandes categorías: la creación de redes sociales internas (ISN) y la creación de redes sociales externos (ESN) como, por ejemplo, MySpace, Bebo y Facebook. Ambos tipos pueden aumentar el sentimiento de comunidad entre las personas. El ISN es un cerrado y privado comunidad que se compone de un grupo de personas dentro de una empresa, asociación, sociedad, el proveedor de educación y organización, o incluso una "invitación", creado por un grupo de usuarios en un ESN. El ESN es una red abierta y la disposición de todos los usuarios de la web para comunicarse y están diseñados para atraer a los anunciantes. Los usuarios pueden añadir una imagen de sí mismos y con frecuencia pueden ser "amigos" con otros usuarios.

Algunas redes sociales tienen funciones adicionales, como la capacidad de crear grupos que comparten intereses comunes o afiliaciones, subir videos, y celebrar debates en los foros. Geosocial networking da la opción de cartografía de los servicios de Internet para organizar la participación de los usuarios en torno a las características geográficas y sus atributos.

II.6.3. Otras herramientas sociales

II.6.3.1. Marcadores Sociales

Los marcadores sociales son una forma sencilla y popular de almacenar, clasificar y compartir enlaces en Internet o en una Intranet. En un sistema de marcadores sociales los usuarios guardan una lista de recursos de Internet que consideran útiles. Las listas pueden ser accesibles públicamente o de forma privada. Otras personas con intereses similares pueden ver los enlaces por categorías, etiquetas o al azar. Entre los más populares se puede mencionar Furl, Del.icios.us, Blinklist , CiteUlike.

II.6.3.2. Redifusión de Contenido

La primera y más importante evolución de la Web 2.0 se refiere a la redifusión del contenido de una Web, usando protocolos estandarizados que permitan a los usuarios finales usar el contenido de la web en otro contexto, ya sea en otra web, en un conector de navegador o en una aplicación de escritorio. Entre los protocolos que permiten redifundir se encuentra el RSS, y Atom, todos ellos basados en XML.

II.6.3.3. Aplicaciones de Publicación Colectiva

Una API permite extraer la información de la base de datos de un gran servicio online (Google, Amazon, Flickr) e incorporarla a cualquier otra nueva aplicación que hayamos creado. Esto permite que podamos, por ejemplo, incluir una caja de búsqueda de Google en cualquier otra página.

Existe una variedad de herramientas que permiten compartir e intercambiar información en contenidos, pudiendo ser esta información de diferentes tipos como ser fotos, videos, imágenes, artículos, archivos, presentaciones, etc. Blogs, Wikis, YouTube, Podcast, Flickr, Vodcast, Slideshare, son algunas de las herramientas de esta categoría.

II.7 Ingeniería de Software

La Ingeniería del software es una disciplina o área de la Informática, que ofrece métodos y técnicas para desarrollar y mantener software de calidad que resuelven problemas de todo tipo. Hoy día es cada vez más frecuente la consideración de la Ingeniería del Software como una nueva área de la Ingeniería, y el Ingeniero del Software comienza a ser una profesión implantada en el mundo laboral internacional, con derechos, deberes y responsabilidades que cumplir, junto a una, ya, reconocida consideración social en el mundo empresarial y, por suerte, para esas personas con brillante futuro.

II.8 Metodologías

Las metodologías imponen un proceso disciplinado sobre el desarrollo de software con el fin de hacerlo más predecible y eficiente. Lo hacen desarrollando un proceso detallado con un fuerte énfasis en la planificación inspirado por otras disciplinas de la ingeniería. De nada sirven buenas notaciones y herramientas si no se proveen directivas para su aplicación.

II.8.1. Metodologías Agiles

Las Metodologías agiles o "ligeras" constituyen un nuevo enfoque en el desarrollo de software, mejor aceptado por los desarrolladores de proyectos de software que las metodologías convencionales, debido a la simplicidad de sus reglas y prácticas, su orientación a equipos pequeños de desarrollo, su flexibilidad ante los cambios y su ideología de colaboración.

Son menos orientados al documento, exigiendo una cantidad más pequeña de documentación para una tarea dada. El software que funciona es la medida principal de progreso. La atención continua a la calidad técnica y al buen diseño mejora la agilidad. La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor.

II.9 Metodología AUP

El Proceso Unificado Ágil (AUP) es una versión simplificada de RUP (Proceso Unificado Racional). Esta metodología describe un fácil y entendible acercamiento al desarrollo de software usando técnicas agiles y conceptos todavía no utilizados de RUP. Esta metodología es una combinación entre XP y el tradicional RUP, es un proceso ágil sin embargo explícitamente incluye actividades y artefactos los cuales son acostumbrados. La metodología AUP adopta muchas de las técnicas agiles de XP y otros procesos agiles todavía retenidos de RUP. Se encuentra bajo los principios de Agile Alliance.

II.10 Ingeniería de Requerimientos

"Ingeniería de Requerimientos ayuda a los ingenieros de software a entender mejor el problema en cuya solución trabajarán. Incluye el conjunto de tareas que conducen a comprender cuál será el impacto del software sobre el negocio, qué es lo que el cliente quiere y cómo interactuarán los usuarios finales con el software". (Pressman, 2006: 155).

Antes de desarrollar o implantar sistemas, es necesario hacer un estudio de las necesidades o requerimientos del entorno que requiere de un sistema, haciendo un análisis de los procesos de trabajo desde un enfoque centrado en cómo fluye y se maneja la información o datos a través de los procesos realizados por el grupo en cuestión. Un paso importante en este análisis, es el modelado de los procesos

II.11 UML (Unified Modeling Language)

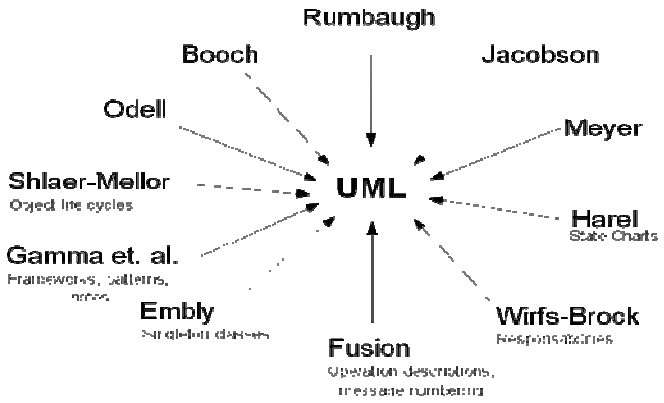

Figura 9: UML unifica enfoques OO

UML es un Lenguaje de Modelado Unificado basado en una notación gráfica la cual permite: especificar, construir, visualizar y documentar los objetos de un sistema programado. Permite representar en mayor o menor medida todas las fases de un proyecto informático: desde el análisis con los casos de uso, el diseño con los diagramas de clases, objetos, etc., hasta la implementación y configuración con los diagramas de despliegue.

Este lenguaje es el resultado de la unificación de los métodos de modelado orientados a objetos de Booch, Rumbaugh (OMT: Object Modeling Technique) y Jacobson (OOSE: Object-Oriented Sotfware Engineering). UML aglutina enfoques orientados a objetos.

Para poder representar correctamente un sistema, UML ofrece trece diagramas para visualizar el sistema desde varias perspectivas. Los diagramas más importantes para diseñar un sistema son los siguientes:

II.11.1. Diagrama de Casos de Uso

Un diagrama de Casos de Uso muestra las distintas operaciones que se esperan de una aplicación o sistema y cómo se relaciona con su entorno (usuario u otras aplicaciones). Es una herramienta esencial para la captura de requerimientos y para la planificación y control de un proyecto interactivo.

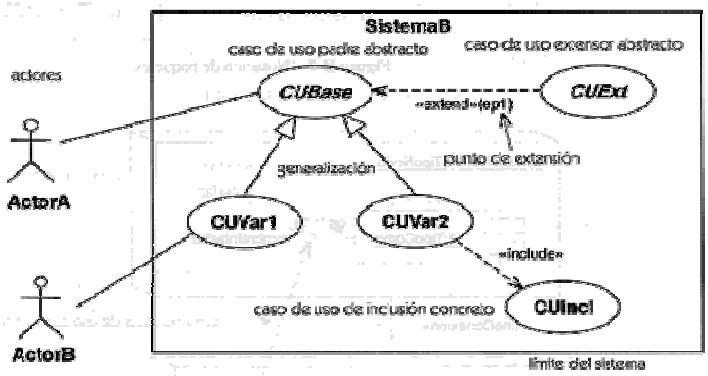

Figura 10: Ejemplo UML: Diagrama de Casos de Uso

Los **casos de Uso** se representan en el diagrama por medio de una elipse que denota un requerimiento solucionando por el sistema. Cada caso de uso es una operación completa desarrollada por los actores y por el sistema en un diálogo. El conjunto de casos de uso representa la totalidad de operaciones desarrolladas por el sistema.

Un **actor** es un usuario del sistema, que necesita o usa alguno de los casos de uso. Un usuario puede jugar más de un rol. Un solo actor puede actuar en muchos casos de uso; recíprocamente, un caso de uso puede tener varios actores. Los actores no necesitan ser humanos pueden ser sistemas externos que necesitan alguna información del sistema actual.

Las relaciones entre los actores y los casos de uso de un determinado sistema puede ser uno de los siguientes tipos:

include: Esta relación sucede cuando un caso de uso incorpora en su funcionamiento otro caso de uso, la ocurrencia del caso de uso incorporado es obligatoria para el buen funcionamiento del caso de uso que lo incorpora.

extend: Esta relación es una extensión de un caso de uso definido, el caso de uso que se extiende describe un comportamiento opcional del sistema.

II.11.2. Diagrama de Secuencia

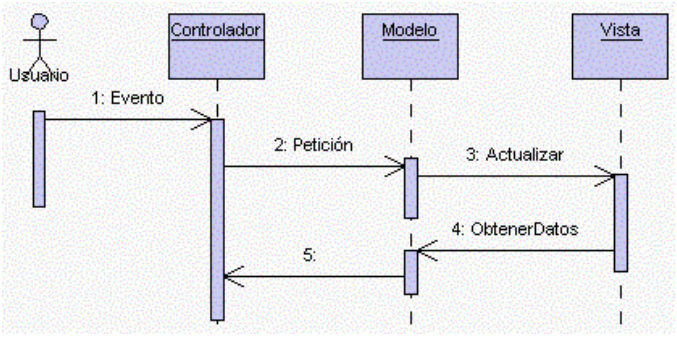

Figura 11: Ejemplo UML: Diagrama de Secuencia

Un diagrama de secuencia muestra la interacción de un conjunto de objetos de una aplicación a través del tiempo. Esta descripción es importante porque puede dar detalle a los casos de uso, aclarándolos al nivel de mensajes de los objetos existentes, como también muestra el uso de los mensajes de las clases diseñadas en el contexto de una operación.

El eje vertical representa el tiempo, y en el eje horizontal se colocan los objetos y actores participantes en la interacción, sin un orden prefijado, aunque por orden lo usual es colocar los objetos de izquierda a derecha y en la parte superior. Cada objeto o actor tiene una línea vertical, y los mensajes se representan mediante flechas entre los distintos objetos.

II.11.3. Diagrama de Clases

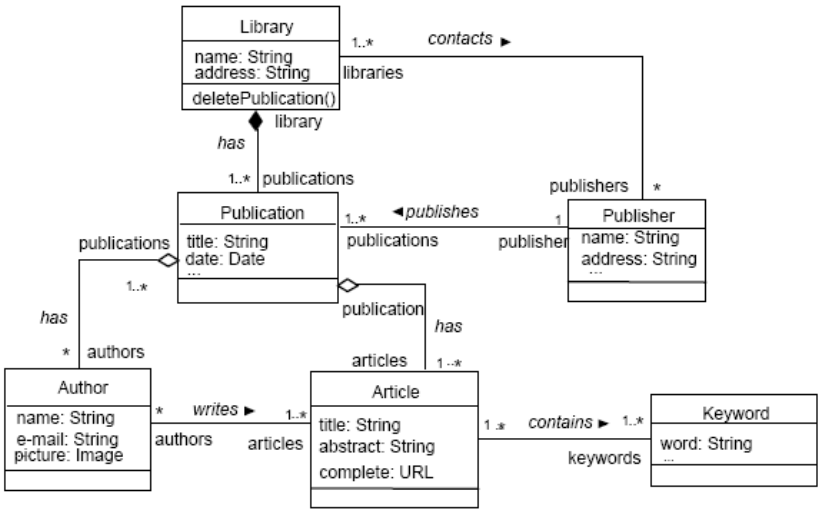

Figura 12: Ejemplo UML: Diagrama de Clases

Un diagrama de clases o estructura estática muestra el conjunto de clases y objetos importantes que forman parte de un sistema, junto con las relaciones existentes entre clases y objetos. Muestra de una manera estática la estructura de información del sistema y la visibilidad que tiene cada una de las clases, dada por sus relaciones con los demás en el modelo.

Una **clase** representa un conjunto de entidades que tienen en común propiedades, operaciones, relaciones y semántica. Una clase es un constructor que define la estructura y comportamiento de una colección de objeto denominados instancia de la clase. En UML la clase está representada por un rectángulo con tres divisiones internas, son los elementos fundamentales del diagrama.

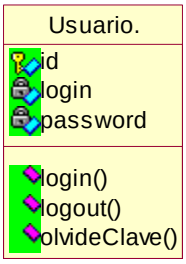

Figura 13: Clase

Los tipos de asociaciones entre clases presentes en un diagrama estático son:

La **Composición** es una asociación fuerte que implica dependencia existencial. El elemento dependiente desaparece al destruirse el que lo contiene y si es de cardinalidad 1, es creado al mismo tiempo. Se puede decir que el objeto contenido es parte constitutiva y vital del que lo contiene. Se denota dibujando un rombo del lado de la clase que contiene a la otra en la relación.

La **Agregación** relaciona una clase ya ensamblada con una clase componente. Es también una relación de composición menos fuerte (no se exige dependencia existencial) y se denota por un rombo sin rellenar en uno de los extremos.

La **Generalización** es un proceso de abstracción en el cual un conjunto de clases existentes, que tienen atributos y métodos comunes, es referido por una clase genérica a un nivel mayor de abstracción. La relación de generalización denota una relación de herencia entre clases. Se representa dibujando un triángulo sin rellenar en el lado de la superclase. La subclase hereda todos los atributos y mensajes descritos en la superclase.

II.11.3.1. Diagrama de Componentes

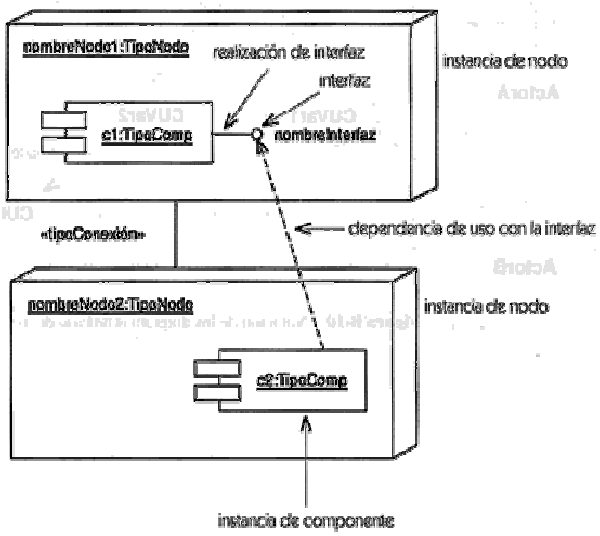

Figura 14: Ejemplo UML: Diagrama de Componentes

Los diagramas de componentes describen los elementos físicos reemplazables del sistema y sus relaciones. Muestran las opciones de realización incluyendo código fuente, binario y ejecutable.

Los **componentes** representan todos los tipos de elementos software que entran en la fabricación de aplicaciones informáticas. Pueden ser simples archivos, librerías, bibliotecas cargadas dinámicamente, etc. Las relaciones de dependencia se utilizan en los diagramas de componentes para indicar que un componente utiliza los servicios ofrecidos por otro componente.

II.11.3.2. Diagrama de Despliegue

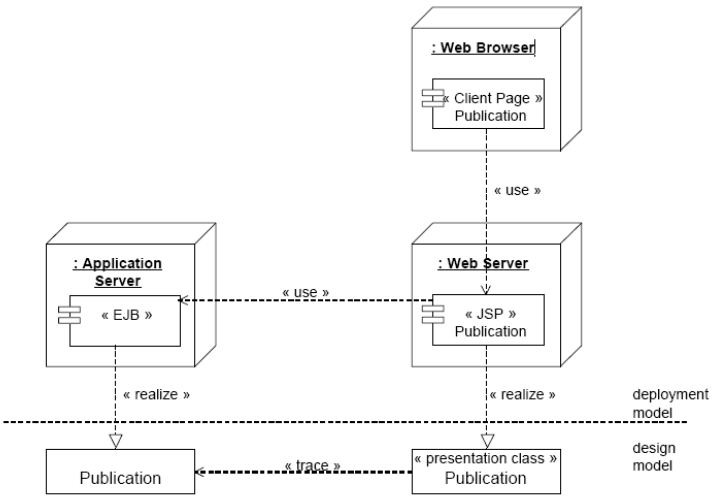

Figura 15: Ejemplo UML: Diagrama de Despliegue

Este tipo de diagramas se enfoca específicamente al hardware de un sistema determinado. El elemento primordial del hardware es un *nodo,* que es un nombre genérico para todo tipo de recurso de cómputo.
Dentro del cubo se puede introducir información sobre el nodo, que puede ser simplemente texto o inclusive componentes, usando los diagramas de componentes anteriormente ejemplificados.

El resto de diagramas muestran distintos aspectos del sistema a modelar. Para modelar el comportamiento dinámico del sistema están los de, *colaboración*, *estados* y *actividades*. Los *diagramas de paquetes, objetos* están enfocados a la implementación del sistema.

II.12 Herramientas CASE

De acuerdo con Kendall y Kendall la ingeniería de sistemas asistida por ordenador es la aplicación de tecnología informática a las actividades, las técnicas y las metodologías propias de desarrollo, su objetivo es acelerar el proceso para el que han sido diseñadas, en el caso de CASE para automatizar o apoyar una o mas fases del ciclo de vida del desarrollo de sistemas.

Cuando se hace la planificación de la base de datos, la primera etapa del ciclo de vida de las aplicaciones de bases de datos, también se puede escoger una herramienta CASE (Computer-Aided Software Engineering) que permita llevar a cabo el resto de tareas del modo más eficiente y efectivo posible. Una herramienta CASE suele incluir:

- Herramientas que permitan desarrollar el modelo de datos corporativo, así como los esquemas conceptual y lógico.
- Herramientas para desarrollar los prototipos de las aplicaciones.
- Un diccionario de datos para almacenar información sobre los datos de la aplicación de bases de datos.
- Herramientas de diseño para dar apoyo al análisis de datos.

El uso de las herramientas CASE puede mejorar la productividad en el desarrollo de una aplicación de bases de datos.

II.12.1. Rational Rose

Rational Rose es una herramienta para "**modelado visual**", que forma parte de un conjunto más amplio de herramientas que juntas cubren todo el ciclo de vida del desarrollo de software.

IBM Rational Rose Enterprise es uno de los productos más completos de la familia Rational Rose. Todos los productos de Rational Rose dan soporte a Unified Modeling Language (UML), pero no

son compatibles con las mismas tecnologías de implementación.

Rational Rose Enterprise es un entorno de modelado que permite generar código a partir de modelos Ada, ANSI C++, C++, CORBA, Java/J2EE, Visual C++ y Visual Basic. Al igual que todos los productos de Rational Rose, ofrece un lenguaje de modelado común que agiliza la creación del software.

II.12.2. Enterprise Architect

Es una herramienta CASE con un conjunto de ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del Ciclo de Vida de desarrollo de un Software (Investigación Preliminar, Análisis, Diseño, Implementación e Instalación.).

II.13 Programación Orientada a Objetos

La Programación Orientada a Objetos (POO u OOP según sus siglas en inglés) es un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas de computadora. Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo y encapsulamiento. Su uso se popularizó a principios de la década de 1990. Actualmente son muchos los lenguajes de programación que soportan la orientación a objetos.

II.14 Modelo Vista Controlador

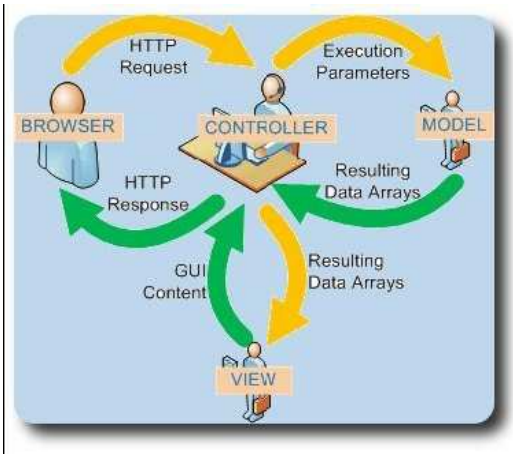

Figura 16: Modelo Vista Controlador

La arquitectura MVC separa la lógica de negocio (el modelo) y la presentación (la vista) por lo que se consigue un mantenimiento más sencillo de las aplicaciones. Si por ejemplo una misma aplicación debe ejecutarse

tanto en un navegador estándar como un navegador de un dispositivo móvil, solamente es necesario crear una vista nueva para cada dispositivo; manteniendo el controlador y el modelo original. El controlador se encarga de aislar al modelo y a la vista de los detalles del protocolo utilizado para las peticiones (HTTP, consola de comandos, email, etc.). El modelo se encarga de la abstracción de la lógica relacionada con los datos, haciendo que la vista y las acciones sean independientes de, por ejemplo, el tipo de gestor de bases de datos utilizado por la aplicación.

II.15 Lenguajes de Programación

Un lenguaje de programación permite a uno o más programadores especificar de manera precisa sobre qué datos debe operar una computadora, cómo estos datos deben ser almacenados o transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. Todo esto, a través de un lenguaje que intenta estar *relativamente* próximo al lenguaje humano o natural, tal como sucede con el lenguaje Léxico. Una característica relevante de los lenguajes de programación es precisamente que más de un programador puedan tener un conjunto común de instrucciones que puedan ser comprendidas entre ellos para realizar la construcción del programa de forma colaborativa.

II.15.1. HTML

Es el lenguaje con el que se escriben las páginas Web. Las páginas Web pueden ser vistas por el usuario mediante un tipo de aplicación llamada navegador. Podemos decir por lo tanto que HTML es el lenguaje usado por los navegadores para mostrar las páginas Web al usuario, siendo hoy en dia la interfaz más extendida en la red. HTML se utilizara para la capa de presentación que utiliza plantillas.

II.15.2. JAVA

Java es un lenguaje de programación con un paradigma orientado a objetos. Apareció en el año 1991 desarrollado por la empresa Sun Microsystems, Java fue desarrollado basándose en el lenguaje C y C++, a diferencia de estos lenguajes tiene un modelo de objetos más simple.

Los programas Java generalmente pasan por cinco fases antes de ejecutarse, estas son: editar, compilar, cargar, verificar y ejecutar.

Java ya no es solamente un lenguaje orientado a la programación de sistemas escritorio, ahora también se lo puede utilizar para desarrollar servicios Web utilizando Servlets como programas CGI (PHP, Perl, Ruby, etc) en el servidor para atender las peticiones de los clientes.

II.15.3. Groovy

Figura 17: Groovy

Groovy es un lenguaje dinámico para la Máquina Virtual Java. Se puede compilar o interpretar. En cualquier caso genera el mismo código bytecode que un compilador Java. Comparte con Java la misma orientación a objetos. Se puede integrar Java y Groovy muy fácilmente, hacer una clase Groovy que herede de una clase Java, y viceversa. Se puede mezclar ambos lenguajes en un mismo proyecto.

Groovy es un lenguaje muy parecido a Java, porque tiene una sintaxis muy similar, pero también incorpora características de lenguajes como Ruby, Python o Smalltalk. La originalidad de Groovy probablemente venga del hecho de que Groovy es un lenguaje nuevo creado con la máquina virtual java en mente. No se trata de una simple copia de un lenguaje de scripting que ya existiera.

II.16 Framework

"Un framework simplifica el desarrollo de una aplicación mediante la automatización de algunos de los patrones utilizados para resolver las tareas comunes. Además, un framework proporciona estructura al código fuente, forzando al desarrollador a crear código más legible y más fácil de mantener. Por último, un framework facilita la programación de aplicaciones, ya que encapsula operaciones complejas en instrucciones sencillas".

II.16.1. Spring

El **Spring Framework** (también conocido simplemente como **Spring**) es un framework de código abierto de desarrollo de aplicaciones para la plataforma Java. La primera versión fue escrita por Rod Jonhson, quien lo lanzó primero con la publicación de su libro *Expert One-on-One Java EE Design and Development* (Wrox Press, octubre 2002). También hay una versión para la plataforma .NET, Spring.net.

A pesar de que Spring Framework no obliga a usar un modelo de programación en una alternativa y sustituto del modelo de Enterprise JavaBean. Por su diseño el framework ofrece mucha libertad a

los desarrolladores en Java y soluciones muy bien documentadas y fáciles de usar para las prácticas comunes en la industria.

II.16.2. Grails

Figura 18: Grails

Framework Open Source orientado a un proceso de desarrollo ágil. Lleva el paradigma de "codificación por convención" al lenguaje Groovy y está orientado al desarrollo de aplicaciones Web 2.0. Está basado en frameworks y tecnologías Java como Spring, Groovy, Hibernate, Sitemesh, Quartz Scheduling, Jetty, HSQLDB, entre otros. Las principales ventajas de este framework son que está orientado al desarrollo de sistemas web, implementando MVC con ORM, además de tener otras características útiles como ser scaffolding dinámico, taglibs simplificadas (muy simple en comparación al desarrollo de taglibs para jsp) e integración de herramientas de testing y logging. Posee integración con distintos IDEs, en particular con Eclipse, el IDE utilizado en este proyecto. GRAILS posee una curva de aprendizaje bastante rápida por ser muy intuitivo y simple, garantiza una productividad alta, por lo que es una excelente opción para proyectos con plazos muy cortos, o para prototipación de sistemas grandes. GRAILS tiene una arquitectura orientada a plugins y posee varios plugins muy útiles. Además GRAILS es un proyecto libre y de código abierto, y su licencia permite desarrollar tanto proyectos libres como proyectos comerciales.

II.17 Técnicas

Las técnicas son herramientas o utilidades que facilitan el desarrollo de aplicaciones Web, permiten agilizar el desarrollo de la aplicación.

II.17.1. Ajax

Ajax representa JavaScript Asíncrono Y XML, no es una nueva tecnología, es una combinación de tecnologías existentes como HTML, JavaScript, DHTML y DOM. Realmente es simplemente un acercamiento innovador y combina estas tecnologías para satisfacer las necesidades de aplicaciones en vías de desarrollo. AJAX colabora en el desarrollo de aplicaciones web para mejorar y conseguir un mejor resultado de cara al usuario final. Funciona en cualquier navegador

moderno, es perfectamente compatible con cualquier tipo de servidor estándar y lenguaje de programación Web. PHP, ASP. ASP.Net, Perl, JSP, Cold Fusion.

II.17.2. jQuery

Es un nuevo tipo de librerias de Javascript que permite simplificar la manera de interactuar con los documentos HTML, permitiendo manejar eventos, desarrollar animaciones, y agregar interacción con la tecnología AJAX a nuestras páginas web. jQuery está diseñado para cambiar la forma de escribir código JavaScript.

II.17.3. ModalBox

ModalBox es una técnica javascript para modernas creaciones (estilo Web 2.0), modal diálogos o secuencias de diálogos, sin usar popups y paginas recargadas. Esta inspirado en los diálogos de Mac OS X. ModalBox esta desarrollado con javascript purop basado en en prototype y scrip.aculo.us y XHTML/CSS valido. ModalBox usa AJAX para cargar el contenido.

II.17.4. JFreeChart

JFreeChart es una ibrería libre para generar cuadros estadísticos para la plataforma JAVA. Está diseñada para usars en aplicaciones, applets, servlets y JSP. JFreeChart esta distribuida con código fuete completo sujeto a los términos de la licencia LGPL. JFreeChart puede generar cuadros estadísticos en barras, líneas , tortas porcentuales, series de tiempo, etc.

II.18 RSS (Really Simple Syndication)

Figura 19: RSS (Really Simple Syndication)

Es una familia de formatos de fuentes web codificados en XML. Se utiliza para suministrar a suscriptores de información actualizada frecuentemente. El formato permite distribuir contenido sin necesidad de un navegador, utilizando un software diseñado para leer estos contenidos RSS (agregador). A pesar de eso, es posible utilizar el mismo navegador para ver los contenidos RSS. Las últimas versiones de los principales navegadores permiten leer los RSS sin necesidad de software adicional. RSS es parte de la familia de los formatos XML desarrollado específicamente

para todo tipo de sitios que se actualicen con frecuencia y por medio del cual se puede compartir la información y usarla en otros sitios web o programas. A esto se le conoce como redifusión web o *sindicación web* (una traducción incorrecta, pero de uso muy común).

II.18.1. Hojas de Estilo CSS

Las hojas de estilo en cascada (Cascading Style Sheets, CSS) son un lenguaje formal usado para definir la presentación de un documento estructurado escrito en HTML o XML (y por extensión en XHTML). El W3C (World Wide Web Consortium) es el encargado de formular la especificación de las hojas de estilo que servirán de estándar para los agentes de usuario o navegadores. La idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación.

II.19 Bases de Datos

Una base de datos es un conjunto de datos almacenados entre los que existen relaciones lógicas y ha sido diseñada para satisfacer los requerimientos de información de una organización. En una base de datos además de los datos también se almacena su descripción. Todos los datos se integran con una mínima cantidad de duplicidad.

II.20 Modelo de Datos

Es un conjunto de conceptos que sirven para describir la estructura de una base de datos las relaciones, los datos, las relaciones entre los datos y las restricciones de consistencia.

II.21 Gestor de Bases de Datos

El sistema gestor de base de datos es una aplicación que permite a los usuarios definir, crear y mantener la base de datos y proporciona acceso controlado a la misma.

Permite la definición de la base de datos mediante (DDL, Lenguaje de Definición de Datos), este lenguaje permite especificar la estructura y el tipo de datos así como las restricciones. Permite la inserción, actualización y eliminación y consultas de datos mediante el lenguaje de manipulación de datos (DML).

Proporciona un acceso controlado a la base de datos mediante:

- Un sistema de seguridad de modo que los usuarios no autorizados no puedan acceder a la base de datos.
- Un sistema de integridad que mantiene la integridad y consistencia de los datos.

- Un sistema de control de concurrencia que permite el acceso compartido a la Base de Datos.
- Un sistema de control de recuperación que restablece a la base de datos, después de que se produzca un fallo del hardware o del software.

II.22 Postgresql

PostgreSQL es un sistema de gestión de base de datos relacional orientada a objetos de software libre, publicado bajo la licencia BSD. Como muchos otros proyectos open source, el desarrollo de PostgreSQL no es manejado por una sola compañía sino que es dirigido por una comunidad de desarrolladores y organizaciones comerciales las cuales trabajan en su desarrollo. Dicha comunidad es denominada el PGDG (*PostgreSQL Global Development Group*).

II.23 Mapeo de Objetos a Bases de Datos (ORM)

Un ORM consiste en una serie de objetos que permiten acceder a los datos y que contienen en su interior cierta lógica de negocio. Una de las ventajas de utilizar estas capas de abstracción de objetos/relacional es que evita utilizar una sintaxis específica de un sistema de bases de datos concreto. Esta capa transforma automáticamente las llamadas a los objetos en consultas SQL optimizadas para el sistema gestor de bases de datos que se está utilizando en cada momento.

De esta forma, es muy sencillo cambiar a otro sistema de bases de datos completamente diferente en mitad del desarrollo de un proyecto. Estas técnicas son útiles por ejemplo cuando se debe desarrollar un prototipo rápido de una aplicación y el cliente aun no ha decidido el sistema de bases de datos que más le conviene.

II.23.1. Hibernate

Al igual que Propel permite la persistencia objeto/relacional y el servicio de consultas a la base de datos para aplicaciones desarrolladas en Java. Hibernate permite desarrollar clases persistentes siguiendo un idioma orientado a objetos incluyendo asociación, herencia, polimorfismo, composición y colecciones. Las consultas en Hibernate pueden ser realizadas en su propia extensión portátil (HQL) y así también el SQL nativo.

II.24 Servidor Web

Un servidor web es un programa que implementa el *protocolo HTTP* (*HyperText Transfer Protocol*), que se ejecuta continuamente en un ordenador (también se emplea el término para referirse al ordenador que lo ejecuta), manteniéndose a la espera de peticiones por parte de un cliente (un navegador web) y que responde a estas peticiones adecuadamente, mediante una *página web* que se exhibirá en el navegador o mostrando el respectivo mensaje si se detectó algún error.

II.24.1. Apache Tomcat

Tomcat (también llamado Jakarta Tomcat o Apache Tomcat) es un servidor web con soporte de servlets y JSPs. Tomcat no es un servidor de aplicaciones, como JBoss o JOnAS. Incluye el compilador Jasper, que compila JSPs convirtiéndolas en servlets. El motor de servlets de Tomcat a menudo se presenta en combinación con el servidor web Apache.

Tomcat puede funcionar como servidor web por sí mismo. En sus inicios existió la percepción de que el uso de Tomcat de forma autónoma era sólo recomendable para entornos de desarrollo y entornos con requisitos mínimos de velocidad y gestión de transacciones. Hoy en día ya no existe esa percepción y Tomcat es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad. Dado que Tomcat fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual Java.

II.25 Hosting

Hosting o alojamiento Web es un servicio ofrecido por una gran cantidad de suministradores que permiten almacenar un sitio web en sus servidores. Estos servidores pueden ser compartidos (shared servers) con otros clientes o pueden ser dedicados a un único cliente (servidor dedicado).
Un servicio hosting se puede diferenciar de otro por el tipo de sistema operativo, uso de base de datos y motor de generación de páginas webs exista en el.

Algunos servicios las comunes que se pueden entregar son el FTP, manejo por páginas web y múltiples clientes en las bases de datos. Existen diferentes tipos de hosting: servidores gratuitos, compartidos dedicados, revendedores, virtuales y de colocación.

II.26 Capacitación

Se entiende por capacitación el conjunto de procesos organizados, relativos tanto a la educación no formal como a la informal, dirigidos a prolongar y a complementar la educación mediante la generación de conocimientos, el desarrollo de habilidades y el cambio de actitudes, con el fin de incrementar la capacidad individual y colectiva para contribuir al cumplimiento de la misión institucional, a la mejor prestación de servicios a la comunidad, al eficaz desempeño del cargo y al desarrollo personal integral.

II.27 Proceso de Enseñanza Aprendizaje

La enseñanza y aprendizaje forman parte de un único proceso. La enseñanza es el acto mediante el cual se transmite conocimientos, enseñanzas clasificados y evaluados por medio de la utilización de diferentes metodologías de apoyo a la enseñanza, todas las actividades que se realicen en este proceso esta inevitablemente vinculado a los procesos de aprendizaje.

En la enseñanza se sintetizan conocimientos. Se va desde el no saber hasta el saber; desde el saber imperfecto, inacabado e insuficiente hasta el saber perfeccionado, suficiente y que sin llegar a ser del todo perfecto se acerca bastante a la realidad objetiva de la representación que con la misma se persigue.

Al aprendizaje se le puede considerar como un proceso de naturaleza extremadamente compleja caracterizado por la adquisición de un nuevo conocimiento, habilidad o capacidad, debiéndose aclarar que para que tal proceso pueda ser considerado realmente como aprendizaje, en lugar de una simple huella o retención pasajera de la misma, debe ser susceptible de manifestarse en un tiempo futuro y contribuir, además, a la solución de situaciones concretas, incluso diferentes en su esencia a las que motivaron inicialmente el desarrollo del conocimiento, habilidad o capacidad.

II.28 Socialización

La socialización es vista por los sociólogos como el proceso mediante el cual se inculca la cultura a los miembros de la sociedad, a través de él, la cultura se va transmitiendo de generación en generación, los individuos aprenden conocimientos específicos, desarrollan sus potencialidades y habilidades necesarias para la participación adecuada en la vida social y se adaptan a las formas de comportamiento organizado característico de su sociedad.

II.29 Medios de Difusión

Los medios de difusión son aquellos que se utiliza para difundir mensajes emanados por las estructuras de dominación que los controlan, es decir la empresas de dedicadas al rubro de la comunicación. Usualmente se utiliza el término "medios de comunicación" para hacer referencia a los medios de comunicación masivos (televisión, radio, prensa escrita, etc), sin embargo, estos medios no cumplen la función de comunicar sino más bien de difundir.

II.30 Publicidad

La publicidad implica una forma de comunicación orientada a difundir. Se trata de un hecho comercial, social, cultural que permite estimular a consumidores para que compren un producto, hagan uso de un servicio o participen de un evento.

Al elaborar la publicidad, los publicistas tienen muy presente el medio de comunicación a través del que se van a emitir. Esto les permite tomar en cuenta los distintos códigos que se utilizan en cada medio, de modo que la publicidad cumpla con eficacia en sus funciones.

II.30.1. Medios Gráficos

En los diarios y revistas, todo anuncio puede ilustrarse y explicarse en detalle. Esto se da porque ambos se valen de la imagen gráfica y del texto escrito: ellos permiten observar con más detenimiento el mensaje transmitido.

- ✓ El **diario** es considerado uno de los medios principales para la publicidad por el número de lectores diarios, los ingresos que permite por publicidad y por la preferencia entre el público como vehículo de publicidad.
- ✓ Las **revistas** se diferencian del diario por su forma de circulación, su formato y su contenido. Este tipo de medio permite a los anunciantes emitir el mensaje publicitario de un modo más selectivo de acuerdo a las preferencias del público al que se dirigen.
- ✓ Los **trípticos** son impresiones formadas por una lámina de papel o cartulina que se dobla en tres partes. Constituye un elemento publicitario ideal para comunicar ideas sencillas sobre un producto, servicio, empresa, evento, etc.
- ✓ Los **afiches** son carteles o avisos expuestos al público, estos carteles son realizados con alguna intención artística, mediante el que se anuncia un evento próximo.

II.30.2. Televisión

Es un medio flexible y favorece el desarrollo creativo a través de los diversos elementos que integra: la imagen y el sonido. Permite hacer presentaciones sencillas, de modo que los perceptores identifiquen los productos o los servicios de una forma fácil y rápida. Para obtener un comercial realmente emotivo, es necesario combinar los elementos racionales con los emotivos.

II.30.3. Radio

Es un medio que permite muchas posibilidades creativas a nivel publicitario. Voz humana, sonidos de la realidad, efectos de sonido, música y silencio son las herramientas que maneja y que hacen de ella un medio muy rico y creativo.

II.30.4. Internet

Es un medio de comunicación que permite llegar la publicidad de forma precisa a grupos poblacionales específicos. Es un medio rápido e interactivo. El resultado de las campañas se puede medir de forma precisa y de manera continua. Permite reiteradas modificaciones y adaptaciones de las campañas en función de los resultados.

III. Componentes

III.1 Componente 1 : Plataforma de Blogs

III.1.1. Descripción General del Proyecto de Software

III.1.1.1. Introducción

La nueva web propicia el desarrollo de capacidades y competencias hasta ahora poco frecuentes: colaboración, equipos de trabajo, conocimiento abierto, trabajos no conclusivos, etc. La información (y por ende, el conocimiento) no es propiedad de nadie, es colectivo. Existen varios servicios Web desarrollados siguiendo esta filosofía de inteligencia colectiva y son denominados herramientas sociales Web 2.0.

En el presente proyecto se desarrollara un gestor de blogs como herramienta que será de gran utilidad para los Colegios de Profesionales.

El proyecto será gestionado bajo la metodología de **Proceso Unificado Ágil (AUP)** es un enfoque al desarrollo de software basado en el Rational Unified Process (RUP) de IBM. El ciclo de vida de Agile UP es serial en lo grande e iterativo en lo pequeño, liberando entregables incrementales en el tiempo.

III.1.1.2. Evaluación de la Organización

Como se menciono anteriormente el proyecto está desarrollado para cubrir las necesidades de los Colegios de Profesionales, para describir a estas organizaciones se tomo como ejemplo el **Colegio de Ingenieros Civiles de Tarija.**

Misión

"Impulsar y participar en iniciativas estratégicas en los ámbitos de mayor impacto económico y social, promoviendo la investigación científica y la correcta aplicación técnica y ética de la ingeniería, y además actuando como referente gremial en apoyo a sus asociados"

Visión

Ser una institución que representará activamente a la totalidad de sus asociados a nivel departamental, nacional e internacional, promoviendo permanentemente capacitación, mejores prácticas profesionales, en cumplimiento de la ética, asegurando calidad y

excelencia en el ejercicio profesional, con el fin de gestar una sociedad más correcta, transparente y respetuosa de la ley.

Estructura Organizacional

Directorio Ejecutivo del Colegio Departamental de Ingenieros Civiles Tarija Gestión 2009	
Presidente	Ing. Ariel Antonio Aguirre
Vicepresidente	Ing. Aníbal Aldana Ortega
Secretario General	Ing. Ricardo Antelo Bustamante
Secretario de Actas	Ing. Gladis López Rueda
Secretario de Hacienda	Ing. José Jaime Zamora Calderón
Secretario de Coordinación y Asuntos Prof.	Ing. Trinidad Baldivieso Montalvo
Secretario de Promoción	Ing. Pedro Arturo Dubravic Celaya
Secretario de Deportes	Ing. Walter R. Cox Hoyos
Vocal	Ing. Ronald G. de los Rios Rissiotti
Vocal	Ing. Jorge Muci Chali
Vocal	Ing. María Angélica Sánchez

Tabla 16. Directorio Ejecutivo del Colegio Departamental de Ingenieros Civiles Tarija

III.1.1.3. Misión

Mediante el desarrollo del gestor de blogs contribuir al mejoramiento de la administración en los Colegios de Profesionales para que estos puedan cumplir adecuadamente su rol como institución, con sus afiliados y la sociedad.

III.1.1.4. Visión

La plataforma de blogs permitirá a los Colegios de Profesionales crear su blog, el blog se convertirá en la herramienta que mejorara la administración de estas instituciones, ya que será de gran utilidad para difundir información relacionada a la institución, como por ejemplo las actividades que se desarrollan, cursos que se organizan, etc. De esta manera estas

instituciones mantendrán permanentemente informados a sus afiliados y la sociedad en general tendrá un mejor conocimiento de estas instituciones de las diferentes actividades que realiza.

Esta herramienta no solo permitirá publicar información para el afiliado, sino también el afiliado podrá publicar información de interés y de esta manera compartirla con los demás afiliados y la sociedad.

III.1.1.5. Objetivos

III.1.1.5.1. Objetivo General

Desarrollar e implementar una Plataforma de Blogs para los Colegios de Profesionales que le permita una mejor administración.

III.1.1.5.2. Objetivos Específicos

Los objetivos específicos que permitirán alcanzar el objetivo general son los siguientes:

- Recolectar y seleccionar toda la información necesaria para conocer los problemas y necesidades de las empresas de este tipo.
- Realizar el análisis y diseño, modelando el sistema utilizando los diagramas UML necesarios.
- Diseñar e implementar la base de datos basándonos en el diagrama de clases analizado.
- Desarrollar el modulo de Administración de Blogs.
- Desarrollar el modulo de Blogs.
- Realizar las pruebas correspondientes a ambos módulos de la plataforma desarrollada.

III.1.1.6. Plan de Proyecto

III.1.1.6.1. Plan de Iteraciones

El desarrollo se llevará a cabo en base a fases con una o más iteraciones en cada una de ellas. La siguiente tabla muestra una la distribución de tiempos y el número de iteraciones de cada fase (para las fases de Construcción y Transición es sólo una aproximación muy preliminar).

Fase	Nro. de Iteraciones	Duración
Fase de Inicio	1	1 mes
Fase de Elaboración	1	1 mes
Fase de Construcción	7	7 meses
Fase de Transición	2	2 mes

Tabla 17. Plan de Iteraciones

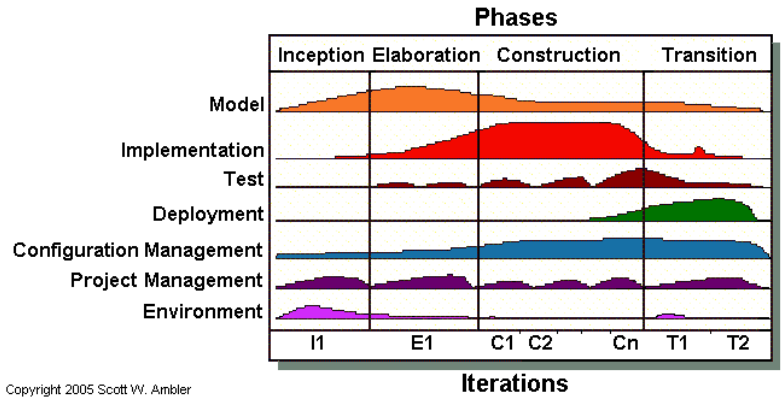

Figura 20. Fases de AUP

Los hitos que marcan el final de cada fase se describen en la siguiente tabla.

Fase	Hito
Inicio	• Acuerdo del Alcance del Proyecto. • Definición Inicial de Requerimientos. • Acuerdo costo inicial y la estimación del cronograma. • Aceptación y mitigación de Riesgos • Aceptación de Proceso. La Metodología • **Viabilidad** técnica, operacional y del negocio. • Plan del Proyecto. • El alcance del proyecto encaja bien en su organización
	• Visión Estable: El proyecto debe tener una visión estable y realística • Arquitectura Estable: Está de acuerdo que la arquitectura es estable y suficiente satisfacer los requisitos.

Elaboración	• Aceptación de Riesgos: Los riesgos se han evaluado para asegurar ellos se han entendido propiamente y se han documentado.
Construcción	• La estabilidad del sistema. El software y la documentación de apoyo son aceptables (estable y madura) para desplegar el sistema a los usuarios. • La empresa o negocio está listo para el sistema a ser desplegado (aunque todavía puede necesitar el entrenamiento).
Transición	• La empresa o negocio está satisfecho con el sistema y acepta el sistema. • Aceptación de funcionalidades del Sistema: La gente responsable para operar el sistema, está satisfecha con las funcionalidades pertinentes y la documentación. • Los actuales gastos son aceptables con respecto a las estimaciones.

Tabla 18. Descripción de Fases

III.1.1.6.2. Cronograma del Proyecto

Para este proyecto se ha establecido el siguiente calendario. La fecha de aprobación indica cuándo el artefacto en cuestión tiene un estado de completitud suficiente para someterse a revisión y aprobación, pero esto no quita la posibilidad de su posterior refinamiento y cambios.

Disciplinas / Artefactos generados o modificados durante la Fase de Inicio	Comienzo	Fin	Aprobación
Modelado			
Modelado de Requerimientos			
Modelado de Procesos de Negocio	01/04/09	05/04/09	20/05/09
Automatización de Oportunidades	05/04/09	10/04/09	20/05/09
Modelado de Casos de Uso	05/04/09	15/04/09	20/05/09
Requerimientos Técnicos	15/04/09	31/04/09	20/05/09
Modelado de Diseño			
Modelo de Objetos	01/05/09	05/05/09	Sig. Fase
Modelo de Datos Físico	05/05/09	15/05/09	Sig. Fase
Modelo de Interfaces de Usuario	15/05/09	20/05/09	Sig. Fase
Descripción General del Sistema	20/05/09	25/05/09	Sig. Fase
Modelo de Despliegue	24/05/09	25/05/09	Sig. Fase
Modelo de Amenazas de Seguridad	20/05/09	25/05/09	Sig. Fase
Implementación			
Sistema	01/06/09	31/12/09	Sig. Fase
Esquema de Base de Datos	05/06/09	10/06/09	Sig. Fase
Pruebas			

Suite de Pruebas de Regresión	31/11/09	10/12/09	Sig. Fase
Despliegue			
Scripts de Instalación	15/12/09	20/12/09	Sig. Fase
Material de Apoyo al Usuario	10/12/09	20/12/09	Sig. Fase
Administración de Configuración	Durante todo el proyecto		
Administración del proyecto	Durante todo el proyecto		
Plan del Proyecto	01/07/09	15/07/09	20/05/09
Glosario	01/09/09	05/09/09	Sig. Fase
Entorno	Durante todo el proyecto		

Tabla 19. Disciplinas / Artefactos generados o modificados durante la Fase de Inicio

Disciplinas / Artefactos generados o modificados durante la Fase de Elaboración	**Comienzo**	**Fin**	**Aprobación**
Modelado			
Modelado de Requerimientos			
Modelado de Procesos de Negocio	01/04/09	05/04/09	Aprobado
Automatización de Oportunidades	05/04/09	10/04/09	Aprobado
Modelado de Casos de Uso	05/04/09	15/04/09	Aprobado
Requerimientos Técnicos	15/04/09	31/04/09	Aprobado
Modelado de Diseño			
Modelo de Objetos	01/05/09	05/05/09	30/05/09
Modelo de Datos Físico	05/05/09	15/05/09	30/05/09
Modelo de Interfaces de Usuario	15/05/09	20/05/09	30/05/09
Descripción General del Sistema	20/05/09	25/05/09	30/05/09
Modelo de Despliegue	24/05/09	25/05/09	30/05/09
Modelo de Amenazas de Seguridad	20/05/09	25/05/09	30/05/09
Implementación			
Sistema	01/06/09	31/12/09	Sig. Fase
Esquema de Base de Datos	05/06/09	10/06/09	Sig. Fase
Pruebas			
Suite de Pruebas de Regresión	31/11/09	10/12/09	Sig. Fase
Despliegue			
Scripts de Instalación	15/12/09	20/12/09	Sig. Fase
Material de Apoyo al Usuario	10/12/09	20/12/09	Sig. Fase
Administración de Configuración	Durante todo el proyecto		
Administración del proyecto	Durante todo el proyecto		
Plan del Proyecto	01/07/09	15/07/09	Aprobado
Glosario	01/09/09	05/09/09	30/05/09
Entorno	Durante todo el proyecto		

Tabla 20. Disciplinas / Artefactos generados o modificados durante la Fase de Elaboración

Disciplinas / Artefactos generados o modificados durante la Fase de Elaboración	Comienzo	Fin	Aprobación
Modelado			
Modelado de Requerimientos			
Modelado de Procesos de Negocio	01/04/09	05/04/09	Aprobado
Automatización de Oportunidades	05/04/09	10/04/09	Aprobado
Modelado de Casos de Uso	05/04/09	15/04/09	Aprobado
Requerimientos Técnicos	15/04/09	31/04/09	Aprobado
Modelado de Diseño			
Modelo de Objetos	01/05/09	05/05/09	Aprobado
Modelo de Datos Físico	05/05/09	15/05/09	Aprobado
Modelo de Interfaces de Usuario	15/05/09	20/05/09	Aprobado
Descripción General del Sistema	20/05/09	25/05/09	Aprobado
Modelo de Despliegue	24/05/09	25/05/09	Aprobado
Modelo de Amenazas de Seguridad	20/05/09	25/05/09	Aprobado
Implementación			
Sistema	01/06/09	31/12/09	12/10/09
Esquema de Base de Datos	05/06/09	10/06/09	12/10/09
Pruebas			
Suite de Pruebas de Regresión	31/11/09	10/12/09	23/11/09
Despliegue			
Scripts de Instalación	15/12/09	20/12/09	Sig. Fase
Material de Apoyo al Usuario	10/12/09	20/12/09	Sig. Fase
Administración de Configuración	Durante todo el proyecto		
Administración del proyecto	Durante todo el proyecto		
Plan del Proyecto	01/07/09	15/07/09	Aprobado
Glosario	01/09/09	05/09/09	Aprobado
Entorno	Durante todo el proyecto		

Tabla 21. Disciplinas / Artefactos generados o modificados durante la Fase de Construcción

Disciplinas / Artefactos generados o modificados durante la Fase de Elaboración	Comienzo	Fin	Aprobación
Modelado			
Modelado de Requerimientos			
Modelado de Procesos de Negocio	01/04/09	05/04/09	Aprobado
Automatización de Oportunidades	05/04/09	10/04/09	Aprobado
Modelado de Casos de Uso	05/04/09	15/04/09	Aprobado
Requerimientos Técnicos	15/04/09	31/04/09	Aprobado
Modelado de Diseño			
Modelo de Objetos	01/05/09	05/05/09	Aprobado
Modelo de Datos Físico	05/05/09	15/05/09	Aprobado
Modelo de Interfaces de Usuario	15/05/09	20/05/09	Aprobado
Descripción General del Sistema	20/05/09	25/05/09	Aprobado

Modelo de Despliegue	24/05/09	25/05/09	Aprobado
Modelo de Amenazas de Seguridad	20/05/09	25/05/09	Aprobado
Implementación			
Sistema	01/06/09	31/12/09	Aprobado
Esquema de Base de Datos	05/06/09	10/06/09	Aprobado
Pruebas			
Suite de Pruebas de Regresión	31/11/09	10/12/09	Aprobado
Despliegue			
Scripts de Instalación	15/12/09	20/12/09	Aprobado
Material de Apoyo al Usuario	10/12/09	20/12/09	Aprobado
Administración de Configuración	Durante todo el proyecto		
Administración del proyecto	Durante todo el proyecto		
Plan del Proyecto	01/07/09	15/07/09	Aprobado
Glosario	01/09/09	05/09/09	Aprobado
Entorno	Durante todo el proyecto		

Tabla 22. Disciplinas / Artefactos generados o modificados durante la Fase de Transición

III.1.1.6.3. Estimación

Los costos de proyecto para completar el proyecto.

ITEM	RUBROS		Aporte IDH	TOTAL (Bs)
21000	**Servicios Básicos**			
	21100	Comunicación - Internet		1620
	21200	Energía Eléctrica		300
Subtotal			**1920**	**1920**
22000	**Viajes y Transporte personal**			
	22100	Transporte personal		300
Subtotal			**300**	**300**
25000	**Servicios Profesionales y Comerciales**			
	25600	Servicios de imprenta ● Empastados		150
	25810	Consultores por producto		2180
	25820	Consultores por línea Director		1680
Subtotal			**4010**	**4010**
3200	**Productos de papel, cartón e impresos**			
	32100	Papel de Escritorio 3. Resma de papel bond		320
	39500	Útiles de Escritorio y Oficina 4. Cartuchos de Tinta a color 5. Cartuchos de Tinta negra 6. Tinta para impresiones		220 190 280
Subtotal			**1010**	**1010**
Total			**7240**	**7240**

Tabla 23. Estimación del Proyecto

III.1.1.6.4. Presupuesto

Una indicación del monto del presupuesto, cuando será recibido y el criterio (si fuera el caso) que su equipo debe cumplir para recibir el fondo para soportar el esfuerzo del proyecto.

III.1.1.6.5. Lista de Riesgos

Una lista de los riesgos identificados, y las estrategias de mitigación (si procede).

RIESGO	DESCRIPCION	PROBABILIDAD %	IMPACTO	PLAN FRENTE AL RIESGO
Cambios de requisitos	Existencia de más cambios de requerimientos de los previstos inicialmente	70%	Reformulación de los requerimientos	Tratar de realizar una buena identificación y formulación de requerimientos.
Retrasos en la especificación	Retrasos en las especificaciones de interfaces esenciales	40%	Retraso en el desarrollo del proyecto	Tratar de cumplir responsablemente con la calendarización
Subestimación del tamaño	El tamaño de un requisito se ha subestimado	40%	Retraso en el desarrollo del proyecto	Cubrir de la mejor forma el impacto negativo en el proyecto
De requisitos	Los clientes no comprenden el impacto de los cambios en los requerimientos	40%	Posible paralización temporal del proyecto a causa de las diferencias.	Concientizar al cliente a cerca de la importancia de los cambios en los requerimientos
De requisitos	Los requisitos no se han definido correctamente. Y su redefinición aumenta el ámbito del proyecto.	40%	Requiere tiempo para el desarrollo de sistema	Definir bien la gestión de requisitos y realizar un estudio continuo de los mismos.
De estimación	El tiempo requerido para desarrollar el proceso de ingeniería de requisitos está subestimado	40%	Retraso en el desarrollo del proyecto	Tratar de realizar una buena estimación del tiempo requerido para desarrollar el proceso de ingeniería de requerimientos
De estimación	Las áreas desconocidas del producto llevan más tiempo del esperado en el diseño y en la implementación.	20%	Requiere tiempo para el desarrollo de sistema	Realizar un análisis más profundo del sistema a desarrollar.
Imposibilidad de trabajo	Personal clave enfermo o no disponible en momentos críticos	50%	Retrasa el desarrollo del Proyecto	Se tratara de no enfermarse

Tabla 24. Lista de Riesgos

III.1.2. Modelo de Requerimientos

III.1.2.1. Introducción

"Ingeniería de Requerimientos ayuda a los ingenieros de software a entender mejor el problema en cuya solución trabajarán. Incluye el conjunto de tareas que conducen a comprender cuál será el impacto del software sobre el negocio, qué es lo que el cliente quiere y cómo interactuarán los usuarios finales con el software". (Pressman, 2006: 155)

"La ingeniería de requerimientos es el proceso de desarrollar una especificación de software. Las especificaciones pretender comunicar las necesidades del sistema del cliente a los desarrolladores del sistema". (Sommerville, 2005:82).

La Ingeniería de Requerimientos se ve plasmada en este modelo que describe los requisitos que la plataforma debe cumplir. Es un comprendido de una variedad de productos de trabajo, potencialmente apruebas de aceptación, oportunidades de automatización, modelos de procesos del negocio, reglas del negocio, modelo del dominio, modelo de la organización, glosario del proyecto, requerimientos técnicos, modelo de casos de uso, y el modelo de interface de usuario.

No es necesario mantener todos los aspectos para el modelo de requerimientos, sólo la porción que resuma el alcance del sistema. En este caso el modelo mantendrá los siguientes productos.

- Modelo de Procesos del Negocio
- Modelo del Dominio
- Automatización de Oportunidades
- Modelo de Casos de Uso
- Requisitos Técnicos
- Modelo de Interfaces de Usuario
- Glosario del Proyecto

III.1.2.2. Captura y Análisis de Requerimientos

El análisis preliminar consiste en la evaluación de las solicitudes de proyectos de desarrollo de sistemas de información, no incluye la recolección de detalles para describir el sistema de la empresa, ni tampoco incluye un diseño detallado de solución; Si no que fundamentalmente es la reunión de la información que permite evaluar los méritos de la solicitud del proyecto y emitir un juicio, con conocimiento de causas con respecto a la factibilidad del proyecto propuesto.

Se estudiaron los procesos de las organizaciones mediante entrevistas, cuestionarios y conversaciones con las personas involucradas, esto con la finalidad de conocer sus opiniones, ideas y proposición de soluciones.

Para la captura de requerimientos se realizo:

- Un análisis a las diferentes plataformas de blogs que actualmente existen para adecuar Blogprof a los diferentes requerimientos que debe contemplar un sistema de este tipo.

- Un estudio a diferentes sistemas Web de Colegios de Profesionales e instituciones parecidas de nuestra ciudad, del país y de otras partes del mundo.

- Para cubrir los requerimientos específicos de los Colegios de Profesionales de la ciudad de Tarija se recopilo la información mediante el uso de cuestionarios y entrevistas que se realizo a algunos Presidentes de Colegios, profesionales afiliados y no afiliados.

- Fueron muy importantes las ideas y recomendanciones que aportaron los docentes guías de la materia.

- También se estudiaron la documentación que nos posibilitaron algunos Colegios de nuestra ciudad, como formularios, reportes.

III.1.2.3. Modelo de Procesos del Negocio

El Modelo del Proceso de Negocio es una descripción de las actividades que se realizan en la empresa, la información que fluye entre ellos, los recursos, y los destinatarios de la información. Los diagramas de flujo de datos (DFDs), el diagrama de actividad UML y los diagramas del workflow son opciones buenas para describir el proceso comercial visualmente. En este caso he utilizado diagramas de actividad UML.

A continuación se describen las actividades de manera general que llevan a cabo de manera similar los distintos Colegios de Profesionales de nuestra ciudad.

III.1.2.3.1. Diagrama de Proceso de Negocio: Registrar Afiliación

Figura 21. Diagrama de Proceso de Negocio: Registrar Afiliación

III.1.2.3.2. Diagrama de Proceso de Negocio: Realizar Actividades

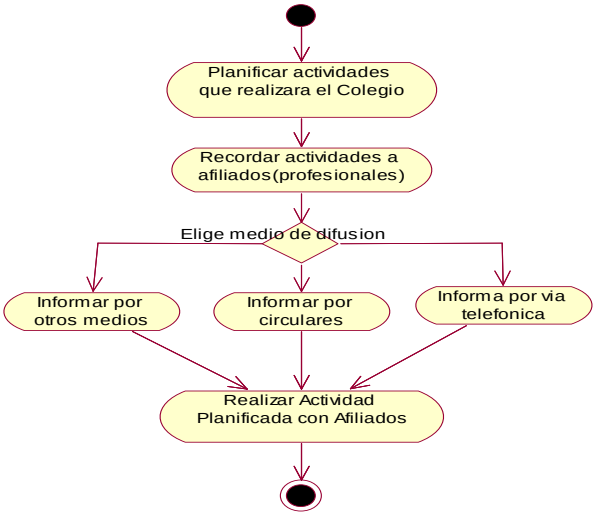

Figura 22. Diagrama de Proceso de Negocio: Realizar Actividades

III.1.2.3.3. Diagrama de Proceso de Negocio: Informar sobre Reuniones

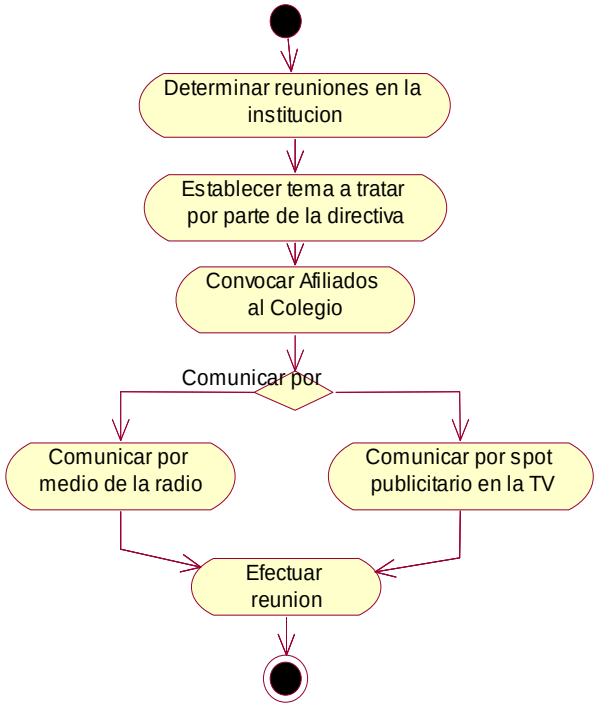

Figura 23. Diagrama de Proceso de Negocio: Informar sobre Reuniones

III.1.2.3.4. Diagrama de Proceso de Negocio: Organizar Cursos-Talleres

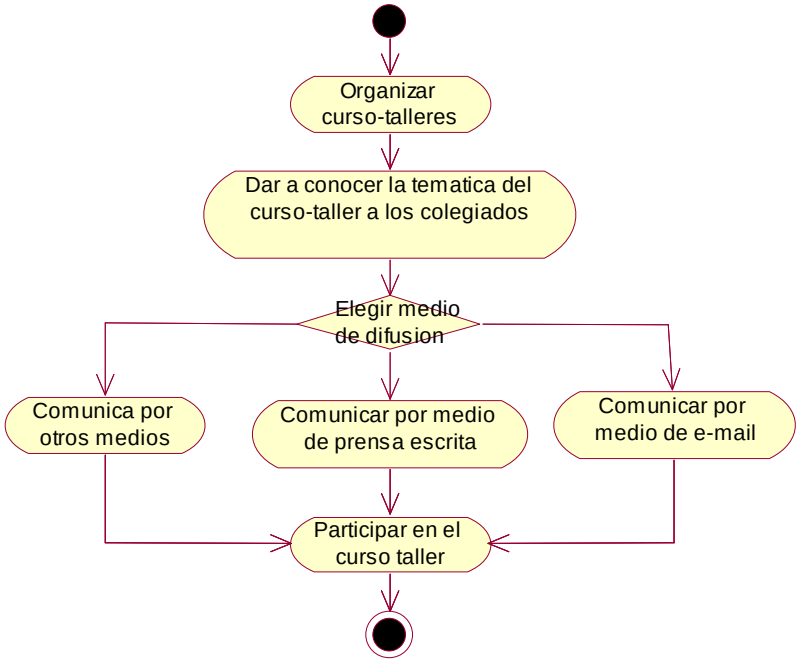

Figura 24. Diagrama de Proceso de Negocio: Organizar Cursos-Talleres

III.1.2.4. Automatización de Oportunidades

Una indicación de actividades manuales en las que potencialmente podrían ser automatizados.

El blog creado bajo la plataforma permitirá automatizar los siguientes procesos que llevan a cabo estas instituciones:

- *Registrar afiliación*

  Todos los Colegios de Profesionales registran a sus afiliados en base a un registro manual de acuerdo a un formato establecido por la institución y luego se archiva en folders de acuerdo a la gestión.
- *Administrar la información de los afiliados registrados.*

  En este caso los datos personales, y de afiliación de los diferentes profesionales que quieran afiliarse.
- *Informar a los afiliados sobre actividades programadas, reuniones, jornadas, científicas, etc.*

  Informar a los afiliados sobre cursos, conferencias, talleres que se organizan en la institución o fuera de ella.
- *Informar a la población sobre actividades que se desarrollan a su favor.*

  Todos este proceso de difusión de información automatizado mediante la publicación de contenidos, los contenidos serán cualquier información que quieran dar a conocer y esta información se ordenara en el blog mediante categorías siguiendo un orden cronológico de publicación.
- *Registrar egresos e ingresos de la institución*

  En este caso el administrador del blog del Colegio podrá registrar los ingresos que se recauden por afiliado o por alguna actividad realizada. También podrá registrar los egresos o gastos.
- *Sacar un reporte de las cuotas realizadas por cada miembro*

  El administrador podrá generar el kardex por miembro.
- *Mostrar el Directorio actual*

III.1.2.5. Modelo de Dominio

Un modelo de Dominio es un modelo conceptual en alto nivel, definiendo física y abstractamente objetos en una área de interés del Proyecto.

Un modelo de Dominio muestra:

- Las unidades organizacionales y físicas del dominio
- Las relaciones entre estas unidades

III.1.2.5.1. Modelo de Dominio: Colegio de Profesionales

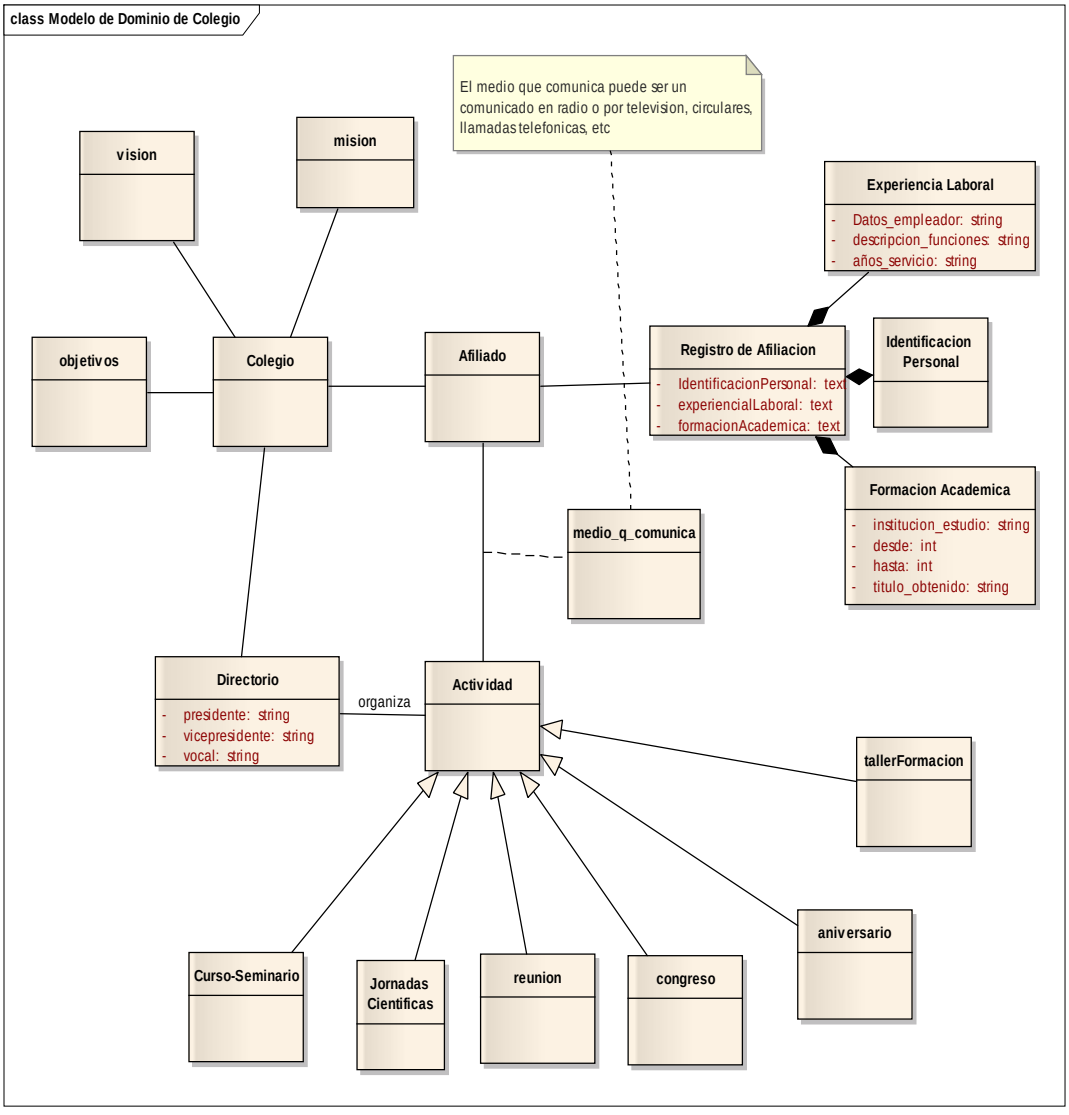

Figura 25: Modelo de Dominio: Colegios de Profesionales

III.1.2.5.2. Modelo de Dominio: Plataforma de Blogs

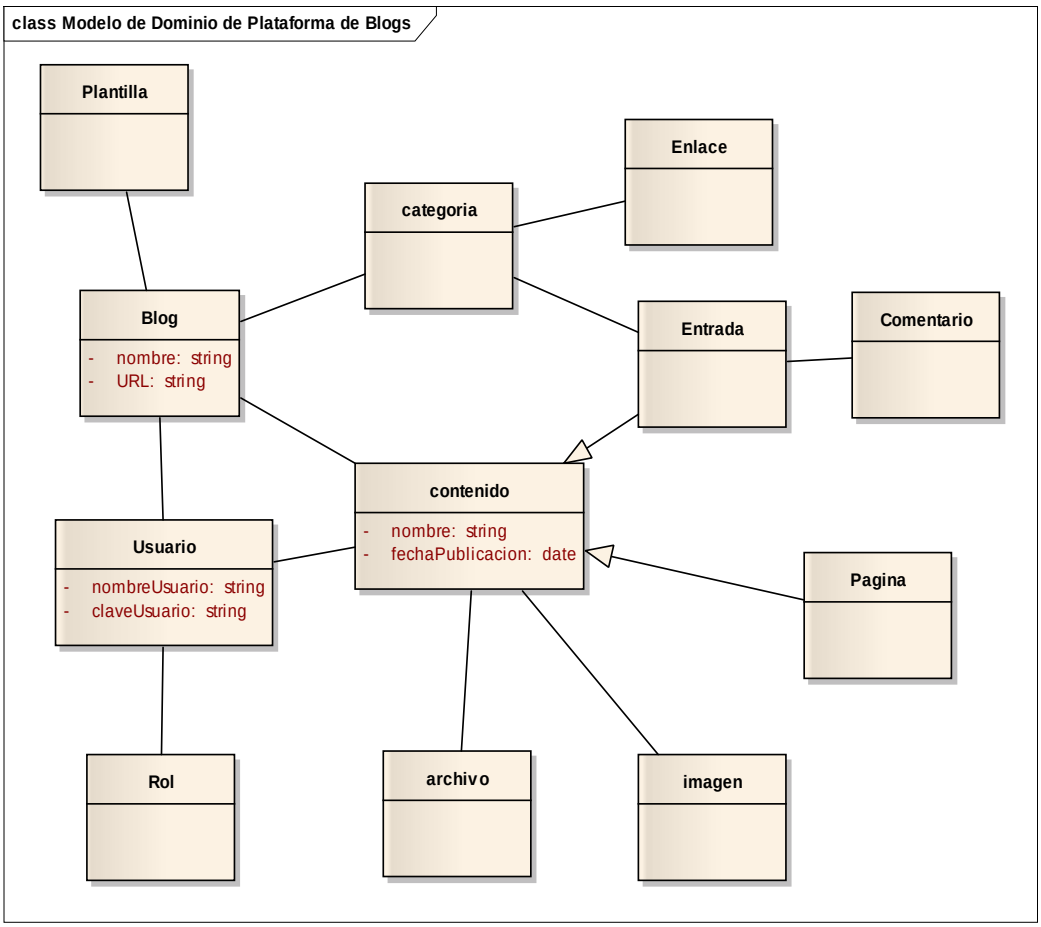

Figura 26: Modelo de Dominio: Plataforma de Blogs

III.1.2.6. Modelo de Casos de Uso

El modelo de casos de Uso representa los requerimientos funcionales que debe ofrecer la plataforma con respecto a los usuarios, consiste en un diagrama general del sistema, descripción de casos de uso, descripción de actores.

El conjunto de casos de uso representa la totalidad de operaciones desarrolladas por el Sistema.

El modelo de casos de uso esta compuesto por cero o más diagramas de casos de uso, descripciones de casos d uso y descripciones de actores.

III.1.2.6.1. Diagrama de Casos de Uso: General

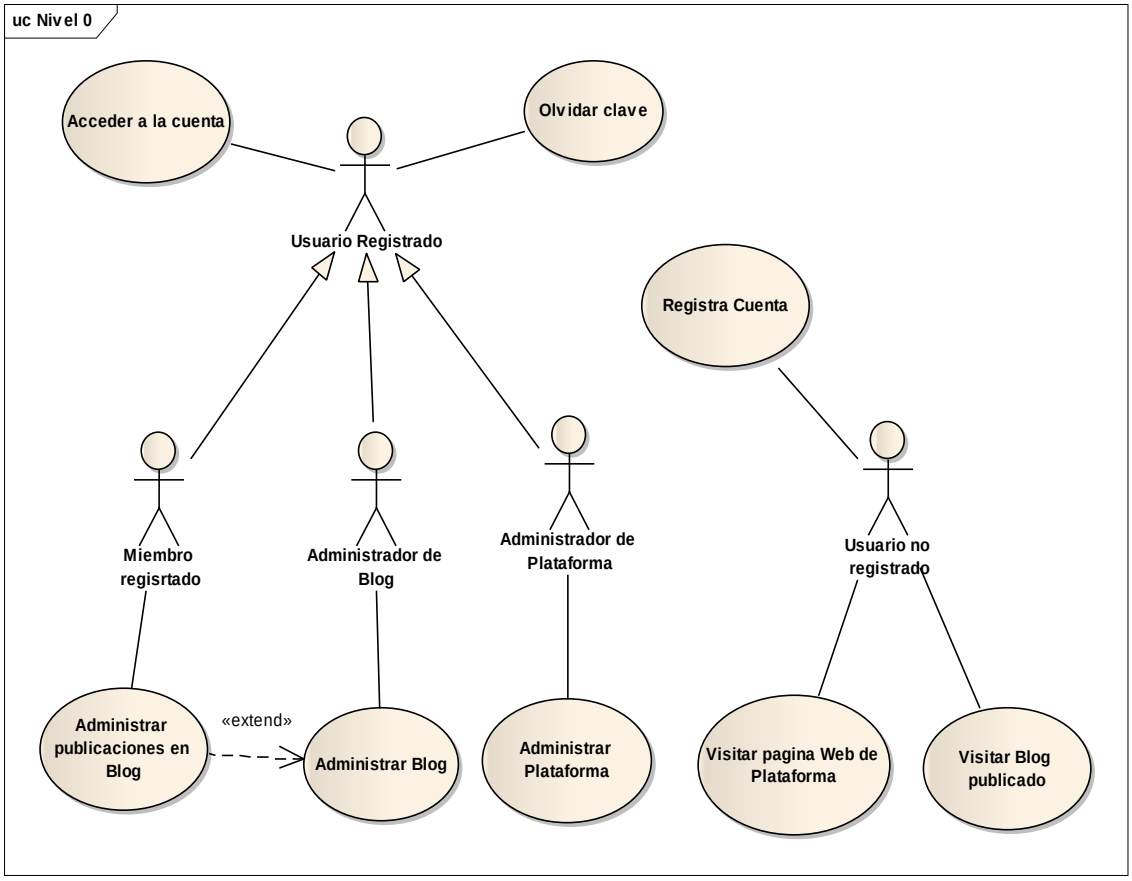

Figura 27. Diagrama de Casos de Uso: General

III.1.2.6.2. Diagrama de Casos de Uso: Visitar Blog publicado

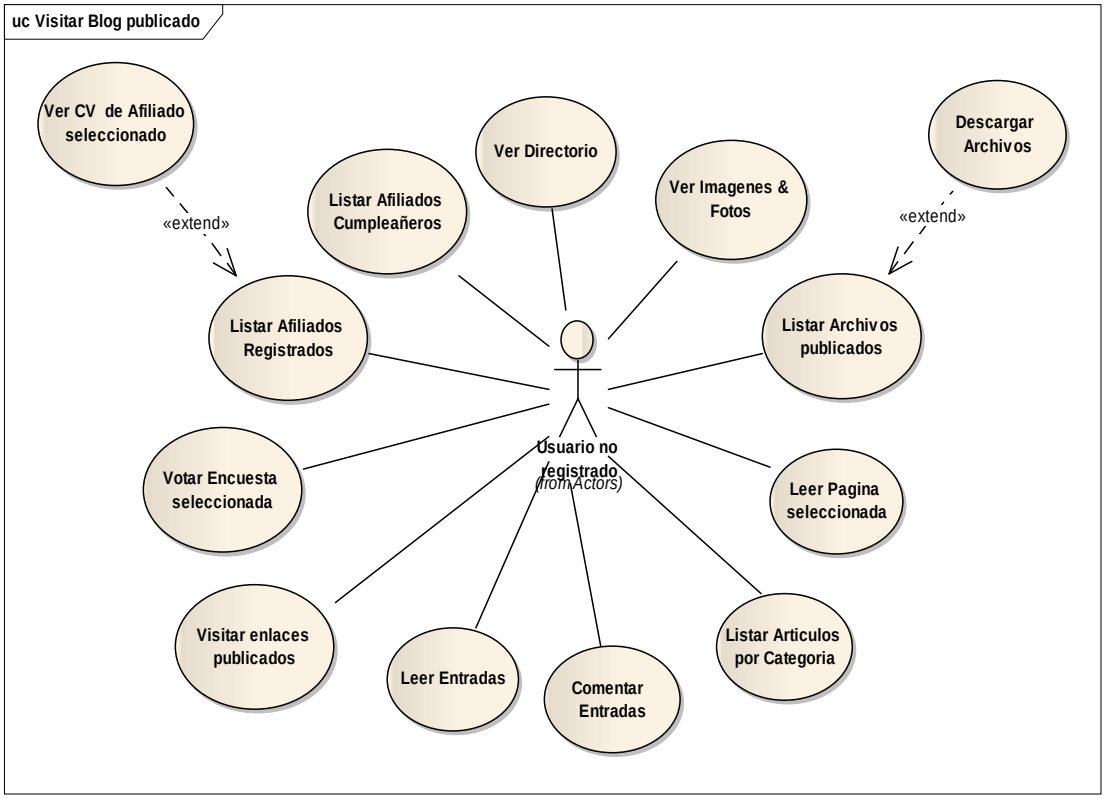

Figura 28. Diagrama de Casos de Uso: Visitar Blog publicado

III.1.2.6.3. Diagrama de Casos de Uso: Visitar pagina Web de la plataforma

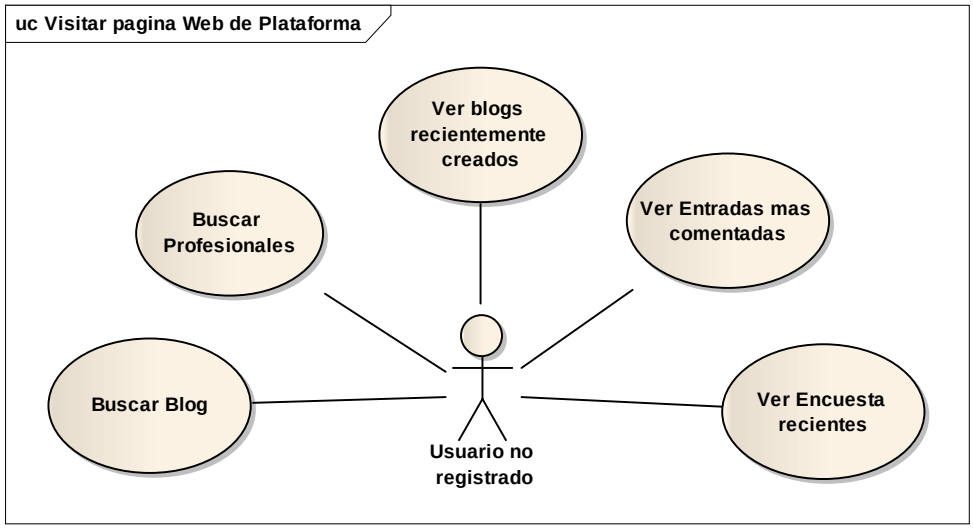

Figura 29. Diagrama de Casos de Uso: Visitar pagina Web de la plataforma

III.1.2.6.4. Diagrama de Casos de Uso: Administración de publicaciones en el Blog

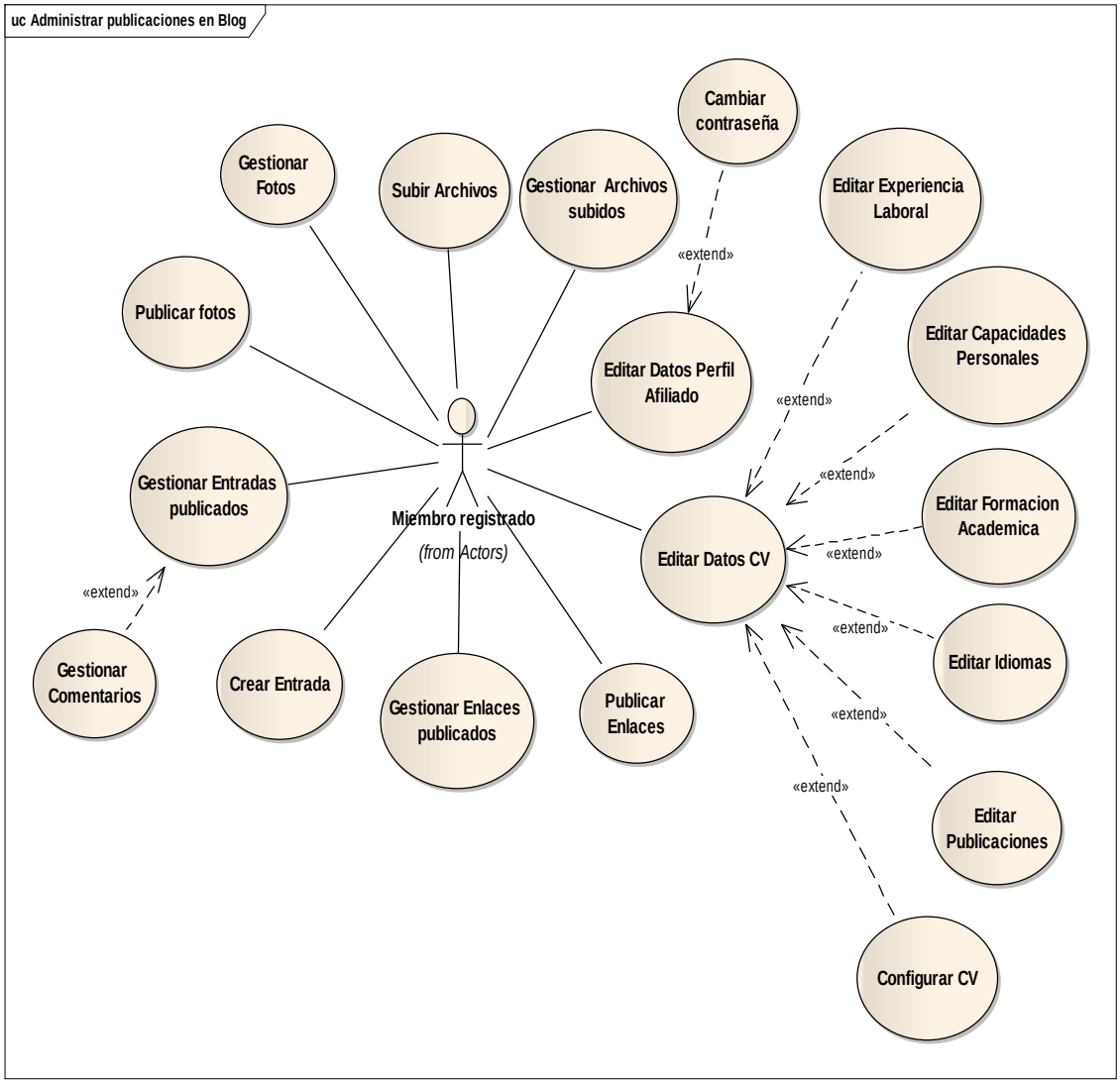

Figura 30. Diagrama de Casos de Uso: Administración de publicaciones en el Blog

III.1.2.6.5. Diagrama de Casos de Uso: Administrar Blog

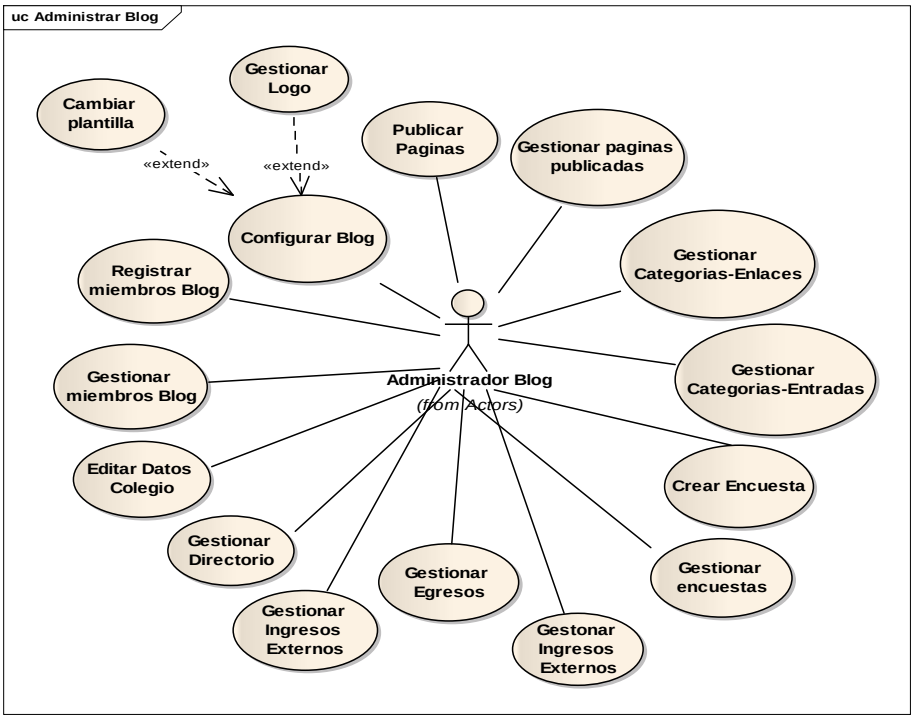

Figura 31. Diagrama de Casos de Uso: Administrar Blog

III.1.2.6.6. Diagrama de Casos de Uso: Administrar Plataforma

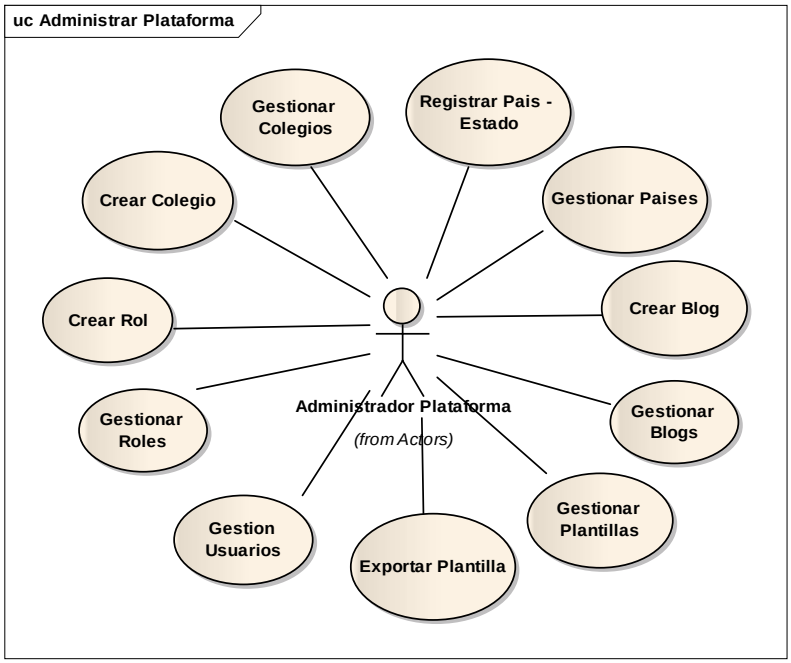

Figura 32. Diagrama de Casos de Uso: Administrar Plataforma

III.1.2.7. Especificaciones de Caso de Uso

A continuación se presenta una descripción informal de los casos de uso es decir un detalle de los requisitos iníciales del sistema en alto nivel.

III.1.2.7.1. Especificación de Caso de Uso: Registrar Cuenta

Nombre Caso de Uso:	**Registrar Cuenta**
Identificador:	**RC**
Descripción: En este caso de uso el usuario no registrado que visita la plataforma registra la cuenta para su Colegio.	
Precondiciones: El usuario visita la pagina Web de la plataforma	
Pos condiciones: La plataforma cuenta con un nuevo blog registrado.	
Flujo Básico de la Acción: **1.** El usuario hace clic en el link Registrar Cuenta del topbar de la página Web de la Plataforma. **2.** Inmediatamente aparece la interfaz que permite ingresar los datos del Colegio y del Administrador. **3.** El usuario ingresa los datos del Colegio. **4.** Presiona Siguiente. **5.** Se valida los datos del Colegio. **6.** Se registra el Colegio y su blog correspondiente en base de datos. **7.** Una vez guardado los datos, se direcciona a la interfaz que permite registrar los datos del Administrador de Blog. **8.** El usuario ingresa sus datos y el control de seguridad. **9.** Presiona Siguiente **10.** Se valida los datos del Administrador de Blog y el Control de Seguridad. **11.** Si todo esta correcto se guardan los datos del nuevo Administrador Blog, se inicia sesión. **12.** Se direcciona a la interfaz que permite elegir la plantilla del blog para darle la apariencia exterior. **13.** El usuario elige una plantilla y presiona Siguiente. **14.** Se direcciona a la interfaz que permite subir el logo o escudo del Colegio para poder identificarse mejor. **15.** Se verifica que el archivo subido sea una imagen, si es así se redimensiona la imagen y se guarda en base de datos.	

16. Si todo el proceso de registro de Cuenta fue realizado correctamente, se direcciona a la interfaz que muestra los datos ingresados del blog creado.
Flujo Alternativo de la Acción: **1.** La opción de Registrar cuenta se encuentra no solo en la página de la plataforma sino en todos los blogs, que redireccionan a la interfaz de registro. **2.** Si los datos requeridos no son ingresados o son incorrectos se mostraran los mensajes de que validen el error. **3.** En el punto 12, el usuario puede omitir el paso de elegir plantilla. **4.** En el punto 14, el usuario puede omitir el paso de Subir logo. **5.** En el punto 15, si el archivo subido no es una imagen no le permite subir.

Tabla 25. Especificación de Caso de Uso: Registrar Cuenta

III.1.2.7.2. Especificación de Caso de Uso: Acceder a Cuenta

Nombre Caso de Uso:	**Acceder a la Plataforma**
Identificador:	**AC**
Descripción: Es el escenario mediante el cual diferentes usuarios acceden a la plataforma; pero con diferentes roles que distinguen los permisos de los usuarios.	
Precondiciones: El usuario visita la pagina Web de la plataforma	
Flujo Básico de la Acción: 1. El usuario ingresa su nombre de usuario y clave en el topbar de la pagina index de la pagina Web 2. Presiona el botón Ingresar o hace un ENTER. 3. Se valida que el usuario exista en la base de datos. 3.1. Se verifica el estado del usuario, este debe está habilitado 3.2. Para comparar la clave de usuario se encripta el clave ingresada y se compara con la clave encriptada de base de datos 4. Si el usuario existe y está habilitado, se crea la sesión para el usuario. 5. Luego el usuario tiene acceso al panel administrador de su blog.	
Flujo Alternativo de la Acción: 1. Existe la opción de ingresar a la cuenta en cualquier blog de la plataforma 2. En el punto 3, si los datos son incorrectos, no se permite el ingreso del usuario y sale el mensaje “Datos de usuario incorrectos, el usuario no existe”. 3. Si el usuario existe en el punto 4 pero esta deshabilitado, saldrá el mensaje “Usuario Deshabilitado” y tampoco podrá ingresar.	

Tabla 26. Especificación de Caso de Uso: Registrar Cuenta

III.1.2.7.3. Especificación de Caso de Uso: Olvidar Clave

Nombre Caso de Uso:	**Olvidar Clave**
Identificador:	**OC**
Descripción: Es el escenario mediante el cual diferentes usuarios que olvidaron su clave o contraseña pueden pedir que se les otorgue otra clave para ingresar.	
Precondiciones: El usuario visita la pagina Web de la plataforma	
Flujo Básico de la Acción: 1. El usuario presiona la opción Olvide mi clave. 2. Aparece una pantallita que le permite ingresar su id de usuario. 3. Internamente se verifica que el nombre de usuario ingresado se encuentre registrado. 4. Se verifica que cuente con cuenta de email. 5. Si todo esta correcto, se genera una clave alfanumérica, se la encripta y actualiza el valor de la clave del usuario. 6. Una vez actualizada la clave del usuario, se manda la nueva clave de usuario mediante un mensaje a la dirección email del usuario y aparece el mensaje "Una nueva clave ha sido generada en tu cuenta email"	
Flujo Alternativo de la Acción: 1. El usuario puede acceder a esta opción desde el blog al que pertenece. 2. En el punto 3, si el nombre de usuario no es encontrado en BD, se muestra el mensaje de advertencia. "El ID Usuario no se encuentra registrado". 3. En el punto 4, si el usuario no ha registrado su cuenta de email, sale el mensaje de aviso: "No se puede localizar su cuenta de email"	

Tabla 27. Especificación de Caso de Uso: Olvidar Clave

III.1.2.7.4. Especificaciones de Caso de Uso: Visitar Blog publicado

III.1.2.7.4.1. Especificación de Caso de Uso: Listar Afiliados Registrados

Nombre Caso de Uso:	**Listar afiliados registrados**
Identificador:	**FAB1**
Descripción: En este caso de uso el usuario que visita el blog puede ver todos los afiliados profesionales registrados en el Colegio	
Precondiciones: El usuario visita un blog publicado por la plataforma	

Pos condiciones: El usuario realiza alguna de las siguientes acciones que se le permite cuando visualiza la lista de afiliados
Flujo Básico de la Acción: **1.** El usuario hace clic en el enlace Afiliados Registrados **2.** Según blog se selecciona de la base de datos todos los profesionales afiliados al Colegio al cual pertenece el blog. **3.** En pantalla se actualiza la pagina mostrando un máximo de diez registros de afiliados y si existen más se puede seguir mediante el paginador. **4.** El usuario puede realizar una de las siguientes acciones: **5.** **FAB1-1 :** *Ver CV de Afiliado seleccionado*

Tabla 28. Especificación de Caso de Uso: Listar afiliados registrados

III.1.2.7.4.2. Especificación de Caso de Uso: Ver CV de Afiliado seleccionado

Nombre Caso de Uso:	**Ver CV de Afiliado seleccionado**
Identificador:	**FAB1-1**
Descripción: En este caso de uso el usuario que visita el blog puede ver el Curriculum Vitae de el afiliado que permita ver su CV	
Precondiciones: El usuario selecciona la realiza el caso de uso FAB1 El usuario presiona la opción Ver CV en el registro del afiliado seleccionado	
Pos condiciones: Aparece en otra pantalla emergente el CV del afiliado seleccionado	
Flujo Básico de la Acción: **1.** El usuario selecciona un afiliado de la lista que tenga la opción Ver CV **2.** Aparece una pantalla emergente en el cual despliega el CV del afiliado seleccionado **3.** El usuario lee el CV del afiliado seleccionado **4.** Posteriormente cierra la ventana emergente.	

Tabla 29. Especificación de Caso de Uso: Ver CV de Afiliado seleccionado

III.1.2.7.4.3. Especificación de Caso de Uso: Listar afiliados cumpleañeros

Nombre Caso de Uso:	**Listar afiliados cumpleañeros**
Identificador:	**FAB2**
Descripción: En este caso de uso el usuario que visita el blog puede listar a los afiliados cumpleañeros registrados	
Precondiciones: El usuario visita un blog publicado por la plataforma	
Flujo Básico de la Acción: **1.** El usuario hace clic en el enlace Afiliados Cumpleañeros **2.** Internamente la plataforma selecciona en base de datos todos los afiliados que cumplieron años en los últimos 30 días. **3.** En pantalla se actualiza la página mostrando los afiliados cumpleañeros.	
Flujo Alternativo de la Acción: **1.** El usuario hace clic en el enlace Afiliados Cumpleañeros **2.** Internamente la plataforma selecciona en base de datos todos los afiliados que cumple años en los últimos 30 días. **3.** En pantalla se actualiza la página pero muestra un mensaje "Hoy no hay cumpleañeros"	

Tabla 30. Especificación de Caso de Uso: Listar afiliados cumpleañeros

III.1.2.7.4.4. Especificación de Caso de Uso: Ver Directorio

Nombre Caso de Uso:	**Ver Directorio**
Identificador:	**FAB3**
Descripción: En este caso de uso el usuario que visita el blog puede ver la estructura orgánica de la directiva del Colegio.	
Precondiciones: El Colegio ha creado su directorio y sus respectivos cargos fueron asignados y registrados.	
Flujo Básico de la Acción: **1.** El usuario selecciona la opción directorio que aparece en el blog **2.** Internamente se selecciona el directorio del Colegio y sus respectivos cargos en Base de datos. **3.** En página se actualiza y muestra las fotografías, nombres y los respectivos cargos que ocupan en el directorio.	

Flujo Alternativo de la Acción:
1. En el punto 3, si el Administrador de Blog no ha creado el Colegio en el blog, no se mostrara más que un mensaje: "Todavía no creo su directorio". **2.** En el punto 3, si el Administrador de Blog ha creado el Directorio pero no ha registrado sus cargos correspondientes.

Tabla 31. Especificación de Caso de Uso: Ver Directorio

III.1.2.7.4.5. Especificación de Caso de Uso: Ver Fotos

Nombre Caso de Uso:	**Ver Fotos**
Identificador:	**FAB4**
Descripción: En este escenario el usuario puede ver todas las fotos que fueron publicadas por los afiliados del Colegio al que pertenece el blog	
Precondiciones: El usuario visita un blog publicado por la plataforma	
Flujo Básico de la Acción: **1.** El usuario selección la opción Fotos **2.** Internamente se busca en base de datos todas las fotos publicadas correspondientes a todos los albums de todos los afiliados en el blog. **3.** El usuario puede ver la foto en tamaño real haciendo click en la misma.	
Flujo Alternativo de la Acción: **1.** En el punto dos, el usuario puede ver las fotos correspondiente a un solo álbum haciendo click en el album al que pertenece. **2.** De esta manera se mostrara solo las fotos correspondientes a ese album.	

Tabla 32. Especificación de Caso de Uso: Ver Fotos

III.1.2.7.4.6. Especificación de Caso de Uso: Listar archivos publicados

Nombre Caso de Uso:	**Listar archivos publicados**
Identificador:	**FAB5**
Descripción: En este caso de uso el usuario que visita el blog puede listar todos los archivos publicados por los afiliados registrados en el blog	
Precondiciones: El usuario visita un blog publicado por la plataforma	
Flujo Básico de la Acción: **1.** El usuario hace clic en el enlace Documentos **2.** Internamente la plataforma selecciona en base de datos todos l los archivos publicados por los afiliados registrados en el blog	

4. En pantalla se actualiza la página mostrando todos los archivos publicados a la fecha. **5.** El usuario decide pasar al caso de uso FAB5-1 *Descargar Archivo*
Flujo Alternativo de la Acción: 1. En el punto 2, si no hay ninguna selección de archivos en ese caso no aparece nada de archivos para descargar en el blog.

Tabla 32. Especificación de Caso de Uso: Listar archivos publicados

III.1.2.7.4.7. Especificación de Caso de Uso: Descargar Archivo

Nombre Caso de Uso:	**Descargar Archivo**
Identificador:	**FAB5-1**
Descripción: En este caso de uso el usuario que visita el blog puede descargar el archivo que de la lista de archivos publicados.	
Precondiciones: El usuario realiza el caso de uso FAB5	
Flujo Básico de la Acción: **1.** El usuario selecciona el documento que quiere descargar **2.** Presiona la opción descargar del archivo seleccionado **3.** Aparece la ventanita del navegador que posibilita bajar el archivo **4.** El usuario elige la opción descargar de la ventanita del navegador **5.** Comienza la descarga del servidor **6.** Una vez que termina de descargar el archivo, aparece el mensaje descarga completa del navegador que se está utilizando	
Flujo Alternativo de la Acción: **1.** El usuario selecciona el documento que quiere descargar **2.** Presiona la opción descargar del archivo seleccionado **3.** Aparece la ventanita del navegador que posibilita bajar el archivo **4.** El usuario elige la opción cancelar de la ventanita del navegador **5.** No se realiza ninguna descarga	

Tabla 33. Especificación de Caso de Uso: Descargar Archivo

III.1.2.7.4.8. Especificación de Caso de Uso: Votar Encuesta seleccionada

Nombre Caso de Uso:	**Votar Encuesta seleccionada**
Identificador:	**FAB6**
Descripción: En este caso de uso el usuario que visita el blog puede votar la encuesta de su interés, de esta manera los Colegios pueden saber la opinión de la población sobre un determinado tema, sobre el cual se publique la encuesta.	
Precondiciones: El usuario selecciona una encuesta de su interés	
Flujo Básico de la Acción: **1.** El usuario hace elige uno de las encuestas que aparece en la lista de Encuestas en el rightbar del blog **2.** La página se actualiza y muestra la encuesta correspondiente y las opciones de respuesta para votar. **3.** El usuario elige una de las opciones de respuesta de la encuesta. **4.** Presiona la opción Votar **5.** Internamente se guarda la votación de las respuestas seleccionadas en la base de datos y se actualiza la pagina mostrando la torta que muestra el porcentaje de votos que tiene cada una de las respuestas seleccionada	
Flujo Alternativo de la Acción: **1.** El usuario hace elige uno de las encuestas que aparece en la lista de Encuestas en el rightbar del blog **2.** La página se actualiza y muestra la encuesta correspondiente y las opciones de respuesta para votar. **3.** El usuario decide no votar y vuelve a la página principal	

Tabla 34. Especificación de Caso de Uso: Votar Encuesta seleccionada

III.1.2.7.4.9. Especificación de Caso de Uso: Visitar enlaces publicados

Nombre Caso de Uso:	**Visitar enlaces publicados**
Identificador:	**FAB7**
Descripción: En este escenario el usuario que visita el blog puede visitar otras paginas Web a travez de los enlaces publicados por los usuarios del Blog	
Precondiciones: El usuario visita un blog publicado en la plataforma	

Pos condiciones: El usuario visita la pagina del enlace seleccionado
Flujo Básico de la Acción: **1.** El usuario selecciona un enlace publicado **2.** Inmediatamente se abre otra página con la URL del enlace seleccionado

Tabla 35. Especificación de Caso de Uso: Visitar enlaces publicados

III.1.2.7.4.10. Especificación de Caso de Uso: Leer Articulo

Nombre Caso de Uso:	**Leer Articulo**
Identificador:	**FAB8**
Descripción: En este escenario el usuario que visita el blog puede leer los Entradas publicados por los afiliados registrados	
Precondiciones: El usuario visita un blog publicado en la plataforma	
Flujo Básico de la Acción: **1.** El usuario selecciona un artículo de la pagina y hace click en Leer Mas **2.** Inmediatamente se actualiza la pagina y se muestra el articulo completo mostrando la fecha la hora y quien publico el articulo **3.** El usuario termina de leer el artículo y puede volver a la página principal haciendo click en Principal	
Flujo Alternativo de la Acción: **1.** En el punto 1, el usuario también puede leer el artículo completo, haciendo click en el titulo. **2.** El usuario puede realizar un comentario al artículo. **3.** El usuario realiza el caso de uso FAB9 Comentar Articulo	

Tabla 36. Especificación de Caso de Uso: Leer Artículo

III.1.2.7.4.11. Especificación de Caso de Uso: Comentar Articulo

Nombre Caso de Uso:	**Comentar Articulo**
Identificador:	**FAB9**
Descripción: En este escenario el usuario que visita el blog puede leer los Artículos publicados por los afiliados registrados	

Precondiciones: El usuario realiza el caso de uso FAB8. Existe la opción para comentar el artículo.
Flujo Básico de la Acción: **1.** El usuario selecciona la opción Comentar **2.** Aparece una pantallita que permite al lector volverse comentador, registrando su comentario. **3.** El usuario ingresa su nombre, email sitio Web, y por supuesto su comentario. **4.** Si todos los datos requeridos son correctos el comentador podrá guardar su comentario. **5.** Internamente se guarda el comentario correspondiente al artículo en base de datos. **6.** Se cierra la pantallita, y el comentario registrado, aparece en la lista de comentarios al pie del artículo.
Flujo Alternativo de la Acción: **1.** En el punto 3, si los datos requeridos no son ingresados el comentador no podrá registrar o guardar su comentario.

Tabla 37. Especificación de Caso de Uso: Comentar Artículo

III.1.2.7.4.12. Especificación de Caso de Uso: Leer pagina seleccionada

Nombre Caso de Uso:	**Leer Pagina seleccionada**
Identificador:	**FAB11**
Descripción: En este escenario el usuario que visita el blog puede leer la pagina que le interese	
Precondiciones: El usuario visita un blog publicado en la plataforma	
Flujo Básico de la Acción: **1.** El usuario selecciona una etiqueta de pagina del menubar del blog **2.** Inmediatamente se actualiza la pagina y se muestra la pagina seleccionada **3.** El usuario termina de leer la pagina y puede volver a la página principal haciendo clic en Principal	
Flujo Alternativo de la Acción: **1.** Si el blog no ha publicado ninguna pagina el usuario que visita el blog no podrá leer ninguna pagina.	

Tabla 38. Especificación de Caso de Uso: Leer pagina seleccionada

III.1.2.7.5. Especificaciones de Caso de Uso: Visitar pagina Web de Plataforma

III.1.2.7.5.1. Especificación de Caso de Uso: Buscar Blog

Nombre Caso de Uso:	**Buscar Blog**
Identificador:	**FAP1**
Descripción: En este escenario el usuario que visite la Web de la Plataforma, podrá buscar el blog que quiera visitar.	
Precondiciones: **1.** El usuario visita la Web de la Plataforma	
Flujo Básico de la Acción: **1.** El usuario introduce el nombre del Blog, que quiere visitar. **2.** Según el parámetro ingresado en el buscador se realiza la búsqueda del blog internamente en base de datos **3.** El usuario identifica el blog que está buscando y hace un ENTER. **4.** Inmediatamente se muestra el blog que solicitaba visitar.	
Flujo Alternativo de la Acción: **1.** Si según el parámetro ingresado no se encuentra ningún blog en la búsqueda en la base de datos. **2.** Sale el mensaje "Blog no encontrado"	

Tabla 39. Especificación de Caso de Uso: Buscar Blog

III.1.2.7.5.2. Especificación de Caso de Uso: Buscar Profesionales

Nombre Caso de Uso:	**Buscar Profesionales**
Identificador:	**FAP2**
Descripción: En este escenario el usuario que visite la Web de la Plataforma, podrá buscar el profesional que necesite contratar	
Precondiciones: El usuario visita la Web de la Plataforma	
Flujo Básico de la Acción: 1. El usuario ingresa el profesional que está buscando (su profesión). 2. El usuario puede seleccionar la opción de buscar en Tarija, en Bolivia, en el mundo 3. Internamente se hace la búsqueda con los parámetros ingresados. 4. Aparece en pantalla la lista de profesionales que se encontró correspondiente a la Búsqueda.	

Tabla 40. Especificación de Caso de Uso: Buscar Profesionales

III.1.2.7.5.3. Especificación de Caso de Uso: Ver blogs recientemente creados

Nombre Caso de Uso:	**Ver blogs recientemente creados**
Identificador:	**FAP3**
Descripción: Este caso de uso el usuario que visita la página de la Plataforma puede ver una lista de los blogs recientemente creados	
Precondiciones: **1.** Un usuario visita la pagina Web de la Plataforma	
Flujo Básico de la Acción: **1.** En el index o página principal de la Web de la Plataforma se tiene la lista de todos los Colegios que tienen su blog activo y recientemente creado. **2.** El visitador elige cualquier blog de la lista. **3.** Inmediatamente se abre en otra ventana el blog elegido mostrando su página principal o index.	

Tabla 41. Especificación de Caso de Uso: Ver blogs recientemente creados

III.1.2.7.5.4. Especificación de Caso de Uso: Ver entradas más comentadas

Nombre Caso de Uso:	**Ver Entradas más comentadas**
Identificador:	**FAP4**
Descripción: Este caso de uso el usuario que visita la página de la Plataforma puede ver las entradas más comentadas de todos los blogs publicados.	
Precondiciones: El usuario no registrado visita la pagina Web de la Plataforma	
Flujo Básico de la Acción: **1.** En el index o página principal de la Web de la Plataforma se tiene la lista de todas las entradas o artículos más comentados. **2.** El usuario elige la entrada que quiere leer. **3.** Inmediatamente se abre en otra ventana el blog al que pertenece la entrada mostrando la entrada o articulo completo para que el usuario pueda seguir leyendo.	

Tabla 42. Especificación de Caso de Uso: Ver entradas mas comentadas

III.1.2.7.5.5. Especificación de Caso de Uso: Ver encuestas recientes

Nombre Caso de Uso:	**Ver encuestas recientes**
Identificador:	**FAP5**
Descripción: Este caso de uso el usuario que visita la página de la Plataforma puede ver las encuestas que recientemente se publicaron dentro los diez últimos días.	
Precondiciones: La persona visita la pagina Web de la Plataforma	
Flujo Básico de la Acción: **1.** En el index o página principal de la Web de la Plataforma se tiene la lista de todas las entradas o artículos más comentados. **2.** El usuario elige la encuesta que le interesa. **3.** Inmediatamente se abre en otra ventana el blog a la que pertenece la encuesta, dándole la opción a la persona para que pueda emitir su voto.	

Tabla 43. Especificación de Caso de Uso: Ver encuestas recientes

III.1.2.7.6. Especificación de Caso de Uso: Administrar publicaciones en Blog

III.1.2.7.6.1. Especificación de Caso de Uso: Crear Entradas

Nombre Caso de Uso:	**Crear Entradas**
Identificador:	**BAB1**
Descripción: Este caso de uso puede ser realizado por el administrador de blog o los afiliados registrados habilitados. Este caso de uso permite crear un artículo o entrada en el blog.	
Precondiciones: **2.** El usuario registrado ingresa correctamente al administrador de su blog **3.** Elige la opción Publicar del menubar	
Pos condiciones: Se actualiza el numero entradas publicadas en el blog	
Flujo Básico de la Acción: **5.** El usuario registrado elige la opción Entrada del submenú Publicar **6.** Inmediatamente aparece en pantalla la interfaz para crear la entrada **7.** El usuario introduce el nombre del articulo **8.** El usuario redacta la entrada en el editor, puede darle el formato, color tipo de	

letra, etc. al contenido del artículo. Si el usuario desea adicionar imágenes puede hacerlo desde la Web con el editor. **9.** Si el publicador quiere que su Entrada se publique en el blog selecciona esta opción **10.** Si el usuario quiere que la entrada o articulo sea comentado selecciona la opción *Permitir Comentarios* **11.** El usuario ingresa un pequeño párrafo de la entrada que publico para mostrarlo en lista. **12.** Selecciona la categoría a la que pertenecerá la entrada **13.** Si el usuario desea insertar en el artículo una imagen o archivo de su computadora, examina el archivo en su sistema. **14.** Una vez realizado todo el proceso para crear el artículo, presiona el botón crea, **15.** La plataforma verifica que todos los campos necesarios estén completos y que no exista ningún error. **16.** Se guarda la entrada en la base de datos **17.** La pantalla se actualiza y se muestra en lista la entrada creada
Flujo Alternativo de la Acción: **3.** En la validación del artículo o entrada creada se encuentra errores. **4.** Se muestra mensajes indicando donde se encuentra el error señalado con rojo

Tabla 44. Especificación de Caso de Uso: Crear Entrada

III.1.2.7.6.2. Especificación de Caso de Uso: Gestionar Entradas Publicadas

Nombre Caso de Uso:	**Gestionar Entradas Publicadas**
Identificador:	**BAB2**
Descripción: Este caso de uso permite administrar las entradas publicadas. El Administrador de Blog puede gestionar todas las entradas publicadas por todos los usuarios afiliados, en cambio el afiliado solo tiene el control sobre sus propias entradas.	
Precondiciones: **1.** El usuario registrado ingresa correctamente al administrador de su blog **2.** Elige la opción Gestionar del menubar	
Pos condiciones: El usuario realiza alguna de las siguientes acciones que se le permite cuando visualiza la lista de entradas	
Flujo Básico de la Acción: **1.** El usuario registrado elige la opción Entradas del submenú	

2. Inmediatamente se muestra en pantalla las entradas publicadas, si el usuario es administrador se muestran todas las entradas del blog, si es afiliado solo se muestran sus entradas publicadas. El usuario puede seleccionar dar de habilitar o deshabilitar la publicación de la entrada, pasar al punto 3; modificar entrada, pasar al punto 4; eliminar entrada, pasar al punto 5, ver comentarios 6.

3. El usuario selecciona habilitar o deshabilitar entrada

3.1. El usuario selecciona una entrada especifica

3.2. Si la entrada está habilitada tendrá la opción de Deshabilitar, si la entrada esta deshabilitada tendrá la opción habilitar.

3.3. Dependiendo de la opción que se muestra el usuario podrá habilitar o deshabilitar la entrada.

3.4. Internamente se cambiara el estado de la entrada en base de datos

3.5. En pantalla se actualizara el estado de la entrada en la lista.

4. El usuario modifica entrada

4.1. El usuario selecciona la entrada que desea modificar

4.2. Presiona la opción modificar de la entrada seleccionada

4.3. Internamente se hace la selección del contenido y los datos de la entrada en base de datos

4.4. En pantalla se muestra la entrada en la interfaz que permite modificar sus datos y contenido.

4.5. El usuario modifica el contenido de la entrada en el editor, puede cambiar el contenido, o aumentarlo, cambiar formato, etc.

4.6. El usuario puede modificar los demás datos de la entrada como titulo, cita, cambiar de categoría, permitir o no permitir comentarios, adjuntar archivo, adjuntar imagen.

4.7. Presiona Modificar Entrada

4.8. Si los datos modificados son validos y correctos se actualiza la entrada en base de datos, caso contrario no se realiza la transacción.

4.9. Se direcciona a la lista de entradas y aparece el mensaje *"Entrada (titulo) modificada"*

5. El usuario elimina entrada

5.1. El usuario selecciona una entrada de la lista

5.2. Presiona su opción para Eliminar

5.3. Aparece un mensaje de confirmación ¿Desea eliminar la entrada?

5.4. Si el usuario elige SI en el mensaje se realiza la eliminación

5.5. Internamente en base de datos se eliminan los comentarios y la entrada en cascada.

5.6. Se actualiza la entrada pero ya no se muestra la entrada porque fue eliminada.

6. El usuario selecciona la opción Comentarios.

6.1. Selecciona una entrada a elegir

6.2. Hace clic en la opción Comentarios de la entrada seleccionada

6.3. Internamente se selecciona todos los comentarios que pertenecen a esa entrada

6.4. En pantalla se muestra la lista de los comentarios seleccionados

6.5. E usuario puede realizar el caso de uso BAB3.

Flujo Alternativo de la Acción: **1.** El usuario registrado elige la opción Entradas del submenú **2.** Internamente se selecciona las entradas publicadas, si el usuario es administrador se muestran todas las entradas del blog, si es afiliado solo se muestran sus entradas publicadas. **3.** Si no se encuentra nada aparece el mensaje "aun no se publico niguna entrada en el blog" para el administrador, para el afiliado aparece el mensaje "Aun no publico ninguna entrada".

Tabla 45. Especificación de Caso de Uso: Gestionar Entradas publicadas

III.1.2.7.6.3. Especificación de Caso de Uso: Gestionar Comentarios

Nombre Caso de Uso:	**Gestionar Comentarios**
Identificador:	**BAB2-1**
Descripción: Este caso de uso permite administrar los comentarios de la entrada que se selecciono. Este caso de uso puede realizarlo tanto el administrador de blog como el afiliado pero el solo gestiona los comentarios de sus entradas.	
Precondiciones: **1.** El usuario registrado ingresa correctamente al administrador de su blog **2.** Elige la opción Gestionar del menubar **3.** El usuario realiza el caso de uso BAB2 en el punto 6	
Pos condiciones: El usuario realiza alguna de las siguientes acciones que se le permite cuando visualiza la lista comentarios	
Flujo Básico de la Acción: **1.** El usuario selecciona un comentario de la lista y se pinta de otro color **2.** El usuario puede seleccionar ver el comentario, pasar al punto 3; eliminar comentario, pasar al punto 4, responder comentario pasar punto 5. **3.** El usuario selecciona ver comentario **3.1.** Internamente se busca el comentario en la base de datos para mostrarlo en pantalla. **3.2.** Se muestra en pantalla el contenido y datos del comentario seleccionado **4.** El usuario elimina comentario **4.1.** El usuario selecciona un comentario de la lista **4.2.** Presiona su opción para Eliminar **4.3.** Aparece un mensaje de confirmación ¿Desea eliminar el comentario seleccionado? **4.4.** Si el usuario elige SI en el mensaje se realiza la eliminación	

4.5. Internamente en base de datos se elimina el comentario.

4.6. Se actualiza la lista de comentarios.

5. El usuario responde comentario.

5.1. El usuario elige la opción contestar.

5.2. Se muestra una pantalla en la que puede responder al comentario seleccionado

5.3. Presiona Guardar en la pantallita emergente.

5.4. Se valida que los datos sean correctos y que los campos obligatorios se hayan llenado.

5.5. Si validación no emite errores, se guarda en base de datos.

Flujo Alternativo de la Acción:

1. El usuario realiza el caso de uso BAB2 en el punto 6

2. Si no se encuentra ningún comentario sale el siguiente mensaje "No hay comentarios para esta entrada"

Tabla 46. Especificación de Casos de Uso: Gestionar Cometarios

III.1.2.7.6.4. Especificación de Caso de Uso: Publicar Enlaces

Nombre Caso de Uso:	**Publicar Enlaces**
Identificador:	**BAB3**

Descripción:

Este caso de uso puede ser realizado por el administrador de blog o los afiliados registrados habilitados. Este caso de uso permite publicar un enlace en el blog.

Precondiciones:

1. El usuario registrado ingresa correctamente al administrador de su blog

2. Elige la opción Publicar del menubar

Pos condiciones:

Se actualiza el numero enlaces publicados en el blog

Flujo Básico de la Acción:

1. El usuario elige la opción Enlace del submenú Publicar

2. Inmediatamente aparece en pantalla la interfaz para publicar el enlace

3. El usuario ingresa los datos del enlace nombre, URL de enlace y si se requiere una breve descripción.

4. El usuario presiona guardar.

5. Se valida que los datos sean correctos, si los datos son incorrectos sale los mensajes indicando el error de entrada de datos.

6. Si los datos introducidos son correctos se guarda el enlace en la base de datos.

Flujo Alternativo de la Acción:

1. En la validación del enlace se encuentra errores.

2. Se muestra mensajes indicando donde se encuentra el error señalado con rojo

Tabla 47. Especificación de Caso de Uso: Publicar Enlaces

III.1.2.7.6.5. Especificación de Caso de Uso: Gestionar Enlaces publicados

Nombre Caso de Uso:	**Gestionar Enlaces publicados**
Identificador:	**BAB4**
Descripción: Este caso de uso permite administrar los enlaces. Este caso de uso puede realizarlo tanto el administrador de blog como el afiliado pero el afiliado solo gestiona los enlaces que publico.	
Precondiciones: **1.** El usuario registrado ingresa correctamente a la parte administrativa del blog **2.** Elige la opción Gestionar del menubar	
Pos condiciones: El usuario realiza alguna de las siguientes acciones que se le permite cuando visualiza la lista de enlaces	
Flujo Básico de la Acción: **1.** El usuario selecciona un enlace de la lista y se pinta de otro color **2.** El usuario puede seleccionar modificar enlace pasar al punto 3; eliminar enlace, pasar al punto 4. **3.** El usuario seleccionar modificar enlace **3.1.** Internamente se busca el enlace en la base de datos para poder modificarlo. **3.2.** En pantalla se muestra la interfaz par modificar los datos del enlace. **3.3.** El usuario modifica los datos que necesite. **3.4.** Presiona el botón Actualizar enlace. **3.5.** Se valida los datos corregidos por el usuario. **3.6.** Si los datos son correctos se actualiza el enlace en base de datos **4.** El usuario elimina enlace **4.1.** El usuario selecciona un enlace de la lista **4.2.** Presiona su opción para Eliminar **4.3.** Aparece un mensaje de confirmación ¿Desea eliminar el enlace seleccionado? **4.4.** Si el usuario elige SI en el mensaje se realiza la eliminación **4.5.** Internamente en base de datos se elimina el enlace. **4.6.** Se actualiza la lista de enlaces.	
Flujo Alternativo de la Acción: En el punto 3, al validar los datos corregidos por el usuario, Si los datos son incorrectos se muestra mensajes de error.	

Tabla 48. Especificación de Caso de Uso: Gestionar Enlaces publicados

III.1.2.7.6.6. Especificación de Caso de Uso: Subir Fotos o Imágenes

Nombre Caso de Uso:	**Subir Fotos o Imágenes**
Identificador:	**BAB5**
Descripción: Este caso de uso permite subir y publicar fotos o imágenes. Este caso de uso puede realizarlo tanto el administrador de blog como el afiliado.	
Precondiciones: **1.** El usuario registrado ingresa correctamente a la parte administrativa del blog **2.** Elige la opción Publicar del menubar	
Pos condiciones: Se publica en el blog la imagen subida	
Flujo Básico de la Acción: **1.** El usuario selecciona la opción Imágenes del submenú de Publicar **2.** Aparece la interfaz que permite subir imágenes al blog. **3.** El usuario ingresa el nombre de la imagen o foto. **4.** Examina en su computadora la imagen que quiere subir, identifica la ruta de la imagen. **5.** Si el usuario quiere puede ingresar una breve descripción. **6.** Presiona Subir y si todo esta correcto la imagen o foto es guardada en la base de datos de la plataforma del correspondiente blog. **7.** Se actualiza la interfaz y se puede ver la imagen subida.	
Flujo Alternativo de la Acción: **1.** Si el archivo subido no es una imagen, saldrá el error la imagen subida no tiene la extensión bmp, jpg, png, etc, ya que solo se permite subir imágenes en todos sus formatos.	

Tabla 49. Especificación de Caso de Uso: Subir Fotos & Imágenes

III.1.2.7.6.7. Especificación de Caso de Uso: Gestionar Imágenes subidas

Nombre Caso de Uso:	**Gestionar Imágenes subidas**
Identificador:	**BAB6**
Descripción: Este caso de uso permite administrar las imágenes subidas. Este caso de uso puede realizarlo tanto el administrador de blog como el afiliado pero el afiliado solo gestiona las imágenes que publico.	
Precondiciones: **1.** El usuario registrado ingresa correctamente a la parte administrativa del blog **2.** Elige la opción Gestionar del menubar	

Pos condiciones: El usuario realiza alguna de las siguientes acciones que se le permite cuando visualizar las imágenes publicadas
Flujo Básico de la Acción: **1.** El usuario selecciona Fotos & Imágenes del submenú de Gestionar **2.** Inmediatamente aparecen todas las imágenes publicadas con las siguientes opciones modificar nombre pasar al punto 4, ver tamaño completo pasar punto 5, eliminar pasar punto 6. **3.** El usuario selecciona una imagen específica y elige una de las opciones. **4.** Selecciona la opción modificar **4.7.** Aparece una ventanita emergente que le permitirá introducir los nuevos datos. **4.8.** El usuario introduce los nuevos datos y presiona Modificar. **4.9.** Se actualiza los datos de la imagen en la base de datos. **5.** El usuario selecciona la opción ver tamaño completo. **5.1.** La imagen seleccionada es mostrada en una ventana emergente en su tamaño completo, con la opción Cerrar Ventana. **6.** El usuario selecciona la opción Eliminar **6.1.** Aparece un mensaje de confirmación ¿Desea eliminar esta imagen? **6.2.** Si el usuario elige SI en el mensaje se realiza la eliminación **6.3.** Internamente en base de datos se elimina la imagen **6.4.** Se actualiza la pagina que muestra las imágenes
Flujo Alternativo de la Acción: **1.** En el punto 6, si el usuario elige la opción NO en el mensaje de confirmación, la imagen no se elimina y no ocurre ningún evento.

Tabla 50. Especificación de Caso de Uso: Gestionar Imágenes subidas

III.1.2.7.6.8. Especificación de Caso de Uso: Subir Archivos

Nombre Caso de Uso:	**Subir Archivos**
Identificador:	**BAB7**
Descripción: Este caso de uso permite subir y publicar archivos. Este caso de uso puede realizarlo tanto el administrador de blog como el afiliado.	
Precondiciones: **3.** El usuario registrado ingresa correctamente a la parte administrativa del blog **4.** Elige la opción Publicar del menubar	

Pos condiciones: Se publica en el blog el archivo subido.
Flujo Básico de la Acción: **8.** El usuario selecciona la opción Archivos del submenú de Publicar **9.** Aparece la interfaz que permite subir archivos. **10.** El usuario ingresa el nombre del archivo. **11.** Examina en su computadora el archivo que quiere subir, identifica la ruta del archivo. **12.** Si el usuario quiere puede ingresar una breve descripción. **13.** Presiona Subir y si todo esta correcto el archivo es guardado en la base de datos de la plataforma del correspondiente blog. **14.** Se actualiza la pantalla y se muestra el link del archivo subido
Flujo Alternativo de la Acción: **2.** Si el archivo subido tiene un tamaño que excede los 350 Mb sale el mensaje tamaño excedido.

Tabla 51. Especificación de Caso de Uso: Subir Archivos

III.1.2.7.6.9. Especificación de Caso de Uso: Gestionar Archivos

Nombre Caso de Uso:	**Gestionar Archivos**
Identificador:	**BAB8**
Descripción: Este caso de uso permite administrar los archivos subidos. Este caso de uso puede realizarlo tanto el administrador de blog como el afiliado pero el afiliado solo gestiona los archivos que subió.	
Precondiciones: **1.** El usuario registrado ingresa correctamente a la parte administrativa del blog **2.** Elige la opción Gestionar del menubar	
Pos condiciones: El usuario realiza alguna de las siguientes acciones que se le permite cuando visualiza los archivos subidos.	
Flujo Básico de la Acción: **1.** El usuario selecciona Archivos del submenú de Gestionar **2.** Inmediatamente aparecen todas los archivos publicados con la opción Descargar pasar punto 3, la opción Eliminar pasar punto 4. **3.** El usuario selecciona la opción Descargar. **3.1.** Este caso de uso es similar al caso de uso **FAB5-1,** claro que en este caso se lo hace desde el administrador del blog.	

4. El usuario selecciona la opción Eliminar 4.1. Aparece un mensaje de confirmación ¿Desea eliminar el archivo? 4.2. Si el usuario elige SI en el mensaje se realiza la eliminación 4.3. Internamente en base de datos se elimina la imagen 4.4. Se actualiza la lista de archivos publicados
Flujo Alternativo de la Acción: **1.** En el punto 6, si el usuario elige la opción NO en el mensaje de confirmación, la imagen no se elimina y no ocurre ningún evento.

Tabla 52. Especificación de Caso de Uso: Gestionar Archivos

III.1.2.7.6.10. Especificación de Caso de Uso: Editar Datos Perfil

Nombre Caso de Uso:	**Editar Datos Perfil**
Identificador:	**BAB9**
Descripción: Este caso de uso permite editar los datos del perfil del usuario. Tanto el administrador como cualquier afiliado puede editar sus datos.	
Precondiciones: **1.** El usuario registrado ingresa correctamente a la parte administrativa del blog **2.** Elige la opción Afiliado-Usuario del menubar	
Pos condiciones: Se actualiza su información personal publicada en el blog.	
Flujo Básico de la Acción: **1.** En pantalla se muestra la interfaz para modificar los datos del usuario. **2.** El usuario modifica los datos que necesite. **3.** Presiona el botón Actualizar. **4.** Se valida los datos corregidos por el usuario. **5.** Si los datos son correctos se actualiza su información personal en base de datos.	
Flujo Alternativo de la Acción: Si los datos son incorrectos se muestra mensajes de error en la validación.	

Tabla 53. Especificación de Caso de Uso: Editar Datos Perfil

III.1.2.7.6.11. Especificación de Caso de Uso: Cambiar contraseña

Nombre Caso de Uso:	**Cambiar contraseña**
Identificador:	**BAB9-1**
Descripción: Este caso de uso le permite al usuario cambiar contraseña.	
Precondiciones: **3.** El usuario registrado ingresa correctamente a la parte administrativa del blog **4.** Elige la opción Afiliado-Usuario del menubar	
Pos condiciones: Se actualiza su información personal publicada en el blog.	
Flujo Básico de la Acción: **6.** En pantalla se muestra la interfaz para modificar la clave del usuario **7.** El usuario debe ingresar el nombre de usuario, la contraseña actual y la nueva contraseña que desea tener. **8.** Presiona Guardar. **9.** Se comprueba que el nombre de usuario y la contraseña actual corresponda al usuario. **10.** Si son correctos estos datos, la nueva contraseña es encriptada y guardada en base de datos..	
Flujo Alternativo de la Acción: Si los datos de comprobación de usuario son incorrectos, sale mensajes de error.	

Tabla 54. Especificación de Caso de Uso: Cambiar contraseña

III.1.2.7.6.12. Especificación de Caso de Uso: Editar Datos CV

Nombre Caso de Uso:	**Editar Datos CV**
Identificador:	**BAB10**
Descripción: Este caso de uso permite editar los datos del Curriculum Vitae del Afiliado registrado. Tanto el administrador como cualquier afiliado puede editar su CV.	
Precondiciones: **1.** El usuario registrado ingresa correctamente a la parte administrativa del blog. **2.** Elige la opción Datos CV del menubar	
Pos condiciones: Se actualiza la información de su CV publicada en el blog.	
Flujo Básico de la Acción: **1.** El usuario puede elegir una de las siguientes opciones, Experiencia Laboral pasar punto 2, Capacidades Personales pasar punto 3, Formación Académica pasar punto 4, Idiomas pasar punto 5, Publicaciones pasar punto 6, Publicar CV pasar	

punto 7.

2. Caso de uso BAB-1 Editar Experiencia Laboral.
3. Caso de uso BAB-2 Editar Capacidades Personales.
4. Caso de uso BAB-3 Editar Formación Académica.
5. Caso de uso BAB-4 Editar Idiomas
6. Caso de uso BAB-5 Editar Publicaciones.
7. Caso de uso BAB-6 Editar Publicar CV.

Tabla 55. Especificación de Caso de Uso: Editar Datos CV

III.1.2.7.6.13. Especificación de Caso de Uso: Editar Experiencia Laboral

Nombre Caso de Uso:	**Editar Experiencia Laboral.**
Identificador:	**BAB10-1**

Descripción:

Este caso de uso permite editar los datos de experiencias laborales del Afiliado para que se muestre en su CV. Tanto el administrador como cualquier afiliado puede editar su CV.

Precondiciones:

1. El usuario registrado ingresa correctamente a la parte administrativa del blog.
2. Elige la opción Datos CV del menubar

Pos condiciones:

Se actualiza la información de su CV publicada en el blog.

Flujo Básico de la Acción:

1. El usuario elige opción Experiencia Laboral
2. Inmediatamente aparece en pantalla la lista de experiencias laborales que ya registro dándole la posibilidad de modificarlas pasar punto 3. y de registrar una nueva pasar punto 4.
3. En este caso aparece la interfaz con los datos de la experiencia laboral que desea modificar.

3.1. El usuario modifica los datos que requiere de actualización.

3.2. Presiona Actualizar, y si los datos ingresados son correctos se guardan en base de datos.

3.3. La pantalla se actualiza y vuelve a mostrar la lista de experiencias laborales.

4. En este caso aparece la interfaz que permite crear una nueva experiencia laboral.

4.1. El usuario ingresa los datos como ser fechas de inicio y fin, labor principal, datos de la empresa o empleador que lo contrato y el area en que se desempeño.

4.2. Una vez ingresados los datos presiona crear.

4.3. Si los datos de la nueva experiencia laboral son correctos se guarda en base de datos.

4.4. La pantalla se actualiza y vuelve a mostrar la lista de experiencias laborales adicionando la recién creada.

Flujo Alternativo de la Acción:
1. Si no existe ninguna experiencia laboral registrada aparecerá el mensaje no registro ninguna Experiencia Laboral. **2.** En caso de ingresar datos incorrectos al momento de registrar (punto 3) o modificar (punto 4) una experiencia laboral, si los datos son incorrectos al momento de validarlos saldrá mensajes de error.

Tabla 56. Especificación de Caso de Uso: Editar Experiencia Laboral

III.1.2.7.6.14. Especificación de Caso de Uso: Editar Capacidades Personales

Nombre Caso de Uso:	**Editar Capacidades Personales**
Identificador:	**BAB10-2**
Descripción: Este caso de uso permite editar los datos de Capacidades Personales del Afiliado para que se muestre en su CV. Tanto el administrador como cualquier afiliado puede editar su CV.	
Precondiciones: **1.** El usuario registrado ingresa correctamente a la parte administrativa del blog. **2.** Elige la opción Datos CV del menubar	
Pos condiciones: Se actualiza la información de su CV publicada en el blog.	
Flujo Básico de la Acción: **1.** El usuario elige opción Capacidades Personales **2.** Inmediatamente aparece en pantalla la lista de Capacidades Personales que ya registro dándole la posibilidad de modificarlas pasar punto 3. y de registrar una nueva pasar punto 4. **3.** En este caso aparece la interfaz con los datos de la Capacidad Personal que desea modificar. **3.1.** El usuario modifica los datos que requiere de actualización. **3.2.** Presiona Actualizar, y si los datos ingresados son correctos se guardan en base de datos. **3.3.** La pantalla se actualiza y vuelve a mostrar la lista de Capacidades Personales. **5.** En este caso aparece la interfaz que permite crear una nueva Capacidad Personal. **5.1.** El usuario ingresa los datos como ser tipo de capacidad, habilidades y logros. **5.2.** Una vez ingresados los datos presiona crear. **5.3.** Si los datos de la nueva Capacidad Personal son correctos se guarda en base de datos. **5.4.** La pantalla se actualiza y vuelve a mostrar la lista de Capacidades Personales adicionando la recién creada.	

Flujo Alternativo de la Acción:
1. Si no existe ninguna experiencia laboral registrada aparecerá el mensaje no registro ninguna Capacidades Personales. **2.** En caso de ingresar datos incorrectos al momento de registrar (punto 3) o modificar (punto 4) una Capacidad Personal, si los datos son incorrectos al momento de validarlos saldrá mensajes de error.

Tabla 57. Especificación de Caso de Uso: Editar Capacidades Personales

III.1.2.7.6.15. Especificación de Caso de Uso: Editar Formación Académica

Nombre Caso de Uso:	**Editar Formación Académica**
Identificador:	**BAB10-3**
Descripción: Este caso de uso permite editar los datos de Formación Académica del Afiliado para que se muestre en su CV. Tanto el administrador como cualquier afiliado puede editar su CV.	
Precondiciones: **1.** El usuario registrado ingresa correctamente a la parte administrativa del blog. **2.** Elige la opción Datos CV del menubar	
Pos condiciones: Se actualiza la información de su CV publicada en el blog.	
Flujo Básico de la Acción: **1.** El usuario elige opción Formación Académica. **2.** Inmediatamente aparece en pantalla la lista de Formaciones Académicas que ya registro dándole la posibilidad de modificarlas pasar punto 3. y de registrar una nueva pasar punto 4. **3.** En este caso aparece la interfaz con los datos de la Formación Académica que desea modificar. **3.1.** El usuario modifica los datos que requiere de actualización. **3.2.** Presiona Actualizar, y si los datos ingresados son correctos se guardan en base de datos. **3.3.** La pantalla se actualiza y vuelve a mostrar la lista de Formaciones Académicas. **4.** En este caso aparece la interfaz que permite crear una nueva Formación Académica. **4.1.** El usuario ingresa los datos como ser el título obtenido, los datos del centro que lo capacito, principales materias aprendidas, el nivel alcanzado y por ultimo fechas de inicio y fin. **4.2.** Una vez ingresados los datos presiona crear. **4.3.** Si los datos de la nueva Formación Académica son correctos se guarda en base de datos. **4.4.** La pantalla se actualiza y vuelve a mostrar la lista de Formaciones Académicas	

adicionando la recién creada.
Flujo Alternativo de la Acción: **1.** Si no existe ninguna experiencia laboral registrada aparecerá el mensaje no registro ninguna Formación Académica. **2.** En caso de ingresar datos incorrectos al momento de registrar (punto 3) o modificar (punto 4) una Formación Académica, si los datos son incorrectos al momento de validarlos saldrá mensajes de error.

Tabla 58. Especificación de Caso de Uso: Editar Formación Académica

III.1.2.7.6.16. Especificación de Caso de Uso: Editar Idiomas

Nombre Caso de Uso:	**Editar Idiomas**
Identificador:	**BAB10-4**
Descripción: Este caso de uso permite registrar y editar los Idiomas que el Afiliado habla o comprende para que se muestre en su CV. Tanto el administrador como cualquier afiliado puede editar su CV.	
Precondiciones: **1.** El usuario registrado ingresa correctamente a la parte administrativa del blog. **2.** Elige la opción Datos CV del menubar	
Pos condiciones: Se actualiza la información de su CV publicada en el blog.	
Flujo Básico de la Acción: **1.** El usuario elige opción Idiomas **2.** Inmediatamente aparece en pantalla la lista de Idiomas que ya registro dándole la posibilidad de modificarlas pasar punto 3 y de registrar un nuevo idioma pasar punto 4. **3.** En este caso aparece la interfaz con los datos del Idioma que desea modificar. **3.1.** El usuario modifica los datos que requiere de actualización. **3.2.** Presiona Actualizar, y si los datos ingresados son correctos se guardan en base de datos. **3.3.** La pantalla se actualiza y vuelve a mostrar la lista de Idiomas. **4.** En este caso aparece la interfaz que permite registrar un nuevo Idioma. **4.1.** El usuario ingresa los datos como ser nombre del idioma, nivel de comprensión auditiva, nivel de comprensión de lectura, nivel de expresión oral, nivel de escritura. **4.2.** Una vez ingresados los datos presiona Registrar. **4.3.** Si los datos del nuevo Idioma son correctos se guarda en base de datos. **4.4.** La pantalla se actualiza y vuelve a mostrar la lista de Idiomas adicionando el	

recién creado.
Flujo Alternativo de la Acción: **1.** Si no existe ningún Idioma registrado aparecerá el mensaje Hasta el momento no registro ningún idioma para ser publicado en su CV. **2.** En caso de ingresar datos incorrectos al momento de registrar (punto 3) o modificar (punto 4) una Idioma, si los datos son incorrectos al momento de validarlos saldrá mensajes de error.

Tabla 59. Especificación de Caso de Uso: Editar Idiomas

III.1.2.7.6.17. Especificación de Caso de Uso: Configurar CV

Nombre Caso de Uso:	**Configurar CV**
Identificador:	**BAB10-6**
Descripción: En este escenario el usuario permite o no la publicación de su CV en el Blog, y determina que datos quiere que aparezca.	
Precondiciones: **1.** El usuario registrado ingresa correctamente a la parte administrativa del blog. **2.** Elige la opción Datos CV del menubar	
Pos condiciones: Se modifica la publicación del CV en el blog	
Flujo Básico de la Acción: **1.** El usuario elige la opción Configurar CV**.** **2.** Inmediatamente aparece en pantalla que permite configurar el CV del afiliado **3.** El usuario selecciona la opción Permitir ver CV. **4.** Enseguida las opciones del CV se habilitan. **5.** El usuario selecciona los datos que quiere que aparezcan en su CV. **6.** Presiona el botón Configurar y la configuración del CV es guardado en el sistema.	
Flujo Alternativo de la Acción: **1.** Si el usuario no selecciona la opción permitir Ver CV las opciones de configuración no se habilitan.	

Tabla 60. Especificación de Caso de Uso: Configurar CV

III.1.2.7.7. Especificaciones de Caso de Uso: Administrar Blog

III.1.2.7.7.1. Especificación de Caso de Uso: Publicar Pagina

Nombre Caso de Uso:	**Publicar Paginas**
Identificador:	**BAB11**

Descripción:

Este caso de uso solo puede ser realizado por el administrador de blog. Este caso de uso permite crear una página para el blog.

Precondiciones:

1. El usuario registrado ingresa correctamente al administrador de su blog
2. Elige la opción Publicar del menubar

Pos condiciones:

Se actualiza el numero paginas publicadas en el blog

Flujo Básico de la Acción:

1. El usuario registrado elige la opción Pagina del submenú Publicar
2. Inmediatamente aparece en pantalla la interfaz para crear la pagina
3. El usuario introduce el nombre de la pagina
4. El usuario redacta la página en el editor, puede darle el formato, color tipo de letra, etc. al contenido de la página.
5. Si el usuario quiere que la página se publique en el blog selecciona esta opción.
6. Si el usuario desea insertar en la página una imagen o archivo de su computadora, examina el archivo en su sistema.
7. Una vez realizado todo el proceso para crear la pagina, presiona el botón crear,
8. La plataforma verifica que todos los campos necesarios estén completos y que no exista ningún error.
9. Se guarda la entrada en la base de datos
10. La pantalla se actualiza y se muestra en lista la página creada.

Flujo Alternativo de la Acción:

1. En la validación de los datos de la página se encuentra errores.
2. Se muestra mensajes indicando donde se encuentra el error señalado con rojo

Tabla 61. Especificación de Caso de Uso: Publicar Pagina

III.1.2.7.7.2. Especificación de Caso de Uso: Gestionar Paginas

Nombre Caso de Uso:	**Gestionar Paginas**
Identificador:	**BAB12**

Descripción:

Este caso de uso permite administrar las páginas publicadas. Solo el Administrador de Blog puede gestionar todas las páginas creadas.

Precondiciones:

1. El usuario registrado ingresa correctamente al administrador de su blog
2. Elige la opción Gestionar del menubar

Pos condiciones:

El usuario realiza alguna de las siguientes acciones que se le permite cuando visualiza la lista de páginas.

Flujo Básico de la Acción:

1. El usuario registrado elige la opción Paginas del submenú
2. Inmediatamente se muestra en pantalla las páginas publicadas. El usuario puede seleccionar habilitar o deshabilitar la publicación de la pagina, pasar al punto 3; modificar pagina, pasar al punto 4; eliminar pagina, pasar al punto 5.
3. El usuario selecciona habilitar o deshabilitar pagina
3.1. El usuario selecciona una página especifica
3.2. Si la entrada está habilitada tendrá la opción de Deshabilitar, si la pagina esta deshabilitada tendrá la opción habilitar.
3.3. Dependiendo de la opción que se muestra el usuario podrá habilitar o deshabilitar la pagina.
3.4. Internamente se cambiara el estado de la pagina en base de datos
3.5. En pantalla se actualizara el estado de la página en la lista.
4. El usuario modifica página
4.1. El usuario selecciona la página que desea modificar
4.2. Presiona la opción modificar de la página seleccionada
4.3. Internamente se hace la selección del contenido y los datos de la página en base de datos
4.4. En pantalla se muestra la página en la interfaz que permite modificar sus datos y contenido.
4.5. El usuario modifica el contenido de la página en el editor, puede cambiar el contenido, o aumentarlo, cambiar formato, etc.
4.6. El usuario puede modificar los demás datos de la página, adjuntar archivo, adjuntar imagen.
4.7. Presiona Modificar Página.
4.8. Si los datos modificados son validos y correctos se actualiza la página en base de datos, caso contrario no se realiza la transacción.
4.9. Se direcciona a la lista de páginas y aparece el mensaje *"Pagina (titulo)*

modificada" **5.** El usuario elimina página **5.1.** El usuario selecciona una página de la lista **5.2.** Presiona su opción para Eliminar **5.3.** Aparece un mensaje de confirmación ¿Desea eliminar la página? **5.4.** Si el usuario elige SI en el mensaje se realiza la eliminación **5.5.** Internamente en base de datos se eliminan los datos de la página. **5.6.** Se actualiza la lista de páginas.
Flujo Alternativo de la Acción: **1.** El usuario registrado elige la opción Paginas del submenú **2.** Internamente se selecciona las páginas publicadas. **3.** Si no se encuentra nada aparece el mensaje "aun no se publico ninguna pagina en el blog que permita presentar al Colegio".

Tabla 62. Especificación de Caso de Uso: Gestionar Páginas

III.1.2.7.7.3. Especificación de Caso de Uso: Gestionar Categorías-Enlaces

Nombre Caso de Uso:	**Gestionar Categorías-Enlaces**
Identificador:	**BAB13**
Descripción: Este caso de uso permite administrar las categorías para publicar enlaces. Solo el Administrador de Blog puede realizar este caso de uso.	
Precondiciones: **1.** El usuario registrado ingresa correctamente al administrador de su blog **2.** Elige la opción Gestionar Categorías del menubar	
Pos condiciones: El usuario realiza alguna de las siguientes acciones que se le permite cuando visualiza la lista de categorías de enlace.	
Flujo Básico de la Acción: **1.** El usuario registrado elige la opción Categorías-Enlaces del submenú **2.** Inmediatamente se muestra en pantalla las categorías para enlaces publicadas. El usuario puede seleccionar modificar categoría, pasar al punto 3; eliminar categoría, pasar al punto 4. **3.** El usuario modifica categoría-enlace **3.1.** El usuario selecciona la categoría que desea modificar **3.2.** Presiona la opción modificar de la categoría seleccionada **3.3.** Internamente se hace la selección de los datos de la categoría en base de datos **3.4.** En pantalla se muestra la categoría en la interfaz que permite modificar sus datos. **3.5.** El usuario modifica los datos de la categoría. **3.6.** Presiona Modificar. **3.7.** Si los datos modificados son validos y correctos se actualiza la categoria en base	

de datos, caso contrario no se realiza la transacción. **3.8.** Se direcciona a la lista de páginas y aparece el mensaje *"Categoria (titulo) modificada"* **4.** El usuario elimina categoría **4.1.** El usuario selecciona una categoría de la lista. **4.2.** Presiona su opción para Eliminar. **4.3.** Aparece un mensaje de confirmación ¿Desea eliminar la categoría? **4.4.** Si el usuario elige SI en el mensaje se realiza la eliminación **4.5.** Internamente en base de datos se eliminan los datos de la categoría. **4.6.** Se actualiza la lista de categorías.
Flujo Alternativo de la Acción: **1.** El usuario registrado elige la opción Categorías-Enlaces del submenú **2.** Internamente se selecciona las categorías para enlaces publicadas. **3.** Si no se encuentra nada aparece el mensaje "*Aun no se publico ninguna categoría por lo tanto no puede mostrar enlaces en el blog".*

Tabla 62. Especificación de Caso de Uso: Gestionar Categoría- Enlaces

III.1.2.7.7.4. Especificación de Caso de Uso: Gestionar Categorías-Entradas

Nombre Caso de Uso:	**Gestionar Categorías-Entradas**
Identificador:	**BAB13**
Descripción: Este caso de uso permite administrar las categorías para publicar entradas. Solo el Administrador de Blog puede realizar este caso de uso.	
Precondiciones: **1.** El usuario registrado ingresa correctamente al administrador de su blog **2.** Elige la opción Gestionar Categorías del menubar	
Pos condiciones: El usuario realiza alguna de las siguientes acciones que se le permite cuando visualiza la lista de categorías de entradas.	
Flujo Básico de la Acción: **1.** El usuario registrado elige la opción Categorías-Entradas del submenú **2.** Inmediatamente se muestra en pantalla las categorías para entradas publicadas. El usuario puede seleccionar modificar categoría, pasar al punto 3; eliminar categoría, pasar al punto 4. **3.** El usuario modifica categoría-entrada **3.1.** El usuario selecciona la categoría que desea modificar **3.2.** Presiona la opción modificar de la categoría seleccionada **3.3.** Internamente se hace la selección de los datos de la categoría en base de datos **3.4.** En pantalla se muestra la categoría en la interfaz que permite modificar sus datos.	

3.5. El usuario modifica los datos de la categoría. **3.6.** Presiona Modificar. **3.7.** Si los datos modificados son validos y correctos se actualiza la categoría en base de datos, caso contrario no se realiza la transacción. **3.8.** Se direcciona a la lista de categorías-entradas y aparece el mensaje *"Categoría (titulo) modificada"* **4.** El usuario elimina categoría-entrada **4.1.** El usuario selecciona una categoría de la lista. **4.2.** Presiona su opción para Eliminar. **4.3.** Aparece un mensaje de confirmación ¿Desea eliminar la categoría? **4.4.** Si el usuario elige SI en el mensaje se realiza la eliminación **4.5.** Internamente en base de datos se eliminan los datos de la categoría y de todas las entradas relacionadas. **4.6.** Se actualiza la lista de categorías-entradas.
Flujo Alternativo de la Acción: **1.** El usuario registrado elige la opción Categorías-Entradas del submenú **2.** Internamente se selecciona las categorías para entradas publicadas. **3.** Si no se encuentra nada aparece el mensaje "*Aun no se publico ninguna categoría por lo tanto no puede mostrar entradas en el blog".*

Tabla 63. Especificación de Caso de Uso: Gestionar Categorías- Entradas

III.1.2.7.7.5. Especificación de Caso de Uso: Crear Encuesta

Nombre Caso de Uso:	**Crear Encuesta**
Identificador:	**BAB14**
Descripción: Este caso de uso solo puede ser realizado por el administrador de blog. Este caso de uso permite crear encuestas en el blog.	
Precondiciones: **1.** El usuario registrado ingresa correctamente al administrador de su blog **2.** Elige la opción Publicar del menubar	
Pos condiciones: Se actualiza el numero de encuestas publicadas en el blog	
Flujo Básico de la Acción: **1.** El usuario registrado elige la opción Encuesta del submenú Publicar **2.** Inmediatamente aparece en pantalla la interfaz para crear la encuesta **3.** El usuario introduce el nombre de la encuesta y la pregunta. **4.** Presiona siguiente y aparece la siguiente pantalla que permite registrar las respuestas. **5.** El usuario registra las respuestas de la encuesta creada, que pueden ser varias	

opciones. **6.** Una vez que se introduce todas las respuestas, el usuario presiona finalizar y todos estos datos se guardan en base de datos **7.** La pantalla se actualiza y s muestra la nueva encuesta creada.
Flujo Alternativo de la Acción: **1.** Si se ingresa datos erróneos, o se verifica datos faltantes en la validación saldrá mensaje de error y no se podrá guardar en base de datos hasta corregirlos.

Tabla 64. Especificación de Caso de Uso: Crear Encuesta

III.1.2.7.7.6. Especificación de Caso de Uso: Gestionar Encuesta

Nombre Caso de Uso:	**Gestionar Encuestas**
Identificador:	**BAB15**
Descripción: Este caso de uso permite administrar las encuestas publicadas. Solo el Administrador de Blog puede gestionar todas las encuestas creadas.	
Precondiciones: **1.** El usuario registrado ingresa correctamente al administrador de su blog **2.** Elige la opción Gestionar del menubar	
Pos condiciones: El usuario realiza alguna de las siguientes acciones que se le permite cuando visualiza la lista de encuestas.	
Flujo Básico de la Acción: **1.** El usuario registrado elige la opción Encuestas del submenú **2.** Inmediatamente se muestra en pantalla las encuestas publicadas. El usuario puede seleccionar habilitar o deshabilitar la publicación de la encuesta, pasar al punto 3; modificar encuesta, pasar al punto 4; eliminar encuesta, pasar al punto 5, respuestas pasar punto 6. **3.** El usuario selecciona habilitar o deshabilitar encuesta. **3.1.** El usuario selecciona una encuesta especifica **3.2.** Si la encuesta está habilitada tendrá la opción de Deshabilitar, si la encuesta esta deshabilitada tendrá la opción habilitar. **3.3.** Dependiendo de la opción que se muestra el usuario podrá habilitar o deshabilitar la encuesta. **3.4.** Internamente se cambiara el estado de la encuesta en base de datos **3.5.** En pantalla se actualizara el estado de la encuesta en la lista. **4.** El usuario modifica encuesta. **4.1.** El usuario selecciona la encuesta que desea modificar **4.2.** Presiona la opción modificar de la encuesta seleccionada **4.3.** Internamente se hace la selección de los datos de la encuesta en base de datos	

4.4. En pantalla se muestra la encuesta en la interfaz que permite modificar sus datos. **4.5.** Presiona Modificar Encuesta. **4.6.** Si los datos modificados son validos y correctos se actualiza la encuesta en base de datos, caso contrario no se realiza la transacción. **4.7.** Se direcciona a la lista de encuestas y aparece el mensaje *"Encuesta (titulo) modificada"* **5.** El usuario elimina encuesta **5.1.** El usuario selecciona una encuesta de la lista **5.2.** Presiona su opción para Eliminar **5.3.** Aparece un mensaje de confirmación ¿Desea eliminar la encuesta? **5.4.** Si el usuario elige SI en el mensaje se realiza la eliminación **5.5.** Internamente en base de datos se eliminan los datos de la encuesta. **5.6.** Se actualiza la lista de encuestas. **6.** El usuario selecciona la opción Ver Respuestas. **6.1.** Selecciona una encuesta a elegir **6.2.** Hace clic en la opción Respuestas de la encuesta seleccionada **6.3.** Internamente se selecciona todas opciones de respuesta que pertenecen a esa encuesta **6.4.** En pantalla se muestra la lista de las respuestas. **6.5.** El usuario puede modificarlas o eliminarlas
Flujo Alternativo de la Acción: **1.** El usuario registrado elige la opción Encuestas del submenú **2.** Internamente se selecciona las encuestas publicadas. **3.** Si no se encuentra nada aparece el mensaje *"Aun no se publico ninguna encuesta en el blog".*

Tabla 65. Especificación de Caso de Uso: Gestionar Encuestas

III.1.2.7.7.7. Especificación de Caso de Uso: Registrar Miembro

Nombre Caso de Uso:	**Registrar Miembro**
Identificador:	**BAB17**
Descripción: Este caso de uso solo puede ser realizado por el administrador de blog. Este caso de uso permite crear Usuarios de Blog.	
Precondiciones: **1.** El usuario registrado ingresa correctamente al administrador de su blog **2.** Elige la opción Gestionar Colegios del menubar	
Pos condiciones: El número de usuarios del blog puede incrementar o disminuir.	

Flujo Básico de la Acción: **1.** El usuario registrado elige la opción Registrar Miembro del submenú Afiliado-Usuario **2.** Inmediatamente aparece en pantalla la interfaz para crear un nuevo usuario **3.** El usuario primero realiza el registro de los datos personales y datos de afiliación del profesional. **4.** Seguidamente procede a registrar los datos de usuario para que pueda ingresar a la parte administrativa del blog. **5.** Presiona guardar y si todos los datos están correctos se guarda en base de datos. **6.** La pantalla se actualiza y se muestra en lista el nuevo usuario creado
Flujo Alternativo de la Acción: **2.** Si se ingresa datos erróneos, o se verifica datos faltantes en la validación saldrá mensaje de error y no se podrá guardar en base de datos hasta corregirlos.

Tabla 66. Especificación de Caso de Uso: Registrar Miembro

III.1.2.7.7.8. Especificación de Caso de Uso: Gestionar Miembros

Nombre Caso de Uso:	**Gestionar Miembros**
Identificador:	**BAB18**
Descripción: Este caso de uso permite gestionar a los Usuarios de Blog. Solo en administrador de blog realiza este caso de uso.	
Precondiciones: **1.** El usuario registrado ingresa correctamente a la parte administrativa de su blog **2.** Elige la opción Gestionar Colegio del menubar	
Pos condiciones: El usuario realiza alguna de las siguientes acciones que se le permite cuando visualiza la lista de Usuarios de Blog.	
Flujo Básico de la Acción: **1.** El usuario registrado elige la opción Gestionar Miembros del submenú **2.** Inmediatamente se muestra en pantalla la lista de Miembros o Afiliados registrados. El usuario puede seleccionar habilitar o deshabilitar usuario, pasar al punto 3; modificar usuarios, pasar al punto 4; eliminar usuario, pasar al punto 5. **3.** El administrador selecciona habilitar o deshabilitar usuario **3.1.** El usuario selecciona un usuario de la lista **3.2.** Si el usuario está habilitado tendrá la opción de Deshabilitar, si el usuario esta deshabilitado tendrá la opción Habilitar. **3.3.** Dependiendo de la opción que se muestra el usuario podrá habilitar o deshabilitar	

al usuario.

3.4. Internamente se cambiara el estado del usuario en base de datos

3.5. En pantalla se actualizara el estado del usuario en lista.

4. El administrador de blog modifica usuario

4.1. El administrador de blog selecciona el usuario que desea modificar

4.2. Presiona la opción modificar del usuario seleccionado

4.3. Internamente se hace la selección de los datos del usuario en base de datos

4.4. En pantalla se muestra el usuario en la interfaz que permite modificar sus datos.

4.5. El usuario modifica los datos del usuario

4.6. Presiona Modificar

4.7. Si los datos modificados son validos y correctos se actualiza en base de datos, caso contrario no se realiza la transacción.

4.8. Se direcciona a la lista de usuarios y aparece el mensaje *"Se modificaron los datos del Usuario: (nombre completo)"*

5. El administrador elimina usuario

5.1. El administrador selecciona un usuario de la lista

5.2. Presiona su opción para Eliminar

5.3. Aparece un mensaje de confirmación: "*Todos los datos del usuario se eliminaran, así también sus publicaciones ¿Eliminara al usuario?"*

5.4. Si el usuario elige SI en el mensaje se realiza la eliminación

5.5. Internamente en base de datos se eliminan los datos del usuario afiliado y todas las publicaciones que tenga en el blog.

5.6. Se actualiza el número de usuarios en lista.

Tabla 67. Especificación de Caso de Uso: Gestionar Miembros

III.1.2.7.7.9. Especificación de Caso de Uso: Configurar Blog

Nombre Caso de Uso:	**Configurar Blog**
Identificador:	**BAB19**

Descripción:

Este escenario permite configurar los datos y apariencia del blog. Este caso de uso es realizado por el administrador de blog.

Precondiciones:

1. El usuario registrado ingresa correctamente a la parte administrativa del blog.
2. Elige la opción Configurar Blog del menubar

Flujo Básico de la Acción:

1. El usuario puede elegir una de las siguientes opciones: Editar Datos de Colegio pasar punto 2, Editar datos Blog pasar punto 3, Cambiar Plantilla pasar punto 4.
2. Caso de uso BAB19-1 Cambiar Plantilla
3. Caso de uso BAB19-2 Cambiar Logo

Tabla 68. Especificación de Caso de Uso: Configurar Blog

III.1.2.7.7.10. Especificación de Caso de Uso: Cambiar Plantilla

Nombre Caso de Uso:	**Cambiar Plantilla**
Identificador:	**BAB19-1**
Descripción: Este caso de uso permite cambiar la plantilla del Blog, Este caso de uso es realizado por el administrador de blog.	
Precondiciones: **1.** El usuario registrado ingresa correctamente a la parte administrativa del blog **2.** Elige la opción Configurar Blog del menubar	
Pos condiciones: Cuando el usuario vea su blog después del cambio de plantilla notara la diferencia en la apariencia.	
Flujo Básico de la Acción: **1.** El usuario elige la opción Cambiar Plantilla del submenú **2.** La pagina se actualizara y mostrara todas las plantillas disponibles **3.** El usuario eligiera una de ellas **4.** Presiona Cambiar **5.** Internamente se cambia la relación del blog con la nueva plantilla en base de datos	
Flujo Alternativo de la Acción: S no se elige ninguna plantilla no se cambia nada en la apariencia del blog. .	

Tabla 69. Especificación de Caso de Uso: Cambiar Plantilla

III.1.2.7.7.11. Especificación de Caso de Uso: Cambiar Logo

Nombre Caso de Uso:	**Cambiar Logo**
Identificador:	**BAB19-2**
Descripción: Este caso de uso permite cambiar el logo del Blog, Este caso de uso es realizado por el administrador de blog.	
Precondiciones: **3.** El usuario registrado ingresa correctamente a la parte administrativa del blog **4.** Elige la opción Configurar Blog del menubar	
Pos condiciones: Cuando el usuario vea su blog después del cambio de logo, se actualizara el logo mostrado en el blog	
Flujo Básico de la Acción: **6.** El usuario elige la opción Cambiar Logo del submenú **7.** La pagina se actualizara y mostrara todas los logos subidos por el Admiistrador	

del Blog, **8.** El usuario elige uno de ellos, para ser mostrado en el blog. **9.** Presiona Cambiar **10.** Internamente se cambia el estado del logo Elegido, y del logo Actual, y se muestra en blog el logo elegido.
Flujo Alternativo de la Acción: Si el administrador del blog no hubiera subido ningún logo, entonces aparecerá la interfaz que permite subir logos. .

Tabla 70. Especificación de Caso de Uso: Cambiar Logo

III.1.2.7.7.12. Especificación de Caso de Uso: Editar Datos Colegio

Nombre Caso de Uso:	**Editar Datos de Colegio**
Identificador:	**BAB20**
Descripción: Este caso de uso permite editar los datos del Colegio, Este caso de uso es realizado por el administrador de blog.	
Precondiciones: **1.** El usuario registrado ingresa correctamente a la parte administrativa del blog **2.** Elige la opción Gestionar Colegio del menubar	
Pos condiciones: Se actualiza la configuración del Blog.	
Flujo Básico de la Acción: **1.** El usuario elige la opción Datos Colegio del submenú **2.** Aparecerá la pantalla con los datos del Colegio **3.** El usuario modifica los datos que requiera. **4.** Presiona el botón Actualizar. **5.** Se valida los datos corregidos por el usuario. **6.** Si los datos son correctos se actualiza los datos del Colegio en base de datos.	
Flujo Alternativo de la Acción: Si los datos son incorrectos se muestra mensajes de error en la validación.	

Tabla 71. Especificación de Caso de Uso: Editar Datos Colegio

III.1.2.7.7.13. Especificación de Caso de Uso: Gestionar Directorio

Nombre Caso de Uso:	**Gestionar Directorio**
Identificador:	**BAB21**
Descripción: Este caso de uso permite gestionar el directorio creado, Este caso de uso es realizado por el administrador de blog.	
Precondiciones: **1.** El usuario registrado ingresa correctamente a la parte administrativa del blog **2.** Elige la opción Gestionar Colegio del menubar	
Pos condiciones: Se actualiza la configuración del Blog.	
Flujo Básico de la Acción: **1.** El usuario elige la opción Directorio del submenú **2.** Aparecerá la pantalla con los cargos creados y asignados. **3.** El usuario administrador del Blog puede modificar los datos del cargo seleccionado o directamente eliminarlo **4.** La interfaz de permite también cambiar la forma de presentación del directorio y registrar Cargo.	
Flujo Alternativo de la Acción: **1.** Si el usuario no ha creado aun su directorio en el blog, aparecerá la interfaz para crear su directorio. **2.** Una vez creado el directorio se direccionara a la interfaz de gestión de directorio. **3.** El usuario administrador de blog ahora podrá registrar los cargos.	

Tabla 71. Especificación de Caso de Uso: Gestionar Directorio

III.1.2.7.7.14. Especificación de Caso de Uso: Gestionar Ingresos Externos

Nombre Caso de Uso:	**Gestionar Ingresos Externos**
Identificador:	**BAB22**
Descripción: Este caso de uso permite gestionar los ingresos externos, que son los ingresos percibidos por actividades realizadas, Este caso de uso es realizado por el administrador de blog.	
Precondiciones: **1.** El usuario registrado ingresa correctamente a la parte administrativa del blog **2.** Elige la opción Gestionar Colegio del menubar	
Pos condiciones: Se actualiza la configuración del Blog.	

Flujo Básico de la Acción: **1.** El usuario elige la opción Ingresos Externos del submenú. **2.** Aparecerá la interfaz con la lista de ingresos externos registrados. **3.** El usuario puede hacer cualquiera de las siguientes acciones. **4.** Registrar un nuevo ingreso externo. **5.** Elige un ingreso externo de la lista y puede modificar sus datos. **6.** Elige un ingreso externo de la lista y puede eliminarlo.
Flujo Alternativo de la Acción: **1.** Si el usuario no ha creado aun no ha registrado ningun ingreso externo, al seleccionar la opción Ingresos Externos. **2.** Se direccionara a la pantalla para registrar ingresos externos.

Tabla 72. Especificación de Caso de Uso: Gestionar Ingresos Externos

III.1.2.7.7.15. Especificación de Caso de Uso: Gestionar Ingresos Internos

Nombre Caso de Uso:	**Gestionar Ingresos Internos**
Identificador:	**BAB22**
Descripción: Este caso de uso permite gestionar los ingresos internos, que son los ingresos percibidos por concepto de cuotas o cancelaciones que realizan los afiliados o miembros, Este caso de uso es realizado por el administrador de blog.	
Precondiciones: **1.** El usuario registrado ingresa correctamente a la parte administrativa del blog **2.** Elige la opción Gestionar Colegio del menubar	
Pos condiciones: Se actualiza la configuración del Blog.	
Flujo Básico de la Acción: **1.** El usuario elige la opción Ingresos Internos del submenú. **2.** Aparecerá la interfaz con la lista de ingresos internos registrados. **3.** El usuario puede hacer cualquiera de las siguientes acciones. **4.** Registrar un nuevo ingreso interno. **5.** Elige un ingreso interno de la lista y puede modificar sus datos. **6.** Elige un ingreso interno de la lista y puede eliminarlo.	
Flujo Alternativo de la Acción: **1.** Si el usuario no ha creado aun no ha registrado ningún ingreso interno, al seleccionar la opción Ingresos Internos. **2.** Se direccionara a la pantalla para registrar ingresos internos.	

Tabla 73. Especificación de Caso de Uso: Gestionar Ingresos Internos

III.1.2.7.7.16. Especificación de Caso de Uso: Gestionar Egresos

Nombre Caso de Uso:	**Gestionar Egresos**
Identificador:	**BAB23**
Descripción: Este caso de uso permite gestionar los egresos, que son los gastos efectuados en el Colegio. Este caso de uso es realizado por el administrador de blog.	
Precondiciones: **3.** El usuario registrado ingresa correctamente a la parte administrativa del blog **4.** Elige la opción Gestionar Colegio del menubar	
Pos condiciones: Se actualiza la configuración del Blog.	
Flujo Básico de la Acción: **7.** El usuario elige la opción Egresos del submenú. **8.** Aparecerá la interfaz con la lista de egresos registrados. **9.** El usuario puede hacer cualquiera de las siguientes acciones. **10.** Registrar un nuevo egreso. **11.** Elige un egreso de la lista y puede modificar sus datos. **12.** Elige un egreso de la lista y puede eliminarlo.	
Flujo Alternativo de la Acción: **3.** Si el usuario no ha creado aun no ha registrado ningún ingreso interno, al seleccionar la opción Egresos. **4.** Se direccionara a la pantalla para registrar egresos.	

Tabla 74. Especificación de Caso de Uso: Gestionar Egresos

III.1.2.7.8. Especificaciones de Casos de Uso: Administrar Plataforma

III.1.2.7.8.1. Especificación de Caso de Uso: Crear Colegio

Nombre Caso de Uso:	**Crear Colegio**
Identificador:	**BAP1**
Descripción: Este caso de uso permite al Administrador de Plataforma Crear un Colegio.	
Precondiciones: **1.** El usuario ingresa a la plataforma **2.** Elige la opción Crear	

Pos condiciones: Existe una nuevo Colegio creado en base de datos
Flujo Básico de la Acción: **1.** El usuario selecciona la Opción Colegio del Menú. **2.** Aparece la interfaz para crear el Colegio **3.** El usuario ingresa los datos del Colegio. **4.** Presiona Crear **5.** Si todos los datos son correctos el Colegio es creado y guardado en base de datos. **6.** Se retorna a la lista de Colegios y sa muestra el mensaje *"Colegio (nombre) creado".*
Flujo Alternativo de la Acción: **1.** En la opción 5 si en la validación de los datos se encuentra errores sale el mensaje. **2.** En la opción 5 si el Colegio ya existe en base de datos saldrá el mensaje *"El Colegio ya fue creado por otra persona".*

Tabla 75. Especificación de Caso de Uso: Crear Colegio

III.1.2.7.8.2. Especificación de Caso de Uso: Gestionar Blogs

Nombre Caso de Uso:	**Gestionar Blogs**
Identificador:	**BAP3**
Descripción: Este escenario involucra todas las operaciones que realiza el administrador de plataforma con todos los blogs que van creando los diferentes Colegios, el administrador de plataforma podrá habilitar, deshabilitar, eliminar los blogs creados.	
Precondiciones: 1. El Administrador de Plataforma (AP) a ingresado correctamente al sistema 2. El Administrador de Plataforma(AP) selecciona la opción gestionar blogs	
Pos condiciones: Se visualiza la pagina correspondiente a la acción que desea realizar	
Flujo Básico de la Acción: **1.** Se visualiza en pantalla la lista de blogs que han sido registrados por los Colegios y por el mismo. **2.** El AP selecciona un blog y cualquiera de las opciones. Las opciones que puede elegir son: habilitar pasar punto 3, deshabilitar pasar punto 4, eliminar pasar punto 5, editar perfil blog pasar punto 6. **3.** El usuario elige la opción habilitar del blog seleccionado **3.1.** El blog aparece en lista con estado deshabilitado y con la opción habilitar	

3.2. El AP hace click en la opción habilitar del blog en lista **3.3.** Internamente se cambia el estado del blog en base de datos **3.4.** Inmediatamente se actualiza la lista mostrando al blog como habilitado **4.** El usuario elige la opción deshabilitar del blog seleccionado. **4.1.** El blog aparece en lista con estado habilitado y con la opción deshabilitar **4.2.** El AP hace click en la opción deshabilitar del blog en lista **4.3.** Internamente se cambia el estado del blog en base de datos **4.4.** Inmediatamente se actualiza la lista mostrando al blog como deshabilitado **5.** El usuario elige la opción eliminar del blog seleccionado **5.1.** Aparece un mensaje para confirmar la eliminación del blog **5.2.** El AP presiona SI en el mensaje **5.3.** Internamente se eliminan todas las entradas, paginas, archivos, imágenes, enlaces, usuarios de ese blog **5.4.** Se elimina el blog de la base de datos **6.** El usuario elige la opción editar del blog seleccionado **6.1.** Aparece la interfaz donde el AP puede modificar los datos del blog
Flujo Alternativo de la Acción: 1. Se visualiza en pantalla la lista de blogs que han sido registrados por los Colegios. 2. El AP no selecciona ninguna opción que se muestra en el submenú

Tabla 76. Especificación de Caso de Uso: Gestionar Blogs

III.1.2.7.8.3. Especificación de Caso de Uso: Gestionar Colegios

Nombre Caso de Uso:	**Gestionar Colegios**
Identificador:	**BAP4**
Descripción: Este escenario involucra todas las operaciones que realiza el administrador de plataforma con todos los Colegios que se registraron, el administrador de plataforma podrá habilitar, deshabilitar, eliminar los colegios creados.	
Precondiciones: 1. El Administrador de Plataforma (AP) a ingresado correctamente al sistema 2. El Administrador de Plataforma(AP) selecciona la opción gestionar Colegios	
Pos condiciones: Se visualiza la pagina correspondiente a la acción que desea realizar	
Flujo Básico de la Acción: **1.** Se visualiza en pantalla la lista de blogs que han sido registrados. **2.** El AP selecciona un Colegio y cualquiera de las opciones. Las opciones que puede elegir son: habilitar pasar punto 3, deshabilitar pasar punto 4, eliminar	

pasar punto 5, editar perfil Colegio pasar punto 6.

3. El usuario elige la opción habilitar del Colegio seleccionado
3.1. El blog aparece en lista con estado deshabilitado y con la opción habilitar
3.2. El AP hace click en la opción habilitar del Colegio en lista
3.3. Internamente se cambia el estado del Colegio en base de datos
3.4. Inmediatamente se actualiza la lista mostrando al blog como habilitado
4. El usuario elige la opción deshabilitar del Colegio seleccionado.
4.1. El Colegio aparece en lista con estado habilitado y con la opción deshabilitar
4.2. El AP hace click en la opción deshabilitar del Colegio en lista
4.3. Internamente se cambia el estado del Colegio en base de datos
4.4. Inmediatamente se actualiza la lista mostrando al Colegio como deshabilitado
5. El usuario elige la opción eliminar del blog seleccionado
5.1. Aparece un mensaje para confirmar la eliminación del Colegio.
5.2. El AP presiona SI en el mensaje.
5.3. Si el Colegio no tiene ningún Colegio relacionado se elimina de base de datos.
6. El usuario elige la opción editar del Colegio seleccionado
6.1. Aparece la interfaz donde el AP puede modificar los datos del Colegio.

Tabla 77. Especificación de Caso de Uso: Gestionar Colegios

III.1.2.7.8.4. Especificación de Caso de Uso: Exportar Plantilla

Nombre Caso de Uso:	**Exportar Plantilla**
Identificador:	**BAP5**

Descripción:

Este caso de uso permite exportar Plantilla al Administrador de Plataforma desde su panel de administración.

Precondiciones:

1. El usuario ingresa a la plataforma
2. Elige la opción Plantillas

Pos condiciones:

Existe una nueva Plantilla en base de datos

Flujo Básico de la Acción:

1. El usuario selecciona la Opción Exportar del Menú.
2. Aparece la interfaz para exportar la Plantilla.
3. El usuario da un nombre a la plantilla que quiere exportar
4. El usuario sube el archivo CSS de la plantilla.
5. El usuario sube el archivo HTML de la plantilla.
6. El usuario sube la imagen de muestra del la plantilla.
7. El usuario exporta la plantilla y se guarda en base de datos de la plataforma.
8. Una vez registrada la plantilla se dirección a la interfaz que muestra la lista de

plantillas exportadas.
Flujo Alternativo de la Acción: 1. En el punto 8. El usuario Administrador de Plataforma puede adicionar imágenes a la plantilla si son necesarias para su presentación.

Tabla 78. Especificación de Caso de Uso: Exportar plantilla

III.1.2.7.8.5. Especificación de Caso de Uso: Gestionar Plantillas

Nombre Caso de Uso:	**Gestionar Plantillas**
Identificador:	**BAP6**
Descripción: Las plantillas que se necesiten para que los Colegios puedan elegir al crear su blog serán gestionados el Administrador de Blogs, la gestión comprende acciones como: Crear, editar y eliminar plantilla.	
Precondiciones: **1.** El Administrador de Plataforma(AP) selecciona la opción gestionar blogs **2.** Selecciona opción Plantillas	
Pos condiciones: Se visualiza la pagina correspondiente a la acción seleccionada	
Flujo Básico de la Acción: **1.** Se visualiza en pantalla la lista de Plantillas que han sido registrados. **2.** El AP selecciona un Plantilla y cualquiera de las opciones. Las opciones que puede elegir son: habilitar pasar punto 3, deshabilitar pasar punto 4, eliminar pasar punto 5, editar Plantilla pasar punto 6. **3.** El usuario elige la opción habilitar del Plantilla seleccionada **3.1.** La Plantilla aparece en lista con estado deshabilitado y con la opción habilitar **3.2.** El AP hace click en la opción habilitar de la Plantilla en lista **3.3.** Internamente se cambia el estado de la Plantilla en base de datos **3.4.** Inmediatamente se actualiza la lista mostrando a la plantilla como habilitada **4.** El usuario elige la opción deshabilitar de la Plantilla seleccionada. **4.1.** La Plantilla aparece en lista con estado habilitada y con la opción deshabilitar **4.2.** El AP hace click en la opción deshabilitar de la plantilla en lista **4.3.** Si la plantilla está siendo usada por algún blog se muestra el siguiente mensaje "*No puede deshabilitar esta plantilla porque está siendo utilizada*" **4.4.** Internamente se cambia el estado de la Plantilla en base de datos **4.5.** Inmediatamente se actualiza la lista mostrando la Plantilla como deshabilitada **5.** El usuario elige la opción eliminar de la Plantilla seleccionada **5.1.** Aparece un mensaje para confirmar la eliminación de la Plantilla. **5.2.** El AP presiona SI en el mensaje. **5.3.** Si la plantilla esta deshabilitada se podrá eliminar de base de datos.	

6. El usuario elige la opción editar de la Plantilla seleccionada **6.1.** Aparece la interfaz donde el AP puede editar el código de la plantilla

Tabla 79. Especificación de Caso de Uso: Gestionar Plantillas

III.1.2.7.8.6. Especificación de Caso de Uso: Crear Rol

Nombre Caso de Uso:	**Crear Rol**
Identificador:	**BAP7**
Descripción: Este caso de uso permite crear Rol al Administrador de Plataforma desde su panel de administración.	
Precondiciones: **1.** El usuario ingresa a la plataforma **2.** Elige la opción Crear	
Pos condiciones: Existe un nuevo Rol creada en base de datos	
Flujo Básico de la Acción: **1.** El usuario selecciona la Opción Rol del Menú. **2.** Aparece la interfaz para crear el Rol. **3.** El ingresa los datos del Rol como nombre, permisos, etc. **4.** Una vez editado y registrado los datos del Rol, presiona Crear **5.** Si todos los datos son correctos el Rol es creado y guardado en base de datos. **6.** Se retorna a la lista de Roles y se muestra el mensaje "Rol *(nombre) creado".*	
Flujo Alternativo de la Acción: **1.** En la opción 5 si en la validación de los datos se encuentra errores sale el mensaje. **2.** En la opción 5 si existe otro Rol con el mismo nombre registrado en base de datos saldrá el mensaje *"El* Rol *ya fue creado".*	

Tabla 80. Especificación de Caso de Uso: Crear Rol

III.1.2.7.8.7. Especificación de Caso de Uso: Gestionar Roles

Nombre Caso de Uso:	**Gestionar Roles**
Identificador:	**BAP8**

Descripción:

Este escenario involucra todas las operaciones que realiza el administrador de plataforma con todos los Roles que se registraron, el administrador de plataforma podrá habilitar, deshabilitar, eliminar los Roles creados.

Precondiciones:

1. El Administrador de Plataforma (AP) a ingresado correctamente al sistema
2. El Administrador de Plataforma(AP) selecciona la opción gestionar Roles

Pos condiciones:

Se visualiza la pagina correspondiente a la acción que desea realizar

Flujo Básico de la Acción:

1. Se visualiza en pantalla la lista de Roles que han sido registrados.
2. El AP selecciona un Rol y cualquiera de las opciones. Las opciones que puede elegir son: habilitar pasar punto 3, deshabilitar pasar punto 4, eliminar pasar punto 5, editar perfil Rol pasar punto 6.
3. El usuario elige la opción habilitar del Rol seleccionado

3.1 El Rol aparece en lista con estado deshabilitado y con la opción habilitar
3.2 El AP hace click en la opción habilitar del Rol en lista
3.3 Internamente se cambia el estado del Rol en base de datos
3.4 Inmediatamente se actualiza la lista mostrando al Rol como habilitado

4. El usuario elige la opción deshabilitar del Rol seleccionado.

4.1 El Rol aparece en lista con estado habilitado y con la opción deshabilitar
4.2 El AP hace click en la opción deshabilitar del Rol en lista
4.3 Internamente se cambia el estado del Rol en base de datos
4.4 Inmediatamente se actualiza la lista mostrando al Rol como deshabilitado

5. El usuario elige la opción eliminar del Rol seleccionado

5.1 Aparece un mensaje para confirmar la eliminación del Rol.
5.2 El AP presiona SI en el mensaje.
5.3 Si el Rol no tiene ningún Rol relacionado se elimina de base de datos.

6. El usuario elige la opción editar del Rol seleccionado

6.1 Aparece la interfaz donde el AP puede modificar los datos del perfil del Rol.

Tabla 81. Especificación de Caso de Uso: Gestionar Roles

III.1.2.7.8.8. Especificación de Caso de Uso: Gestionar Usuarios

Nombre Caso de Uso:	**Gestionar Usuarios**
Identificador:	**BAB9**

Descripción:

Este caso de uso permite gestionar a los Usuarios de todos los Blogs creados. Solo en administrador de plataforma realiza este caso de uso.

Precondiciones:

1. El usuario registrado ingresa correctamente a la parte administrativa de Plataforma
2. Elige la opción Gestionar del menubar

Pos condiciones:

El usuario realiza alguna de las siguientes acciones que se le permite cuando visualiza la lista de Usuarios.

Flujo Básico de la Acción:

1. El usuario registrado elige la opción Gestionar del submenú
2. Inmediatamente se muestra en pantalla la lista de Usuarios. El usuario puede seleccionar habilitar o deshabilitar usuario, pasar al punto 3; modificar usuarios, pasar al punto 4; eliminar usuario, pasar al punto 5.
3. El administrador selecciona habilitar o deshabilitar usuario
3.1 El usuario selecciona un usuario de la lista
3.2 Si el usuario está habilitado tendrá la opción de Deshabilitar, si el usuario esta deshabilitado tendrá la opción Habilitar.
3.3 Dependiendo de la opción que se muestra el usuario podrá habilitar o deshabilitar al usuario.
3.4 Internamente se cambiara el estado del usuario en base de datos
3.5 En pantalla se actualizara el estado del usuario en lista.
4. El administrador de plataforma modifica usuario
4.1 El administrador de blog selecciona el usuario que desea modificar
4.2 Presiona la opción modificar del usuario seleccionado
4.3 Internamente se hace la selección de los datos del usuario en base de datos
4.4 En pantalla se muestra el usuario en la interfaz que permite modificar sus datos.
5. El administrador AP elimina usuario
10.1 El administrador selecciona un usuario de la lista
5.7. Presiona su opción para Eliminar
5.8. Aparece un mensaje de confirmación: "*Todos los datos del usuario se eliminaran, así también sus publicaciones ¿Eliminara al usuario?"*
5.9. Si el usuario elige SI en el mensaje se realiza la eliminación
5.10. Internamente en base de datos se eliminan los datos del usuario afiliado y todas las publicaciones que tenga en el blog.
Se actualiza el número de usuarios en lista.

Tabla 82. Especificación de Caso de Uso: Gestionar Usuarios

III.1.2.8. Descripción de Actores

III.1.2.8.1. Descripción del Actor : Administrador de Blogs o de Plataforma

Nombre de Actor	Administrador de Blogs o de Plataforma
Definición	Es la persona encargada de la administración de la plataforma, tendrá todos los permisos, y completo control en la plataforma. Es un usuario registrado .
Notas	➢ Esta persona podrá administrar todas las publicaciones hechas sobre entradas, paginas, enlaces en todos los blogs creados. ➢ Es la persona encargada de crear nuevos roles, nuevos usuarios para acceder a este modulo administrador otorgándole permisos. ➢ Esta persona gestiona todos los blogs registrados, y también puede ver la información proveniente de ellos,

Tabla 83. Descripción del Actor: Administrador de Blogs o de Plataforma

III.1.2.8.2. Descripción del Actor : Administrador de Blog

Nombre de Actor	Administrador de Blog
Definición	Es la persona encargada de administrar el blog perteneciente a un Colegio, tiene completo acceso a todas las opciones para administrar el blog. Es un usuario registrado.
Notas	➢ Tiene acceso a todas las publicaciones efectuadas por el mismo y por los Afiliados publicadores. ➢ Es la persona encargada de registrar a los afiliados del Colegio. ➢ Es la persona encargada de crear nuevos usuarios para acceder al blog. ➢ Es la persona encargada del mantenimiento del blog

Tabla 84. Descripción del Actor: Administrador de Blog

III.1.2.8.3. Descripción del Actor : Afiliado Registrado

Nombre de Actor	Afiliado Registrado
Definición	El afiliado puede corresponder a cualquier Colegio que cuente con su blog, se convierte en publicador cuando el Administrador de Blog le otorga una cuenta de usuario. Es un usuario registrado.
Notas	➢ Puede publicar contenidos o enlaces en cualquier categoría que exista en el blog. ➢ Puede administrar sus propias publicaciones, editarlos o eliminarlos. ➢ Una vez que haya publicado podrá ver su publicación en la página con la opción Ver Blog.

Tabla 85. Descripción de Actor: Afiliado Registrado

III.1.2.8.4. Descripción del Actor : Usuario no registrado

Nombre del Actor	Usuario no registrado
Definición	➢ El Usuario no registrado podrá ser cualquier usuario que ingrese a la Web y vea el blog publicado. ➢ Actor que tiene la posibilidad de comentar los contenidos publicados por los afiliados y administrador. ➢ También simboliza al usuario que visita la pagina Web de la plataforma y realiza las acciones que se le permite.
Notas	

Tabla 86. Descripción de Actor: Usuario no registrado

III.1.2.9. Requerimientos Técnicos

Un requerimiento técnico según la metodología pertenece a los aspectos técnicos que el sistema debe cumplir, como ser aspectos relacionados con la ejecución, seguridad y disponibilidad. Estos tipos de requerimientos son también llamados requerimientos de calidad de servicio (siglas en ingles, QoS). La documentación de un requerimiento técnico en este caso incluye una descripción, un ejemplo, un recurso y la referencia a requerimientos técnicos relacionados.

III.1.2.9.1. Requerimientos de Ejecución

Nombre:	Tiempo de búsqueda
Identificador:	RTE1
Descripción:	El tiempo de búsqueda de cualquier entidad en la base de datos debe ocurrir en menos de 3 segundos de tiempo.
Ejemplo:	Un usuario que visita la pagina Web necesita personal para su empresa y busca un profesional por especialidad.

Tabla 87. Requerimiento de Ejecución: Tiempo de Búsqueda

Nombre:	Subdominios ilimitados
Identificador:	RTE1
Descripción:	Bajo el dominio de la plataforma debe poder albergarse ilimitados subdominios porque no se sabe la cantidad de blogs que se puedan crear.
Ejemplo:	Se está registrando el blog número 3.695.410.342 y su URL debe ser única en la WWW.

Tabla 88. Requerimiento de Ejecución: Subdominios ilimitados

Nombre:	Mínimamente Hosting VPS
Identificador:	RTE2
Descripción:	La plataforma debe estar alojada mínimamente en un hosting VPS debido de que sus características son apropiadas para el proyecto, en caso de que se necesite de más recursos y una administración completa se puede adoptar un alojamiento Cloud. **Características de Hosting VPS (Servidor Privado Virtual)** ➢ Recursos del servidor compartidos por todos los VPS alojados en el mismo. Se garantizan recursos dedicados. ➢ Acceso como administrador.

	➢ Panel de control de Hosting opcional. ➢ Panel de control de la virtualización para instalar software y gestionar la configuración básica del servidor. ➢ Instalación de librerías, configuración específica de programas de servidor, tareas programadas, etc. Total libertad de instalación, a cargo del cliente (dentro de los recursos del VPS). ➢ Excelente escalabilidad. Ampliable, según las necesidades de su proyecto, de forma casi inmediata. ➢ El punto de entrada más asequible, técnica y económicamente, para iniciarse en la administración de servidores, o para disfrutar de un entorno dedicado.
Ejemplo:	Es necesario crear nuevas funcionalidades en la plataforma para los blogs

Tabla 89. Requerimiento de Ejecución: Mínimamente Hosting VPS

III.1.2.9.2. Requerimiento de Seguridad

Nombre:	La base de datos de la plataforma debe estar protegida
Identificador:	RTS1
Descripción:	La base de datos de la plataforma debe estar protegida de intrusos que quieran corromper sus datos lo que produciría una pérdida de confianza en la plataforma.
Ejemplo:	Un atacante podría conectarse remotamente a la base de datos, si es una base de datos de SQL Server esto lo podría hacer a través del puerto TCP 1433 o UDP 1434

Tabla 90. Requerimiento de Seguridad: La base de datos de la plataforma debe estar protegida

Nombre:	Desastres naturales
Identificador:	RTS2
Descripción:	El entorno físico desde donde se administre la plataforma debe estar protegido de los desastres naturales que podrían ocasionar perdidas de activos. Para estos casos es necesario tener un plan de contingencia.
Ejemplo:	Desastres naturales: Huracanes, terremotos, maremotos, etc

Tabla 91. Requerimiento de Seguridad: Desastres naturales

III.1.2.10. Modelo de Interfaces de Usuario

Describe las interfaces de usuario de su sistema. Pueden ser escritas en papel o en pizarra de pantallas son suficientes para obtener el punto para conocer qué es lo que se desea construir.

Las interfaces de Usuario son útiles durante las Iniciación y quizás la Elaboración, pero durante la Construcción se debe escribir el código actual de trabajo y no el código del prototipo.

III.1.2.10.1. Modelo de Interfaz Administrador de Blog

Figura 33. Modelo de Interfaz Administrador de Blog

III.1.2.10.2. Modelo de Interfaz del Blog creado

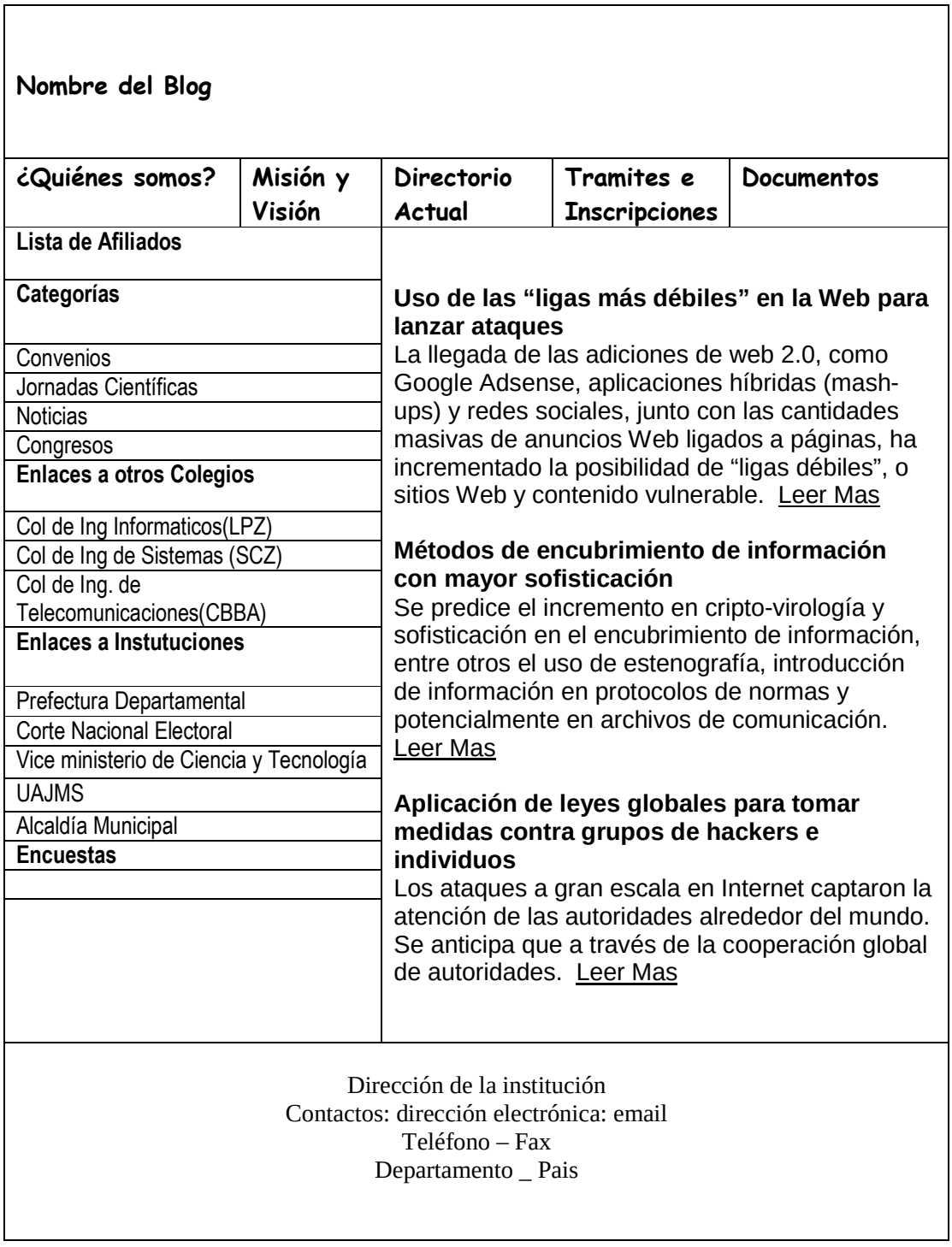

Figura 34. Modelo de Interfaz de Blog creado

III.1.2.11. Glosario del proyecto

Blog: Son páginas en el Internet muy fáciles de crear y actualizar. No se necesita ser un ingeniero de sistemas o saber mucho sobre el diseño o programación de páginas web. Solo se requiere un poco de practica y dedicación para poder contar sus experiencias y tener un espacio propio en el Internet. Estas páginas permiten que cualquier ciudadano con acceso al Internet tenga la misma oportunidad y derecho de escribir y publicar sobre sus ideas y experiencias, y además poder interactuar con otros que lean y comentan.

Weblog: Un weblog, también conocido como blog o bitácora, es un *sitio web* periódicamente actualizado que recopila cronológicamente textos y/o Entradas de uno o varios autores, donde el más reciente aparece primero, con un uso o temática en particular, siempre conservando el autor la libertad de dejar publicado lo que crea pertinente.
Los weblogs usualmente están escritos con un estilo personal e informal. Existen variadas herramientas de mantenimiento de blogs que permiten, muchas de ellas gratuitamente y sin necesidad de elevados conocimientos técnicos, administrar todo el weblog, coordinar, borrar o reescribir los Entradas, moderar los comentarios de los lectores, etc., de una forma casi tan sencilla como administrar el correo electrónico.

Blogósfera: Término usado para referirse a la totalidad de weblogs, traducción del término en inglés *blogosphere*. En ocasiones el concepto blogosfera se utiliza para referirse al conjunto de todos los blogs, en otras se utiliza para referirse a agrupaciones parciales de blog, como por ejemplo la blogosfera hispana, la blogosfera política, etc. La blogosfera en tanto que espacio de comunicación compartida, es resultado de la interconexión de los blogs a través de mecanismos como los *hiperenlaces*, los comentarios, etc.

Entrada: Son publicaciones o posts que consisten en texto, imágenes y también puede ser que el publicador requiera subir un documento. Estas publicaciones cuentan con una estructura definida con la cual se publicara en el blog y se relaciona con una categoría.

Página: Son publicaciones únicas, es decir solamente habrá una publicación referente a ese tema al igual que los contenidos consisten en texto, imágenes y también puede ser que el publicador requiera subir un documento. Estas publicaciones también cuentan con una estructura definida con la cual se publicara en el blog.

Enlace: Los enlaces son hipervínculos a otras páginas externas al blog pudiendo ser estas enlaces a otros blogs o sitios Web. Se relacionan con categorías de enlace.

Categoría: Una categoría agrupa una serie de publicaciones sobre un mismo tema, sirve para guardar y organizar los contenidos según la categoría relacionada.

Plantilla: Las plantillas o templates son archivos con código fuente en HTML y CSS que son utilizados para el diseño de los blogs.

Banner: Son animaciones realizadas en Flash o Potoshop para encabezar el blog, creadas y gestionadas por el administrador de la plataforma.

III.1.3. Modelo de Diseño

El objetivo del modelado de un sistema es capturar las partes esenciales del sistema. Para facilitar este modelado, se realiza una abstracción y se plasma en una notación gráfica. Esto se conoce como modelado visual.

El modelado visual permite manejar la complejidad de los sistemas a analizar o diseñar. De la misma forma que para construir una choza no hace falta un modelo, cuando se intenta construir un sistema complejo como un rascacielos, es necesario abstraer la complejidad en modelos que el ser humano pueda entender. UML sirve para el modelado completo de sistemas complejos, tanto en el diseño de los sistemas software como para la arquitectura hardware donde se ejecuten.

Según la metodología que se está utilizando para el desarrollo del proyecto, el modelo de diseño consta de una variedad de productos de trabajo, incluye potencialmente un modelo de despliegue, un modelo de objetos, un modelo de datos físico(PDM), un modelo de seguridad de amenazas, un documento de resumen del sistema y un modelo de interfaces de usuario. No es necesario crear todos los modelos que se nombro anteriormente, sólo las que sean adecuadas para representar el sistema.

El presente modelo de diseño consta de los siguientes productos:

- **Modelo de Objetos** Se mantendrá este modelo para mostrar por medio de unos cuantos diagramas la interacción con el sistema por medio de diagramas de secuencia y para ver la estructura estática por medio del diagrama de clases y diagramas de componentes.

- **Modelo de datos Físico** (PDM) Es obligatorio mantener este modelo porque muestra el esquema físico de la base de da tos

- **Modelo de Despliegue** Se mantendrá este modelo que describe cómo se va a organizar los aspectos de hardware y software del sistema.

III.1.3.1. Modelo de Objetos

III.1.3.1.1. Diagramas de Secuencia

III.1.3.1.1.1. Diagrama de Secuencia para Caso de Uso: FAB1 Listar Afiliados registrados

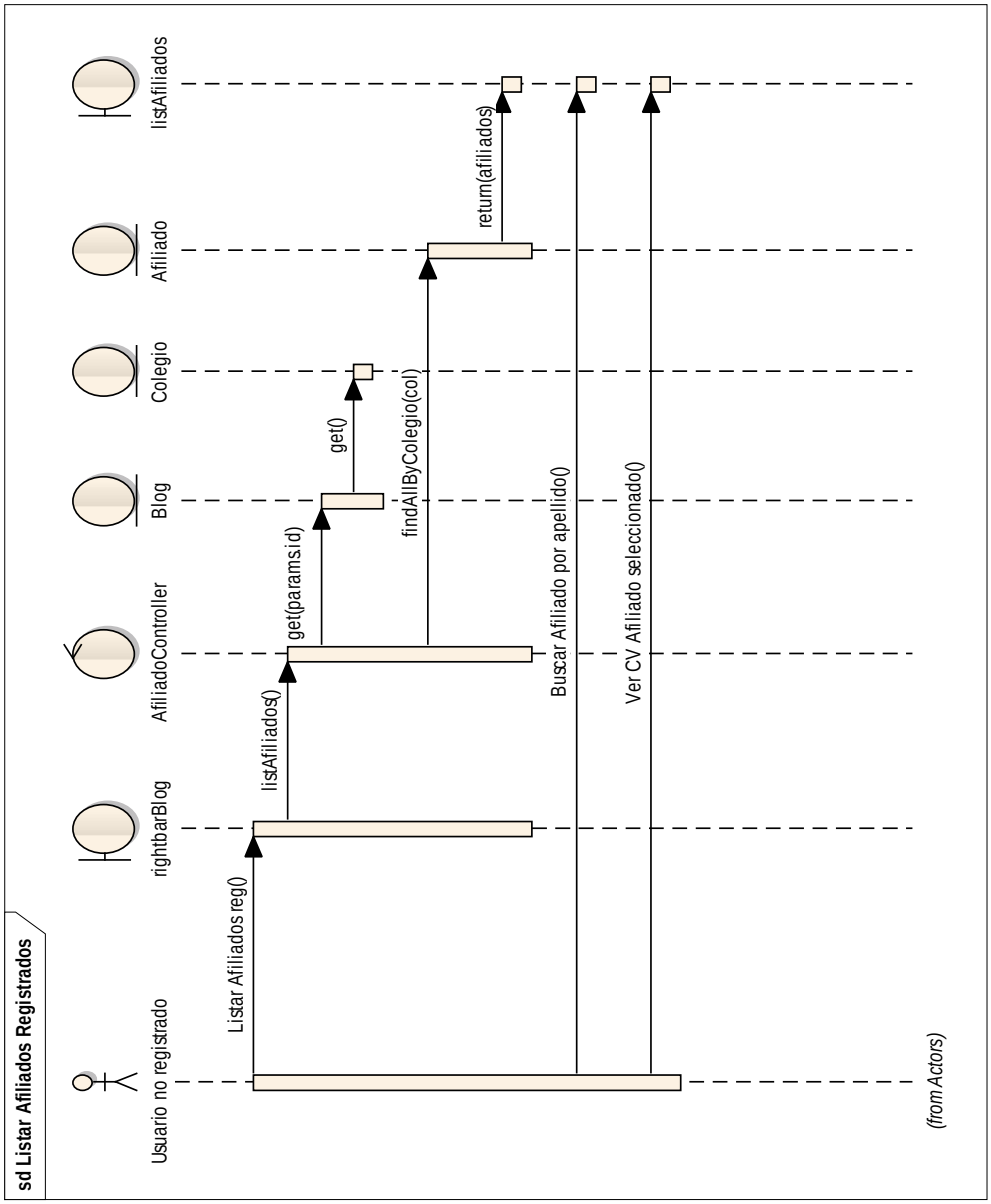

Figura 35. Diagrama de Secuencia: Listar Afiliados registrados

III.1.3.1.1.2. Diagrama de Secuencia para Caso de Uso: FAB1-1 Ver CV de Afiliado seleccionado

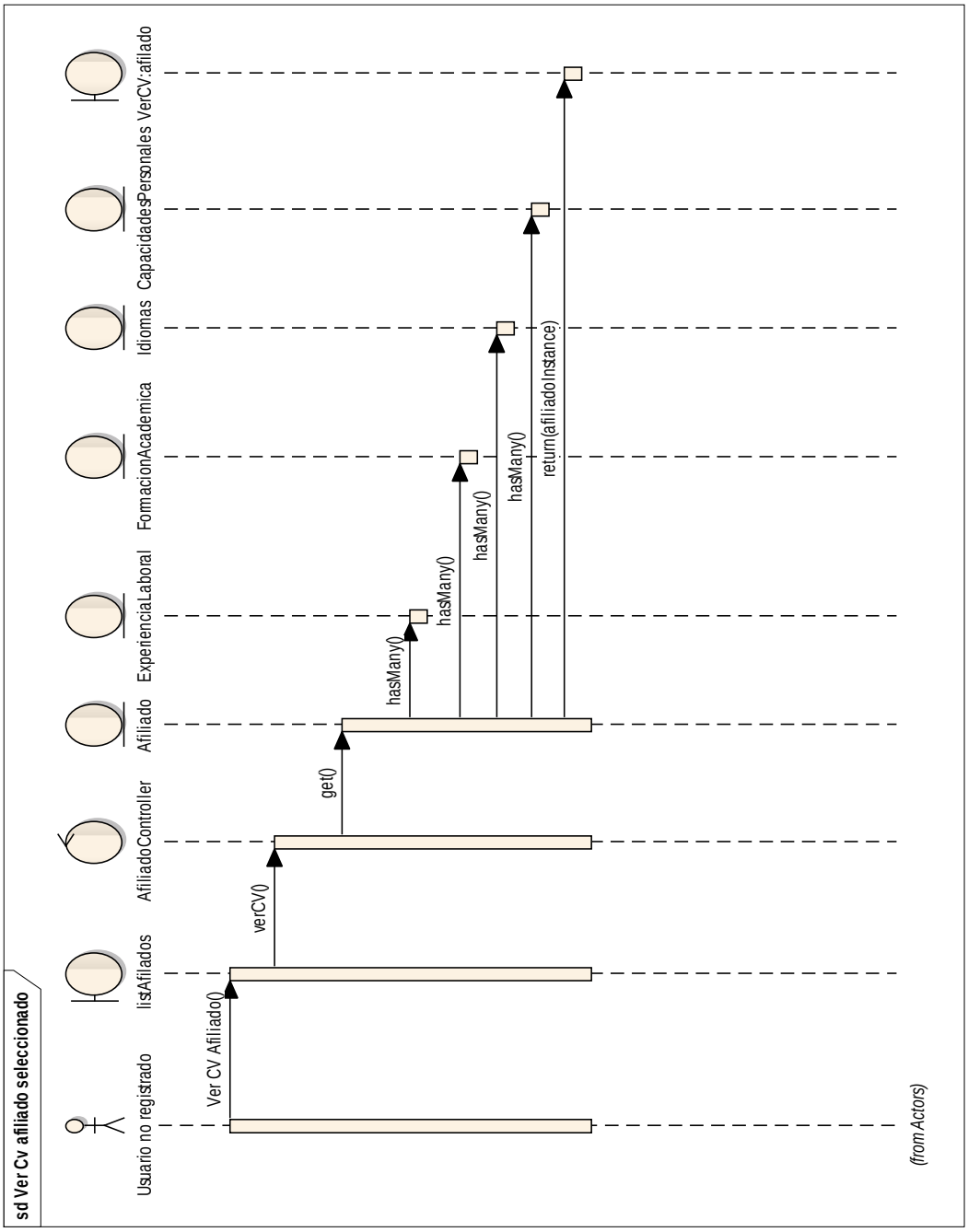

Figura 35. Diagrama de Secuencia: Ver CV de Afiliado seleccionado

III.1.3.1.1.3. Diagrama de Secuencia para Caso de Uso: FAB6 Votar Encuesta seleccionada

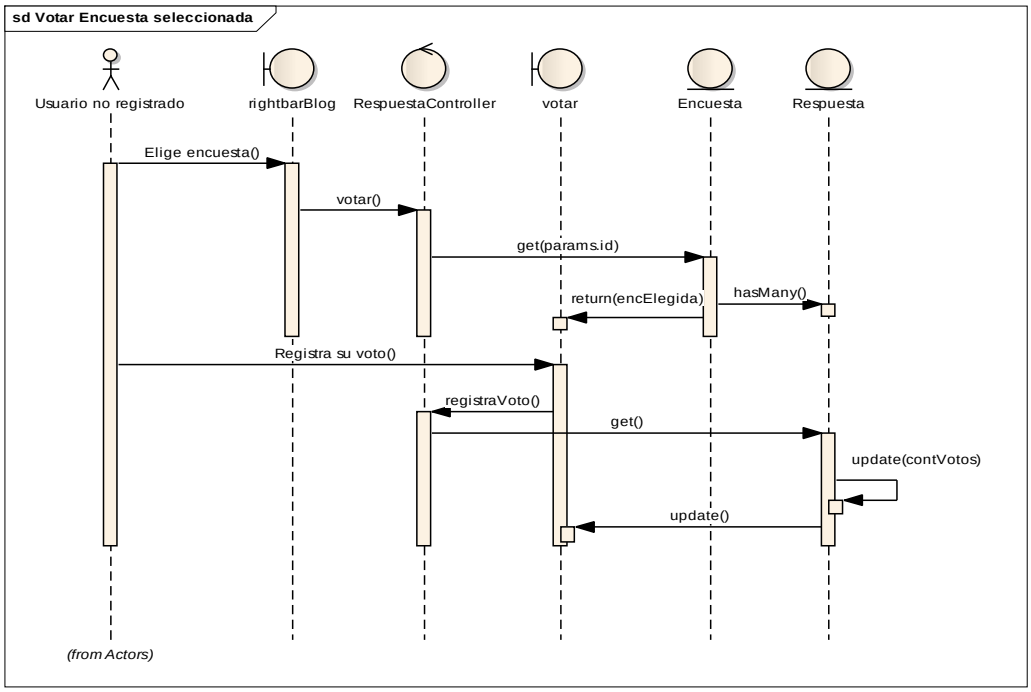

Figura 36. Diagrama de Secuencia: Votar Encuesta seleccionada

III.1.3.1.1.4. Diagrama de Secuencia para Caso de Uso: FAB11 Leer Pagina seleccionada

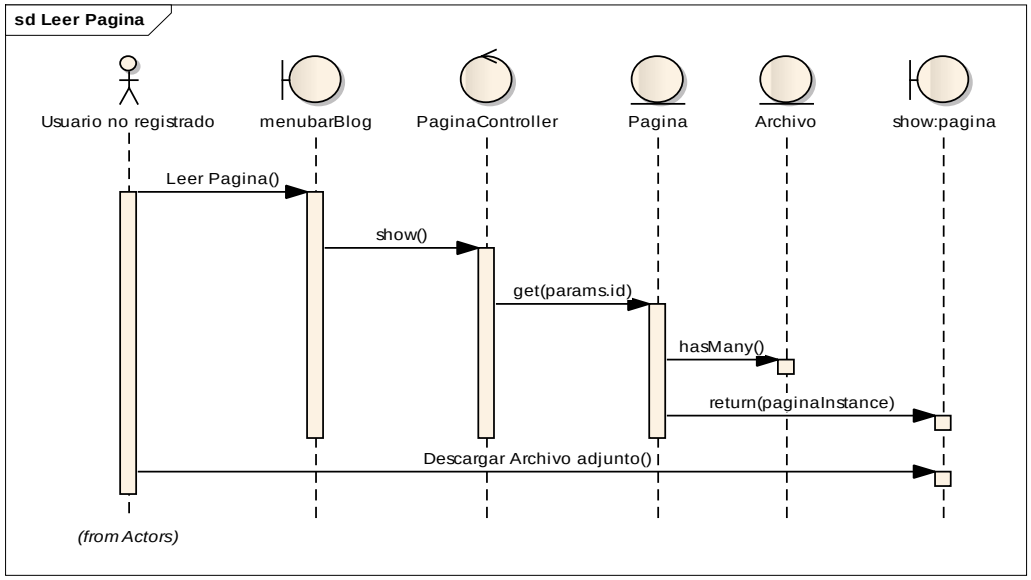

Figura 37. Diagrama de Secuencia: Leer Pagina seleccionada

III.1.3.1.1.5. Diagrama de Secuencia para Caso de Uso: FAB9 Comentar Entrada

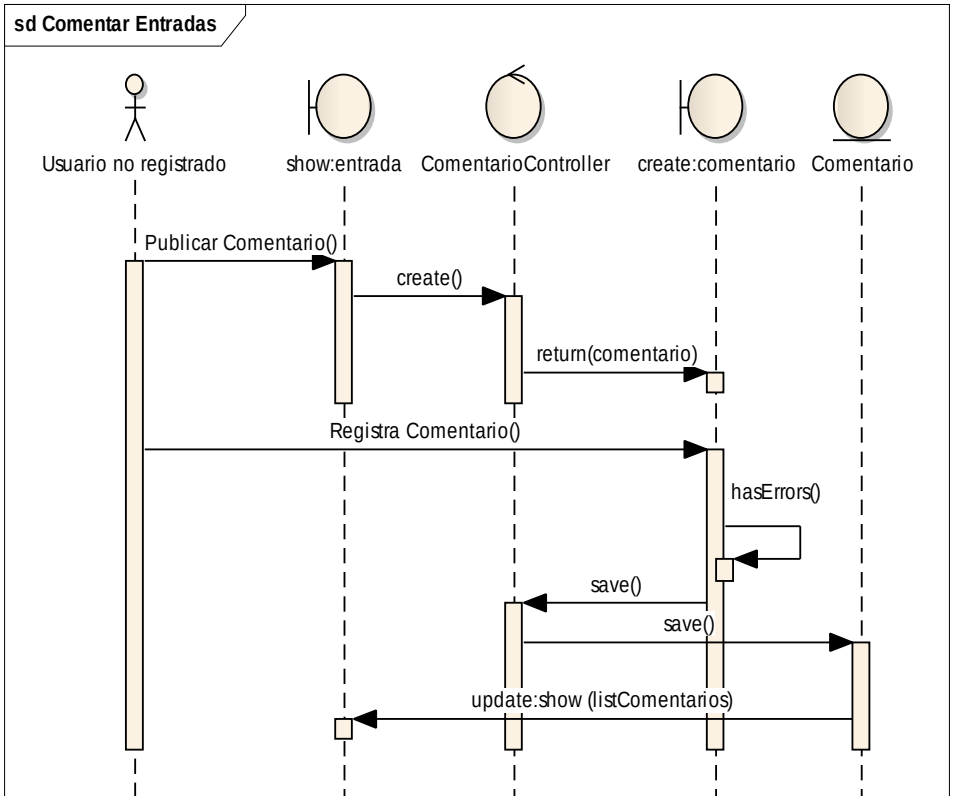

Figura 38. Diagrama de Secuencia: Comentar Entrada

III.1.3.1.1.6. Diagrama de Secuencia para Caso de Uso: BAB1 Crear Entrada

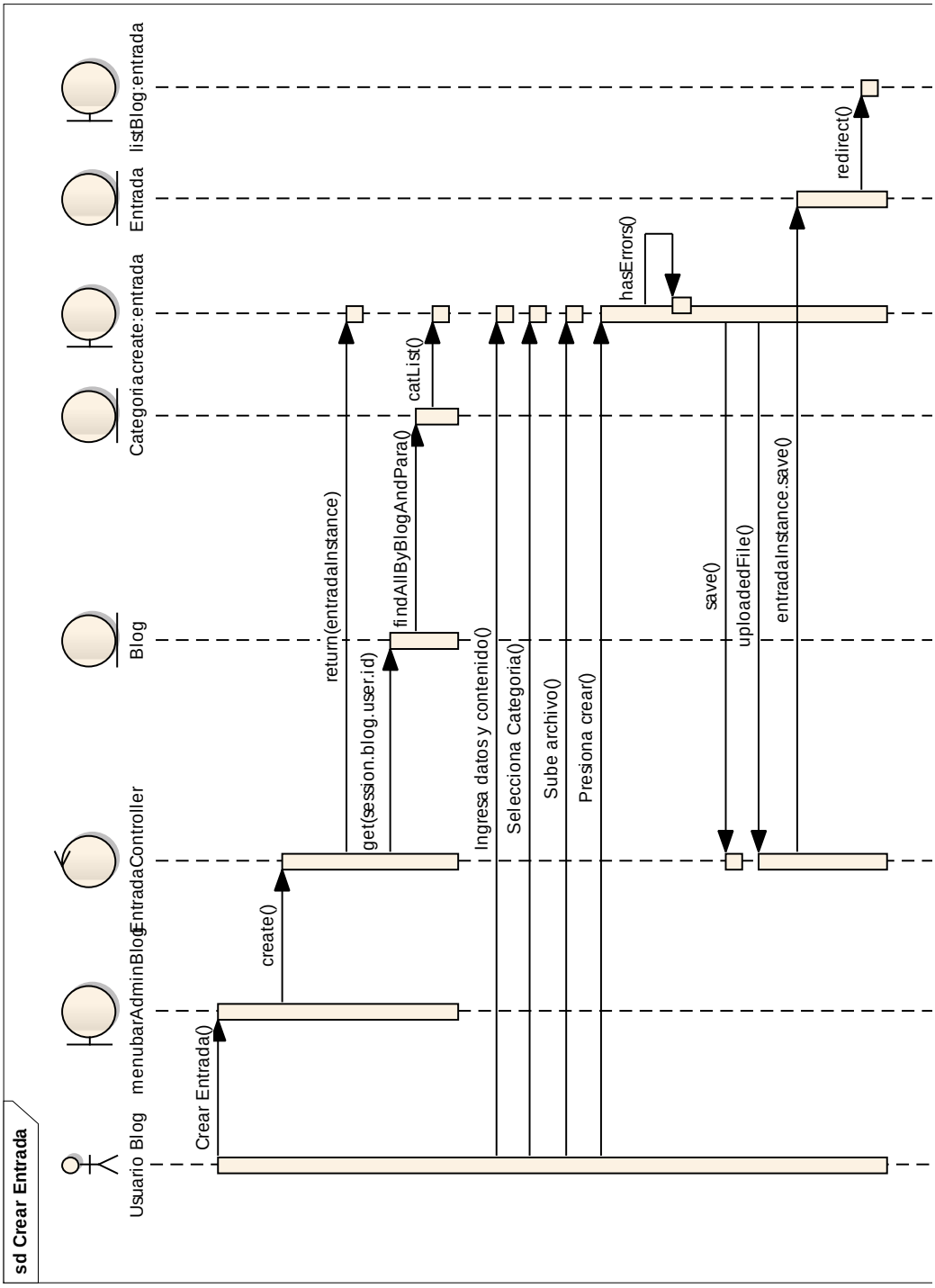

Figura 39. Diagrama de Secuencia: Crear Entrada

III.1.3.1.1.7. Diagrama de Secuencia para Caso de Uso: BAB2 Gestionar Entradas

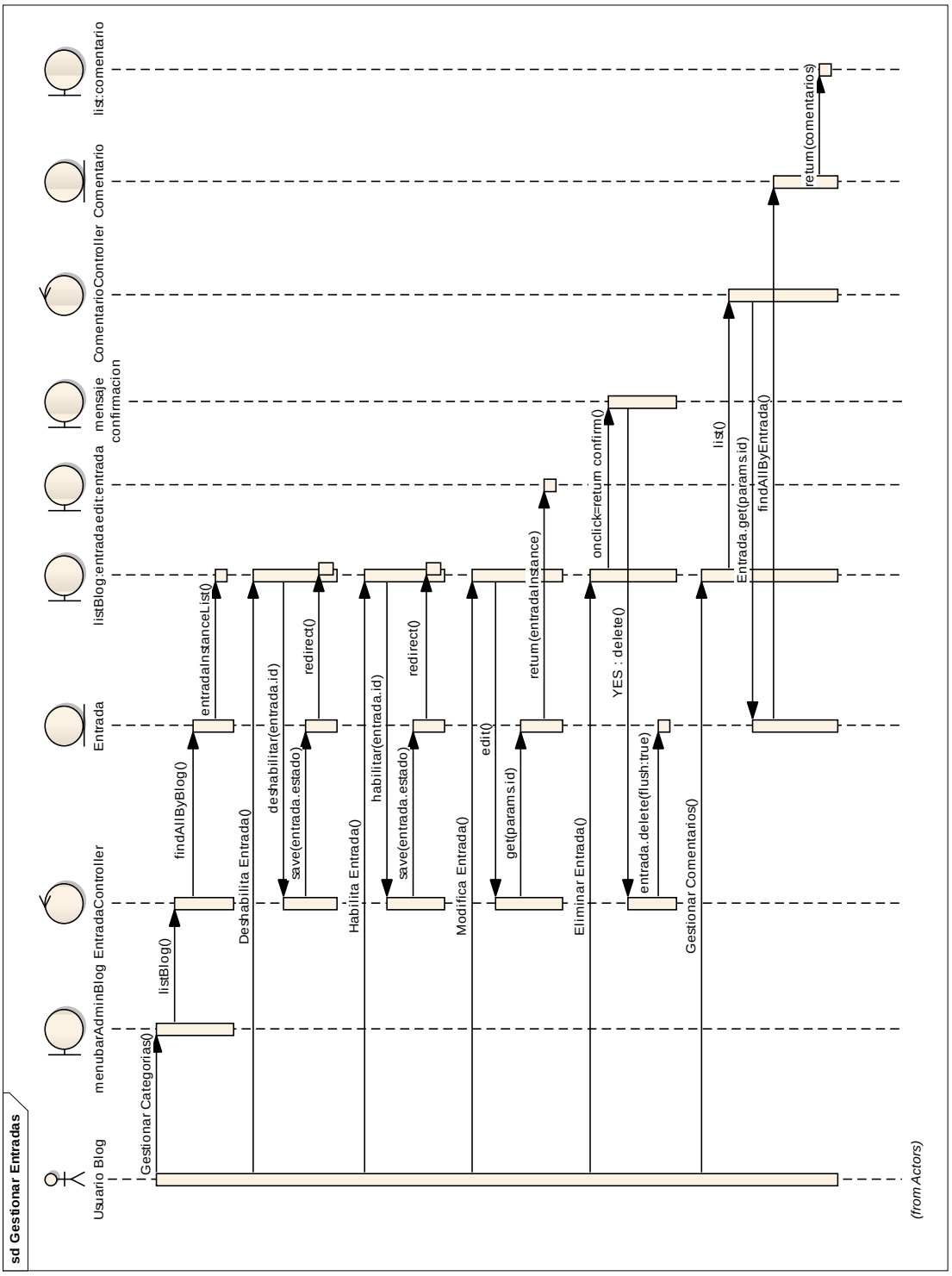

Figura 40. Diagrama de Secuencia: Gestionar Entradas

III.1.3.1.1.8. Diagrama de Secuencia para Caso de Uso: BAB2-1 Gestionar Comentarios

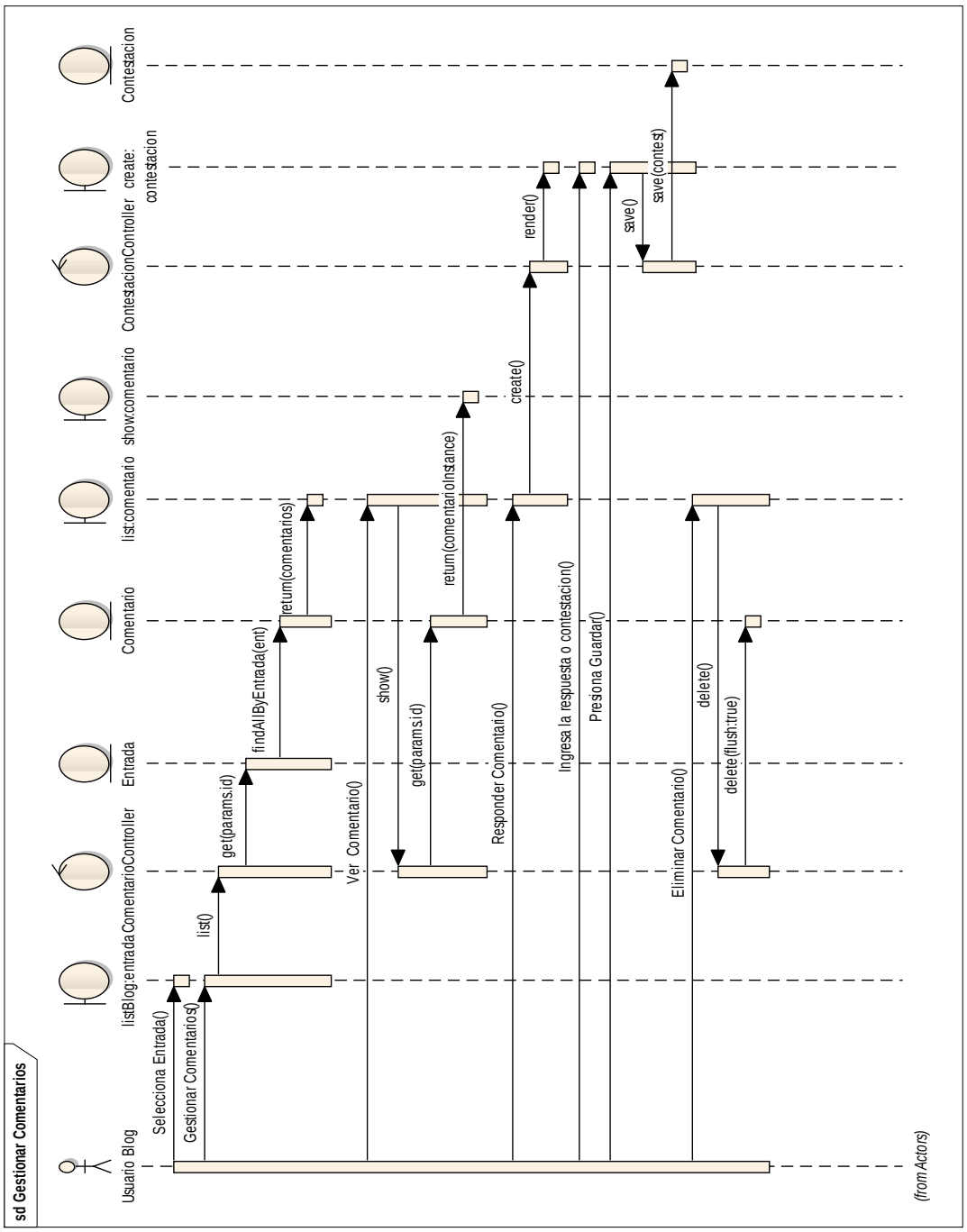

Figura 41. Diagrama de Secuencia: Gestionar Cometarios

III.1.3.1.1.9. Diagrama de Secuencia para Caso de Uso: BAB3 Publicar Enlaces

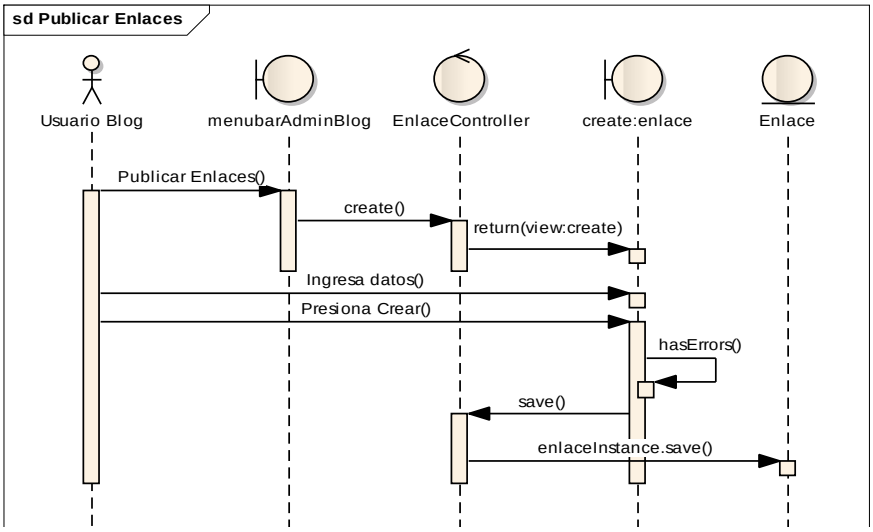

Figura 42. Diagrama de Secuencia: Publicar Enlaces

III.1.3.1.1.10. Diagrama de Secuencia para Caso de Uso: BAB4 Gestionar Enlaces publicados

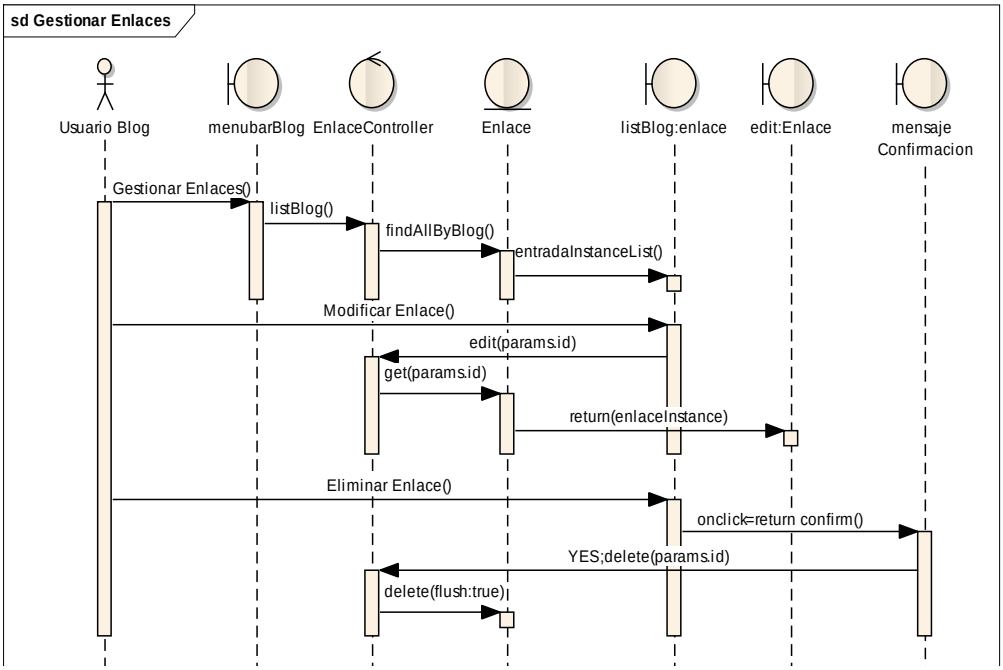

Figura 43. Diagrama de Secuencia: Gestionar enlaces publicados

III.1.3.1.1.11. Diagrama de Secuencia para Caso de Uso: BAB5 Subir Fotos

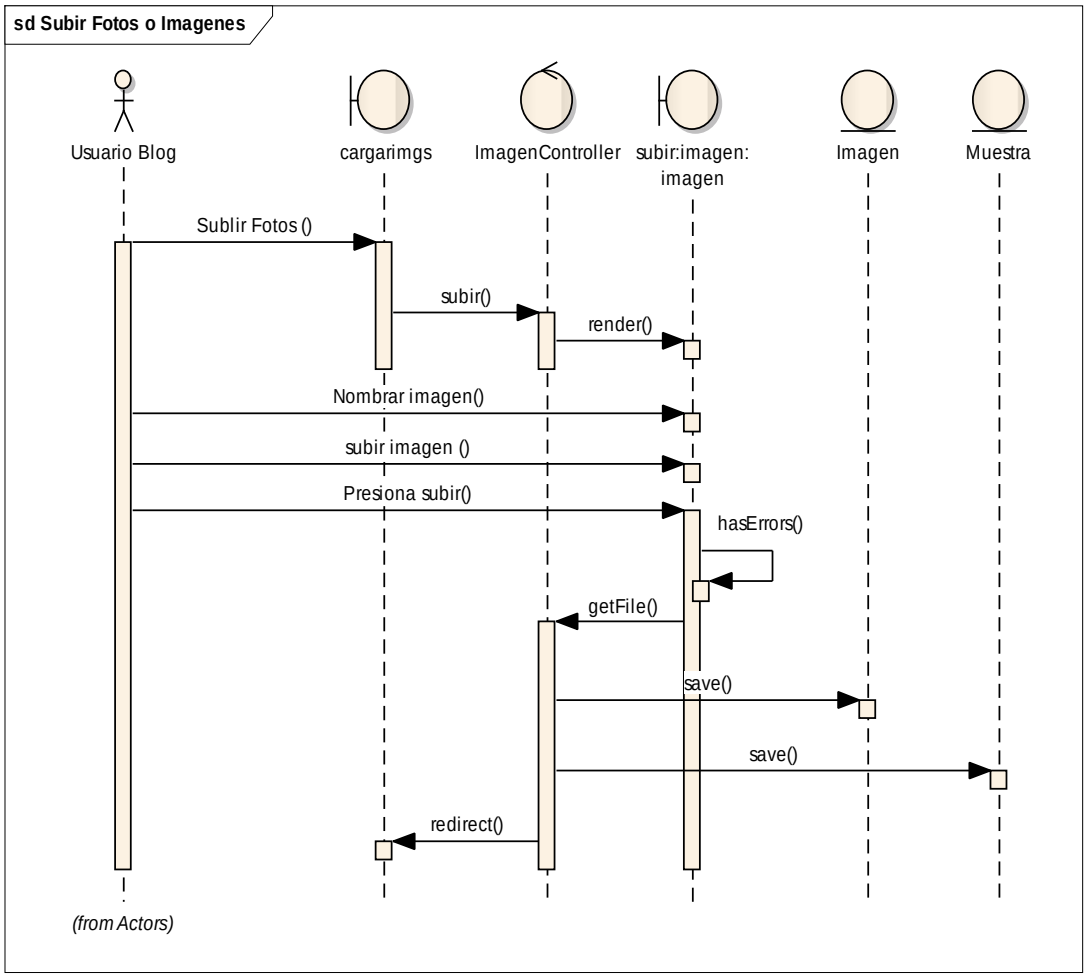

Figura 44. Diagrama de Secuencia: Subir Fotos

III.1.3.1.1.12. Diagrama de Secuencia para Caso de Uso: BAB6 Gestionar Fotos

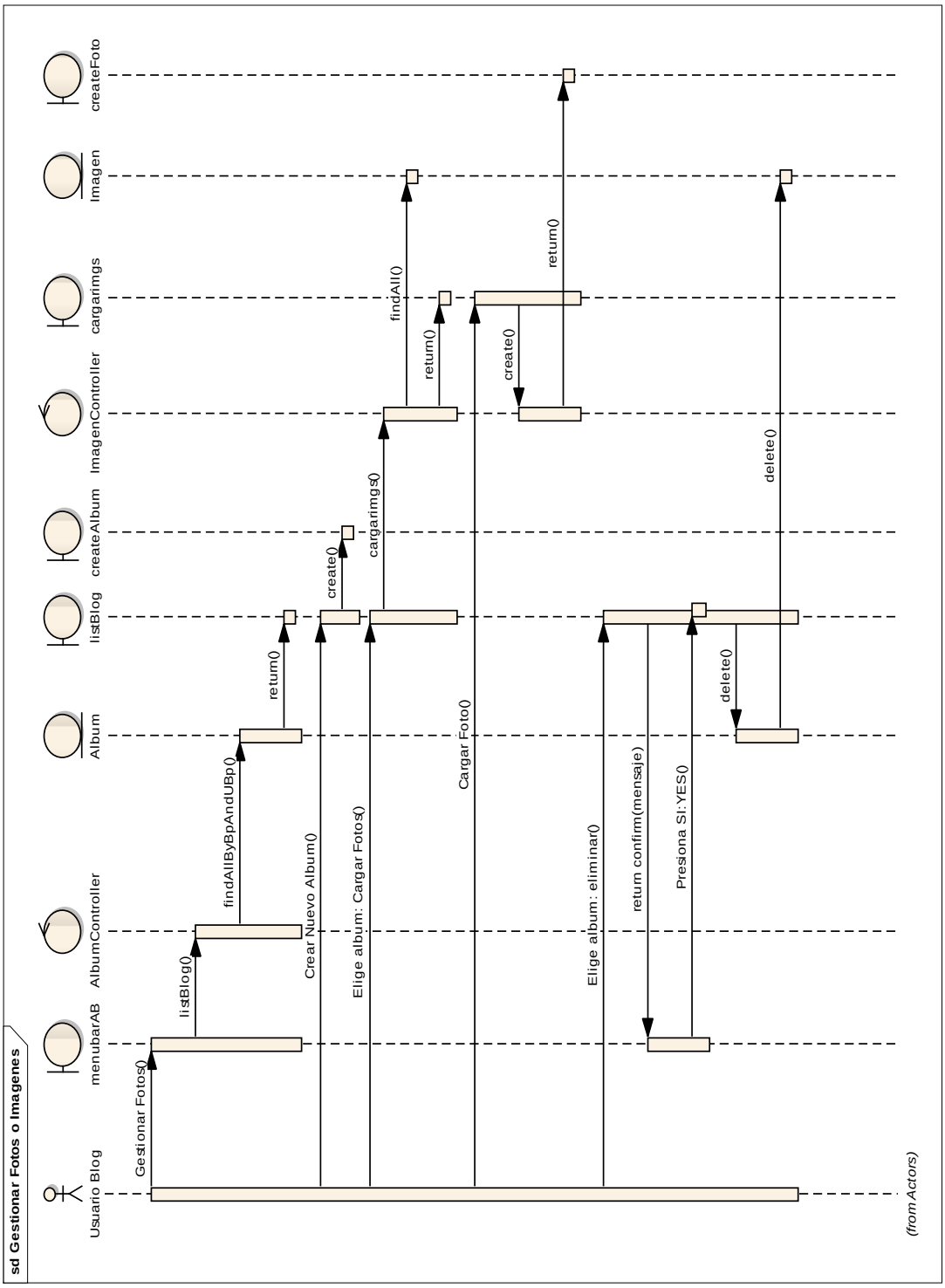

Figura 45. Diagrama de Secuencia: Gestionar Fotos

III.1.3.1.1.13. Diagrama de Secuencia para Caso de Uso: RC Registrar Cuenta

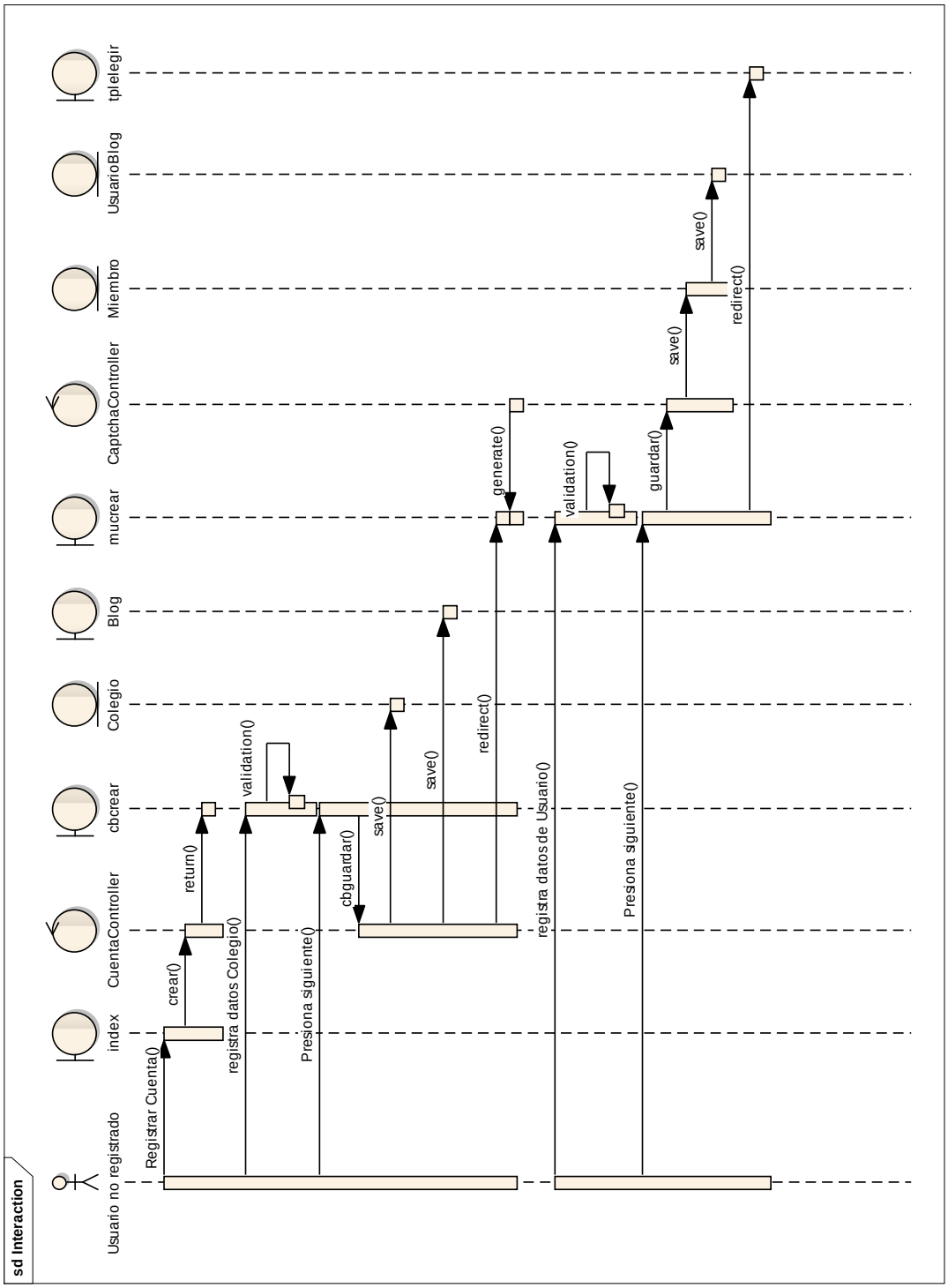

Figura 46. Diagrama de Secuencia: Registrar Cuenta

III.1.3.1.1.14. Diagrama de Secuencia para Caso de Uso: AC Acceder a la Cuenta

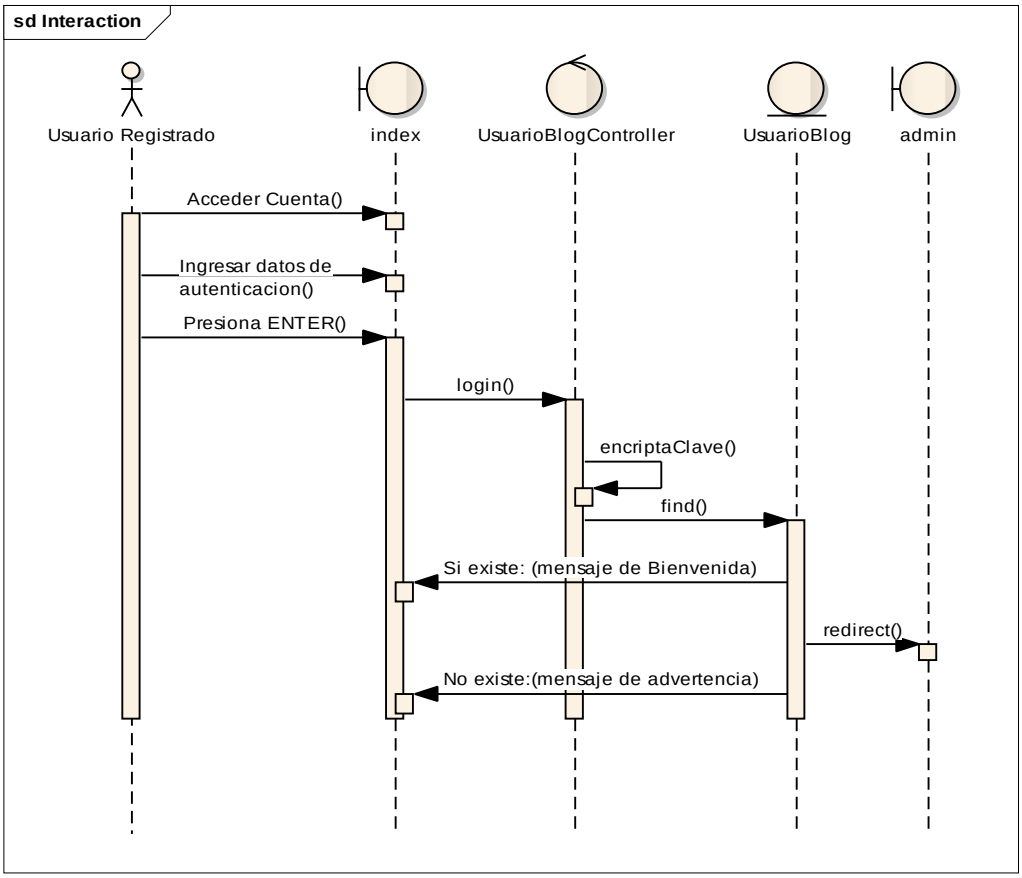

Figura 47. Diagrama de Secuencia: Acceder a Cuenta

III.1.3.1.2. Diagramas de Componentes

III.1.3.1.2.1. Diagrama de Componentes : Listar Afiliados registrados

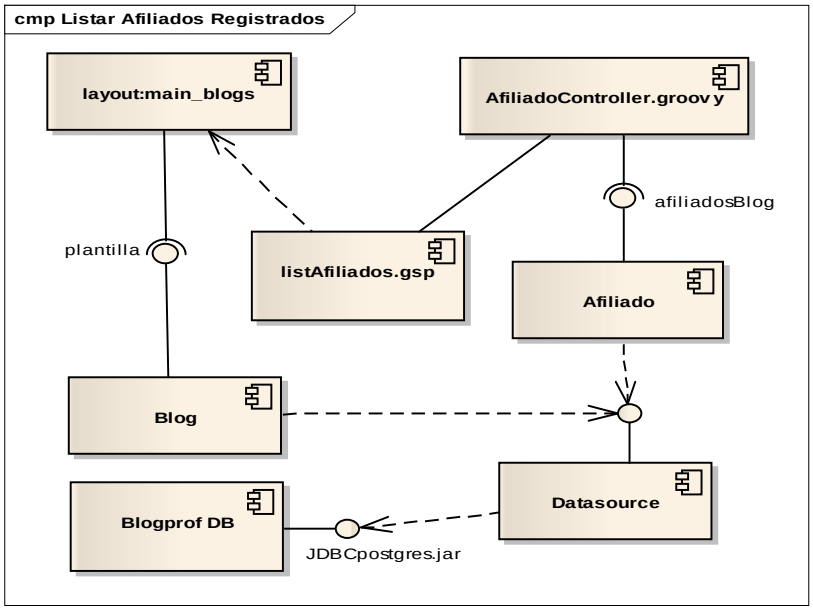

Figura 48. Diagrama de Componentes: Listar Afiliados registrados

III.1.3.1.2.2. Diagrama de Componentes: Ver CV de Afiliado seleccionado

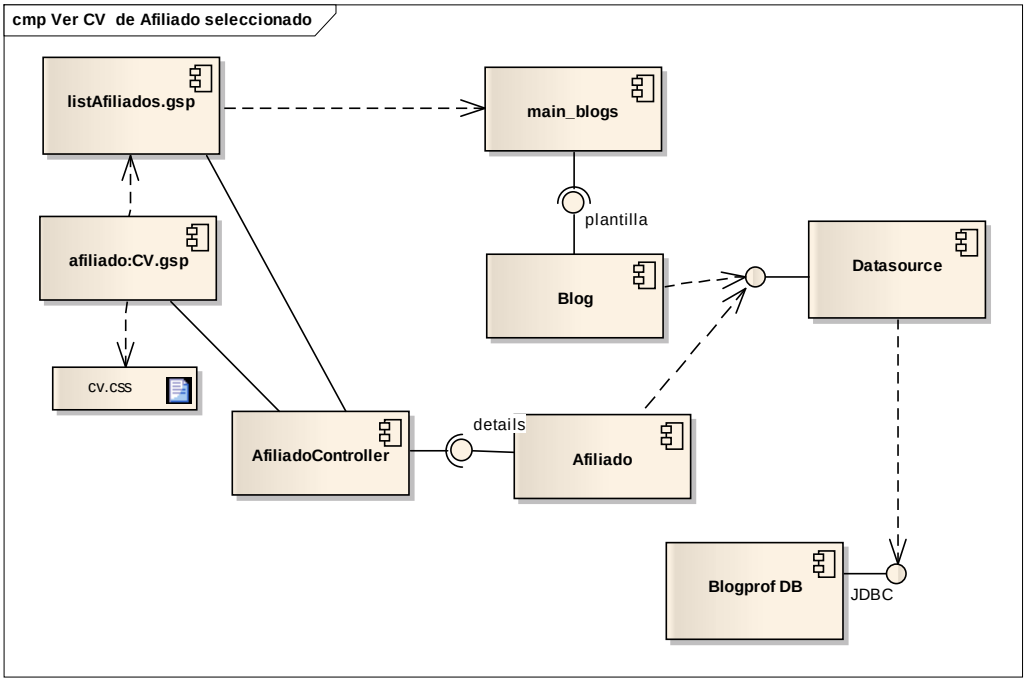

Figura 49. Diagrama de Componentes: Ver CV de Afiliado seleccionado

III.1.3.1.2.3. Diagrama de Componentes: Votar Encuesta seleccionada

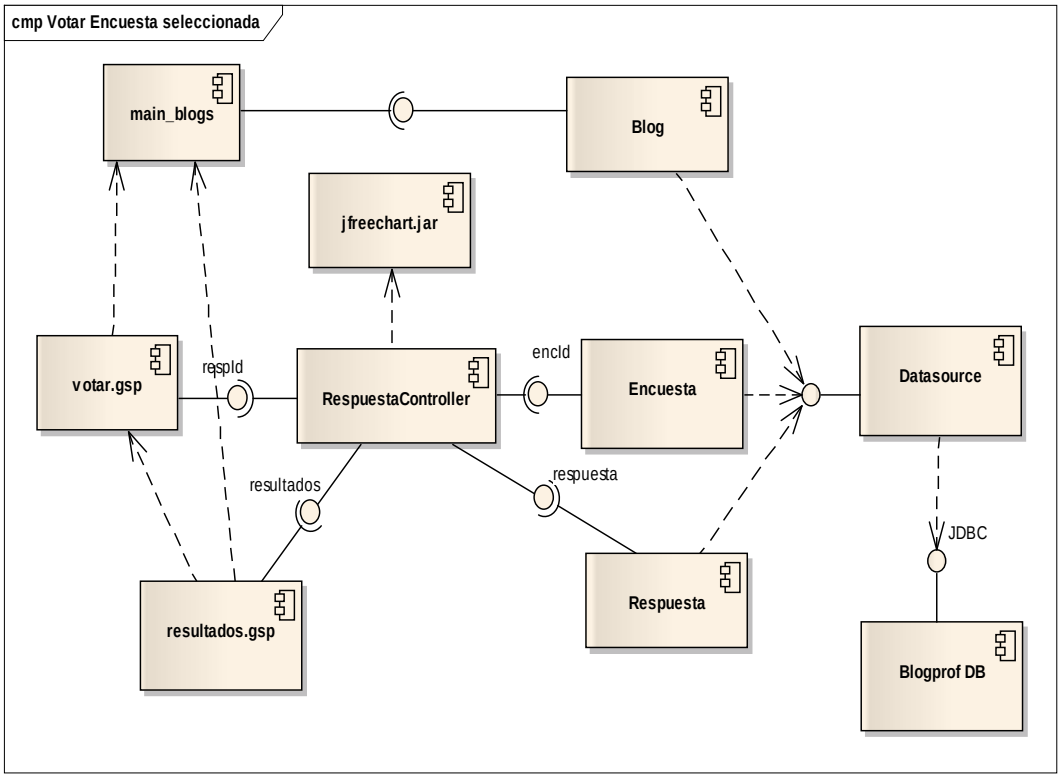

Figura 50. Diagrama de Componentes: Votar Encuesta seleccionada

III.1.3.1.2.4. Diagrama de Componentes : Leer Pagina seleccionada

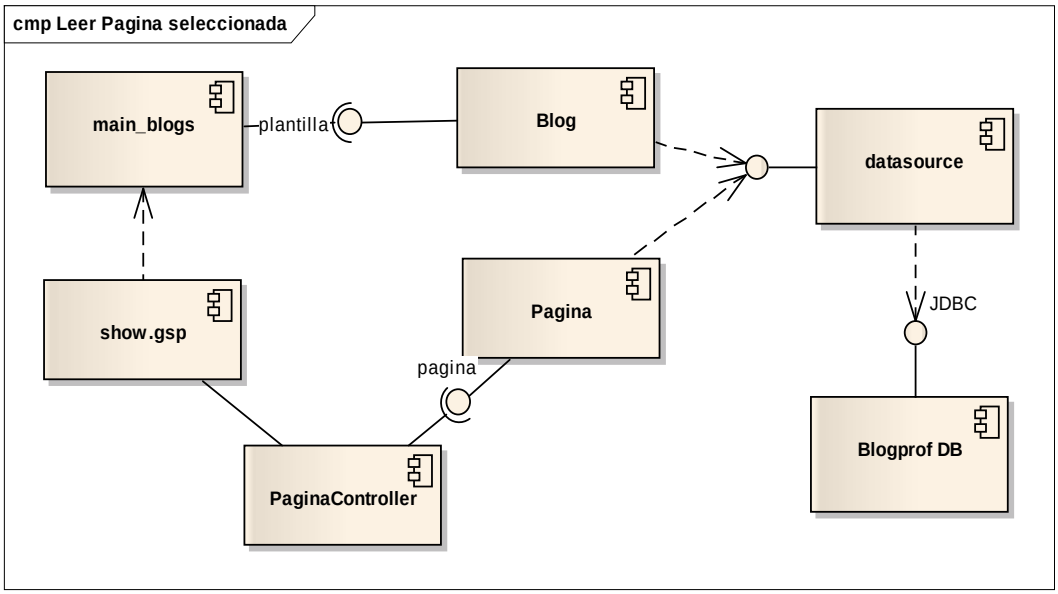

Figura 51. Diagrama de Componentes: Leer Pagina seleccionada

III.1.3.1.2.5. Diagrama de Componentes: Comentar Entrada

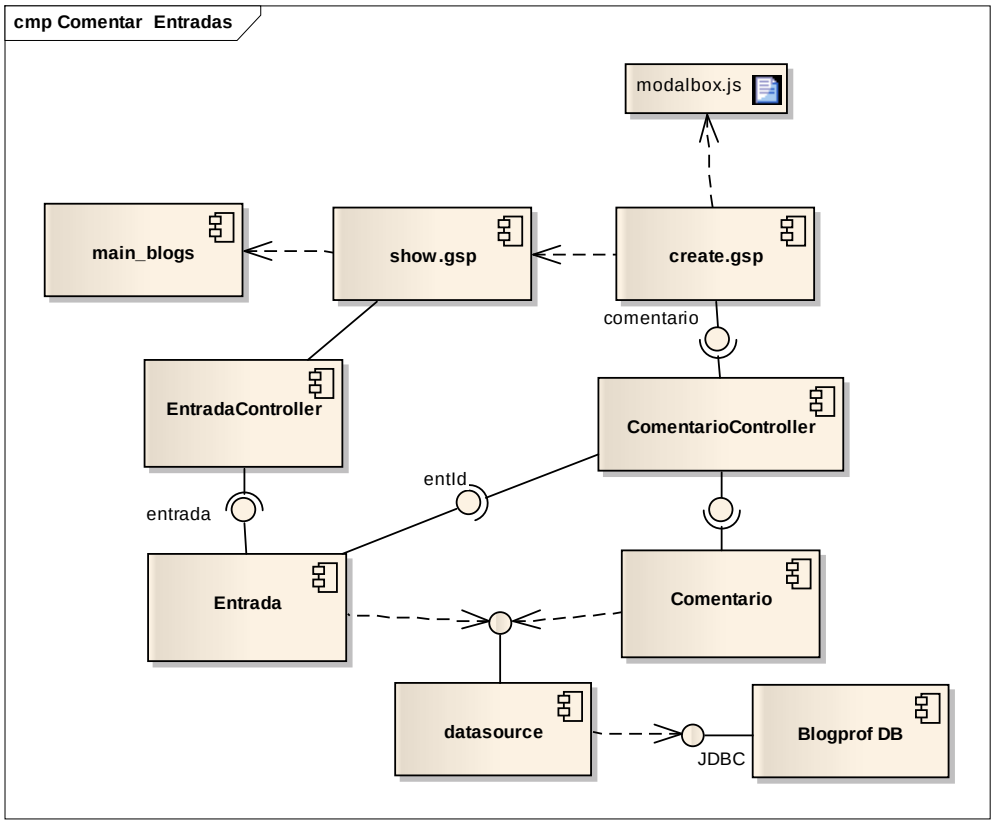

Figura 52. Diagrama de Componentes: Comentar Entrada

III.1.3.1.2.6. Diagrama de Componentes: Crear Entrada

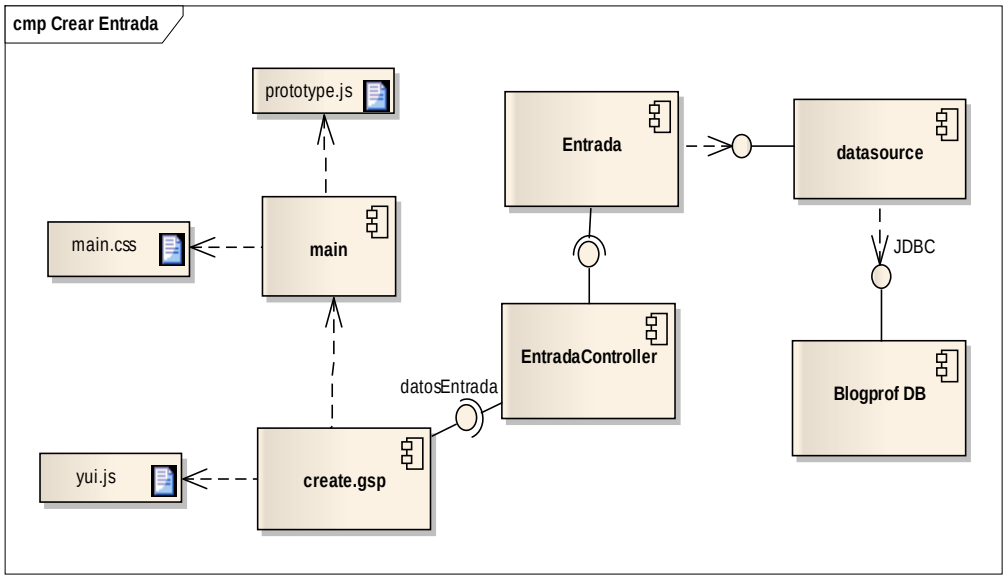

Figura 53. Diagrama de Componentes: Crear Entrada

III.1.3.1.2.7. Diagrama de Componentes: Gestionar Entradas

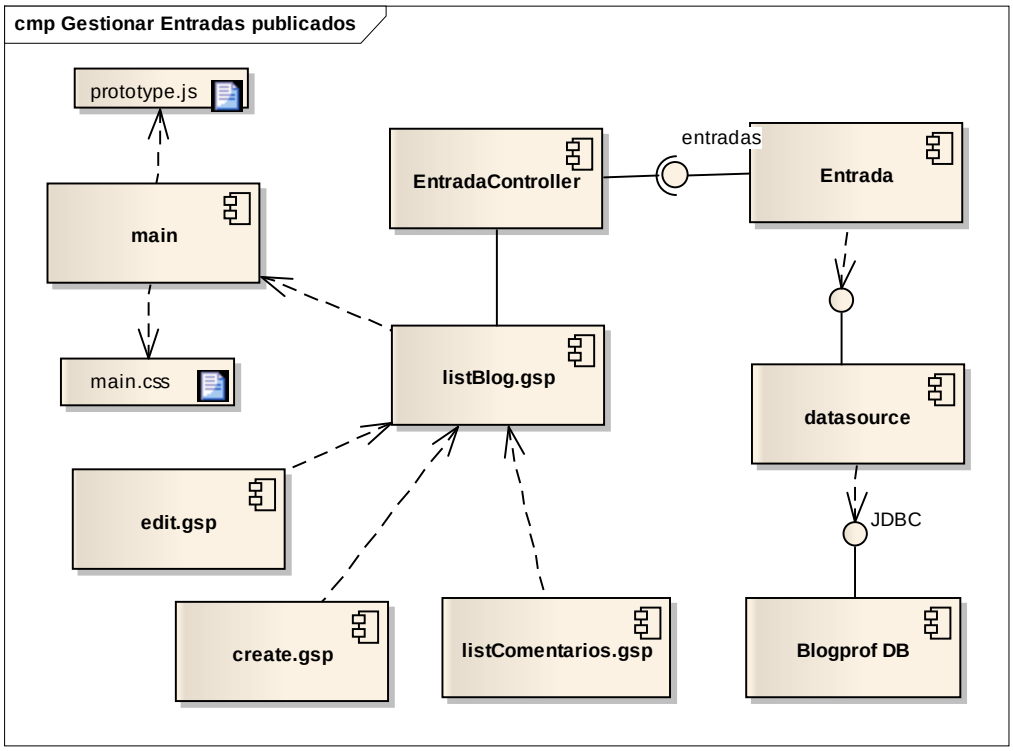

Figura 54. Diagrama de Componentes: Gestionar Entradas

III.1.3.1.2.8. Diagrama de Componentes: Gestionar Comentarios

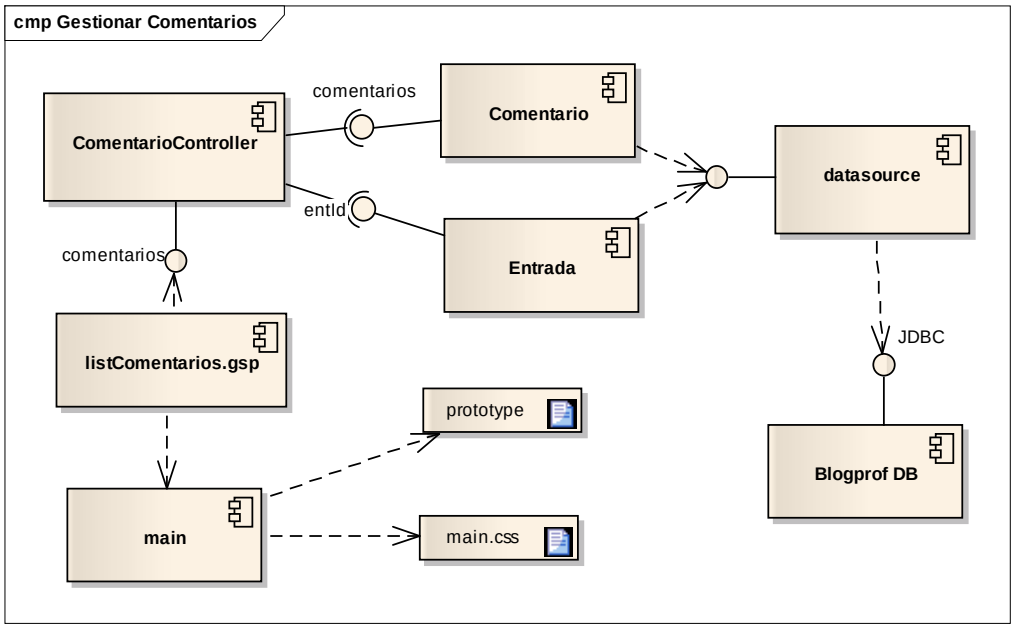

Figura 55. Diagrama de Componentes: Gestionar Comentarios

III.1.3.1.2.9. Diagrama de Componentes: Publicar Enlaces

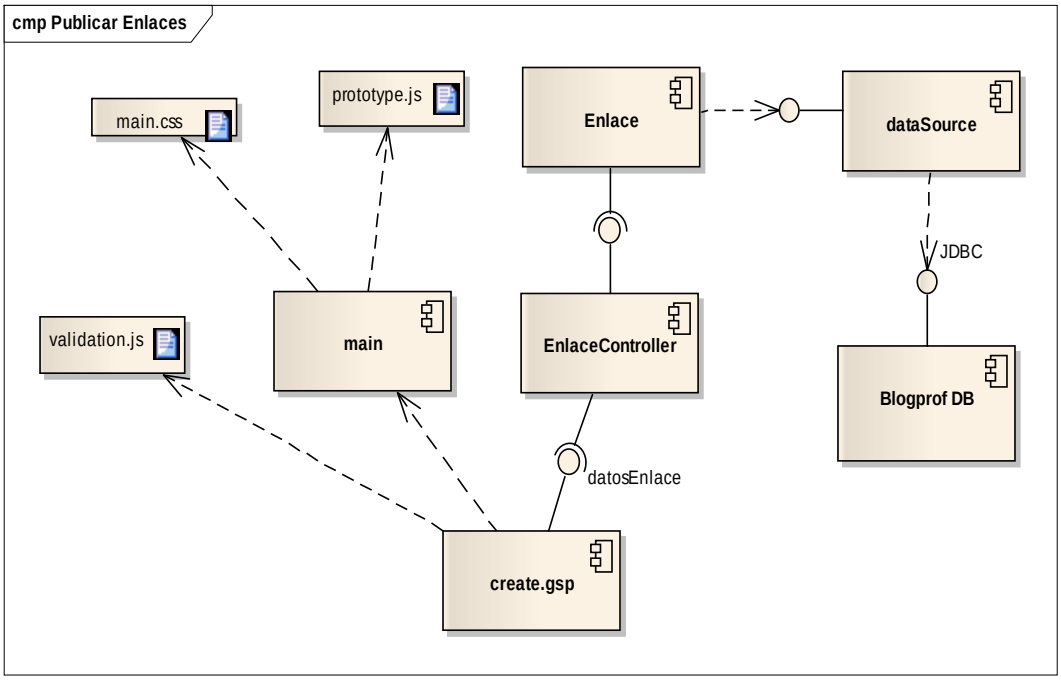

Figura 56. Diagrama de Componentes: Publicar Enlaces

III.1.3.1.2.10. Diagrama de Componentes: Gestionar Enlaces publicados

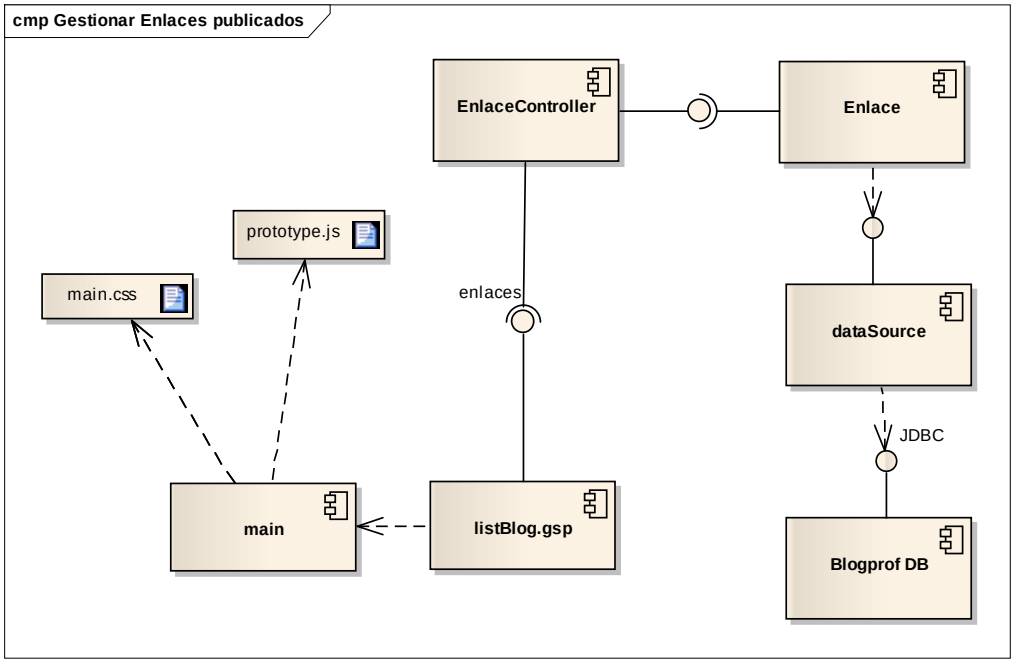

Figura 57. Diagrama de Componentes: Gestionar Enlaces publicados

III.1.3.1.2.11. Diagrama de Componentes: Subir Fotos

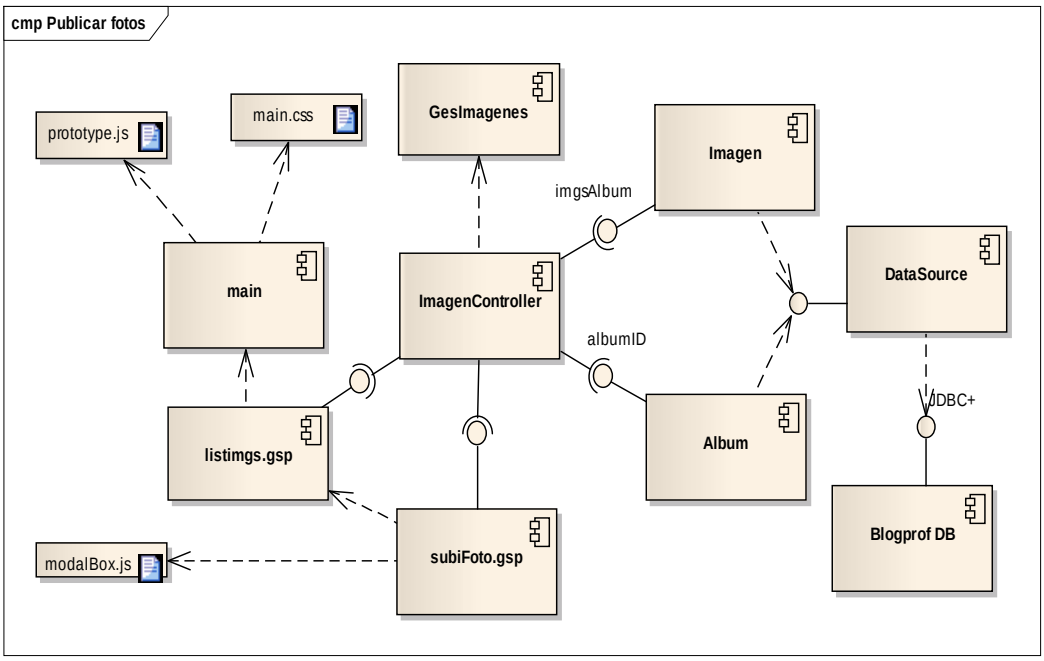

Figura 58. Diagrama de Componentes: Subir Fotos

III.1.3.1.2.12. Diagrama de Componentes: Gestionar Fotos

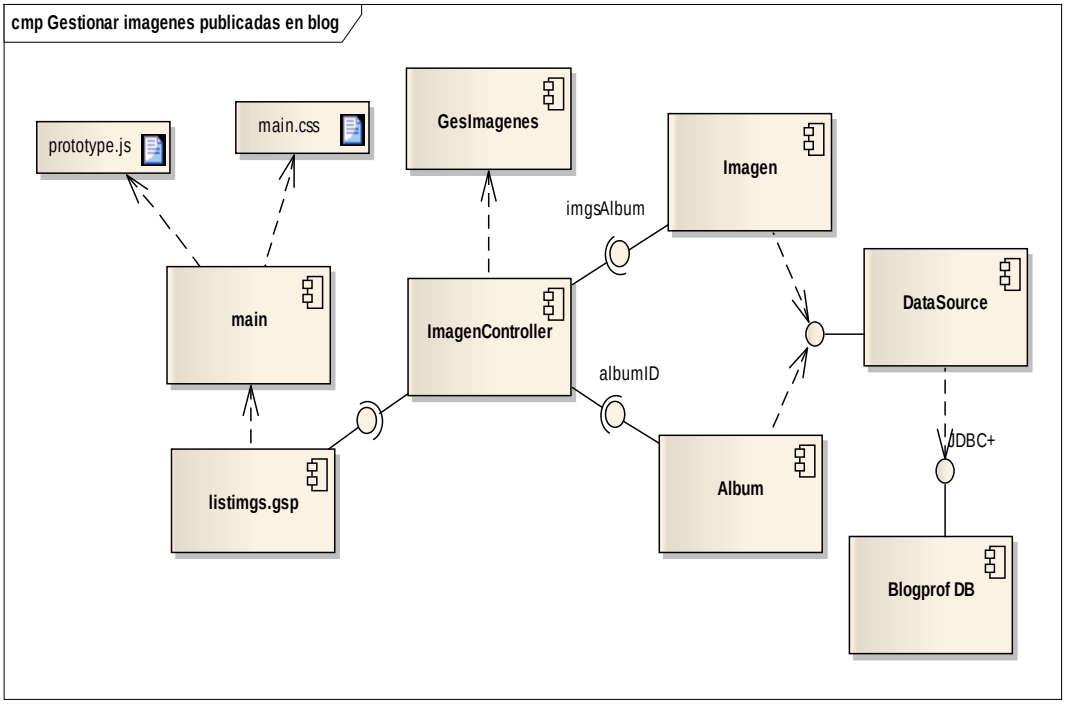

Figura 59. Diagrama de Componentes: Gestionar Fotos

III.1.3.1.2.13. Diagrama de Componentes: Registrar Cuenta

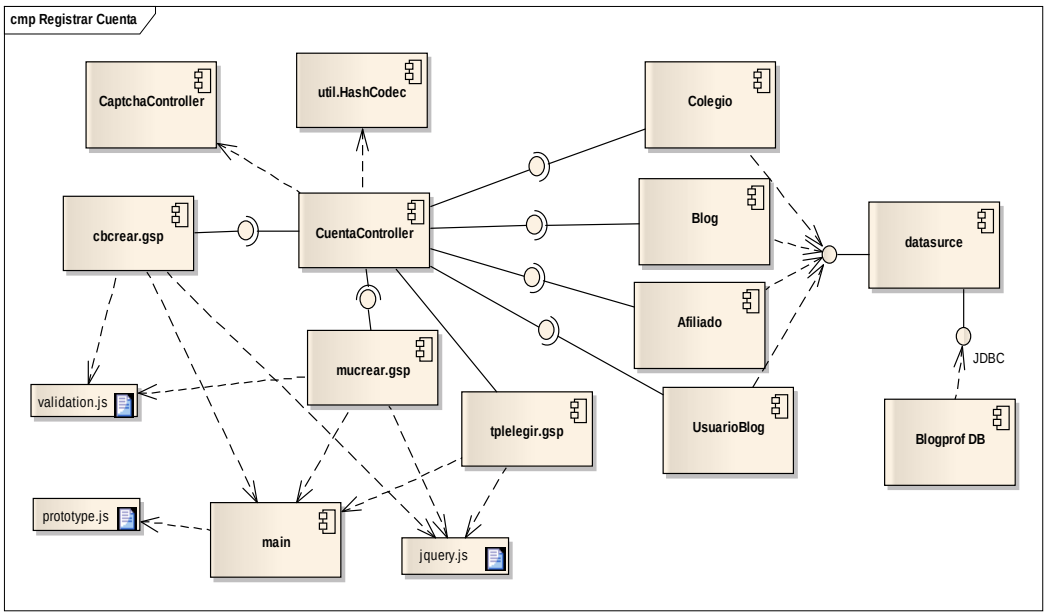

Figura 60. Diagrama de Componentes: Registrar Cuenta

III.1.3.1.2.14. Diagrama de Componentes: Acceder a la Cuenta

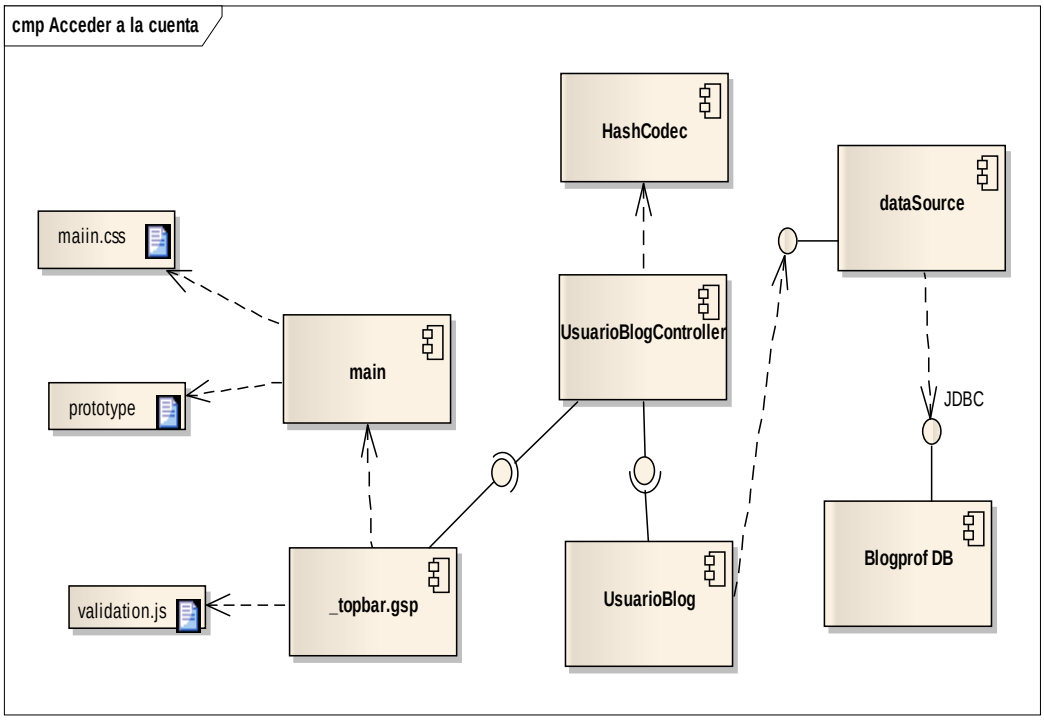

Figura 61. Diagrama de Componentes: Acceder a la Cuenta

III.1.3.1.3. Diagrama de Clases

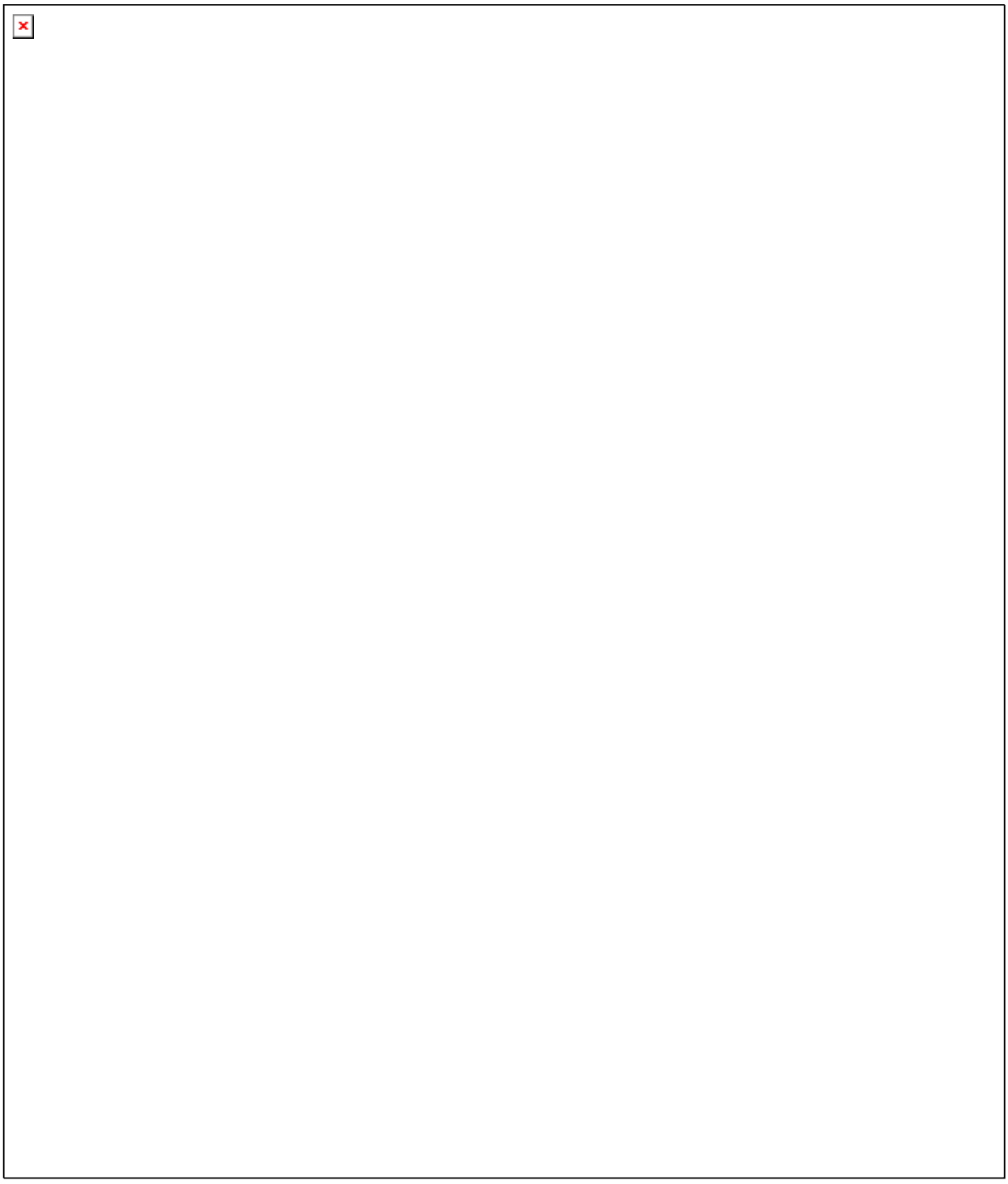

Figura 62. Diagrama de Clases

III.1.3.2. Modelo de Datos Físico

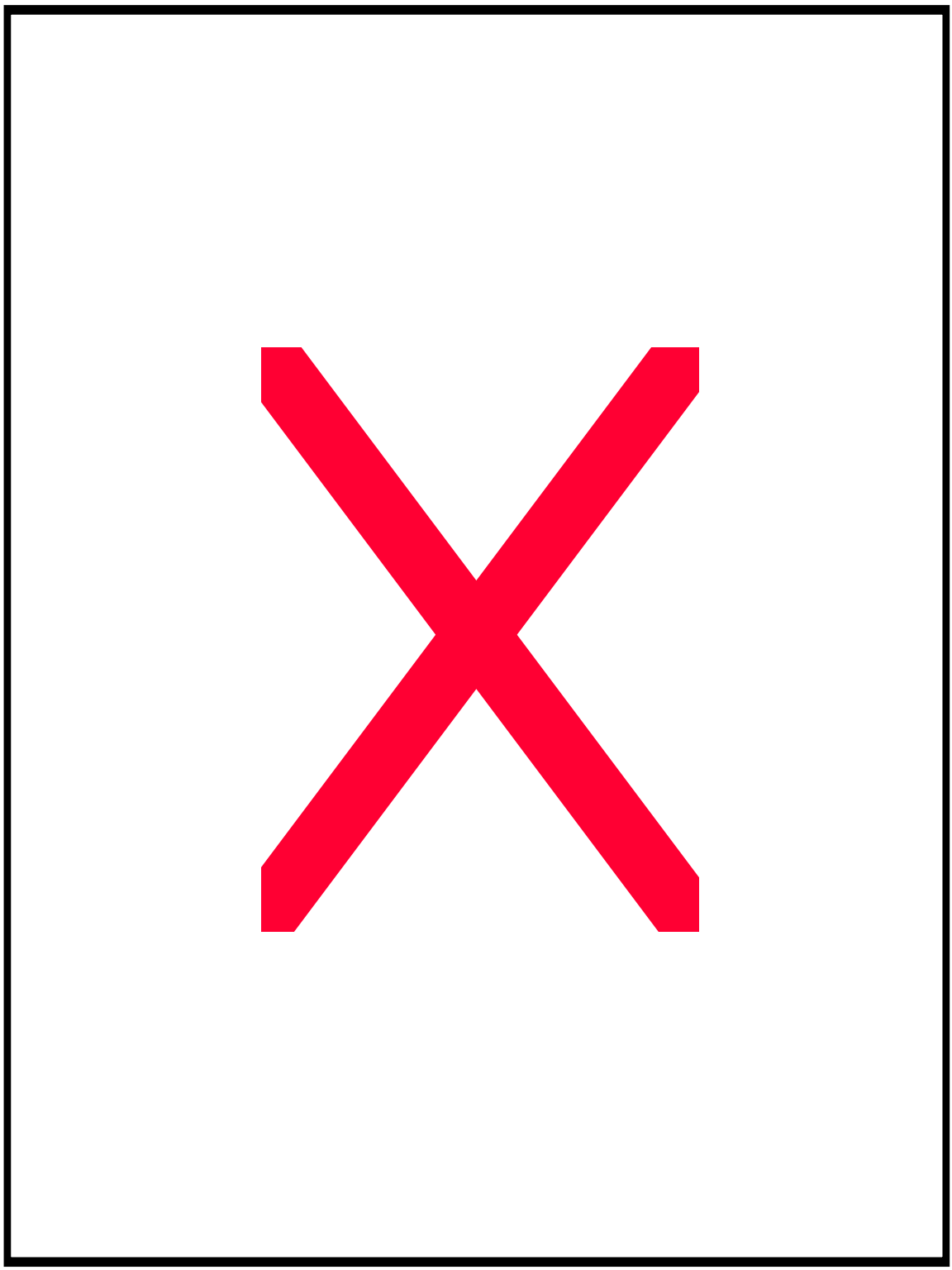

Figura 63. Modelo Físico de Datos

III.1.3.3. Modelo de Despliegue

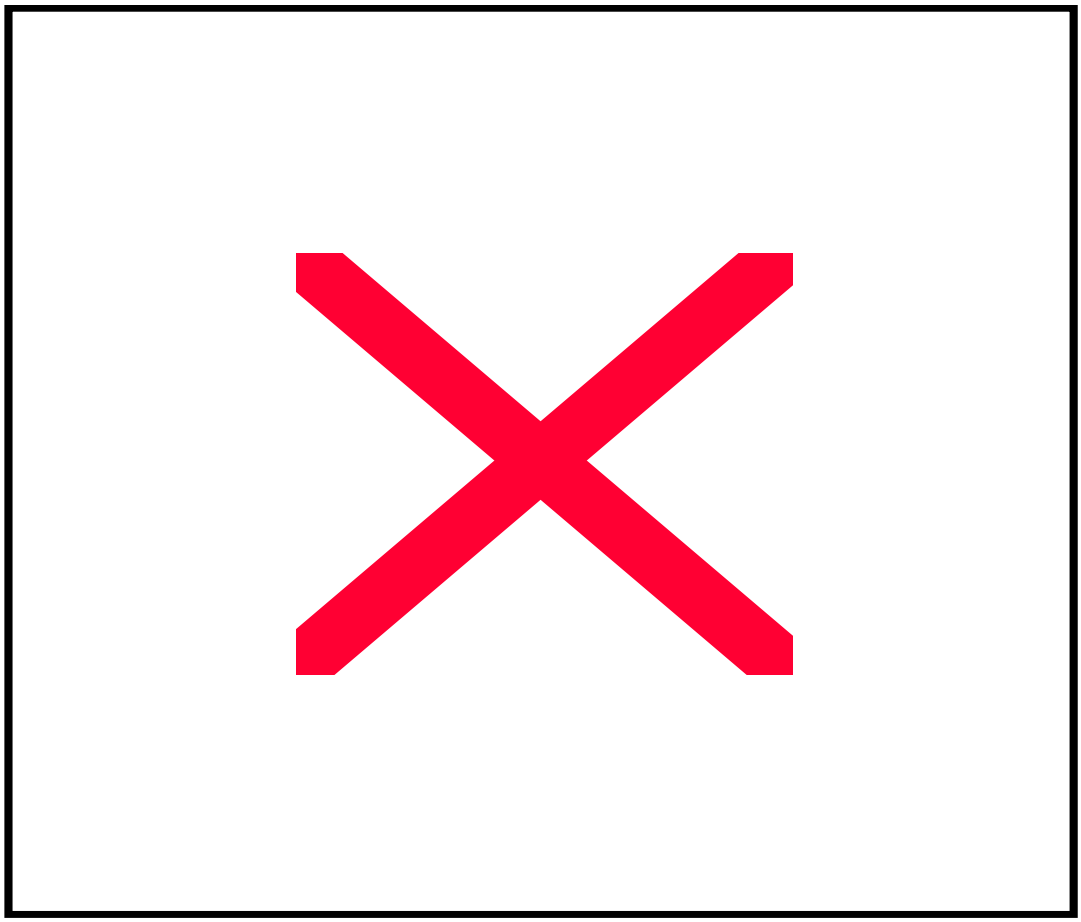

Figura 64. Modelo de Despliegue

III.1.4. Código Fuente

III.1.4.1. Clases

III.1.4.1.1. Clase Album

```
class Album {
   UsuarioBlog usuarioBlog
   Blog blog
   String nombre
   String descripcion
   SortedSet imagenes
   Integer imgCount = 0
   Date fechaCreacion = new Date()
   Date fechaModificacion = new Date()
   static belongsTo = [usuarioBlog:UsuarioBlog, blog:Blog]
   static hasMany = [ imagenes : Imagen ]
   static optionals = [ 'descripcion' ]
   static constraints = {
      nombre(size: 1..40, blank: false)
      descripcion()
   }
   static mapping = {
      imagenes cascade: 'all-delete-orphan', lazy: true, inverse: true
      descripcion type: 'text'
   }
   String toString (){
         return nombre
                  }

   int compareTo(obj) {
      obj.id.compareTo(id)
               }
}
```

III.1.4.1.2. Clase Archivo

```
class Archivo {
  String nombre
  String descripcion
  int size
  String extension
  ArchivoData archivoData
  Date fechaCreacion
  Date ultimaModificacion
  Blog blog
  UsuarioBlog usuarioBlog
   static belongsTo = Blog, UsuarioBlog
  static constraints = {
      archivoData( nullable: false )
```

```
        nombre(nullable:false, blank:false, length:1..50)
            size( nullable: false )
        extension( nullable: false )
       }

  def getDownloadName() {
   return "${nombre}.${extension}"
                }
    }
```

III.1.4.1.3. Clase ArchivoData

```
class ArchivoData {
    private static final int TEN_MEG_IN_BYTES = 1024*1024*10
    Archivo archivo
     static belongsTo = [archivo:Archivo]
     byte[] data
     static constraints = {
        data( nullable: false, minSize: 1, maxSize: TEN_MEG_IN_BYTES )
                    }
      }
```

III.1.4.1.4. Clase AskComment

```
class  AskComment implements Comparable {
    Entrada entrada
    Comentario comment
    String contenido
    UsuarioBlog usuBlog
    Date fechaCreacion
     static belongsTo = [usuBlog:UsuarioBlog]
     static constraints = {
        contenido(nullable:false, blank:false)
        fechaCreacion(nullable:false)
                    }
    public int compareTo(Object o) {
        return fechaCreacion.compareTo(o.fechaCreacion)
        }
}
```

III.1.4.1.5. Clase Blog

```
class Blog {
   String urlBlog
   String identificador
   String nombre
   Date fechaRegistro
   String estado
  Organizacion organizacion
   static belongsTo = Organizacion
```

```
  Plantilla plantilla
   static hasMany = [usuariosBlog:UsuarioBlog,
catEntradas:CategoriaEntrada, catEnlaces:CategoriaEnlace,
albums:Album, entradas:Entrada, paginas:Pagina, encuestas:Encuesta,
enlaces: Enlace, archivos:Archivo, imagenes:Imagen, logos:Logo ]
      static def constraints = {
        identificador(unique:true)
         urlBlog(blank:false)
         nombre(nullable:false)
                  }
      }
```

III.1.4.1.6. Clase CapacidadPersonal

```
class CapacidadPersonal {
    String area
    String habilidades
    String logros
   Miembro miembro
  static belongsTo = Miembro
   static constraints = {
         area(inList: ["Sociales","Organizativas", "Tecnicas",
"Informaticas","Artisticas", "Otras"])
          habilidades(blank:false, length: 1..400)
               logros(length: 1..400)
                        }
    }
```

III.1.4.1.7. Clase Cargo

```
class Cargo {
       String cargo
    int nivelCargo
    Miembro miembro
    Directorio directorio
    static belongsTo = Directorio
    static hasMany = [miembro: Miembro]
      static constraints = {
    cargo (nullable:false)
    nivelCargo(inList:[1,2,3,4,5,6,7,8,9,10,11,12,13,14])
    miembro (nullable:false)
             }
    }
```

III.1.4.1.8. Clase CategoriaEnlace

```
class CategoriaEnlace {
      String nombre
    String nombreBase
    String descripcion
```

```
    int    contador = 0
     Blog blog
    static belongsTo = Blog
    static hasMany = [ enlaces: Enlace]
    static constraints = {
                nombre(nullable:false, blank: false, length: 1..20)
                    }
     String toString ()
            {  return nombre   }
}
```

III.1.4.1.9. Clase CategoriaEntrada

```
class CategoriaEntrada {
   String nombre
    String nombreBase
    String descripcion
    int    contador = 0
    Blog blog
    static belongsTo = Blog
    static hasMany = [ entradas: Entrada]
   static constraints = {
    nombre(nullable:false, blank: false, length: 1..20)
    }
   String toString () {
     return nombre
     }
}
```

III.1.4.1.10. Clase Comentador

```
class Comentador {
   String nombre
   String url
   String email
   Comentario comentario
   static belongsTo = Comentario
   static constraints = {
       nombre(nullable:false, blank:false)
       url(nullable:true, blank:false, url:true)
       email(nullable:false, blank:true, email:true)
        }
}
```

III.1.4.1.11. Clase Comentario

```
class Comentario {
   Entrada entrada
   String contenido
   Comentador who = new Comentador()
```

```
    Date fechaCreacion
    int askComment = 0
      static belongsTo = Entrada
    static constraints = {
       contenido(nullable:false, blank:false)
       who(nullable:false)
       fechaCreacion(nullable:false)
            }
 }
```

III.1.4.1.12. Clase Cuota

```
class Cuota {
    Date fecha
    String concepto
    Float importe=0.00
    Colegio colegio
    Miembro miembro
   static belongsTo = Colegio, Miembro
    static constraints = {
         concepto(nullable:false)
         importe(nullable:false, scale:2)
          }
}
```

III.1.4.1.13. Clase Directorio

```
class Directorio {
     Organizacion organizacion
     String formaPresentacion
      static hasMany= [cargos: Cargo]
      static constraints = {
           formaPresentacion(inList:["Lista", "Arbol"])
                }
}
```

III.1.4.1.14. Clase Egreso

```
class Egreso {
     Date fecha
     String concepto
     Float importe=0.00
     Organizacion organizacion
     static belongsTo = Organizacion
        static constraints = {
     concepto(nullable:false)
     importe(nullable:false, scale:2)
              }
}
```

III.1.4.1.15. Clase Encuesta

```
class Encuesta {
      String titulo
      String textoPregunta
      Date fechaCreacion
      int contVotos = 0
     Boolean estado = false
     Blog blog
       static belongsTo = Blog
       static hasMany = [respuestas:Respuesta]
       static constraints = {
                  textoPregunta(nullable:false, blank:false, length:1..2000)
               }
     String toString (){
      return nombre   }
  }
```

III.1.4.1.16. Clase Enlace

```
class Enlace {
    String nombre
    String descripcion
    String urlEnlace
    Boolean estado = true
    Blog blog
    UsuarioBlog usuarioBlog
    CategoriaEnlace catEnlace
            static belongsTo = Blog, UsuarioBlog, CategoriaEnlace
            static constraints = {
                    nombre(blank: false, length: 1..20)
                     urlEnlace(nullable: false, url:true)
                }
}
```

III.1.4.1.17. Clase EntimgData

```
class EntimgData {
    private static final int ONE_MEG_IN_BYTES = 1024*1024*1
     byte[] data
    EntradaImagen entradaImagen
    static belongsTo = [entradaImagen:EntradaImagen]
     static constraints = {
        data( nullable: false, minSize: 1, maxSize: ONE_MEG_IN_BYTES
)
                }
    }
```

III.1.4.1.18. Clase Entrada

```
class Entrada {
        String titulo
    String contenido
    Boolean estado = false
     Date fechaCreacion
          Date fechaModificacion
          String cita
          Boolean permComentarios = false
          int contComentarios = 0
          SortedSet comentarios
          CategoriaEntrada catEntrada
          Blog blog
          UsuarioBlog usuarioBlog
          static belongsTo = Blog, UsuarioBlog
          static hasMany = [comentarios:Comentario]
          static mapping = {
                sort "fechaModificacion":"desc"
                comentarios cascade: 'all-delete-orphan', lazy: true,
inverse: true
                descripcion type: 'text'
                     }

        static constraints = {
              titulo(nullable:false,unique:true, blank:false, length:1..50)
              contenido(nullable:false, blank:false, length:1..5000)
              estado(nullable:false)
              cita(blank: false, nullable: false, length: 1..100)
           }
  }
```

III.1.4.1.19. Clase EntradaImagen

```
class EntradaImagen {
  String nombre
  int ancho = 100
  int alto = 100
  String posicion
  int size
  String extension
  EntimgData entimgData
  Entrada entrada
  static belongsTo = Entrada
   static  constraints = {
      entimgData( nullable: false )
      nombre(nullable:false, blank:false, length:1..50)
      size( nullable: false )
      posicion(inList:['top', 'bottom'])
      extension(inList:['jpg', 'jpeg', 'png', 'gif','bmp'] )
```

```
        }
    def getDownloadName() {
            return "${nombre}.${extension}"
              }
}
```

III.1.4.1.20. Clase Estado

```
class Estado {
        Pais pais
        String nombre
String toString () {
      return nombre  }
}
```

III.1.4.1.21. Clase ExperienciaLaboral

```
class ExperienciaLaboral {
     Date fechaInicio
     Date fechaFin
     String cargoOcupado
     String laborPrincipal
     String nombreEmpleador
     String dirEmpleador
     String sectorEmpleador
           Miembro miembro
    static belongsTo = Miembro
   static constraints = {
    fechaInicio(nullable:true)
    fechaFin(nullable:true)
    cargoOcupado(blank: false)
    nombreEmpleador(blank: false)
            }
}
```

III.1.4.1.22. Clase FormacionAcademica

```
class Idioma {
     Date fechaInicio
     Date fechaFin
     String tituloObtenido
     String capacidadesAprendidas
     String centroCapacitador
     String tipocentroCap
     String nivelAlcanzado

     Afiliado afiliado
     static belongsTo = Afiliado
```

III.1.4.1.23. Clase FotoPersona

```
class FotoPersona {
   Persona persona
   Integer size
   byte[] data
   String contentType ='image/jpg'
   Integer ancho = 200
   Integer alto = 200
   static belongsTo = [ Persona ]

   static mapping = {
     data type: 'binary'
   }

  static constraints = {
      data(maxSize: MAX_SIZE)
   }

   static final Integer MAX_SIZE = 5 * 1024 * 1024
   static final Integer Original = 1
   static final Integer Large = 2
   static final Integer Medium = 3
   static final Integer Small = 4
   static final Integer[] anchos =  [ 0, 0, 500, 250, 150 ]
   static final Integer[] altos = [ 0, 0, 500, 250, 150 ]
   static final String[] Names = [ '', 'Original', 'Large', 'Medium', 'Small' ]


}
```

III.1.4.1.24. Clase Idioma

```
class Idioma {
          String nombreIdioma
     String comprensionAuditiva
     String comprensionLectura
     String expresionOral
     String nivelEscritura
     Miembro miembro
     static belongsTo = Miembro

   static constraints = {

nombreIdioma(inList:['arabe','albanes','aymara','quechua','guarani','wejnay
ek',
    'español','ingles','frances','japones','aleman',
'griego','latin','portugues','finlandes','italiano',
          'koreano','mapuche'])
```

```
comprensionAuditiva(inList:['bajo','regular','bueno','optimo','excelente'])

comprensionLectura(inList:['bajo','regular','bueno','optimo','excelente'])
        expresionOral(inList:['bajo','regular','bueno','optimo','excelente'])
        nivelEscritura(inList:['bajo','regular','bueno','optimo','excelente'])
                                }
}
```

III.1.4.1.25. Clase Imagen

```
class Imagen {
  String nombre
  String contentType
  Integer ancho
  Integer alto
  Integer size
   byte[] data
  Date fechaCreacion = new Date()
   Blog blog
   UsuarioBlog usuarioBlog
   Album album
      static belongsTo = [ blog:Blog, usuarioBlog:UsuarioBlog,
album:Album ]
      static constraints = {
            nombre(blank:false, length:1..50)
            contentType(inList:['jpg', 'jpeg', 'png', 'gif','bmp'], length:1..10 )
             }
 static mapping = {
    data type: 'binary'
          }
  }
```

III.1.4.1.26. Clase ImagenPTL

```
class ImagenPTL {

  String nombre
   String imagen

        Plantilla plantilla
        static belongsTo = Plantilla

   static constraints = {
        nombre(size:5..50, blank:false, unique:true)
   }
}
```

III.1.4.1.27. Clase InformacionActual

```
class InformacionActual {

        Persona persona
        String estadoCivil = 'Soltera/o'
        String email
          int telefono = 0
        String dirDomicilio
        String pais
        String estado
        String ciudad
          int codPostal=0
        static belongsTo = Persona

   static constraints = {
        estadoCivil(inList:['Soltera/o', 'Casada/a','Divorciada/o', 'Viuda/o',
'Prometida/o', 'Una Relacion'])
        email(email:true)
        dirDomicilio(nullable:true)
        pais(nullable:true)
        estado(nullable:true)
        ciudad(nullable:true)
   }
}
```

III.1.4.1.28. Clase Ingreso

```
class Ingreso {

        Date fecha
        String concepto
        Float importe=0.00
        Organizacion organizacion

        static belongsTo = Organizacion

   static constraints = {
        concepto(nullable:false)
        importe(nullable:false, scale:2)
   }
}
```

III.1.4.1.29. Clase Logo

```
class Logo {

   String nombre
     int size
   String contentType
```

```
    Boolean estado = false
     int ancho
     int alto
    byte[] data
    Date fechaCreacion = new Date()

          Blog blog
          static belongsTo = Blog

   static constraints = {
           data(maxSize: MAX_SIZE)
          nombre(nullable:false, blank:false, length:1..50)
          size( nullable: false )
          contentType(inList:['jpg', 'jpeg', 'png', 'gif','bmp'], length:1..20 )
    }

   static mapping = {
     data type: 'binary'
          }

          static final Integer MAX_SIZE = 5 * 1024 * 1024

  def getDownloadName() {
   return "${nombre}.${extension}"
          }

}
```

III.1.4.1.30. Clase LogoData

```
class LogoData {
     private static final int ONE_MEG_IN_BYTES = 1024*1024
     Logo logo
     static belongsTo = [logo:Logo]
     byte[] data
       static constraints = {
         data( nullable: false, minSize: 1, maxSize:
ONE_MEG_IN_BYTES )
                              }
          }
```

III.1.4.1.31. Clase Afiliado

```
class Afiliado extends Persona {
      Date fechaAfiliacion = new Date()
      String especialidad
      Colegio colegio
      static belongsTo = Colegio
      static hasMany = [ trabajosActuales: TrabajoActual,
experienciasLaborales : ExperienciaLaboral,     capacidadesPersonales :
```

```
CapacidadPersonal, formacionAcademica : FormacionAcademica,
idiomas:Idioma,ingresos:Ingreso, cargos:Cargo]

    static constraints = {
        especialidad(nullable:true)
              }
     }
```

III.1.4.1.32. Clase Muestra

```
class Muestra {
    Blog blog
    UsuarioBlog usuarioBlog
    Album album
    Imagen imagen
    Integer size
    byte[] data
    String contentType ='image/jpg'
    Integer ancho = 200
    Integer alto = 200

    static belongsTo = [ Blog, UsuarioBlog, Album, Imagen ]
    static mapping = {
      data type: 'binary'
    }

    static optionals = [ 'data' ]
    static constraints = {
       data(maxSize: MAX_SIZE)
    }

    static final Integer MAX_SIZE = 10 * 1024 * 1024

    static final Integer Original = 1
    static final Integer Large = 2
    static final Integer Medium = 3
    static final Integer Small = 4

    static final Integer[] anchos =  [ 0, 0, 500, 250, 150 ]
    static final Integer[] altos = [ 0, 0, 500, 250, 150 ]

    static final String[] Names = [ '', 'Original', 'Large', 'Medium', 'Small' ]

    String filename() {
       Muestra.filename(imagen.id, size)
          }
     }
```

III.1.4.1.33. Clase Colegio

```
class Colegio {
      String nombre
        int fax = 0
        int telefono = 0
      String email
      String pais
      String estado
      String ciudad
      String categoria
      String direccion ='no tiene'
          static hasMany = [afiliados:Afiliado,ingresos:Ingreso,
egresos:Egreso]
          static constraints = {
                  nombre(size:1..50,blank:false)
                  email(email:true,unique:true, blank:false)
                    }
     String toString ()
     {
      return nombre
           }
}
```

III.1.4.1.34. Clase PagimgData

```
class PagimgData {
 private static final int ONE_MEG_IN_BYTES = 1024*1024*1
          PaginaImagen paginaImagen
            static belongsTo = [paginaImagen:PaginaImagen]
            byte[] data
           static constraints = {
                  data( nullable: false, minSize: 1, maxSize:
ONE_MEG_IN_BYTES )
                    }
  }
```

III.1.4.1.35. Clase Pagina

```
class Pagina {
       String titulo
       String publicacion
       Boolean estado = false
       Date fechaCreacion = new Date()
       Date fechaModificacion = new Date()
        Blog blog
        UsuarioBlog usuarioBlog
        static belongsTo = Blog, UsuarioBlog
        static mapping = {
                publicacion type: 'text'
```

```
    }
      static constraints = {
            titulo(nullable:false,blank:false, length:1..50)
       publicacion(nullable:false, blank:false, length:1..5000)
      estado(nullable:false)
    }
}
```

III.1.4.1.36. Clase PaginaImagen

```
class PaginaImagen {
  String nombre
  int ancho = 100
  int alto = 100
  String posicion
  int size
  String extension
  PagimgData pagimgData
   Pagina pagina
  static belongsTo = Pagina
   static constraints = {
     pagimgData( nullable: false )
     nombre(nullable:false, blank:false, length:1..50)
     size( nullable: false )
     posicion(inList:['top', 'bottom'])
     extension(inList:['jpg', 'jpeg', 'png', 'gif','bmp'] )
              }
   def getDownloadName() {
        return "${nombre}.${extension}"
              }
   }
```

III.1.4.1.37. Clase País

```
class Pais {
  String nombre
   static hasMany = [estados:Estado]
 String toString () {
       return nombre
            }
  }
```

III.1.4.1.38. Clase Persona

```
class Persona {
      int ci = 0
    String nombres
    String apellidos
     Date fechaNacimiento = new Date()
    String sexo
```

```
    String profesion= 'estudiante'
    String pais
    String estado
    String ciudad
    boolean foto = false
      InformacionActual infAct = new InformacionActual()
     static constraints = {
    nombres(blank:false, length: 1..30)
          apellidos(blank:false, length:1..30)
    sexo(inList:['F','M'])
               pais(nullable:true)
    estado(nullable:true)
    ciudad(nullable:true)
                    }
    String toString () {
     return nombres
          }
}
```

III.1.4.1.39. Clase Plantilla

```
class Plantilla {
      String nombre
      String muestra
      String archivoCSS
      String archivoHTML
      String mainGSP
      Date fechaCreacion
      Date ultimaModificacion
       static hasMany = [blogs:Blog, imagenestpl: ImagenPTL]

       static constraints = {
              nombre(size:5..50, blank:false, unique:true)
                   }
         }
```

III.1.4.1.40. Clase Respuesta

```
class Respuesta {
      String opcionRespuesta
      int contVotos = 0
      Encuesta encuesta
      static belongsTo =  Encuesta
     static constraints = {
          opcionRespuesta(nullable:false, blank:false, length:1..50)
            }
  }
```

III.1.4.1.41. Clase Rol

```
class Rol {
      static hasMany=[ usuarios:Usuario ]
       /** descripcion */
      String descripcion
      /** ROLE String */
     String nombreRol="ROLE_"
     static def constraints = {
                nombreRol(blank:false)
                descripcion()
               }
  }
```

III.1.4.1.42. Clase TrabajoActual

```
class TrabajoActual {
        String tipoOrg
     String nombreOrg
     String cargo
     Date fechaContratacion = new Date()
     String direccion
      int telefono = 0
     Miembro miembro
     static belongsTo = Miembro
        static constraints = {
              tipoOrg(inList:['Ninguna', 'Institucion Publica', 'Institucion
Privada',
                        'ONG', 'Empresa', 'Consultora'])
                nombreOrg(length:1..30)
                          }
            }
```

III.1.4.1.43. Clase Usuario

```
class Usuario {
     Persona persona
     Rol roles
     static transients = ["pass"]
     static hasMany=[ roles:Rol, persona:Persona ]
     static belongsTo = [roles:Rol]
     /** Username */
     String idUsuario
     /** MD5 Password */
     String clave
     /** enabled */
     boolean estado = false
     Date ultimoAcceso
     /** plain password to create a MD5 password */
     String pass="[secret]"
```

```
    static def constraints = {
            idUsuario(blank:false,unique:true)
            clave(blank:false)
            estado()
    }
}
```

III.1.4.1.44. Clase UsuarioBlog

```
class UsuarioBlog extends Usuario {
      Blog blog
    Miembro miembro
    int contEntradasPub = 0
    int contEnlacesPub = 0
    int contAlbumPub = 0
    int contImgPub = 0
    String genPw='nn'
    static belongsTo =[ g:Blog, miembro:Miembro, roles:Rol ]
    Date fechaRegistracion
    static hasMany = [entradas:Entrada, enlaces:Enlace, albums:Album,
        respComment:AskComment, paginas:Pagina, archivos:Archivo,
imagenes:Imagen ]
   }
```

III.1.4.2. Controladores

III.1.4.2.1. Controlador Album

```
class AlbumController {

   def listBlog = {
     if(session.usuario)
         { makeAlbumList()
            } else {
        flash.warning ="Usted necesita iniciar sesion"
         redirect(uri:"/") }
           }

   def ver = {
    def album = Album.get(params.id)
    def blogInstance = album.blog
    def imagenes = Muestra.findAllByBlogAndAlbum( blogInstance,
album )
    def logo = Logo.findByBlogAndEstado(blogInstance,true)
    def categorias = CategoriaEntrada.findAllByBlog(blogInstance)
    def pagina = Pagina.findAllByBlogAndEstado(blogInstance, true)
    def catLink = CategoriaEnlace.findAllByBlog(blogInstance)
    def link = Enlace.findAllByBlogAndEstado(blogInstance, true)
    def enc = Encuesta.findAllByBlogAndEstado(blogInstance, true)

         return [ blogInstance : blogInstance,
catList:categorias,pagList:pagina, imagenes:imagenes,
        catLink:catLink, link: link, enc: enc, logo:logo  ]
           }

    def makeAlbumImagenList(Album album) {
          prepareList()
          params.max = null
         Imagen.findAllByAlbum(album, params)
        }

   def makeAlbumList() {
     prepareList()
    def user = UsuarioBlog.get(session.usuario.id)
    def blog = Blog.get(user.g.id )
    def rol = Rol.get( user.roles.id )
     if (rol.nombreRol == 'ROLE_ADMINBLOG') {
        params.max = null
        def list = Album.findAllByBlog(blog)
        return [ list: list, paginateCount: list.size() ]
     } else {
              params.max = null
              def listUser = Album.findAllByBlogAndUsuarioBlog(blog,
user)
```

```
            return [ list: listUser, paginateCount: listUser.size() ]
        }
}


def delete = {
   if (session.usuario){
           def album = Album.get(params.id)
           def user = album?.usuarioBlog
           def blog = album?.blog
           def imgs = Imagen.findAllByBlogAndAlbum( blog, album )
          if (imgs){
             if (user.contAlbumPub > 0) {
                 user.contAlbumPub--
                      }
              imgs.each { image ->
                if  (user.contImgPub > 0) {
                   user.contImgPub--
                   def muestra = Muestra.findByImagen ( image )
                       muestra.delete(flush:true)
                       image.delete(flush:true)
                                       }
                                  }
                    album.delete()
          flash.message = "Album ${params.nombre} borrado"
          redirect(action: listBlog)
           } else {
      if (user.contAlbumPub > 0) {
                user.contAlbumPub--
                                    }
             album.delete()
            flash.message = "Album ${params.nombre} borrado"
             redirect(action: listBlog)
                     }
              }else{
                  redirect(uri:"/") }
          }

def create = {
  if(session.usuario){
      def album = new Album()
      album.properties = params
      [ album: album ]
   }else{ redirect(uri:"/")   }
}
def save = {
     if(session.usuario) {
          def albumInstance = new Album(params)
           albumInstance.blog = session.usuario.g
          albumInstance.usuarioBlog = session.usuario
       if (!albumInstance.hasErrors()) {
```

```
                def user = UsuarioBlog.get( session.usuario.id )
                user.contAlbumPub++
                if (albumInstance.save()) {
                        flash.message = "Album  ${albumInstance.nombre}
creado"
                        redirect(controller:'imagen', action:'cargarimgs',
params:[albumId: albumInstance.id])
                    } else {
                        render(view: 'create', model: [ albumInstance:
albumInstance ])                        }
            } else {
               render(view: 'create', model: [ albumInstance: albumInstance ])
              }
           }else{
             redirect(uri:"/")  }
    }

}
```

III.1.4.2.2. Controlador Archivo

```
import org.springframework.web.multipart.MultipartFile

class ArchivoController {

    def index = { redirect(action:list,params:params) }
   // the delete, save and update actions only accept POST requests
   static allowedMethods = [save:'POST', update:'POST']

   def list = {
         if (session.usuario){
                def bp = Blog.get(session.usuario?.blog?.id)
                def user = UsuarioBlog.get(session.usuario.id)
                def rol = Rol.get(user.roles.id)
                if (rol.nombreRol == 'ROLE_ADMINBLOG' )
                  {
                  def archivos = Archivo.findAllByBlog(bp)
                  params.max = Math.min( params.max ?
params.max.toInteger() : 10,  100)
                   [ archivos: archivos, archivoInstanceTotal:
Archivo.countByBlog(bp) ]
                 } else {
                 def archivosUser =
Archivo.findAllByBlogAndUsuarioBlog(bp, user)
                   params.max = Math.min( params.max ?
params.max.toInteger() : 10,  100)
                   [ archivos: archivosUser, archivoInstanceTotal:
Archivo.countByBlog(bp) ]
                          }
                 }else{ redirect(uri:"/") }
```

```
    }

    def publicados = {
        def blogInstance = Blog.findByIdentificador( params.bp )
         if(!blogInstance) {
             flash.message = "Blog  no encontrado"
              redirect(action:list)
           } else {
       def archivos = Archivo.findAllByBlog( blogInstance)
       def logo = Logo.findByBlogAndEstado(blogInstance,true)
       def categorias = CategoriaEntrada.findAllByBlog(blogInstance)
       def pagina = Pagina.findAllByBlogAndEstado(blogInstance, true)
       def entrada = Entrada.findAllByBlogAndEstado(blogInstance, true)
       def catLink = CategoriaEnlace.findAllByBlog(blogInstance)
       def link = Enlace.findAllByBlogAndEstado(blogInstance, true)
       def enc = Encuesta.findAllByBlogAndEstado(blogInstance, true)

            return [ blogInstance : blogInstance,archivos:archivos,
catList:categorias,pagList:pagina, entrada: entrada,
                    catLink:catLink, link: link, enc: enc, logo:logo  ]
                }
    }

def delete = {
            if (session.usuario) {
        def archivoInstance = Archivo.get( params.id )
        if(archivoInstance) {
            try {
                archivoInstance.delete(flush:true)
                flash.message = "Archivo eliminado "
                redirect(action:list)
            }
            catch(org.springframework.dao.DataIntegrityViolationException
e) {
                flash.message = "Archivo no pudo ser eliminado"
                redirect(action:show,id:params.id)
            }
                } else {
            flash.message = "Archivo no encontrado"
            redirect(action:list)
          }
            } else {
            redirect(uri:"/") }
    }



    def create = {
            if (session.usuario){
        return [ archivo: new Archivo() ]
             }else{
```

```
            redirect(uri:"/") }
    }

   def save = {
            if ( session.usuario ) {
       def archivo = new Archivo( params )
        archivo.usuarioBlog = session.usuario
            archivo.blog = session.usuario.g
       MultipartFile f = request.getFile( 'archivoData.data' )
       archivo.size = f.getSize() / 1024
       archivo.extension = extractExtension( f )
       archivo.fechaCreacion = new Date()
            archivo.ultimaModificacion = new Date()
       if(archivo.save()) {
          flash.message = "Archivo [${archivo.nombre}] ha sido subido
exitosamente."
           redirect(action: 'list')
              } else {
          render(view: 'create', model: [archivo: archivo])
                 }
            } else {
           redirect(uri:"/") }
         }

             def extractExtension( MultipartFile archivo ) {
           String nombrearchivo = archivo.getOriginalFilename()
      return nombrearchivo.substring(nombrearchivo.lastIndexOf( "." ) + 1)
       }

  def  bajar = {
   def archivo = Archivo.get(params.id)
   response.setContentType( "application-xdownload")
   response.setHeader("Content-Disposition", "attachment;
filename=${archivo.downloadName}")
response.getOutputStream() << new ByteArrayInputStream( archivo.
archivoData.data )
            }
}
```

III.1.4.2.3. Controlador AskComment

```
class AskCommentController {

    def index = { redirect(action:list,params:params) }

    // the delete, save and update actions only accept POST requests
    static allowedMethods = [save:'POST', update:'POST']

    def list = {
            if (session.usuario){
```

```
        params.max = Math.min( params.max ? params.max.toInteger() : 10,
100)
        [ askCommentInstanceList: AskComment.list( params ),
askCommentInstanceTotal: AskComment.count() ]
            } else{
            flash.waring ="Necesita estar logueado"
            redirect(uri:"/")}
    }

    def edit = {
            if(session.usuario){
        def com = Comentario.get( params.comId )
        def ask = AskComment.findByComment(com )
        if(!ask) {
            flash.message = "Repuesta no encontrada"
            redirect(controller:'comentario', action:'list', id:com.entrada.id)
        }
        else {
            return [ askCommentInstance : ask ]
             }
            } else{
            flash.waring ="Necesita estar logueado"
            redirect(uri:"/")}
    }

    def update = {
     if (session.usuario){
        def ask = AskComment.get( params.id )
            def comment = Comentario.get(ask.comment.id)
        if(ask) {
            if(params.version) {
                def version = params.version.toLong()
                if(ask.version > version) {

                    ask.errors.rejectValue("version",
"askComment.optimistic.locking.failure", "Otro usuario modificaba esta
respuesta mientras estabas editando.")
                    render(view:'edit',model:[askCommentInstance:ask])
                    return
                }
            }
            ask.properties = params
            if(!ask.hasErrors() && ask.save()) {
                flash.message = "Comentario actualizado"
                        redirect(controller:'comentario', action:'list',
id:comment.entrada.id)
            }
            else {

render(view:'edit',model:[askCommentInstance:askCommentInstance])
            }
```

```
        }
        else {
            flash.message = "Respuesta no encontrada"
             redirect(controller:'comentario', action:'list',
id:comment.entrada.id)
                    }
            } else{
            flash.waring ="Necesita estar logueado"
            redirect(uri:"/") }
    }

     def create = {
            if (session.usuario) {
            def comment = Comentario.get(params.comentario.id)
          render(view:'create',
                model:[ askCommentInstance : new AskComment(),
                      'comentarioId': params.comentario.id, comment:comment])
              } else {
                redirect (uri:"/")
                  }
            }

    def save = {
            if (session.usuario ) {
                  def askCommentInstance = new AskComment()
                  askCommentInstance.properties = params
                  askCommentInstance.usuBlog = session.usuario
                  def comm = Comentario.get(
askCommentInstance.comment.id )
                  askCommentInstance.entrada = comm.entrada
                  askCommentInstance.fechaCreacion = new Date()

                   if(!askCommentInstance.hasErrors() &&
askCommentInstance.save()) {
                            def com =
Comentario.get(askCommentInstance.comment.id)
                                com.askComment = askCommentInstance.id
                        flash.message = "Comentario respondido"
                        redirect(controller:'comentario', action:'list',
id:com.entrada.id)
                    } else {

render(view:'create',model:[askCommentInstance:askCommentInstance])
                            }
            } else {
              redirect (uri:"/")
            }
      }
}
```

III.1.4.2.4. Controlador Blog

```
class BlogController {

   def index = { redirect(action:show,params:params)          }

  def show = {

     def blogInstance = Blog.findByIdentificador( params.idbp )
      if(!blogInstance) {
         flash.message = "Blog no encontrado"
         redirect(uri:"/")
       }
       else {
          if (blogInstance.estado == true){
               def now = new Date()
               def lastWeek = now - 7
               def logo = Logo.findByBlogAndEstado(blogInstance, true)
               def categorias =
CategoriaEntrada.findAllByBlog(blogInstance)
               def pagina = Pagina.findAllByBlogAndEstado(blogInstance,
true)
               def entradas =
Entrada.findAllByBlogAndFechaModificacionBetween(blogInstance,last
Week, now )
               def catLink = CategoriaEnlace.findAllByBlog(blogInstance)
               def link = Enlace.findAllByBlogAndEstado(blogInstance,
true)
               def enc = Encuesta.findAllByBlogAndEstado(blogInstance,
true)
           return [ blogInstance : blogInstance,
catList:categorias,pagList:pagina, entradas: entradas,
                    catLink:catLink, link: link, enc: enc, logo:logo  ]
               } else {
                        flash.warning = "El blog que precisa ver fue
deshabilitado"
               redirect(uri:"/")
               }
       }
    }

   def searchAJAX = {
     def blogs = Blog.findAllByNombreLike("%${params.query}%")

       //Create XML response
      render(contentType: "text/xml") {
           results() {
               blogs.each { blog ->
                  result(){
                  name(blog.nombre)
      //Optional id which will be available in onItemSelect
```

```
                    id(blog.identificador)
                      }
                    }
                  }
                }
              }
            }
          }
```

III.1.4.2.5. Controlador CapacidadPersonal

```
class CapacidadPersonalController {

   def beforeInterceptor = [action:this.&auth]

   def auth() {
         def user = session.usuario
         if(!user) {
          flash.warning ="Usted necesita iniciar sesion"
              redirect(uri:"/")
               return false
      }
   }

   def index = { redirect(action:list,params:params) }

   // the delete, save and update actions only accept POST requests
   static allowedMethods = [save:'POST', update:'POST']

   def list = {
   def userBlog = UsuarioBlog.get(session.usuario.id)
    if (userBlog){
      def ub = UsuarioBlog.get(userBlog?.id)
         def afi = Miembro.get( ub.miembro.id )
         def capacidadesPersonales =
CapacidadPersonal.findAllByMiembro(afi)
       params.max = Math.min( params.max ? params.max.toInteger() : 10,
100)
       [ capacidadPersonalInstanceList: capacidadesPersonales, capPerT:
CapacidadPersonal.countByMiembro(afi) ]
      } else {
         flash.warning ="Usted necesita ser usuario de algun Blog"
         redirect(uri:"/") }
   }

   def delete = {
    def userBlog = UsuarioBlog.get(session.usuario.id)
         if (userBlog){
           def capacidadPersonalInstance = CapacidadPersonal.get(
params.id )
            if(capacidadPersonalInstance) {
```

```
                try {
                    capacidadPersonalInstance.delete(flush:true)
                    flash.message = "Capacidad Personal eliminada"
                    redirect(action:list)
                }

catch(org.springframework.dao.DataIntegrityViolationException e) {
                    flash.message = "CapacidadPersonal ${params.id} could
not be deleted"
                    redirect(action:list)
                }
            }
            else {
                flash.message = "Capacidad Personal no encontrada"
                redirect(action:list)
            }
         } else {
        flash.warning ="Usted necesita ser usuario de algun Blog"
        redirect(uri:"/") }
    }

    def edit = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
             def capacidadPersonalInstance = CapacidadPersonal.get(
params.id )
              if(!capacidadPersonalInstance) {
             flash.message = "Capacidad Personal no encontrada"
             redirect(action:list)
              } else {
                 return [ capacidadPersonalInstance :
capacidadPersonalInstance ]
        }
         } else {
        flash.warning ="Usted necesita ser usuario de algun Blog"
        redirect(uri:"/") }
    }

    def update = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
            def capacidadPersonalInstance = CapacidadPersonal.get(
params.id )
             if(capacidadPersonalInstance) {
                  if(params.version) {
                     def version = params.version.toLong()
                     if(capacidadPersonalInstance.version > version) {
               capacidadPersonalInstance.errors.rejectValue("version", "otro
usuario estubo editando")
                redirect(action:list)
                return
```

```
                }
            }
                capacidadPersonalInstance.properties = params
                if(!capacidadPersonalInstance.hasErrors() &&
capacidadPersonalInstance.save()) {
                    flash.message = "Capacidad Personal actualizada"
                    redirect(action:list)
                } else {
                    redirect(action:list)
                }
        } else {
         flash.message = "Capacidad Personal no encontrada"
         redirect(action:list)
          }
         } else {
         flash.warning ="Usted necesita ser usuario de algun Blog"
         redirect(uri:"/") }
   }

   def create = {
         def userBlog = UsuarioBlog.get(session.usuario.id)
         if (userBlog){
       def capacidadPersonalInstance = new CapacidadPersonal()
       capacidadPersonalInstance.properties = params
       return ['capacidadPersonalInstance':capacidadPersonalInstance]
         } else {
         flash.warning ="Usted necesita ser usuario de algun Blog"
         redirect(uri:"/") }
   }

   def save = {
         def userBlog = UsuarioBlog.get(session.usuario.id)
         if (userBlog){
          def ub = UsuarioBlog.get(session.usuario?.id)
      def capacidadPersonalInstance = new CapacidadPersonal(params)
         capacidadPersonalInstance.miembro = Miembro.get(
ub.miembro.id )
      if(!capacidadPersonalInstance.hasErrors() &&
capacidadPersonalInstance.save()) {
         flash.message = "Capacidad Personal creada"
         redirect(action:list)
        } else {

render(view:'create',model:[capacidadPersonalInstance:capacidadPersonal
Instance])
         }
          } else {
         flash.warning ="Usted necesita ser usuario de algun Blog"
         redirect(uri:"/") }
   }
}
```

III.1.4.2.6. Controlador Cargo

```
class CargoController {

   def beforeInterceptor = [action:this.&auth]

   def auth() {
         def user = session.usuario
         if(!user) {
          flash.warning ="Usted necesita iniciar sesion"
              redirect(uri:"/")
               return false
      }
   }

   // the delete, save and update actions only accept POST requests
   static allowedMethods = [save:'POST', update:'POST']

   def delete = {
   def userBlog = UsuarioBlog.get(session.usuario.id)
    if (userBlog){
                def cargoInstance = Cargo.get( params.id )
                if(cargoInstance) {
                   try {
                      cargoInstance.delete(flush:true)
                      flash.message = "Cargo eliminado"
                      redirect(controller:'directorio', action:'show',
id:cargoInstance.directorio.id)
                   }

catch(org.springframework.dao.DataIntegrityViolationException e) {
                      flash.message = "Cargo no pudo ser eliminado"
                      redirect(controller:'directorio', action:'show',
id:cargoInstance.directorio.id)
                   }
                } else {
                   flash.message = "Cargo no encontrado"
                   redirect(controller:'directorio', action:'show',
id:cargoInstance.directorio.id)
         }
         } else {
         flash.warning ="Usted necesita ser usuario de algun Blog"
         redirect(uri:"/") }
   }

   def edit = {
         def userBlog = UsuarioBlog.get(session.usuario.id)
    if (userBlog){
                def cargoInstance = Cargo.get( params.id )
```

```
                def blog = Blog.get(session.usuario?.g?.id)
                def asignados = Miembro.findAllByN( blog.organizacion )
                 if(!cargoInstance) {
                    flash.message = "Cargo not found with id ${params.id}"
                  redirect(controller:'directorio', action:'show',
id:cargoInstance.directorio.id )
                 } else {
            return [ cargoInstance : cargoInstance, asignados:asignados  ]
          }
          } else {
          flash.warning ="Usted necesita ser usuario de algun Blog"
          redirect(uri:"/") }
    }

    def update = {
     def userBlog = UsuarioBlog.get(session.usuario.id)
      if (userBlog){
                def cargoInstance = Cargo.get( params.id )
                if(cargoInstance) {
                   if(params.version) {
                      def version = params.version.toLong()
                      if(cargoInstance.version > version) {
                    cargoInstance.errors.rejectValue("version",
"cargo.optimistic.locking.failure", "Otro usuario estuvo editando")
                render(view:'edit',model:[cargoInstance:cargoInstance])
                return
              }
          }
             cargoInstance.properties = params
          if(!cargoInstance.hasErrors() && cargoInstance.save()) {
               cargoInstance.cargo = params.cargo.toUpperCase()
             flash.message = "Cargo ${params.cargo} actualizado"
             def directorioInstance =
Directorio.get(cargoInstance.directorio.id)
               def cargosCreados = Cargo.findAllByDirectorio(
directorioInstance )
          redirect(controller:'directorio', action:'show',
id:cargoInstance.directorio.id )
          }
          else {
                def blog = Blog.get(session.usuario?.g?.id)
                def asignados = Miembro.findAllByN( blog.organizacion )
            redirect(controller:'directorio', action:'show',
id:cargoInstance.directorio.id )
          }
          }
       else {
          flash.message = "Cargo no encontrado"
         redirect(controller:'directorio', action:'show',
id:cargoInstance.directorio.id )
              }
```

```
        } else {
        flash.warning ="Usted necesita ser usuario de algun Blog"
        redirect(uri:"/") }
    }

    def create = {
      def userBlog = UsuarioBlog.get(session.usuario.id)
       if (userBlog){
                 def cargoInstance = new Cargo()
                cargoInstance.properties = params
                def blog = Blog.get(session.usuario?.g?.id)
                def dir = Directorio.get( params.directorio.id )
                def cargosCreados = Cargo.findAllByDirectorio(
dir,[order:"asc", sort:"nivelCargo"] )
                def asignados = Miembro.findAllByN( blog.organizacion )
                  return ['cargoInstance':cargoInstance, asignados:asignados]
        } else {
        flash.warning ="Usted necesita ser usuario de algun Blog"
        redirect(uri:"/") }
    }

    def save = {
     def userBlog = UsuarioBlog.get(session.usuario.id)
      if (userBlog){
          def cargoInstance = new Cargo()
          cargoInstance.properties = params
          def dir = Directorio.get(params.directorio.id)
          def nivel =
Cargo.findByNivelCargoAndDirectorio(params.nivelCargo, dir)
          if( !nivel ){
              if(!cargoInstance.hasErrors() && cargoInstance.save()) {
              cargoInstance.cargo = params.cargo.toUpperCase()
         flash.message = "Cargo ${cargoInstance.cargo} creado y
asignado"
              redirect(controller:"directorio", action:"show",
id:cargoInstance.directorio.id )
        } else {
        redirect(controller:"directorio", action:"show",
id:cargoInstance.directorio.id )
        }
        } else {
              flash.message = "Este nivel de cargo ya fue asignado a
${nivel.miembro.nombres} ${nivel.miembro.apellidos}"
          redirect(action:'create', params:['directorio.id':dir.id])
        }
        } else {
        flash.warning ="Usted necesita ser usuario de algun Blog"
        redirect(uri:"/") }
    }
}
```

III.1.4.2.7. Controlador Categoria

```
class CategoriaController {

    def beforeInterceptor = [action:this.&auth]

    def auth() {
          def user = session.usuario
          if(!user) {
           flash.warning ="Usted necesita iniciar sesion como usuario de
blog"
                redirect(uri:"/")
                 return false
               }
      }

    // the delete, save and update actions only accept POST requests
   static allowedMethods = [save:'POST', update:'POST']

   def listBlogEnlaces = {
     def userBlog = UsuarioBlog.get(session.usuario.id)
      if (userBlog){
                def bp = Blog.get(session.usuario?.g?.id)
                def user = UsuarioBlog.get(session.usuario.id)
                def rol = Rol.get(user.roles.id)

                def categorias = CategoriaEnlace.findAllByBlog(bp)
                  params.max = Math.min( params.max ?
params.max.toInteger() : 10,  100)
                  [ categoriaInstanceList:categorias , categoriaInstanceTotal:
CategoriaEnlace.countByBlog(bp), rol:rol ]
           } else {
          flash.warning ="Usted necesita ser usuario de algun Blog"
          redirect(uri:"/") }
   }

    def listBlogEntradas = {
          def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
                def bp = Blog.get(session.usuario?.g?.id)
                def user = UsuarioBlog.get(session.usuario.id)
                def rol = Rol.get(user.roles.id)
                def categorias = CategoriaEntrada.findAllByBlog(bp)
                  params.max = Math.min( params.max ?
params.max.toInteger() : 10,  100)
                  [ categoriaInstanceList:categorias , categoriaInstanceTotal:
CategoriaEntrada.countByBlog(bp), rol:rol ]
           } else {
          flash.warning ="Usted necesita ser usuario de algun Blog"
          redirect(uri:"/") }
   }
```

```
    def show = {
          if(session.usuario){
                def categoriaInstance = Categoria.get( params.id )
                if(!categoriaInstance) {
                   flash.message = "Categoria not found with id
${params.id}"
                   redirect(action:list)
                }
                else { return [ categoriaInstance : categoriaInstance ] }
          } else {
          flash.warning ="Usted necesita iniciar sesion"
          redirect(uri:"/") }
    }

    def deleteCT = {
     def userBlog = UsuarioBlog.get(session.usuario.id)
     if (userBlog){
       def categoriaInstance = CategoriaEntrada.get( params.id )
          String nombre = categoriaInstance.nombre
          if(categoriaInstance) {
            if (categoriaInstance.contador == 0) {
          try {
                categoriaInstance.delete(flush:true)
             flash.message = "Categoria ${nombre} eliminada"
             redirect(action:listBlogEntradas)
             }
         catch(org.springframework.dao.DataIntegrityViolationException e)
{
             flash.message = "Categoria ${nombre} no pudo ser eliminado"
             redirect(action:listBlogEntradas)
          }
            } else {
                flash.message = "Esta categoria no puede ser borrada porque
tiene entradas publicadas"
                 redirect(action:listBlogEntradas)
               }
       } else {
          flash.message = "Categoria no encontrada"
          redirect(action:listBlogEntradas)
              }
           } else {
          flash.warning ="Usted necesita ser usuario de algun Blog"
          redirect(uri:"/") }
    }

    def deleteCL = {
      def userBlog = UsuarioBlog.get(session.usuario.id)
      if (userBlog){
        def categoriaInstance = CategoriaEnlace.get( params.id )
          String nombre = categoriaInstance.nombre
```

```
            if(categoriaInstance) {
              if (categoriaInstance.contador == 0) {
            try {
                  categoriaInstance.delete(flush:true)
               flash.message = "Categoria ${nombre} eliminada"
               redirect(action:listBlogEnlaces)
               }
            catch(org.springframework.dao.DataIntegrityViolationException
e) {
               flash.message = "Categoria ${nombre} no pudo ser eliminado"
               redirect(action:listBlogEnlaces)
            }
              } else {
                  flash.message = "Esta categoria no puede ser borrada porque
tiene enlaces publicados"
                   redirect(action:listBlogEnlaces)
                  }
          } else {
            flash.message = "Categoria no encontrada"
            redirect(action:listBlogEnlaces)
             }
             } else {
            flash.warning ="Usted necesita ser usuario de algun Blog"
            redirect(uri:"/") }
      }

      def editCL = {
       def userBlog = UsuarioBlog.get(session.usuario.id)
       if (userBlog){
                  def categoriaInstance = CategoriaEnlace.get( params.id )
                   return [ categoriaInstance : categoriaInstance ]
            } else {
            redirect(uri:"/") }
      }

      def editCT = {
       def userBlog = UsuarioBlog.get(session.usuario.id)
       if (userBlog){
                  def categoriaInstance = CategoriaEntrada.get( params.id )
                      return [ categoriaInstance : categoriaInstance ]
             } else {
            flash.warning ="Usted necesita ser usuario de algun Blog"
            redirect(uri:"/") }
      }

      def updateCL = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
          def categoriaInstance = CategoriaEnlace.get( params.id )
          if(categoriaInstance) {
            if(params.version) {
```

```
            def version = params.version.toLong()
            if(categoriaInstance.version > version) {
                categoriaInstance.errors.rejectValue("version",
"categoria.optimistic.locking.failure", "Otro usuario estubo actualizando
esta categoria mientras usted estaba editando")

render(view:'edit',model:[categoriaInstance:categoriaInstance])
                return
            }
        }
        categoriaInstance.properties = params
        if(!categoriaInstance.hasErrors() && categoriaInstance.save()) {
            flash.message = "Categoria ${params.nombre} actualizada"
                redirect(action:listBlogEnlaces)
        } else {
            redirect(action:listBlogEnlaces)
        }
      } else {
        flash.message = "Categoria not found with id ${params.id}"
        redirect(action:list)
       }
    } else {
        flash.warning ="Usted necesita ser usuario de algun Blog"
        redirect(uri:"/") }
  }

  def updateCT = {
   def userBlog = UsuarioBlog.get(session.usuario.id)
   if (userBlog){
    def categoriaInstance = CategoriaEntrada.get( params.id )
    if(categoriaInstance) {
        if(params.version) {
            def version = params.version.toLong()
            if(categoriaInstance.version > version) {
                categoriaInstance.errors.rejectValue("version",
"categoria.optimistic.locking.failure", "Otro usuario estubo actualizando
esta categoria mientras usted estaba editando")
                 redirect(action:listBlogEntradas)
                return
            }
        }
        categoriaInstance.properties = params
        if(!categoriaInstance.hasErrors() && categoriaInstance.save()) {
            flash.message = "Categoria ${params.nombre} actualizada"
                redirect(action:listBlogEntradas)
        } else {

render(view:'editCT',model:[categoriaInstance:categoriaInstance])
        }
      } else {
        flash.message = "Categoria no encontrada"
```

```
            redirect(action:listBlogEntradas)
                }
        } else {
            flash.warning ="Usted necesita ser usuario de algun Blog"
            redirect(uri:"/") }
    }

    def createCatEntrada = {
      def userBlog = UsuarioBlog.get(session.usuario.id)
      if (userBlog){
                def bp = Blog.get(session.usuario?.g?.id)
                def user = UsuarioBlog.get(session.usuario.id)
                def rol = Rol.get(user.roles.id)

                  def categoriaInstance = new CategoriaEntrada()
                  categoriaInstance.properties = params
                  return ['categoriaInstance':categoriaInstance, 'rol':rol]
           } else {
            flash.warning ="Usted necesita ser usuario de algun Blog"
            redirect(uri:"/") }
    }

    def createCatEnlace = {
     def userBlog = UsuarioBlog.get(session.usuario.id)
      if (userBlog){
                def bp = Blog.get(session.usuario?.g?.id)
                def user = UsuarioBlog.get(session.usuario.id)
                def rol = Rol.get(user.roles.id)

                  def categoriaInstance = new CategoriaEnlace()
                  categoriaInstance.properties = params
                  return [ 'categoriaInstance':categoriaInstance, 'rol':rol ]
           } else {
            flash.warning ="Usted necesita ser usuario de algun Blog"
            redirect(uri:"/") }
    }

     def saveCatEntradas = {
      def userBlog = UsuarioBlog.get(session.usuario.id)
      if (userBlog){
                  def categoriaInstance = new CategoriaEntrada()
                categoriaInstance.properties = params
                categoriaInstance.blog = Blog.get(session.usuario?.g?.id)
                categoriaInstance.nombreBase =
categoriaInstance.nombre.toLowerCase()
                if(!categoriaInstance.hasErrors() && categoriaInstance.save())
{
                                flash.message = "Categoria
${categoriaInstance.nombre} creada"
                    redirect(action:listBlogEntradas)
            } else {
```

```
render(view:'createCatEntrada',model:[categoriaInstance:categoriaInstanc
e])
            }
             } else {
            flash.warning ="Usted necesita ser usuario de algun Blog"
            redirect(uri:"/") }
    }

    def saveCatEnlaces = {
      def userBlog = UsuarioBlog.get(session.usuario.id)
       if (userBlog){
                  def categoriaInstance = new CategoriaEnlace()
                categoriaInstance.properties = params
                categoriaInstance.blog = Blog.get(session.usuario?.g?.id)
                categoriaInstance.nombreBase =
categoriaInstance.nombre.toLowerCase()
                if(!categoriaInstance.hasErrors() && categoriaInstance.save())
{
                    flash.message = "Categoria ${categoriaInstance.nombre}
creada"
                    redirect(action:listBlogEnlaces)
                          } else {

render(view:'createCatEnlace',model:[categoriaInstance:categoriaInstance]
)
                }
             } else {
            flash.warning ="Usted necesita ser usuario de algun Blog"
            redirect(uri:"/") }
    }

}
```

III.1.4.2.8. Controlador Comentario

```
class ComentarioController {

    def beforeInterceptor = [action:this.&auth, except:["create", "save"]]

    def auth() {
          def user = session.usuario
          if(!user) {
           flash.warning ="Usted necesita iniciar sesion"
               redirect(uri:"/")
                return false
              }
       }

    // the delete, save and update actions only accept POST requests
```

```
    static allowedMethods = [save:'POST', update:'POST']

    def list = {
          def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
               def ent = Entrada.get( params.id )
                 def comentarios=Comentario.findAllByEntrada( ent )
                 params.max = Math.min( params.max ?
params.max.toInteger() : 10,  100)
                 [ comentarioInstanceList:comentarios, comenTotal:
Comentario.countByEntrada(ent), ent:ent ]
           } else {
          flash.warning ="Usted necesita ser usuario de algun Blog"
          redirect(uri:"/") }
    }


    def show = {
          def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
        def comentarioInstance = Comentario.get( params.id )

          if(!comentarioInstance) {
                    flash.message = "Comentario not found with id
${params.id}"
                              redirect(action:list)
          }
                else { return [ comentarioInstance : comentarioInstance ] }
           } else {
          flash.warning ="Usted necesita ser usuario de algun Blog"
          redirect(uri:"/") }
    }

    def delete = {
          def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
        def comentarioInstance = Comentario.get( params.id )
          def ent = Entrada.get (comentarioInstance.entrada.id)
        if(comentarioInstance) {
          try {
               def ask = AskComment.findByComment(comentarioInstance)
               if (ask)
               { ask.delete(flush:true) }
             comentarioInstance.delete(flush:true)
               ent.contComentarios--
             flash.message = "Comentario borrado"
             redirect(action:list, id:ent.id)
          }
          catch(org.springframework.dao.DataIntegrityViolationException
e) {
             flash.message = "Comentario no pudo ser borrado"
```

```
            redirect(action:list, id:ent.id)
          }
       } else {
          flash.message = "Comentario no encontrado"
          redirect(action:list)
                }
    } else {
          flash.warning ="Usted necesita ser usuario de algun Blog"
          redirect(uri:"/") }
   }



   def create = {

       render(view:'create',
            model:[ comentario:new Comentario(),
                 'entradaId':params.ed])

             }

   def save = {
         def comentario = new Comentario()
         comentario.properties = params
       comentario.fechaCreacion = new Date()
         comentario.entrada = Entrada.get(params.ed)
        if(!comentario.hasErrors() && comentario.save()) {
             def ent = Entrada.get(comentario.entradaId)
             ent.contComentarios++
         redirect( controller:'entrada', action:'show',
params:[titulo:comentario.entrada.titulo])
        } else {
        redirect( controller:'entrada', action:'show',
params:[titulo:comentario.entrada.titulo])
            }
         }
}
```

III.1.4.2.9. Controlador Cuenta

```
import org.springframework.web.multipart.MultipartFile
import java.awt.image.BufferedImage
import javax.imageio.ImageIO

class CuentaController {

 def beforeInterceptor = [action:this.&auth, except:["cbcrear",
"cbguardar", "mucrear", "muguardar"]]

   def auth() {
```

```
        def user = session.usuario
        if(!user) {
         flash.warning ="Usted necesita iniciar sesion "
             redirect(uri:"/")
              return false
    }
  }

  def index = { redirect(action:'cbcrear') }

        // the delete, save and update actions only accept POST requests
  static allowedMethods = [delete:'POST', save:'POST', update:'POST']

        def cbcrear = {
         def colegioInstance = new Colegio()
    colegioInstance.properties = params
    return ['colegioInstance':colegioInstance]
        }

   def cbguardar = {
        def colegioInstance = new Colegio()
        colegioInstance.properties = params
        def result = Colegio.findByEmail( colegioInstance.email )
        if (!result){
          if(!colegioInstance.hasErrors() && colegioInstance.save()) {

             String vemeil = colegioInstance.email
             int pos=0
             for (int i=0; i <= vemeil.length(); i++)
             if(String.valueOf(vemeil.charAt(i)) == "@")
                       { pos = i
                         break; }
             String    cuenta = vemeil.substring(0,pos)
             String idblog = cuenta.replace(".","_")
             def blogInstance = new Blog()
             blogInstance.identificador = idblog
             blogInstance.colegio = colegioInstance
             blogInstance.nombre = colegioInstance.nombre
             blogInstance.urlBlog = 'http://localhost:8080/nucleus/'+ idblog
             blogInstance.fechaRegistro = new Date()
             blogInstance.estado = 'D'
             def template = Plantilla.findByNombre("AnotherSquare")
             blogInstance.plantilla = template
             blogInstance.save()
             colegioInstance.blog = 'yes'
             println ">>> idblog: " + idblog

redirect(action:'mucrear',params:['n.id':colegioInstance.id,'g.id':blogInstan
ce.id])
             }else {
                       redirect( uri:"/")
```

```
                    flash.message = "Ingrese datos correctos"
            }
        }else{
            flash.warning ="Ya se registro un blog para la org :
${colegioInstance.nombre} "
            redirect(action:'cbcrear')
           }
        }


        def mucrear = {
                    def miembroInstance = new Miembro()
        miembroInstance.properties = params
             def usuarioBlogInstance = new UsuarioBlog()
        usuarioBlogInstance.properties = params
            return ['miembroInstance':miembroInstance,
'usuarioBlogInstance':usuarioBlogInstance]
        }

        def muguardar = {
            def miembroInstance = new Miembro(params)
         miembroInstance.fechaAfiliacion = new Date()
          def usuarioBlogInstance = new UsuarioBlog()
        String iduser = params.idUsuario
        def usuario = (Usuario.findByIdUsuario(iduser))
        if (usuario)
        { flash.message ="El ID Usuario que ingreso ya se encuentra
registrado elija otro opcion"

redirect(action:'mucrear',params:['n.id':params.n.id,'g.id':params.g.id])
        } else {
         if(!miembroInstance.hasErrors() && miembroInstance.save()) {
              usuarioBlogInstance.properties = params
              usuarioBlogInstance.persona = miembroInstance
              usuarioBlogInstance.miembro = miembroInstance
               usuarioBlogInstance.fechaRegistracion = new Date()
            usuarioBlogInstance.estado = true
              usuarioBlogInstance.ultimoAcceso = new Date()
              usuarioBlogInstance.clave=params.clave.encodeAsHash()
              def nrol = Rol.findByNombreRol('ROLE_ADMINBLOG')
            usuarioBlogInstance.roles = Rol.get(nrol.id)
              if (params.captcha.toUpperCase() == session.captcha){
                    usuarioBlogInstance.save()
                    def user = usuarioBlogInstance
                    session.usuario = user
                    redirect(action:'tplelegir',
params:['id':usuarioBlogInstance.g.id])
                    } else {
                      flash.warning = "El texto introducido no es el mismo
de la imagen, intente de nuevo por favor"
```

```
redirect(action:'mucrear',params:['n.id':params.n.id,'g.id':params.g.id])
                                }
        }else {
           render(view:'mucrear',model:['miembroInstance':miembroInstance,
'usuarioBlogInstance':usuarioBlogInstance])
            }
             }
            }

            def tplelegir = {
            def userBlog = UsuarioBlog.get(session.usuario.id)
            if (userBlog){
                    def blogInstance = Blog.get(params.id )
                            blogInstance.properties = params
                    def plantillas = Plantilla.list()
                    return [ 'blogInstance' : blogInstance, 'plt' : plantillas ]
        } else {
            flash.warning ="Usted necesita iniciar sesion como usuario blog"
            redirect(uri:"/") }
            }

            def tpleguardar = {
            def userBlog = UsuarioBlog.get(session.usuario.id)
            if (userBlog){
                  def blogInstance = Blog.get( session.usuario.g.id )
        if(blogInstance) {
              blogInstance.properties = params
            if(!blogInstance.hasErrors() && blogInstance.save()) {
              redirect(action:lsubir, params:['id':blogInstance.id])
            }
            else {
                 def plantillas = Plantilla.list()
                 render(view:'tplelegir',model:[blogInstance:blogInstance, plt :
plantillas ])
            }
        } else {
            flash.message = "Ocurrio un error debe sus volver al principio"
            redirect(uri:"/")
                 }
            } else {
            flash.warning ="Usted necesita iniciar sesion como usuario blog"
            redirect(uri:"/") }
        }

            def lsubir = {
            def userBlog = UsuarioBlog.get(session.usuario.id)
            if (userBlog){
                  def blogInstance = Blog.get( params.id )
                   def logoInstance = new Logo()
```

```
                return [ blogInstance : blogInstance, logoInstance:
logoInstance ]
          } else {
          flash.warning ="Usted necesita iniciar sesion como usuario blog"
          redirect(uri:"/") }
          }

          def lguardar = {
          def userBlog = UsuarioBlog.get(session.usuario.id)
           if (userBlog){
             def logo = new Logo( params )
                def multiPartFile = request.getFile('data')
                 String extension = extractExtension( multiPartFile )
   if ((extension =='jpeg') || (extension =='jpg') || (extension =='png') ||
(extension =='bmp') || (extension =='gif'))
                                    {
                                    BufferedImage original =
ImageIO.read(multiPartFile.inputStream)
                                    logo.contentType = extension.toLowerCase()
                                    logo.size = multiPartFile.getSize()
                                    logo.nombre = multiPartFile.originalFilename
                                    logo.ancho = original.width
                           logo.alto = original.height
                                    logo.save()

                                    def img = null
                                    def manImagenes = new ManImagenes()

                           img = manImagenes.createImg(logo,
multiPartFile.inputStream)
                            flash.message = "logo subida"
                         redirect(action:blogcreado)
                      } else {
                     flash.message = "Este formato no esta permitido"
                              redirect(action:blogcreado)
                       }
          } else {
          flash.warning ="Usted necesita iniciar sesion como usuario blog"
          redirect(uri:"/") }
     }

          def extractExtension( MultipartFile logoInstance ) {
         String nombrelogo = logoInstance.getOriginalFilename()
         return nombrelogo.substring(nombrelogo.lastIndexOf( "." ) + 1 )
     }

          def blogcreado = {
          def userBlog = UsuarioBlog.get(session.usuario.id)
          if (userBlog){
               def blogInstance = Blog.get( session.usuario.g.id )
               def logoInstance = Logo.findByBlog (blogInstance)
```

```
                return [ blogInstance : blogInstance, logoInstance:
logoInstance ]
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
        }
}
```

III.1.4.2.10. Controlador Cuota

```
class CuotaController {

        def beforeInterceptor = [action:this.&auth]

    def auth() {
        def user = session.usuario
        if(!user) {
         flash.warning ="Usted necesita iniciar sesion como usuario de
blog"
            redirect(uri:"/")
             return false
           }
     }

   def index = { redirect(action:list,params:params) }

   // the delete, save and update actions only accept POST requests
   static allowedMethods = [save:'POST', update:'POST']

   def list = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
     if (userBlog){
         def bp = Blog.get(session.usuario?.g?.id)
         def org = Colegio.get(bp.colegio.id)
         def cuotas = Cuota.findAllByColegio(org)
         if (cuotas){
           def user = UsuarioBlog.get(session.usuario.id)
           def rol = Rol.get(user.roles.id)
           def totalC = Cuota.executeQuery("select sum(i.importe) from
Cuota i where i.colegio.id = ? ",org.id )
           def cuotasT = Cuota.countByColegio(org)
        if (rol.nombreRol == 'ROLE_ADMINBLOG') {
            render(view:'list', model:[ cuotaInstanceList:cuotas, totalC:
totalC, cuotasT:cuotasT ] )
          } else {
            render(view:'listUser', model:[ cuotaInstanceList:cuotas,
totalC: totalC, cuotasT:cuotasT ] )
            }
        } else  {
            redirect(action:'create')
```

```
                }
            } else {
            flash.warning ="Usted necesita ser usuario de algun Blog"
            redirect(uri:"/") }
        }

    def delete = {
      def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
        def cuotaInstance = Cuota.get( params.id )
        if(cuotaInstance) {
            try {
                cuotaInstance.delete(flush:true)
                flash.message = "Ingreso interno eliminado"
                redirect(action:list)
            }
            catch(org.springframework.dao.DataIntegrityViolationException
e) {
                flash.message = "No se pudo eliminar este ingreso"
                redirect(action:'list')
            }
        }
        else {
            flash.message = "Ingreso interno no encontrado"
            redirect(action:'list')
            }
        } else {
            flash.warning ="Usted necesita ser usuario de algun Blog"
            redirect(uri:"/") }
    }

    def edit = {
          def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
         def cuotaInstance = Cuota.get( params.id )
            if(!cuotaInstance) {
            flash.message = "Cuota no encontrada"
            redirect(action:'list')
          } else {
                return ['cuotaInstance':cuotaInstance]
            }
            } else {
            flash.warning ="Usted necesita ser usuario de algun Blog"
            redirect(uri:"/") }
    }

    def update = {
          def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
        def cuotaInstance = Cuota.get( params.id )
        if(cuotaInstance) {
```

```
            if(params.version) {
                def version = params.version.toLong()
                if(cuotaInstance.version > version) {

                    cuotaInstance.errors.rejectValue("version",
"cuota.optimistic.locking.failure", "Otro usuario actualizo esta cuota
mientras usted estaba editando.")
                    redirect(action:list)
                    return
                }
            }
            cuotaInstance.properties = params
            if(!cuotaInstance.hasErrors() && cuotaInstance.save()) {
                flash.message = "Ingreso interno modificado"
                redirect(action:list)
            }
            else {
                render(view:'edit',model:[cuotaInstance:cuotaInstance])
            }
        } else {
            flash.message = "Ingreso no encontrado"
            redirect(action:list)
             }
             } else {
            flash.warning ="Usted necesita ser usuario de algun Blog"
            redirect(uri:"/") }
    }

    def create = {
            def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
                 def cuotaInstance = new Cuota()
            cuotaInstance.properties = params
                 def blog = Blog.get(session.usuario?.g?.id)
                 def user = UsuarioBlog.get(session.usuario.id)
                 def rol = Rol.get(user.roles.id)
                 def miembros = Miembro.findAllByN( blog.colegio )
            return ['cuotaInstance':cuotaInstance, m:miembros, 'rol':rol]
             } else {
            flash.warning ="Usted necesita ser usuario de algun Blog"
            redirect(uri:"/") }
    }

    def save = {
            def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
                 def cuotaInstance = new Cuota(params)
                def blog = Blog.get(session.usuario?.g?.id)
                def org = Colegio.get(blog.colegio.id)
                cuotaInstance.colegio = org
            if(!cuotaInstance.hasErrors() && cuotaInstance.save()) {
```

```
            flash.message = "Ingreso registrado"
            redirect(action:list)
        } else {
            redirect(action:create)
        }
         } else {
        flash.warning ="Usted necesita ser usuario de algun Blog"
        redirect(uri:"/") }
    }
}
```

III.1.4.2.11. Controlador Directorio

```
class DirectorioController {

        def beforeInterceptor = [action:this.&auth, except:["mostrar"]]

    def auth() {
        def user = session.usuario
        if(!user) {
         flash.warning ="Usted necesita iniciar sesion como usuario de
blog"
            redirect(uri:"/")
             return false
           }
      }

        def index = { redirect(action:list,params:params) }

   // the delete, save and update actions only accept POST requests
   static allowedMethods = [delete:'POST', save:'POST', update:'POST']

   def list = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
     if (userBlog){
            def bp = Blog.get(userBlog.g.id)
            def org = Colegio.get(bp.colegio.id)
            def dir = Directorio.findAllByColegio(org)
        params.max = Math.min( params.max ? params.max.toInteger() :
10,  100)
        [ directorioInstanceList:dir , dirTotal:
Directorio.countByColegio(org) ]
         } else {
        flash.warning ="Usted necesita ser usuario de algun Blog"
        redirect(uri:"/") }
   }

    def show = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
     if (userBlog){
```

```
                def directorioInstance = Directorio.get(params.id)
                def rol = Rol.get(userBlog.roles.id)
                def cargosCreados = Cargo.findAllByDirectorio(
directorioInstance,[order:"asc", sort:"nivelCargo"] )
                if (rol.nombreRol == 'ROLE_ADMINBLOG')
                  {
                    render(view:'show', model: [ miembrosDir:cargosCreados ,
mTotal:Cargo.countByDirectorio( directorioInstance ),
dir:directorioInstance ])
                  } else {
                        render(view:'showAutor', model: [
miembrosDir:cargosCreados , mTotal:Cargo.countByDirectorio(
directorioInstance ), dir:directorioInstance ])
                        }
           } else {
             flash.warning ="Usted necesita ser usuario de algun Blog"
             redirect(uri:"/") }
            }

  def edit = {
          def userBlog = UsuarioBlog.get(session.usuario.id)
       if (userBlog){
              def blog = Blog.get(userBlog.g?.id)
          def directorioInstance = Directorio.findByColegio( blog.colegio )
              def cargosCreados = Cargo.findAllByDirectorio(
directorioInstance )
               if(!directorioInstance) {
             flash.message = "Directorio no encontrado "
             redirect(action:create)
          } else {
             return [ directorioInstance : directorioInstance,
cargosCreados:cargosCreados ]
          }
           } else {
          flash.warning ="Usted necesita ser usuario de algun Blog"
          redirect(uri:"/") }
    }

    def update = {
          def userBlog = UsuarioBlog.get(session.usuario.id)
       if (userBlog){
                def directorioInstance = Directorio.get( params.id )
          if(directorioInstance) {
             if(params.version) {
                def version = params.version.toLong()
                if(directorioInstance.version > version) {
                   directorioInstance.errors.rejectValue("version",
"directorio.optimistic.locking.failure",
                         "Otro usuario estuvo actualizando mientras tu estabas
editando.")
                   redirect(action:'show', id:directorioInstance.id)
```

```
                return
            }
        }
        directorioInstance.properties = params
        if(!directorioInstance.hasErrors() && directorioInstance.save())
{
            flash.message = "La forma de Presentacion fue cambiada"
                redirect(action:'show', id:directorioInstance.id)
        }
        else {
            redirect(action:'show', id:directorioInstance.id)
        }
      }
      else {
         flash.message = "Directorio no encontrado"
        redirect(action:'show', id:directorioInstance.id)
      }
       } else {
      flash.warning ="Usted necesita ser usuario de algun Blog"
      redirect(uri:"/") }
  }

  def create = {
      def userBlog = UsuarioBlog.get(session.usuario.id)
    if (userBlog){
          def bp = Blog.get(session.usuario.g.id)
          def org = Colegio.get(bp.colegio.id)
          def dir = Directorio.findByColegio(org)
          if(dir){
          redirect(action:'show', id:dir.id)
          } else {
                def rol = Rol.get(userBlog.roles.id)
                def directorioInstance = new Directorio()
          directorioInstance.properties = params
          return ['directorioInstance':directorioInstance, 'rol':rol]
          }
       } else {
      flash.warning ="Usted necesita ser usuario de algun Blog"
      redirect(uri:"/") }
  }

  def save = {
      def userBlog = UsuarioBlog.get(session.usuario.id)
    if (userBlog){
          def blog = Blog.get(userBlog?.g.id)
          def directorioInstance = new Directorio()
          directorioInstance.properties = params
          directorioInstance.colegio = blog.colegio
      if(!directorioInstance.hasErrors() && directorioInstance.save()) {

         flash.message = "Directorio creado"
```

```
            redirect(controller:"cargo", action:"create",
params:['directorio.id':directorioInstance.id])
          }
          else {

render(view:'create',model:[directorioInstance:directorioInstance])
          }
           } else {
          flash.warning ="Usted necesita ser usuario de algun Blog"
          redirect(uri:"/") }
    }

            def mostrar = {
          def blogInstance = Blog.findByIdentificador( params.bp )
        if(!blogInstance) {
          flash.message = "Blog no encontrado"
          redirect(uri:"/")
        } else {
          def categorias = CategoriaEntrada.findAllByBlog(blogInstance)
          def logo = Logo.findByBlogAndEstado(blogInstance, true)
          def pagina = Pagina.findAllByBlogAndEstado(blogInstance, true)
          def entrada = Entrada.findAllByBlogAndEstado(blogInstance,
true)
          def catLink = CategoriaEnlace.findAllByBlog(blogInstance)
          def link = Enlace.findAllByBlogAndEstado(blogInstance, true)
          def enc = Encuesta.findAllByBlogAndEstado(blogInstance, true)
          def directorioInstance =
Directorio.findByColegio(blogInstance.colegio)

          if (!directorioInstance){
          render (view:"mostrar",model:[blogInstance : blogInstance,
catList:categorias,pagList:pagina, entrada: entrada,
                  catLink:catLink, link: link, enc: enc, logo:logo ])

          } else if (directorioInstance.formaPresentacion == 'Lista'){
              def cargosCreados = Cargo.findAllByDirectorio(
directorioInstance,[order:"asc", sort:"nivelCargo"] )
              def cargos = Cargo.countByDirectorio( directorioInstance )
             render (view:"mostrarL",model: [blogInstance : blogInstance,
catList:categorias,pagList:pagina,
               entrada: entrada, catLink:catLink, link: link, enc: enc,
directorioInstance : directorioInstance,
cargosCreados:cargosCreados, cargos:cargos, logo:logo  ])
          }else{
              def presidente =
Cargo.findByNivelCargoAndDirectorio('1',directorioInstance)
              def cargosCreados =
Cargo.findAllByDirectorioAndNivelCargoNotEqual( directorioInstance,
'1',[order:"asc", sort:"nivelCargo"] )
              def cargos = Cargo.countByDirectorio( directorioInstance )
```

```
            render (view:"mostrarA",model:[ blogInstance : blogInstance,
catList:categorias,pagList:pagina, entrada: entrada, catLink:catLink, link:
link, enc: enc, directorioInstance : directorioInstance,
cargosCreados:cargosCreados, presidente:presidente, cargos:cargos,
logo:logo ])
                }
            }
    }

}
```

III.1.4.2.12. Controlador Egreso

```
class EgresoController {

    def beforeInterceptor = [action:this.&auth]

     def auth() {
          def user = session.usuario
          if(!user) {
           flash.warning ="Usted necesita iniciar sesion como usuario de
blog"
               redirect(uri:"/")
                return false
              }
       }

    def index = { redirect(action:list,params:params) }

    // the delete, save and update actions only accept POST requests
    static allowedMethods = [save:'POST', update:'POST']

    def list = {
           def userBlog = UsuarioBlog.get(session.usuario.id)
       if (userBlog){
                 def bp = Blog.get(userBlog.g?.id)
                def org = Colegio.get(bp.colegio.id)
                def egresos = Egreso.findAllByColegio( org )
                if ( egresos ){
                 def rol = Rol.get(userBlog.roles.id)
                 def totalE = Egreso.executeQuery("select sum(e.importe)
from Egreso e where e.colegio.id = ? ",org.id)
                   def egresosT = Egreso.countByColegio(org)
                    if (rol.nombreRol == 'ROLE_ADMINBLOG') {
                       render(view:'list', model:[ egresoInstanceList:egresos,
totalE: totalE, egresosT:egresosT ] )
               } else {
                       render(view:'listUser', model:[ egresoInstanceList:egresos,
totalE: totalE, egresosT:egresosT ] )
                           }
```

```
            } else  {
                    redirect(action:'create')
             }
        } else {
        flash.warning ="Usted necesita ser usuario de algun Blog"
        redirect(uri:"/") }
    }


    def delete = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
      if (userBlog){
            def egresoInstance = Egreso.get( params.id )
        if(egresoInstance) {
           try {
              egresoInstance.delete(flush:true)
              flash.message = "Egreso eliminado"
              redirect(action:list)
           }

catch(org.springframework.dao.DataIntegrityViolationException e) {
              flash.message = "Este egreso no pudo ser eliminado"
              redirect(action:list)
           }
        }
        else {
           flash.message = "Egreso no encontrado"
           redirect(action:list)
        }
        } else {
        flash.warning ="Usted necesita ser usuario de algun Blog"
        redirect(uri:"/") }
    }

    def edit = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
      if (userBlog){
            def egresoInstance = Egreso.get( params.id )
            if(!egresoInstance) {
           flash.message = "Egreso no encontrado"
           redirect(action:list)
        }
        else {
           return [ egresoInstance : egresoInstance ]
        }
        } else {
        flash.warning ="Usted necesita ser usuario de algun Blog"
        redirect(uri:"/") }
    }

    def update = {
```

```
        def userBlog = UsuarioBlog.get(session.usuario.id)
      if (userBlog){
            def egresoInstance = Egreso.get( params.id )
        if(egresoInstance) {
          if(params.version) {
            def version = params.version.toLong()
            if(egresoInstance.version > version) {
              egresoInstance.errors.rejectValue("version",
"egreso.optimistic.locking.failure", "Otro usuario actualizo los datos de
este egreso.")
              redirect(action:list)
              return
            }
          }
          egresoInstance.properties = params
          if(!egresoInstance.hasErrors() && egresoInstance.save()) {
            flash.message = "Egreso modificado"
            redirect(action:list)
            } else {
             render(view:'edit',model:[egresoInstance:egresoInstance])
             }
            } else {
              flash.message = "Egreso no encontrado"
          redirect(action:list)
        }
        } else {
        flash.warning ="Usted necesita ser usuario de algun Blog"
        redirect(uri:"/") }
    }

    def create = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
      if (userBlog){
            def egresoInstance = new Egreso()
        egresoInstance.properties = params
            def rol = Rol.get(userBlog.roles.id)
        return ['egresoInstance':egresoInstance, 'rol':rol ]
        } else {
        flash.warning ="Usted necesita ser usuario de algun Blog"
        redirect(uri:"/") }
    }

    def save = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
      if (userBlog){
            def egresoInstance = new Egreso(params)
          def blog = Blog.get(userBlog?.g?.id)
          egresoInstance.colegio = blog.colegio
        if(!egresoInstance.hasErrors() && egresoInstance.save()) {
          flash.message = "Egreso registrado"
          redirect(action:list)
```

```
            }
            else {
               render(view:'create',model:[egresoInstance:egresoInstance])
            }
            } else {
            flash.warning ="Usted necesita ser usuario de algun Blog"
            redirect(uri:"/") }
      }
}
```

III.1.4.2.13. Controlador Encuesta

```
class EncuestaController {

    def beforeInterceptor = [action:this.&auth]

    def auth() {
          def user = session.usuario
          if(!user) {
           flash.warning ="Usted necesita iniciar sesion como usuario de
blog"
               redirect(uri:"/")
                return false
              }
     }

          def index = { redirect(action:listBlog,params:params) }

      // the delete, save and update actions only accept POST requests
      static allowedMethods = [save:'POST', update:'POST']


      def listBlog = {
            def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
               def bp = Blog.get(userBlog?.g?.id)
               def rol = Rol.get(userBlog.roles.id)

                def encuestas = Encuesta.findAllByBlog(bp)
            params.max = Math.min( params.max ? params.max.toInteger() :
10,  100)
          [ encuestaInstanceList:encuestas , encuestaInstanceTotal:
Encuesta.countByBlog(bp), rol:rol]
          } else {
          flash.warning ="Usted necesita ser usuario de algun Blog"
          redirect(uri:"/") }
      }

      def delete = {
            def userBlog = UsuarioBlog.get(session.usuario.id)
```

```
        if (userBlog){
                def encuestaInstance = Encuesta.get( params.id )
            if(encuestaInstance) {
               try {
                  encuestaInstance.delete(flush:true)
                  flash.message = "Encuesta eliminada"
                  redirect(action:listBlog)
               }

catch(org.springframework.dao.DataIntegrityViolationException e) {
                  flash.message = "Encuesta no pudo ser eliminada"
                  redirect(action:listBlog)
               }
            }
            else {
               flash.message = "Encuesta no encontrada"
               redirect(action:listBlog)
            }
            } else {
            flash.warning ="Usted necesita ser usuario de algun Blog"
            redirect(uri:"/") }
      }

      def edit = {
             def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
                def encuestaInstance = Encuesta.get( params.id )
                if(!encuestaInstance) {
               flash.message = "Encuesta no encontrada"
               redirect(action:listBlog)
            } else {
                    return [ encuestaInstance : encuestaInstance ]
            }
            } else {
            flash.warning ="Usted necesita ser usuario de algun Blog"
            redirect(uri:"/") }
      }

      def update = {
             def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
                def encuestaInstance = Encuesta.get( params.id )
            if(encuestaInstance) {
               if(params.version) {
                  def version = params.version.toLong()
                  if(encuestaInstance.version > version) {

                     encuestaInstance.errors.rejectValue("version",
"encuesta.optimistic.locking.failure", "Another user has updated this
Encuesta while you were editing.")
```

```
render(view:'edit',model:[encuestaInstance:encuestaInstance])
                    return
                }
            }
            encuestaInstance.properties = params
            if(!encuestaInstance.hasErrors() && encuestaInstance.save()) {
                flash.message = "Encuesta actualizada"
                redirect(action:listBlog)
            } else {

render(view:'edit',model:[encuestaInstance:encuestaInstance])
            }
        } else {
            flash.message = "Encuesta no encontrada"
                redirect(action:list)
        }
        } else {
        flash.warning ="Usted necesita iniciar sesion"
        redirect(uri:"/") }
    }

    def create = {
         def userBlog = UsuarioBlog.get(session.usuario.id)
      if (userBlog){
             def bp = user.g
             def rol = Rol.get(userBlog.roles.id)
        def encuestaInstance = new Encuesta()
        encuestaInstance.properties = params
        return [encuestaInstance:encuestaInstance, rol:rol]
        } else {
        flash.warning ="Usted necesita ser usuario de algun Blog"
        redirect(uri:"/") }
    }

    def save = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
      if (userBlog){
             def encuestaInstance = new Encuesta(params)
            encuestaInstance.blog = session.usuario.g
            encuestaInstance.fechaCreacion = new Date()
        if(!encuestaInstance.hasErrors() && encuestaInstance.save()) {
            flash.message = "Encuesta ${encuestaInstance.titulo} creada"
            redirect(action:'edit',params:[id:encuestaInstance.id])
        }
        else {
            render(view:'create',model:[encuestaInstance:encuestaInstance])
        }
        } else {
        flash.warning ="Usted necesita ser usuario de algun Blog"
        redirect(uri:"/") }
```

```
    }

        def deshabilitar = {
         def userBlog = UsuarioBlog.get(session.usuario.id)
       if (userBlog){
              def encuestaInstance = Encuesta.get( params.id )
          if(encuestaInstance) {
             encuestaInstance.properties = params
                encuestaInstance.estado = false
             if(!encuestaInstance.hasErrors() && encuestaInstance.save()) {
                flash.message = "La Encuesta ${encuestaInstance.titulo} fue
deshabilitada"
                redirect(action:listBlog)
             } else {
                        flash.message = "La Encuesta
${encuestaInstance.titulo} no fue deshabilitada"
                 redirect(action:listBlog)
              }
             } else {
          flash.message = "Encuesta no encontrada"
          redirect(action:listBlog)
          }
          } else {
          flash.warning ="Usted necesita ser usuario de algun Blog"
          redirect(uri:"/") }
    }

      def habilitar = {
          def userBlog = UsuarioBlog.get(session.usuario.id)
       if (userBlog){
              def encuestaInstance = Encuesta.get( params.id )
          if(encuestaInstance) {
                 encuestaInstance.properties = params
                      encuestaInstance.estado = true
             if(!encuestaInstance.hasErrors() && encuestaInstance.save()) {
                flash.message = "La Encuesta ${encuestaInstance.titulo} fue
habilitada"
                redirect(action:listBlog)
             } else {
                        flash.message = "La Encuesta
${encuestaInstance.titulo} no fue deshabilitada"
                 redirect(action:listBlog)
             }
          } else {
             flash.message = "Encuesta no encontrada"
             redirect(action:listBlog)
          }
          } else {
          flash.warning ="Usted necesita ser usuario de algun Blog"
          redirect(uri:"/") }
    }
```

```
}
```

III.1.4.2.14. Controlador Enlace

```
class EnlaceController {

    def beforeInterceptor = [action:this.&auth]

    def auth() {
          def user = session.usuario
          if(!user) {
           flash.warning ="Usted necesita iniciar sesion"
               redirect(uri:"/")
                return false
              }
      }

   def index = { redirect(action:listBlog,params:params) }

   // the delete, save and update actions only accept POST requests
   static allowedMethods = [save:'POST', update:'POST']


   def listBlog = {
          def userBlog = UsuarioBlog.get(session.usuario.id)
      if (userBlog){
               def bp = Blog.get(userBlog?.g?.id)
               def rol = Rol.get(userBlog.roles.id)
               if (rol.nombreRol == 'ROLE_ADMINBLOG' )
                {
                  def enlaces = Enlace.findAllByBlog(bp)
             params.max = Math.min( params.max ? params.max.toInteger() :
10,  100)
              [ enlaceInstanceList:enlaces , enlaceInstanceTotal:
Enlace.countByBlog(bp)]
               }else{
                 def enlacesUser = Enlace.findAllByBlogAndUsuarioBlog(bp,
userBlog)
             params.max = Math.min( params.max ? params.max.toInteger() :
10,  100)
             [ enlaceInstanceList:enlacesUser , enlaceInstanceTotal:
Enlace.countByBlogAndUsuarioBlog(bp, userBlog)]
               }
           } else {
          flash.warning ="Usted necesita ser usuario de algun Blog"
          redirect(uri:"/") }
   }

   def delete = {
```

```
        def userBlog = UsuarioBlog.get(session.usuario.id)
      if (userBlog){
            def enlaceInstance = Enlace.get( params.id )
        if(enlaceInstance) {
          try {
            enlaceInstance.delete(flush:true)
                  userBlog.contEnlacesPub--
                  def cat =
CategoriaEnlace.get(enlaceInstance.catEnlace.id)
                  cat.contador--
                  userBlog.save()
            flash.message = "Enlace ${params.nombre} borrado"
            redirect(action:listBlog)
          }

catch(org.springframework.dao.DataIntegrityViolationException e) {
            flash.message = "Enlace no pudo ser borrado"
            redirect(action:listBlog)
          }
        }
        else {
          flash.message = "Enlace no encontrado"
          redirect(action:listBlog)
        }
         } else {
        flash.warning ="Usted necesita ser usuario de algun Blog"
        redirect(uri:"/") }
    }

    def edit = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
      if (userBlog){
            def enlaceInstance = Enlace.get( params.id )
           def bp = Blog.get(session.usuario?.g?.id)
           def categorias = CategoriaEnlace.findAllByBlog(bp)
        if(!enlaceInstance) {
          flash.message = "Enlace no encontrado"
          redirect(action:list)
        } else {
          return [ enlaceInstance : enlaceInstance, catList:categorias ]
        }
         } else {
        flash.warning ="Usted necesita ser usuario de algun Blog"
        redirect(uri:"/") }
    }

    def update = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
      if (userBlog){
            def enlaceInstance = Enlace.get( params.id )
        if(enlaceInstance) {
```

```
            if(params.version) {
                def version = params.version.toLong()
                if(enlaceInstance.version > version) {
                    enlaceInstance.errors.rejectValue("version",
"enlace.optimistic.locking.failure", "Otro usuario ha actualizado este
enlace mientras tu estabas editando.")
                    redirect(action:listBlog)
                    return
                }
            }
        if (enlaceInstance.catEnlace.id != params.catEnlace.id)
                    { def catAnt =
CategoriaEnlace.get(enlaceInstance.catEnlace.id)
                        catAnt.contador--
                      def catAct = CategoriaEnlace.get(
params.catEnlace.id )
                        catAct.contador++
                    }
                            enlaceInstance.properties = params
                if(!enlaceInstance.hasErrors() && enlaceInstance.save()) {
              flash.message = "Enlace modificado"
            redirect(action:listBlog)
            } else {
                render(view:'edit',params:[id:enlaceInstance.id])
            }
        } else {
            flash.message = "No fue encontrada esta Enlace"
            redirect(action:listBlog)
        }
         } else {
        flash.warning ="Usted necesita ser usuario de algun Blog"
        redirect(uri:"/") }
   }


        def deshabilitar = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
      if (userBlog){
            def enlaceInstance = Enlace.get( params.id )
        if(enlaceInstance) {
             enlaceInstance.properties = params
                    enlaceInstance.estado = false
           if(!enlaceInstance.hasErrors() && enlaceInstance.save()) {
              flash.message = "El enlace ${enlaceInstance.nombre} fue
deshabilitado"
              redirect(action:listBlog)
            } else {
                    flash.message = "El enlace ${enlaceInstance.nombre}
no fue deshabilitado"
                redirect(action:listBlog)
            }
```

```
            } else {
                flash.message = "Enlace no encontrado"
                redirect(action:listBlog)
            }
             } else {
            flash.warning ="Usted necesita ser usuario de algun Blog"
            redirect(uri:"/") }
    }

            def habilitar = {
            def userBlog = UsuarioBlog.get(session.usuario.id)
          if (userBlog){
                    def enlaceInstance = Enlace.get( params.id )
            if(enlaceInstance) {
                    enlaceInstance.properties = params
                            enlaceInstance.estado = true
                  if(!enlaceInstance.hasErrors() && enlaceInstance.save()) {
                    flash.message = "El enlace ${enlaceInstance.nombre} fue
habilitado"
                    redirect(action:listBlog)
                  } else {
                            flash.message = "La entrada  ${enlaceInstance.nombre}
no fue deshabilitado"
                      redirect(action:listBlog)
                   }
            } else {
                flash.message = "Entrada no encontrada"
                redirect(action:listBlog)
            }
             } else {
            flash.warning ="Usted necesita ser usuario de algun Blog"
            redirect(uri:"/") }
    }

    def create = {
            def userBlog = UsuarioBlog.get(session.usuario.id)
          if (userBlog){
                    def enlaceInstance = new Enlace()
            enlaceInstance.properties = params
                 def bp = Blog.get(session.usuario?.g?.id)
                 def categorias = CategoriaEnlace.findAllByBlog(bp)
            return [ enlaceInstance:enlaceInstance,
catList:categorias,catTotal:CategoriaEnlace.countByBlog(bp) ]
             } else {
            flash.warning ="Usted necesita ser usuario de algun Blog"
            redirect(uri:"/") }
    }


    def save = {
            def userBlog = UsuarioBlog.get(session.usuario.id)
```

```
        if (userBlog){
                def enlaceInstance = new Enlace()
                enlaceInstance.properties = params
                enlaceInstance.blog = session.usuario.g
                enlaceInstance.usuarioBlog = userBlog
            if(!enlaceInstance.hasErrors() && enlaceInstance.save()) {
                        userBlog.contEnlacesPub++
                        def cat = CategoriaEnlace.get(params.catEnlace.id)
                        cat.contador++
               flash.message = "Enlace ${enlaceInstance.nombre} creado"
                redirect(action:listBlog)
            } else {
               render(view:'create',model:[enlaceInstance:enlaceInstance])
            }
             } else {
            flash.warning ="Usted necesita ser usuario de algun Blog"
            redirect(uri:"/") }
    }
}
```

III.1.4.2.15. Controlador Entrada

```
class EntradaController {

   def beforeInterceptor = [action:this.&auth, except:["buscarCatEntradas",
"show", "atom"]]

    def auth() {
          def user = session.usuario
          if(!user) {
           flash.warning ="Usted necesita iniciar sesion "
               redirect(uri:"/")
                return false
       }
    }

    // the save and update actions only accept POST requests
    static allowedMethods = [save:'POST', update:'POST']

   def listBlog = {
          def userBlog = UsuarioBlog.get(session.usuario.id)
          if (userBlog){
               def bp = Blog.get(userBlog?.g?.id)
               def rol = Rol.get(userBlog.roles.id)
               if (rol.nombreRol == 'ROLE_ADMINBLOG' )
```

```
                {
                  def entradas = Entrada.findAllByBlog(bp)
                  params.max = Math.min( params.max ?
params.max.toInteger() : 5,  100)
            [ entradaInstanceList:entradas , enTotal:
Entrada.countByBlog(bp)]
                } else {
                  def entradasUser =
Entrada.findAllByBlogAndUsuarioBlog(bp, userBlog)
                  params.max = Math.min( params.max ?
params.max.toInteger() : 5,  100)
           [ entradaInstanceList:entradasUser , enTotal:
Entrada.countByBlogAndUsuarioBlog(bp, userBlog)]
                }
          } else {
          flash.warning ="Usted necesita iniciar sesion como usuario blog"
          redirect(uri:"/") }
   }

   def edit = {
          def userBlog = UsuarioBlog.get(session.usuario.id)
          if (userBlog){
                def entradaInstance = Entrada.get( params.id )
               def entImg = EntradaImagen.findByEntrada( entradaInstance )
               def bp = Blog.get(userBlog.g?.id)
               def categorias = CategoriaEntrada.findAllByBlog(bp)
          if(!entradaInstance) {
             flash.message = "Entrada no encontrada"
             redirect(action:listBlog)
          } else {
             return [ entradaInstance : entradaInstance, 'catList':categorias,
'entImg':entImg ]
          }
          } else {
          flash.warning ="Usted necesita iniciar sesion como usuario blog"
          redirect(uri:"/") }

   }

   def update = {
          def userBlog = UsuarioBlog.get(session.usuario.id)
          if (userBlog){
                def entradaInstance = Entrada.get( params.id )
          if(entradaInstance) {
             if(params.version) {
                def version = params.version.toLong()
                if(entradaInstance.version > version) {
                  entradaInstance.errors.rejectValue("version",
"entrada.optimistic.locking.failure", "Otro usuario ha actualizado esta
entrada mientras tu estabas editando.")
                redirect(action:listBlog)
```

```
                return
            }
        }
        if (entradaInstance.catEntrada.id != params.catEntrada.id)
                { def catAnt =
CategoriaEntrada.get(entradaInstance.catEntrada.id)
                  catAnt.contador--
                 def catAct = CategoriaEntrada.get(
params.catEntrada.id )
                  catAct.contador++
                }
            entradaInstance.properties = params
          if(!entradaInstance.hasErrors() && entradaInstance.save()) {
            flash.message = "Entrada modificada"
          redirect(action:listBlog)
          } else {
             render(view:'edit',params:[id:entradaInstance.id])
          }
        } else {
          flash.message = "No fue encontrada esta Entrada"
          redirect(action:listBlog)
        }
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
    }

    def create = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
            def entradaInstance = new Entrada()
        entradaInstance.properties = params
            def bp = Blog.get(userBlog.g?.id)
            def categorias = CategoriaEntrada.findAllByBlog(bp)
        return ['entradaInstance':entradaInstance, 'catList':categorias,
catTotal:CategoriaEntrada.countByBlog(bp)]
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
    }

    def save = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
            def entradaInstance = new Entrada()
          entradaInstance.properties = params
          entradaInstance.blog = userBlog?.g
          entradaInstance.usuarioBlog = userBlog
          entradaInstance.fechaModificacion= new Date()
          entradaInstance.contComentarios = 0
```

```
                if(!entradaInstance.hasErrors() && entradaInstance.save()) {
                        userBlog.contEntradasPub++
                        def cat = CategoriaEntrada.get(params.catEntrada.id)
                        cat.contador++
                   flash.message = "Entrada ${entradaInstance.titulo} creada"
                   redirect(action:listBlog)
               } else {
                     def bp = Blog.get(userBlog?.g?.id)
                     def categorias = CategoriaEntrada.findAllByBlog(bp)
                   render(view:'create',model:[entradaInstance:entradaInstance,
'catList':categorias])
               }
               } else {
               flash.warning ="Usted necesita iniciar sesion como usuario blog"
               redirect(uri:"/") }
     }

               def deshabilitar = {
               def userBlog = UsuarioBlog.get(session.usuario.id)
               if (userBlog){
                     def entradaInstance = Entrada.get( params.id )
               if(entradaInstance) {
                    entradaInstance.properties = params
                             entradaInstance.estado = false
                     if(!entradaInstance.hasErrors() && entradaInstance.save()) {
                        flash.message = "La Entrada ${entradaInstance.titulo} fue
deshabilitada"
                        redirect(action:listBlog)
                     } else {
                                flash.message = "La Entrada
${entradaInstance.titulo} no fue deshabilitada"
                        redirect(action:listentrada)
                        }
                  } else {
                    flash.message = "Entrada no encontrada"
                    redirect(action:listBlog)
                   }
               } else {
               flash.warning ="Usted necesita ser usuario de algun blog"
               redirect(uri:"/") }
     }

                def habilitar = {
               def userBlog = UsuarioBlog.get(session.usuario.id)
               if (userBlog){
                     def entradaInstance = Entrada.get( params.id )
                     if(entradaInstance) {
                             entradaInstance.properties = params
                             entradaInstance.estado = true
                        if(!entradaInstance.hasErrors() && entradaInstance.save())
{
```

```
                    flash.message = "La Entrada ${entradaInstance.titulo}
fue habilitada"
                    redirect(action:listBlog)
                }
                else {
                        flash.message = "La entrada
${entradaInstance.titulo} no fue deshabilitada"
                    redirect(action:listBlog)
                }
            }
            else {
                flash.message = "Entrada no encontrada"
                redirect(action:listBlog)
            }
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
    }

        def delete = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
             def entradaInstance = Entrada.get( params.id )
            def coment = Comentario.countByEntrada( entradaInstance )
            if(entradaInstance) {
                    if ( coment == 0 ){
                            def entImg = EntradaImagen.findByEntrada(
entradaInstance )
                            if (entImg){ entImg.delete(flush:true) }
                    entradaInstance.delete(flush:true)
                            flash.message = "Entrada ${params.titulo}
borrada"
                            def cat =
CategoriaEntrada.get(entradaInstance.catEntrada.id)
                            cat.contador--
                            def usuBlog =
UsuarioBlog.get(entradaInstance.usuarioBlog.id)
                            usuBlog.contEntradasPub--
                    redirect(action:listBlog)
                            } else {
                             flash.message = "Para eliminar esta entrada
debe eliminar primero sus comentarios y sus
                         respectivas respuestas si las tienen"
                        redirect(action:listBlog)
                            }
            } else {
                         flash.message = "Entrada no encontrada :
${params.titulo}"
                redirect(action:listBlog)
            }
            } else {
```

```
                flash.warning ="Usted necesita iniciar sesion"
                redirect(uri:"/") }
        }

            def buscarCatEntradas = {
            def cat = CategoriaEntrada.get( params.cat )
            def blogInstance = Blog.get( cat.blog.id )
            def categorias = CategoriaEntrada.findAllByBlog(blogInstance)
            def logo = Logo.findByBlogAndEstado(blogInstance, true)
            def pagina = Pagina.findAllByBlog(blogInstance)
            def entrada = Entrada.findAllWhere(blog:blogInstance,
estado:true, catEntrada:cat)
            def catLink = CategoriaEnlace.findAllByBlog(blogInstance)
            def link = Enlace.findAllByBlogAndEstado(blogInstance, true)
            def enc = Encuesta.findAllByBlogAndEstado(blogInstance, true)

            return [ blogInstance : blogInstance,
catList:categorias,pagList:pagina, entrada: entrada,
                    catLink:catLink, link: link, enc: enc, logo: logo ]
            }

            def show = {
        def entradaInstance = Entrada.findByTitulo( params.titulo )
            def entImg = EntradaImagen.findByEntrada( entradaInstance )
            def blogInstance = Blog.get(entradaInstance.blog.id)
            def pagina = Pagina.findAllByBlogAndEstado(blogInstance, true)
            def logo = Logo.findByBlogAndEstado(blogInstance, true)
            def categorias = CategoriaEntrada.findAllByBlog(blogInstance)
            def catLink = CategoriaEnlace.findAllByBlog(blogInstance)
            def link = Enlace.findAllByBlogAndEstado(blogInstance, true)
            def enc = Encuesta.findAllByBlogAndEstado(blogInstance, true)
            def contest = AskComment.findAllByEntrada( entradaInstance )
        if(!entradaInstance) {
            flash.message = "Entrada no encontrada con id ${params.id}"
            redirect(action:list)
        }
        else { return [ entradaInstance : entradaInstance, entImg:entImg,
blogInstance: blogInstance, catList:categorias, pagList: pagina,
catLink:catLink, link: link, enc: enc, cont: contest, logo:logo ] }
    }

 def atom = {
            if(!params.max) params.max = 10
            def list = Entrada.list( params )
            def fechaModificacion = list[0].fechaModificacion
            [ entryInstanceList:list, fechaModificacion:fechaModificacion ]
 }
```

III.1.4.2.16. Controlador ExperienciaLaboral

```
class ExperienciaLaboralController {

  def beforeInterceptor = [action:this.&auth, except:["show", "mostrar"]]

   def auth() {
        def user = session.usuario
        if(!user) {
         flash.warning ="Usted necesita iniciar sesion "
             redirect(uri:"/")
              return false
     }
   }

   // the delete, save and update actions only accept POST requests
   static allowedMethods = [save:'POST', update:'POST']

   def list = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
          def afi = Miembro.get( userBlog.miembro.id )
          def experienciaLaborales =
ExperienciaLaboral.findAllByMiembro(afi)
       params.max = Math.min( params.max ? params.max.toInteger() :
10,  100)
           [ experienciaLaboralInstanceList: experienciaLaborales ,
expLabT: ExperienciaLaboral.countByMiembro(afi) ]
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
   }

  def delete = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
             def experienciaLaboralInstance = ExperienciaLaboral.get(
params.id )
        if(experienciaLaboralInstance) {
           try {
             experienciaLaboralInstance.delete(flush:true)
             flash.message = "Experiencia Laboral eliminada"
             redirect(action:list)
           }

catch(org.springframework.dao.DataIntegrityViolationException e) {
             flash.message = "Experiencia Laboral no pudo ser eliminada"
             redirect(action:list)
                 }
        } else {
                flash.message = "Experiencia Laboral no encontrada"
```

```
            redirect(action:list)
        }
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
    }

    def edit = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
            def experienciaLaboralInstance = ExperienciaLaboral.get(
params.id )
            if(!experienciaLaboralInstance) {
                flash.message = "Experiencia Laboral no encontrada con
id: ${params.id}"
                redirect(action:list)
            }
            else {
                return [ experienciaLaboralInstance :
experienciaLaboralInstance ]
            }
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
    }

    def update = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
            def experienciaLaboralInstance = ExperienciaLaboral.get(
params.id )
        if(experienciaLaboralInstance) {
            if(params.version) {
                def version = params.version.toLong()
                if(experienciaLaboralInstance.version > version) {

experienciaLaboralInstance.errors.rejectValue("version",
"experienciaLaboral.optimistic.locking.failure", "Another user has
updated this ExperienciaLaboral while you were editing.")

render(view:'edit',model:[experienciaLaboralInstance:experienciaLaboralI
nstance])
                return
                    }
                }
                experienciaLaboralInstance.properties = params
                if(!experienciaLaboralInstance.hasErrors() &&
experienciaLaboralInstance.save()) {
                    flash.message = "Experiencia Laboral actualizada"
                    redirect(action:list)
                } else {
```

```
render(view:'edit',model:[experienciaLaboralInstance:experienciaLaboralI
nstance])
                }
                 } else {
                     flash.message = "Experiencia Laboral no encontrada"
                     redirect(action:list)
            }
            } else {
            flash.warning ="Usted necesita iniciar sesion como usuario blog"
            redirect(uri:"/") }
    }

    def create = {
            def userBlog = UsuarioBlog.get(session.usuario.id)
            if (userBlog){
                  def experienciaLaboralInstance = new ExperienciaLaboral()
            experienciaLaboralInstance.properties = params
            return ['experienciaLaboralInstance':experienciaLaboralInstance]
            } else {
            flash.warning ="Usted necesita iniciar sesion como usuario blog"
            redirect(uri:"/") }
    }

    def save = {
            def userBlog = UsuarioBlog.get(session.usuario.id)
            if (userBlog){
                 def experienciaLaboralInstance = new
ExperienciaLaboral(params)
                 experienciaLaboralInstance.miembro = Miembro.get(
userBlog.miembro.id )
            if(!experienciaLaboralInstance.hasErrors() &&
experienciaLaboralInstance.save()) {
                flash.message = "Experiencia Laboral creada"
                redirect(action:'list')
            } else {

render(view:'create',model:[experienciaLaboralInstance:experienciaLabor
alInstance])
            }
            } else {
            flash.warning ="Usted necesita iniciar sesion como usuario blog"
            redirect(uri:"/") }
    }
}
```

III.1.4.2.17. Controlador FormacionAcademica

```
class FormacionAcademicaController {

  def beforeInterceptor = [action:this.&auth]

   def auth() {
        def user = session.usuario
        if(!user) {
         flash.warning ="Usted necesita iniciar sesion "
             redirect(uri:"/")
              return false
     }
   }

   def index = { redirect(action:list,params:params) }

   // the delete, save and update actions only accept POST requests
   static allowedMethods = [save:'POST', update:'POST']

   def list = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
              def afi = Miembro.get( userBlog.miembro.id )
              def forAcademicas =
FormacionAcademica.findAllByMiembro(afi)
        params.max = Math.min( params.max ? params.max.toInteger() :
10,  100)
        [ formacionAcademicaInstanceList: forAcademicas, formAcadT:
FormacionAcademica.countByMiembro(afi) ]
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
   }


   def delete = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
              def formacionAcademicaInstance =
FormacionAcademica.get( params.id )
        if(formacionAcademicaInstance) {
           try {
              formacionAcademicaInstance.delete(flush:true)
              flash.message = "Formacion Academica eliminada"
              redirect(action:list)
           }

catch(org.springframework.dao.DataIntegrityViolationException e) {
              flash.message = "Formacion Academica
${params.tituloObtenido} no puede ser borrado"
```

```
                redirect(action:list)
            }
        }
        else {
            flash.message = "FormacionAcademica no encontrado :
${params.tituloObtenido}"
            redirect(action:list)
        }
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
    }

    def edit = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
            def formacionAcademicaInstance =
FormacionAcademica.get( params.id )
                    if(!formacionAcademicaInstance) {
            flash.message = "Formacion Academica no encontrada"
            redirect(action:list)
        }
        else {
            return [ formacionAcademicaInstance :
formacionAcademicaInstance ]
        }
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
    }

    def update = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
            def formacionAcademicaInstance =
FormacionAcademica.get( params.id )
        if(formacionAcademicaInstance) {
            if(params.version) {
                def version = params.version.toLong()
                if(formacionAcademicaInstance.version > version) {

                    formacionAcademicaInstance.errors.rejectValue("version",
"formacionAcademica.optimistic.locking.failure", "Another user has
updated this FormacionAcademica while you were editing.")

render(view:'edit',model:[formacionAcademicaInstance:formacionAcade
micaInstance])
                    return
                }
            }
            formacionAcademicaInstance.properties = params
```

```
            if(!formacionAcademicaInstance.hasErrors() &&
formacionAcademicaInstance.save()) {
                flash.message = "Formacion Academica
${params.tituloObtenido} actualizada"
                redirect(action:list)
            }
            else {

render(view:'edit',model:[formacionAcademicaInstance:formacionAcade
micaInstance])
            }
        }
        else {
flash.message = "Formacion Academica:${params.tituloObtenido} no
encontrada "
            redirect(action:list)
        }
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
    }

    def create = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
            def formacionAcademicaInstance = new
FormacionAcademica()
        formacionAcademicaInstance.properties = params
        return
['formacionAcademicaInstance':formacionAcademicaInstance]
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
    }

    def save = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
            def formacionAcademicaInstance = new
FormacionAcademica(params)
            formacionAcademicaInstance.miembro = Miembro.get(
userBlog.miembro.id )
        if(!formacionAcademicaInstance.hasErrors() &&
formacionAcademicaInstance.save()) {
            flash.message = "Formacion Academica
${formacionAcademicaInstance.tituloObtenido} creada"
            redirect(action:list)
        }
        else {
```

```
render(view:'create',model:[formacionAcademicaInstance:formacionAcad
emicaInstance])
        }
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
   }
}
```

III.1.4.2.18. Controlador Idioma

```
class IdiomaController {

  def  beforeInterceptor = [action:this.&auth]

   def auth() {
        def user = session.usuario
        if(!user) {
         flash.warning ="Usted necesita iniciar sesion "
             redirect(uri:"/")
              return false
      }
   }

  def index = { redirect(action:list,params:params) }

   // the delete, save and update actions only accept POST requests
   static allowedMethods = [save:'POST', update:'POST']

   def list = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
              def afi = Miembro.get( userBlog.miembro.id )
              def idiomas = Idioma.findAllByMiembro(afi)
        params.max = Math.min( params.max ? params.max.toInteger() :
10,  100)
        [ idiomaInstanceList: idiomas, idiomaInstanceTotal:
Idioma.countByMiembro(afi) ]
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
   }

   def delete = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
              def idiomaInstance = Idioma.get( params.id )
        if(idiomaInstance) {
           try {
```

```
                idiomaInstance.delete(flush:true)
                flash.message = "Idioma ${params.nombreIdioma} eliminado"
                redirect(action:list)
                        }

catch(org.springframework.dao.DataIntegrityViolationException e) {
                flash.message = "Idioma ${params.nombreIdioma} no pudo
ser borrado"
                redirect(action:list)
            }
        }  else {
            flash.message = "Idioma no encontrado"
            redirect(action:list)
        }
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
    }

    def edit = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
            def idiomaInstance = Idioma.get( params.id )
            if(!idiomaInstance) {
            flash.message = "Idioma no encontrado"
            redirect(action:list)
        }
        else {
            return [ idiomaInstance : idiomaInstance ]
        }
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
    }

    def update = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
            def idiomaInstance = Idioma.get( params.id )
        if(idiomaInstance) {
            if(params.version) {
                    def version = params.version.toLong()
                if(idiomaInstance.version > version) {

                    idiomaInstance.errors.rejectValue("version",
"idioma.optimistic.locking.failure", "Another user has updated this Idioma
while you were editing.")
                    render(view:'edit',model:[idiomaInstance:idiomaInstance])
                    return
                }
            }
```

```
            idiomaInstance.properties = params
            if(!idiomaInstance.hasErrors() && idiomaInstance.save()) {
               flash.message = "Idioma ${params.nombreIdioma}
actualizada"
               redirect(action:list)
            }
            else {
               render(view:'edit',model:[idiomaInstance:idiomaInstance])
            }
         }
         else {
            flash.message = "Idioma ${params.nombreIdioma} no
encontrado "
            redirect(action:list)
         }
         } else {
         flash.warning ="Usted necesita iniciar sesion como usuario blog"
         redirect(uri:"/") }
   }

   def create = {
         def userBlog = UsuarioBlog.get(session.usuario.id)
         if (userBlog){
               def idiomaInstance = new Idioma()
         idiomaInstance.properties = params
         return ['idiomaInstance':idiomaInstance]
         } else {
         flash.warning ="Usted necesita iniciar sesion como usuario blog"
         redirect(uri:"/") }
   }

   def save = {
         def userBlog = UsuarioBlog.get(session.usuario.id)
         if (userBlog){
              def idiomaInstance = new Idioma(params)
              idiomaInstance.miembro = Miembro.get( userBlog.miembro.id
)
         if(!idiomaInstance.hasErrors() && idiomaInstance.save()) {
            flash.message = "Idioma ${idiomaInstance.nombreIdioma}
registrado"
            redirect(action:list)
         }
         else {
            render(view:'create',model:[idiomaInstance:idiomaInstance])
         }
         } else {
         flash.warning ="Usted necesita iniciar sesion como usuario blog"
         redirect(uri:"/") }
   }
}
```

III.1.4.2.19. Controlador Imagen

```
import org.springframework.web.multipart.MultipartFile
import java.awt.image.BufferedImage
import javax.imageio.ImageIO

class ImagenController {

   def beforeInterceptor = [action:this.&auth, except:["show",
"publicados"]]

   def auth() {
         def user = session.usuario
         if(!user) {
          flash.warning ="Usted necesita iniciar sesion "
              redirect(uri:"/")
               return false
      }
   }

  // the delete, save and update actions only accept POST requests
   static allowedMethods = [save:'POST', update:'POST']

  def delete = {
         def userBlog = UsuarioBlog.get(session.usuario.id)
         if (userBlog){
               def imagenInstance = Imagen.get( params.id )
              def muestraInstance = Muestra.findByImagen (
imagenInstance )
              def album = imagenInstance.album
              if(imagenInstance) {
             try {
              if ((userBlog.contImgPub > 0) && (album.imgCount > 0))
                        {       userBlog.contImgPub--
                                album.imgCount--
                        }
                      muestraInstance.delete(flush:true)
               imagenInstance.delete(flush:true)
               flash.message = "Imagen eliminada"
               redirect (action:'cargarimgs', params:[albumId:
imagenInstance.album.id])
              }

catch(org.springframework.dao.DataIntegrityViolationException e) {
               flash.message = "Imagen ${imagenInstance.nombre} no pudo
ser eliminada"
              redirect (action:'cargarimgs', params:[albumId:
imagenInstance.album.id])
               }
         }
         else {
```

```
            flash.message = "Imagen no encontrada"
            redirect (action:'cargarimgs', params:[albumId:
imagenInstance.album.id])
             }
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
   }


   def cargarimgs = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
            def album = Album.get( params.albumId )
            def blog = Blog.get( userBlog?.g.id )
            def imagenes = Muestra.findAllByBlogAndAlbum( blog,
album )
        return [albumInstance:album, imagenes:imagenes ]
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
   }

   def create = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
            def album = Album.get( params.albumId )
        def imagenInstance = new Imagen()
        imagenInstance.properties = params
        return ['imagenInstance':imagenInstance, 'albumInstance':album ]
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
  }

   def save = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
             def album = Album.get(params.album.id)
            def blog = album.blog
          def multiPartFile = request.getFile('data')
            BufferedImage original =
ImageIO.read(multiPartFile.inputStream)
              String ext = extractExtension( multiPartFile )
            if ((ext == 'jpeg') || (ext == 'jpg') || (ext == 'png') || (ext ==
'bmp') || (ext == 'gif'))
                {
                    if (multiPartFile.getSize() <
Imagen.TWO_MEG_IN_BYTES) {
                                def image = new Imagen()
                                 image.properties = params
```

```
                                        image.contentType = ext.toLowerCase()
                                        image.size = multiPartFile.getSize()
                                        image.blog = album.blog
                                        image.usuarioBlog = userBlog
                                        image.album = album
                                        image.ancho = original.width
                                image.alto = original.height
                                        image.save(flush:true)
                                        def muestra = null
                                        def gesMuestras = new GesMuestras()
                                 muestra = gesMuestras.createMuestra(blog,
userBlog, album, image, multiPartFile.inputStream)
                                        userBlog.contImgPub++
                                album.imgCount++
                                        flash.message = "Imagen ${image.nombre} creada"
                               redirect (action:'cargarimgs', params:[albumId: album.id])
                                    } else {
                          flash.message = "La foto no puede tener un tamaño mayor a 5MB"
                               redirect (action:'cargarimgs', params:[albumId: album.id])
                                            }
                                } else {
                                        flash.message = "Este formato no esta permitido"
                               redirect (action:'cargarimgs', params:[albumId: album.id])
                                        }
                          } else {
                          flash.warning ="Usted necesita iniciar sesion como usuario blog"
                          redirect(uri:"/") }
    }

          def extractExtension( MultipartFile imagenInstance ) {
         String nombrearchivo = imagenInstance.getOriginalFilename()
      return nombrearchivo.substring(nombrearchivo.lastIndexOf( "." ) + 1 )
       }

    def show = {
      def imagenInstance = Imagen.get( params.id )
   response.outputStream << imagenInstance.data // write the image to the
outputstream
           response.outputStream.flush()
    }

    def publicados = {
          def blogInstance = Blog.findByIdentificador( params.bp )
       if(!blogInstance) {
          flash.message = "Blog no encontrado"
          redirect(uri:"/")
       }
       else {
          def imagenes = Muestra.findAllByBlog( blogInstance )
          def logo = Logo.findByBlogAndEstado(blogInstance, true)
          def categorias = CategoriaEntrada.findAllByBlog(blogInstance)
```

```
        def pagina = Pagina.findAllByBlogAndEstado(blogInstance, true)
        def catLink = CategoriaEnlace.findAllByBlog(blogInstance)
        def link = Enlace.findAllByBlogAndEstado(blogInstance, true)
        def enc = Encuesta.findAllByBlogAndEstado(blogInstance, true)

        return [ blogInstance : blogInstance,
catList:categorias,pagList:pagina, imagenes:imagenes,
                catLink:catLink, link: link, enc: enc, logo:logo  ]
            }
        }
}
```

III.1.4.2.20. Controlador Ingreso

```
class IngresoController {

   def beforeInterceptor = [action:this.&auth]

   def auth() {
        def user = session.usuario
        if(!user) {
         flash.warning ="Usted necesita iniciar sesion "
            redirect(uri:"/")
             return false
     }
   }

   def index = { redirect(action:list,params:params) }
   // the delete, save and update actions only accept POST requests
   static allowedMethods = [save:'POST', update:'POST']

    def list = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
          def bp = Blog.get(userBlog?.g?.id)
          def org = Colegio.get(bp.colegio.id)
          def ingresos = Ingreso.findAllByColegio(org)
          if (ingresos){
            def rol = Rol.get(userBlog.roles.id)
            def totalI = Ingreso.executeQuery("select sum(i.importe) from
Ingreso i where i.colegio.id = ? ",org.id )
            def ingresosT = Ingreso.countByColegio(org)
            if (rol.nombreRol == 'ROLE_ADMINBLOG') {
        render(view:'list', model:[ ingresoInstanceList:ingresos, totalI:
totalI, ingresosT:ingresosT ])
             } else {
             render(view:'listUser', model:[ ingresoInstanceList:ingresos,
totalI: totalI, ingresosT:ingresosT ])
             }
           } else {
```

```
                redirect(action:'create')
                }
            } else {
            flash.warning ="Usted necesita iniciar sesion como usuario blog"
            redirect(uri:"/") }
        }

    def delete = {
            def userBlog = UsuarioBlog.get(session.usuario.id)
            if (userBlog){
                def ingresoInstance = Ingreso.get( params.id )
            if(ingresoInstance) {
               try {
                  ingresoInstance.delete(flush:true)
                  flash.message = "Ingreso eliminado"
                  redirect(action:list)
               }

    catch(org.springframework.dao.DataIntegrityViolationException e) {
                  flash.message = "Ingreso no pudo ser eliminado"
                  redirect(action:list)
               }
            }
            else {
               flash.message = "Ingreso no encontrado"
               redirect(action:list)
            }
            } else {
            flash.warning ="Usted necesita iniciar sesion como usuario blog"
            redirect(uri:"/") }
    }

    def edit = {
            def userBlog = UsuarioBlog.get(session.usuario.id)
            if (userBlog){
                def ingresoInstance = Ingreso.get( params.id )
                if(!ingresoInstance) {
               flash.message = "Ingreso no encontrado"
               redirect(action:list)
            }
            else {
               return [ ingresoInstance : ingresoInstance ]
            }
            } else {
            flash.warning ="Usted necesita iniciar sesion como usuario blog"
            redirect(uri:"/") }
    }

    def update = {
            def userBlog = UsuarioBlog.get(session.usuario.id)
            if (userBlog){
```

```
                def ingresoInstance = Ingreso.get( params.id )
            if(ingresoInstance) {
               if(params.version) {
                  def version = params.version.toLong()
                  if(ingresoInstance.version > version) {

                     ingresoInstance.errors.rejectValue("version",
"ingreso.optimistic.locking.failure", "Another user has updated this
Ingreso while you were editing.")
                    redirect(action:list)
                     return
                  }
               }
               ingresoInstance.properties = params
               if(!ingresoInstance.hasErrors() && ingresoInstance.save()) {
                  flash.message = "Ingreso externo modificado"
                  redirect(action:list)
               }
               else {
                   redirect(action:list)
               }
            }
            else {
               flash.message = "Ingreso no encontrado"
               redirect(action:list)
            }
            } else {
            flash.warning ="Usted necesita iniciar sesion como usuario blog"
            redirect(uri:"/") }
      }

      def create = {
            def userBlog = UsuarioBlog.get(session.usuario.id)
            if (userBlog){
                  def ingresoInstance = new Ingreso()
            ingresoInstance.properties = params
                 def blog = Blog.get(session.usuario?.g?.id)
                 def rol = Rol.get(userBlog.roles.id)
                 def afiliados = Miembro.findAllByN( blog.colegio )
            return ['ingresoInstance':ingresoInstance, afiliados:afiliados,
'rol':rol]
            } else {
            flash.warning ="Usted necesita iniciar sesion como usuario blog"
            redirect(uri:"/") }
      }

      def save = {
            def userBlog = UsuarioBlog.get(session.usuario.id)
            if (userBlog){
                  def ingresoInstance = new Ingreso(params)
                 def blog = Blog.get(userBlog?.g?.id)
```

```
                ingresoInstance.colegio = blog.colegio
            if(!ingresoInstance.hasErrors() && ingresoInstance.save()) {
               flash.message = "Ingreso registrado"
               redirect(action:list)
            }
            else {
               render(view:'create',model:[ingresoInstance:ingresoInstance])
            }
            } else {
            flash.warning ="Usted necesita iniciar sesion como usuario blog"
            redirect(uri:"/") }
      }
   }
```

III.1.4.2.21. Controlador Logo

```
import org.springframework.web.multipart.MultipartFile
import java.awt.image.BufferedImage
import javax.imageio.ImageIO

class LogoController {

   def beforeInterceptor = [action:this.&auth, except:["show"]]

   def auth() {
         def user = session.usuario
         if(!user) {
          flash.warning ="Usted necesita iniciar sesion "
              redirect(uri:"/")
               return false
      }
   }

   def index = { redirect(action:list,params:params) }

   // the delete, save and update actions only accept POST requests
   static allowedMethods = [save:'POST', update:'POST']

   def list = {
         def userBlog = UsuarioBlog.get(session.usuario.id)
         if (userBlog){
              def bp = Blog.get(userBlog.g.id)
              def logos = Logo.findAllByBlog(bp)
              if(logos){
                 def rol = Rol.get(userBlog.roles.id)
                  [ logoInstanceList:logos, logosT:Logo.countByBlog(bp),
rol:rol ]
              } else{
              redirect(action:cargar)
              }
```

```
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }

    }

    def show = {
        def logoInstance = Logo.get( params.id )
        response.outputStream << logoInstance.data // write the image to
the outputstream
        response.outputStream.flush()
    }

    def delete = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
            def logoInstance = Logo.get( params.id )
        if(logoInstance) {
            try {
                logoInstance.delete(flush:true)
                flash.message = "Logo eliminado"
                redirect(action:list)
            }

catch(org.springframework.dao.DataIntegrityViolationException e) {
                flash.message = "Logo no pudo ser eliminado"
               redirect(action:list)
            }
        }
        else {
            flash.message = "Logo no encontrado"
            redirect(action:list)
        }
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
    }

    def mostrar = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
            def logoInstance = Logo.get( params.id )
        if(!logoInstance) {
            flash.message = "Logo no encontrado"
            redirect(action:list)
        }
        else {
            def bp = Blog.get( logoInstance.blog.id)
            def logoAnt = Logo.findByBlogAndEstado(bp, true)
          if (logoAnt){
              logoAnt.estado = false
```

```
                logoInstance.estado = true
                flash.message = "El logo elegido sera mostrado en el blog"
            redirect(action:list)
                } else {
                logoInstance.estado = true
                flash.message = "El logo elegido sera mostrado en el blog"
            redirect(action:list)
                }
        }
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
    }

    def cargar = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
            def logoInstance = new Logo()
        logoInstance.properties = params
            def rol = Rol.get(userBlog.roles.id)
        return ['logoInstance':logoInstance, 'rol':rol]
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
    }

    def save = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
            def logo = new Logo(params)
            def bp = Blog.get(userBlog.g.id)
            def multiPartFile = request.getFile('data')
             String extension = extractExtension( multiPartFile )
          if ((extension == 'jpeg') || (extension == 'jpg') || (extension ==
'png') || (extension == 'bmp') || (extension == 'gif'))
                                    {       logo.blog = bp
                                            BufferedImage original =
ImageIO.read(multiPartFile.inputStream)
                                            logo.contentType =
extension.toLowerCase()
                                            logo.size = multiPartFile.getSize()
                                            logo.nombre =
multiPartFile.originalFilename
                                            logo.ancho = original.width
                                    logo.alto = original.height
                                            logo.save(flush:true)

                                            def img = null
                                            def manImagenes = new
ManImagenes()
```

```
                                    img = manImagenes.createImg(logo,
 multiPartFile.inputStream)
                                     flash.message = "logo subida"
                                redirect(action:list)
                             } else {
                                flash.message = "Este formato no esta permitido"
                                 redirect(action:list)
                                     }
                      } else {
                        flash.warning ="Usted necesita iniciar sesion como usuario
 blog"
                        redirect(uri:"/") }
                     }

                      def extractExtension( MultipartFile logoInstance ) {
                     String nombrelogo = logoInstance.getOriginalFilename()
                     return nombrelogo.substring(nombrelogo.lastIndexOf( "." ) + 1 )
                 }
 }
```

III.1.4.2.22. Controlador Afiliado

```
class MiembroController {

   def beforeInterceptor = [action:this.&auth, except:["listMiembros",
"listCumpleMiembros", "CV"]]
   def auth() {
           def user = session.usuario
           if(!user) {
           flash.warning ="Usted necesita iniciar sesion "
                redirect(uri:"/")
                 return false
      }
   }

   // the delete, save and update actions only accept POST requests
   static allowedMethods = [delete:'POST', save:'POST', update:'POST']

   def list = {
           def userBlog = UsuarioBlog.get(session.usuario.id)
           if (userBlog){
                def bp = Blog.get(userBlog.g.id)
                def u = UsuarioBlog.findAllByG(bp)
                def uT = UsuarioBlog.countByG(bp)
                def rol = Rol.get(userBlog.roles.id)
                 if (rol.nombreRol == 'ROLE_ADMINBLOG') {
                  render(view:'list', model:[ miembroInstanceList: u,
miembroInstanceTotal: uT ])
                } else {
```

```
                params.max = Math.min( params.max ?
params.max.toInteger() : 10,  100)
                render(view:'listUser', model:[ miembroInstanceList: u,
miembroInstanceTotal: uT ])
                }
            } else {
            flash.warning ="Usted necesita iniciar sesion como usuario blog"
            redirect(uri:"/") }
    }

   def edit = {
            def userBlog = UsuarioBlog.get(session.usuario.id)
            if (userBlog){
                def usuarioBlogInstance =
UsuarioBlog.findByIdUsuario(params.m)
                if(!usuarioBlogInstance) {
               flash.message = "Miembro no encontrado"
               redirect(action:list)
            } else {
                 def miembroInstance = Miembro.get(
usuarioBlogInstance.miembro.id )
              return[miembroInstance:miembroInstance,
usuarioBlogInstance:usuarioBlogInstance]
            }
            } else {
            flash.warning ="Usted necesita iniciar sesion como usuario blog"
            redirect(uri:"/") }
    }

    def editP = {
            def userBlog = UsuarioBlog.get(session.usuario.id)
            if (userBlog){
                def miembroInstance = Miembro.get( userBlog.miembro.id )
            if(!miembroInstance) {
               flash.message = "Miembro no encontrado"
               redirect(action:list)
            } else {
               return [ miembroInstance : miembroInstance ]
            }
            } else {
            flash.warning ="Usted necesita iniciar sesion como usuario blog"
            redirect(uri:"/") }
    }

    def editA = {
            def userBlog = UsuarioBlog.get(session.usuario.id)
            if (userBlog){
                def miembroInstance = Miembro.get( userBlog.miembro.id )
                def foto = FotoPersona.findByPersona( miembroInstance )
            if(!miembroInstance) {
               flash.message = "Miembro no encontrado"
```

```
            redirect(action:list)
        } else {
            return [ miembroInstance : miembroInstance, foto:foto ]
             }
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
    }

        def update = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
            def usuarioBlogInstance =
UsuarioBlog.findByIdUsuario(params.m)
        def miembroInstance = Miembro.get(
usuarioBlogInstance.miembro.id )
            if(miembroInstance) {
                if(params.version) {
              def version = params.version.toLong()
              if(miembroInstance.version > version) {
                miembroInstance.errors.rejectValue("version",
"miembro.optimistic.locking.failure", "Otro usuario ha actualizado tus
datos mientras tu estabas editando.")
                 redirect(action:'editP')
                return
              }
            }
            miembroInstance.properties = params
                if(!miembroInstance.hasErrors() &&
miembroInstance.save()) {
                flash.message = "Miembro ${params.nombres} actualizado"
          redirect(action:'list')
            } else {
              redirect(action:'list')
            }
        }
        else {
            flash.message = "Miembro no encontrado"
             redirect(action:'list')
        }
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
    }

    def updateP = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
            def miembroInstance = Miembro.get( userBlog.miembro.id )
        if(miembroInstance) {
            if(params.version) {
```

```
                def version = params.version.toLong()
                if(miembroInstance.version > version) {
                    miembroInstance.errors.rejectValue("version",
"miembro.optimistic.locking.failure", "Otro usuario ha actualizado tus
datos mientras tu estabas editando.")
                     redirect(action:'editP')
                    return
                }
            }
            miembroInstance.properties = params
            if(!miembroInstance.hasErrors() && miembroInstance.save()) {
                flash.message = "Miembro ${params.nombres} actualizado"
                redirect(action:'editP')
            } else {
                redirect(action:'editP')
            }
        }
        else {
            flash.message = "Miembro no encontrado"
             redirect(action:'editP')
        }
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
    }

        def updateA = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
             def miembroInstance = Miembro.get( userBlog.miembro.id )
             def blog = Blog.get(userBlog.g.id)
        if(miembroInstance) {
            if(params.version) {
                def version = params.version.toLong()
                if(miembroInstance.version > version) {
                    miembroInstance.errors.rejectValue("version",
"miembro.optimistic.locking.failure", "Otro usuario ha actualizado tus
datos mientras tu estabas editando.")
                     redirect(action:'editA')
                    return
                }
            }
            miembroInstance.properties = params
            if(!miembroInstance.hasErrors() && miembroInstance.save()) {
                    flash.message = "Informacion Actual actualizada"
                redirect(action:'editA')
            } else {
                redirect(action:'editA')
            }
        }
        else {
```

```
                flash.message = "Miembro no encontrado"
                 redirect(action:'editA')
            }
            } else {
            flash.warning ="Usted necesita iniciar sesion como usuario blog"
            redirect(uri:"/") }
    }

    def habilitar = {
            def userBlog = UsuarioBlog.get(session.usuario.id)
            if (userBlog){
                 def usuarioBlogInstance =
UsuarioBlog.findByIdUsuario(params.m)
            if(usuarioBlogInstance) {
                 usuarioBlogInstance.estado = true
                          flash.message = "La cuenta del miembro fue
habilitada"
                  redirect(action:'list')
            }
            } else {
            flash.warning ="Usted necesita iniciar sesion como usuario blog"
            redirect(uri:"/") }
    }

    def deshabilitar = {
            def userBlog = UsuarioBlog.get(session.usuario.id)
            if (userBlog){
                 def usuarioBlogInstance =
UsuarioBlog.findByIdUsuario(params.m)
            if(usuarioBlogInstance) {
                 usuarioBlogInstance.estado = false
                          flash.message = "La cuenta del miembro fue
deshabilitada"
                  redirect(action:'list')
            }
             } else {
            flash.warning ="Usted necesita iniciar sesion como usuario blog"
            redirect(uri:"/") }
    }

    def create = {
            def userBlog = UsuarioBlog.get(session.usuario.id)
            if (userBlog){
                 def miembroInstance = new Miembro()
                 def usuarioBlogInstance = new UsuarioBlog()
            miembroInstance.properties = params
                 usuarioBlogInstance.properties = params
                 def rol = Rol.get(userBlog.roles.id)
            return[miembroInstance:miembroInstance,
usuarioBlogInstance:usuarioBlogInstance, rol:rol  ]
            } else {
```

```
            flash.warning ="Usted necesita iniciar sesion como usuario blog"
            redirect(uri:"/") }
    }

    def save = {
            def userBlog = UsuarioBlog.get(session.usuario.id)
            if (userBlog){
                 def miembroInstance = new Miembro()
                miembroInstance.properties = params
                String iduser = params.idUsuario
                def usuario = (Usuario.findByIdUsuario(iduser))
                if (usuario)
                { flash.message ="El ID Usuario que ingreso ya se encuentra
registrado elija otro opcion"
                  redirect(action:'create')
                } else {
                def bp = Blog.get(userBlog.g.id)
                def org = Colegio.get(bp.colegio.id)
                miembroInstance.n = org
                if(!miembroInstance.hasErrors() && miembroInstance.save())
{
                        def usuarioBlogInstance = new UsuarioBlog()
                        usuarioBlogInstance.properties = params
                        usuarioBlogInstance.g = bp
                        usuarioBlogInstance.persona = miembroInstance
                        usuarioBlogInstance.miembro = miembroInstance
                    usuarioBlogInstance.fechaRegistracion = new Date()
                  usuarioBlogInstance.ultimoAcceso = new Date()

usuarioBlogInstance.clave=params.clave.encodeAsHash()
                        def nrol =
Rol.findByNombreRol('ROLE_AUTORBLOG')
                  usuarioBlogInstance.roles = Rol.get(nrol.id)
                        usuarioBlogInstance.save()
                       flash.message = "Miembro registrado"
                        redirect(action:'list')
                 }else {
                     flash.message = "No se pudo registrar al miembro"
                redirect(action:'list')
                }
                }
            } else {
            flash.warning ="Usted necesita iniciar sesion como usuario blog"
            redirect(uri:"/") }
    }

     def kardex = {
            def userBlog = UsuarioBlog.get(session.usuario.id)
            if (userBlog){
                def usuarioBlog = UsuarioBlog.findByIdUsuario( params.id )
                if (usuarioBlog){
```

```
                def miembroInstance = Miembro.get(
usuarioBlog.miembro.id )
                def ingresosMiembro = Cuota.findAllByMiembro(
miembroInstance )
                def totalIM = Cuota.executeQuery("select sum(i.importe) from
Cuota i where i.colegio.id = ? and i.miembro.id = ?",
[miembroInstance.n.id, miembroInstance.id]  )
                def blog = userBlog.g
                def logo = Logo.findByBlogAndEstado(blog, true)
                return [ miembroInstance:miembroInstance,logo:logo,
cuotas:ingresosMiembro, totalIM:totalIM ]
             } else {
             flash.warning ="Miembro no encontrado"
             redirect(uri:"/") }
           } else {
           flash.warning ="Usted necesita iniciar sesion como usuario blog"
           redirect(uri:"/") }
    }

   def listMiembros = {
           def bp = Blog.findByIdentificador( params.bp)
            def miembros = UsuarioBlog.findAllByG(bp)
            def categorias = CategoriaEntrada.findAllByBlog(bp)
           def pagina = Pagina.findAllByBlogAndEstado(bp, true)
           def logo = Logo.findByBlogAndEstado(bp, true)
           def entrada = Entrada.findAllByBlogAndEstado(bp, true)
           def catLink = CategoriaEnlace.findAllByBlog(bp)
           def link = Enlace.findAllByBlog(bp)
           def enc = Encuesta.findAllByBlogAndEstado(bp, true)
 params.max = Math.min( params.max ? params.max.toInteger() : 10,  100)
       return [ miembros:miembros, blogInstance : bp,
catList:categorias,pagList:pagina, entrada: entrada, catLink:catLink, link:
link, enc: enc, logo:logo]
   }

  def listCumpleMiembros = {
           def bp = Blog.findByIdentificador( params.bp)
           def org = Colegio.get(bp.colegio.id)
            def miembros =
Miembro.findAllByNAndFechaNacimientoGreaterThan(org, new Date()-
30)
            def categorias = CategoriaEntrada.findAllByBlog(bp)
           def pagina = Pagina.findAllByBlogAndEstado(bp, true)
           def logo = Logo.findByBlogAndEstado(bp, true)
           def entrada = Entrada.findAllByBlogAndEstado(bp, true)
           def catLink = CategoriaEnlace.findAllByBlog(bp)
           def link = Enlace.findAllByBlog(bp)
           def enc = Encuesta.findAllByBlogAndEstado(bp, true)
           if (miembros){
params.max = Math.min( params.max ? params.max.toInteger() : 10,  100)
```

```
        render (view:'listCumpleMiembros', model:[ miembros:miembros,
blogInstance : bp, catList:categorias,pagList:pagina, entrada: entrada,
catLink:catLink, link: link, enc: enc, logo:logo])
            } else {
params.max = Math.min( params.max ? params.max.toInteger() : 10,  100)
        render (view:'nohayCumple', model:[ miembros:miembros,
blogInstance : bp, catList:categorias,pagList:pagina, entrada: entrada,
catLink:catLink, link: link, enc: enc, logo:logo])
              }
            }

    def CV = {
            def usuarioBlog = UsuarioBlog.findByIdUsuario( params.id )
            if (usuarioBlog){
       def miembroInstance = Miembro.get( usuarioBlog.miembro.id )
      def formAc = FormacionAcademica.findAllByMiembro(
miembroInstance )
            def expLab = ExperienciaLaboral.findAllByMiembro(
miembroInstance )
            def capPer = CapacidadPersonal.findAllByMiembro(
miembroInstance )
            def idi = Idioma.findAllByMiembro( miembroInstance )
            def foto = FotoPersona.findByPersona( miembroInstance )
            def org = Colegio.get(miembroInstance.n.id)
            def blog = Blog.findByColegio(org)
            def logo = Logo.findByBlogAndEstado(blog, true)
            return [ miembroInstance : miembroInstance, formAc:formAc,
expLab:expLab, capPer:capPer, idi:idi, foto:foto, org:org, logo:logo ]
            } else {
              flash.warning ="Miembro no encontrado"
              redirect(uri:"/") }
        }
}
```

III.1.4.2.23. Controlador Muestra

```
class MuestraController {

    def index = { redirect(action:show,params:params) }

    def show = {
        def muestraInstance = Muestra.get( params.id )
            response.outputStream << muestraInstance.data // write the image
to the outputstream
            response.outputStream.flush()
        }
    }
```

III.1.4.2.24. Controlador Colegio

```
class ColegioController {

  def beforeInterceptor = [action:this.&auth]

   def auth() {
        def user = session.usuario
        if(!user) {
         flash.warning ="Usted necesita iniciar sesion "
             redirect(uri:"/")
              return false
     }
   }

   // the delete, save and update actions only accept POST requests
   static allowedMethods = [delete:'POST', save:'POST', update:'POST']

  def show = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
             def blog = Blog.get(userBlog.g?.id)
             def colegioInstance= Colegio.get( blog.colegio.id )
             if(!colegioInstance) {
            flash.message = "Colegio no encontrada"
        } else {
             def miembrosTotal = Miembro.countByN( colegioInstance )

             return [ colegioInstance : colegioInstance,
miembrosTotal:miembrosTotal ]
              }
        } else {
          flash.warning ="Usted necesita iniciar sesion"
          redirect(uri:"/") }
   }
  def edit = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
             def bp = Blog.get(userBlog?.g?.id)
             def rol = Rol.get(userBlog.roles.id)
             def colegioInstance = Colegio.get( bp.colegio.id )
             if(!colegioInstance) {
            flash.message = "Colegio no encontrada"
        } else {
                        if (rol.nombreRol == 'ROLE_ADMINBLOG')
                        {
                         return [ colegioInstance : colegioInstance ]
                        } else{
                           redirect (action:'show')
                        }
        }
```

```
        } else {
          flash.warning ="Usted necesita iniciar sesion"
          redirect(uri:"/") }
    }

    def update = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
            def blog = Blog.get(userBlog.g.id)
        def colegioInstance = Colegio.get( blog.colegio.id )
        if(colegioInstance) {
           if(params.version) {
              def version = params.version.toLong()
              if(colegioInstance.version > version) {

                 colegioInstance.errors.rejectValue("version",
"colegio.optimistic.locking.failure", "Otra persona actualizo los datos de
la colegio mientras tu estabas editando")
                 render(view:'edit',model:[colegioInstance:colegioInstance])
                 return
              }
           }
           colegioInstance.properties = params
                    blog.nombre = params.nombre
           if(!colegioInstance.hasErrors() && colegioInstance.save()) {
              flash.message = "Los datos fueron actualizados"
              redirect(action:'edit')
           }
           else {
              render(view:'edit',model:[colegioInstance:colegioInstance])
           }
        }
        else {
           flash.message = "Colegio no encontrada"
           redirect(action:'edit')
        }
        } else {
          flash.warning ="Usted necesita iniciar sesion"
          redirect(uri:"/") }
          }
  }
```

III.1.4.2.25. Controlador Pagina

```
class PaginaController {

    def beforeInterceptor = [action:this.&auth, except:["show"]]

    def auth() {
        def user = session.usuario
```

```
        if(!user) {
         flash.warning ="Usted necesita iniciar sesion "
             redirect(uri:"/")
              return false
      }
   }

  // the delete, save and update actions only accept POST requests
   static allowedMethods = [save:'POST', update:'POST']

 def listBlog = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
             def bp = Blog.get(userBlog?.g?.id)
             def rol = Rol.get(userBlog.roles.id)

              def paginas = Pagina.findAllByBlog(bp)
               params.max = Math.min( params.max ?
params.max.toInteger() : 10,  100)
               [ paginaInstanceList:paginas , paginaInstanceTotal:
Pagina.countByBlog(bp), rol:rol]
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
   }



   def delete = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
           def paginaInstance = Pagina.get( params.id )
             if(paginaInstance) {
           try {
                     def pagImg = PaginaImagen.findByPagina(
paginaInstance )
                     if (pagImg){ pagImg.delete(flush:true) }
              paginaInstance.delete(flush:true)
              flash.message = "La pagina fue eliminada"
              redirect(action:listBlog)
           }

catch(org.springframework.dao.DataIntegrityViolationException e) {
                   flash.message = "La pagina no pudo ser eliminada"
                   redirect(action:listBlog)
           }
        }
        else {
           flash.message = "La pagina no fue encontrada"
           redirect(action:listBlog)
        }
```

```
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
    }

    def edit = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
            def paginaInstance = Pagina.get( params.id )
           def pagImg = PaginaImagen.findByPagina( paginaInstance )
            if(!paginaInstance) {
                flash.message = "Pagina no encontrada"
                redirect(action:listBlog)
            }
            else {
                return [ paginaInstance : paginaInstance, pagImg:pagImg ]
            }
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
    }

    def update = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
            def paginaInstance = Pagina.get( params.id )
        if(paginaInstance) {
            if(params.version) {
                def version = params.version.toLong()
                if(paginaInstance.version > version) {

                    paginaInstance.errors.rejectValue("version",
"pagina.optimistic.locking.failure", "Otro usuario ha actuaizado esta
entrada mientras tu estabas editando.")
                    render(view:'edit',model:[paginaInstance:paginaInstance])
                    return
                }
            }
            paginaInstance.properties = params
            if(!paginaInstance.hasErrors() && paginaInstance.save()) {
                flash.message = "Pagina ${paginaInstance.titulo}
actualizada"
                 redirect(action:listBlog)
            }
            else {
                render(view:'edit',params:[id:paginaInstance.id])
            }
        }
        else {
            flash.message = "Pagina no encontrada"
            redirect(action:listBlog)
```

```
        }
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
}

def create = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
             def rol = Rol.get(userBlog.roles.id)
        def paginaInstance = new Pagina()
        paginaInstance.properties = params
             return ['paginaInstance':paginaInstance, rol:rol ]
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
}

def save = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
              def paginaInstance = new Pagina()
             paginaInstance.properties = params
             paginaInstance.blog = session.usuario?.g
             paginaInstance.usuarioBlog = userBlog
             paginaInstance.fechaModificacion= new Date()
            if(!paginaInstance.hasErrors() && paginaInstance.save()) {
           flash.message = "Pagina ${paginaInstance.titulo} creada"
           redirect(action:listBlog)
        }
        else {
               flash.message = "El titulo de esta pagina ya fue registrado"
           redirect(action:'create')
        }
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
}

        def deshabilitar = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
        def pag = Pagina.get( params.id )
        if(pag) {
                pag.properties = params
                  pag.estado = false
             if(!pag.hasErrors() && pag.save()) {
              flash.message = "La Pagina ${pag.titulo} fue deshabilitada"
              redirect(action:listBlog)
             } else {
```

```
                    flash.message = "La Pagina ${pag.titulo} no fue
deshabilitada"
                    redirect(action:listBlog)
                   }
                   } else {
                  flash.message = "Pagina no encontrada"
                  redirect(action:listBlog)
                   }
               } else {
              flash.warning ="Usted necesita iniciar sesion como usuario blog"
              redirect(uri:"/") }
        }

        def habilitar = {
              def userBlog = UsuarioBlog.get(session.usuario.id)
              if (userBlog){
                    def paginaInstance = Pagina.get( params.id )
              if(paginaInstance) {
                 paginaInstance.properties = params
                      paginaInstance.estado = true
                 if(!paginaInstance.hasErrors() && paginaInstance.save()) {
                    flash.message = "La Pagina ${paginaInstance.titulo} fue
habilitada"
                    redirect(action:listBlog)
                 } else {
                              flash.message = "La Pagina  ${paginaInstance.titulo}
no fue deshabilitada"
                     redirect(action:listBlog)
                 }
              } else {
                 flash.message = "Pagina no fue encontrada"
                 redirect(action:list)
              }
              } else {
              flash.warning ="Usted necesita iniciar sesion como usuario blog"
              redirect(uri:"/") }
        }

       def show = {
              def paginaInstance = Pagina.findByTituloAndBlog( params.titulo,
Blog.findByIdentificador(params.bp) )
              def pagImg = PaginaImagen.findByPagina( paginaInstance )
              def blogInstance = Blog.get(paginaInstance.blog.id)
              def pagina = Pagina.findAllByBlogAndEstado(blogInstance, true)
              def logo = Logo.findByBlogAndEstado(blogInstance, true)
              def categorias = CategoriaEntrada.findAllByBlog(blogInstance)
              def catLink = CategoriaEnlace.findAllByBlog(blogInstance)
              def link = Enlace.findAllByBlogAndEstado(blogInstance, true)
              def enc = Encuesta.findAllByBlogAndEstado(blogInstance, true)
            if(!paginaInstance) {
              flash.message = "Pagina not found with id ${params.id}"
```

```
            redirect(action:list)
        }
        else { return [ paginaInstance : paginaInstance, pagImg:pagImg,
blogInstance: blogInstance, catList:categorias, pagList:pagina,
catLink:catLink, link: link, enc: enc, logo:logo] }
    }
}
```

III.1.4.2.26. Controlador PaginaImagen

```
import org.springframework.web.multipart.MultipartFile

class PaginaImagenController {

  def beforeInterceptor = [action:this.&auth, except:["show"]]

   def auth() {
         def user = session.usuario
         if(!user) {
          flash.warning ="Usted necesita iniciar sesion "
              redirect(uri:"/")
               return false
      }
   }

   // the delete, save and update actions only accept POST requests
   static allowedMethods = [save:'POST', update:'POST']

    def show = {
     def paginaImagenInstance = PaginaImagen.get( params.id )
         if (paginaImagenInstance){
         response.outputStream << paginaImagenInstance.pagimgData.data
// write the image to the outputstream
         response.outputStream.flush()
         } else {
           flash.warning ="Imagen de pagina no encontrada"
           redirect(uri:"/") }
   }

   def delete = {
         def userBlog = UsuarioBlog.get(session.usuario.id)
         if (userBlog){
               def paginaImagenInstance = PaginaImagen.get( params.id )
         if(paginaImagenInstance) {
            try {
               paginaImagenInstance.delete(flush:true)
                     flash.message = "La imagen fue borrada "
               redirect(controller:"pagina", action:"edit",
id:paginaImagenInstance.pagina.id)
             }
```

```
catch(org.springframework.dao.DataIntegrityViolationException e) {
                flash.message = "La imagen no pudo ser borrada de la
pagina"
                redirect(controller:"pagina", action:"edit",
id:paginaImagenInstance.pagina.id)
            }
        }
        else {
           flash.message = "La imagen no fue encontrada"
           redirect(action:list)
        }
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
   }

   def create = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
              def pagina = Pagina.get( params.id )
        def paginaImagenInstance = new PaginaImagen()
        paginaImagenInstance.properties = params
        return ['paginaImagenInstance':paginaImagenInstance,
'pagina':pagina]
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
   }

   def save = {
    def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
              def paginaImagenInstance = new PaginaImagen( params )
             // guardando imagen
              MultipartFile f = request.getFile( 'pagimgData.data' )
              paginaImagenInstance.size = f.getSize() / 1024
              paginaImagenInstance.extension = extractExtension( f )
                if(paginaImagenInstance.save()) {
                 flash.message = "Imagen
[${paginaImagenInstance.nombre}] subida exitosamente."
                        redirect(controller:"pagina", action:"edit",
id:paginaImagenInstance.pagina.id)
                         } else {
                          redirect(controller:"pagina", action:"edit",
id:paginaImagenInstance.pagina.id)
                }
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
    }
```

```
def extractExtension( MultipartFile paginaImagenInstance ) {
   String nombrearchivo = paginaImagenInstance.getOriginalFilename()
  return nombrearchivo.substring(nombrearchivo.lastIndexOf( "." ) + 1 )
      }
}
```

III.1.4.2.27. Controlador País

```
class PaisController {

 def updateSelect = {

   def paisSelected = Pais.find("from Pais as p where p.nombre=:nombre",
[nombre:params.selectedValue])
         render (template:"selectEstados", model : ['paisSelected' :
paisSelected])
         }
   }
```

III.1.4.2.28. Controlador Persona

```
class PersonaController {

    def index = { redirect(action:list,params:params) }

    // the delete, save and update actions only accept POST requests
    static allowedMethods = [delete:'POST', save:'POST', update:'POST']

    def list = {
         if ( session.usuario ) {
               params.max = Math.min( params.max ?
params.max.toInteger() : 10,  100)
         [ personaInstanceList: Persona.list( params ),
personaInstanceTotal: Persona.count() ]
         } else {
           flash.warning ="Usted necesita iniciar sesion"
           redirect(uri:"/") }
    }

    def show = {
         if (session.usuario){
               def personaInstance = Persona.get( params.id )
         if(!personaInstance) {
            flash.message = "Persona not found with id ${params.id}"
            redirect(action:list)
         }
         else { return [ personaInstance : personaInstance ] }
         } else {
           flash.warning ="Usted necesita iniciar sesion"
```

```
            redirect(uri:"/") }
    }

    def delete = {
        if (session.usuario) {
            def personaInstance = Persona.get( params.id )
        if(personaInstance) {
            try {
                personaInstance.delete(flush:true)
                flash.message = "Persona ${params.id} deleted"
                redirect(action:list)
            }

catch(org.springframework.dao.DataIntegrityViolationException e) {
                flash.message = "Persona ${params.id} could not be deleted"
                redirect(action:show,id:params.id)
            }
        }
        else {
            flash.message = "Persona not found with id ${params.id}"
            redirect(action:list)
        }
        } else {
          flash.warning ="Usted necesita iniciar sesion"
          redirect(uri:"/") }
    }

    def edit = {
        if (session.usuario) {
            def personaInstance = Persona.get( params.id )
            if(!personaInstance) {
            flash.message = "Persona not found with id ${params.id}"
            redirect(action:list)
        }
        else {
            return [ personaInstance : personaInstance ]
        }
        } else {
          flash.warning ="Usted necesita iniciar sesion"
          redirect(uri:"/") }
    }

    def update = {
        if (session.usuario) {
            def personaInstance = Persona.get( params.id )
        if(personaInstance) {
                if(params.version) {
            def version = params.version.toLong()
            if(personaInstance.version > version) {
```

```
                    personaInstance.errors.rejectValue("version",
"persona.optimistic.locking.failure", "Another user has updated this
Persona while you were editing.")

render(view:'edit',model:[personaInstance:personaInstance])
                    return
                }
            }
            personaInstance.properties = params
            if(!personaInstance.hasErrors() && personaInstance.save()) {
                flash.message = "Persona ${params.id} updated"
                redirect(action:show,id:personaInstance.id)
            }
            else {
                render(view:'edit',model:[personaInstance:personaInstance])
            }
        }
        else {
            flash.message = "Persona not found with id ${params.id}"
            redirect(action:list)
        }
        } else {
          flash.warning ="Usted necesita iniciar sesion"
          redirect(uri:"/") }
    }

    def create = {
        if (session.usuario) {
            def personaInstance = new Persona()
        personaInstance.properties = params
        return ['personaInstance':personaInstance]
        } else {
          flash.warning ="Usted necesita iniciar sesion"
          redirect(uri:"/") }
    }

    def save = {
        if (session.usuario) {
            def personaInstance = new Persona(params)
        if(!personaInstance.hasErrors() && personaInstance.save()) {
            flash.message = "Persona ${personaInstance.id} created"
            redirect(action:show,id:personaInstance.id)
        }
        else {
            render(view:'create',model:[personaInstance:personaInstance])
        }
        } else {
          flash.warning ="Usted necesita iniciar sesion"
          redirect(uri:"/") }
    }
}
```

III.1.4.2.29. Controlador Plantilla

```
import org.springframework.web.multipart.commons.CommonsMultipartFile

class PlantillaController {

    def beforeInterceptor = [action:this.&auth]

   def auth() {
                def user = session.usuario
                if(!user) {
           flash.warning ="Usted necesita iniciar sesion "
                    redirect(uri:"/")
                     return false
       }
   }

  // the delete, save and update actions only accept POST requests
   static allowedMethods = [update:'POST']



                def changeTPL = {
                def userBlog = UsuarioBlog.get(session.usuario.id)
                if (userBlog){
                      def blogInstance = Blog.get(userBlog?.g?.id )
                     blogInstance.properties = params
                     def plantillas = Plantilla.list()
                     def rol = Rol.get(userBlog.roles.id)
                 return [ 'blogInstance' : blogInstance, 'plt' : plantillas, 'rol':rol ]
                  } else {
                flash.warning ="Usted necesita iniciar sesion como usuario blog"
                redirect(uri:"/") }
                }

                def updateTPL = {
                 def userBlog = UsuarioBlog.get(session.usuario.id)
                   if (userBlog){
                      def blogInstance = Blog.get(userBlog.g?.id )
                        if(blogInstance) {
                          def tpl = Plantilla.get(params.plantilla.id)
                            blogInstance.plantilla = tpl
                          flash.message = "La plantilla de su Blog fue cambiada"
                            redirect(action:'changeTPL')
                        } else {
                          flash.message = "Ocurrio un error debe volver al principio"
                          redirect(action:list)
                          }
                      } else {
```

```
            flash.warning ="Usted necesita iniciar sesion como usuario blog"
            redirect(uri:"/") }
                        }
            }
```

III.1.4.2.30. Controlador Respuesta

```
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartUtilities;
import org.jfree.chart.JFreeChart;
import org.jfree.data.general.DefaultPieDataset;
import java.io.File;

class RespuestaController {

   def beforeInterceptor = [action:this.&auth, except:["votar",
"registrarVoto", "resultados"]]
   def auth() {
         def user = session.usuario
         if(!user) {
          flash.warning ="Usted necesita iniciar sesion "
              redirect(uri:"/")
               return false
      }
   }

   // the delete, save and update actions only accept POST requests
   static allowedMethods = [save:'POST', update:'POST']

  def delete = {
         def userBlog = UsuarioBlog.get(session.usuario.id)
         if (userBlog){
               def respuestaInstance = Respuesta.get( params.id )
         if(respuestaInstance) {
            try {
               respuestaInstance.delete(flush:true)
               flash.message = "Respuesta ${respuestaInstance.nombre}
eliminada"
                redirect(controller:'encuesta', action:'edit',
params:[id:respuestaInstance.encuesta.id])
            }

catch(org.springframework.dao.DataIntegrityViolationException e) {
               flash.message = "Respuesta no pudo ser eliminada"
                redirect(controller:'encuesta', action:'edit',
params:[id:respuestaInstance.encuesta.id])
            }
         }
         else {
            flash.message = "Respuesta no encontrada"
```

```
            redirect(controller:'encuesta', action:'edit',
params:[id:respuestaInstance.encuesta.id])
        }
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
    }

  def addRes = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
            def enc = Encuesta.get( params.id )
        def respuestaInstance = new Respuesta()
        respuestaInstance.properties = params
        return ['respuestaInstance':respuestaInstance, 'enc': enc]
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
    }

    def saveRes = {
        def userBlog = UsuarioBlog.get(session.usuario.id)
        if (userBlog){
            def respuestaInstance = new Respuesta(params)
        if(!respuestaInstance.hasErrors() && respuestaInstance.save()) {
            flash.message = "Respuesta creada"
            redirect(controller:'encuesta', action:'edit',
params:[id:respuestaInstance.encuesta.id])
        }
        else {
            redirect(controller:'encuesta', action:'edit',
params:[id:respuestaInstance.encuesta.id])
        }
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
    }

  def votar = {
        def encElegida = Encuesta.get( params.enc )
        def blogInstance = Blog.get( encElegida.blog.id)
      def logo = Logo.findByBlogAndEstado(blogInstance, true)
       def categorias = CategoriaEntrada.findAllByBlog(blogInstance)
        def pagina = Pagina.findAllByBlog(blogInstance)
     def entrada = Entrada.findAllByBlogAndEstado(blogInstance, true)
        def catLink = CategoriaEnlace.findAllByBlog(blogInstance)
        def link = Enlace.findAllByBlog(blogInstance)
        def enc = Encuesta.findAllByBlogAndEstado(blogInstance, true)
```

```
            return [encElegida: encElegida, enc: enc, blogInstance :
blogInstance, catList:categorias,pagList:pagina, entrada: entrada,
catLink:catLink, link: link, logo:logo]

        }
        def registrarVoto = {
            def resp = Respuesta.get(params.respuesta)
            def enc = Encuesta.get( resp.encuesta.id )
            enc.contVotos++
            resp.contVotos++
            def respuestas = Respuesta.findAllByEncuesta( enc )
            DefaultPieDataset pieDataset = new DefaultPieDataset();
             respuestas.each { respuesta ->
               pieDataset.setValue(respuesta.opcionRespuesta, new
Integer(respuesta.contVotos));
              }
            JFreeChart chart = ChartFactory.createPieChart(
            "${enc.titulo}", pieDataset, true, true, false);
            try {
            def webRootDir = servletContext.getRealPath("/")
            ChartUtilities.saveChartAsJPEG(new File(webRootDir,
"/images/PieChart2.jpg"), chart, 500,300);
            } catch (Exception e) {
            System.out.println("Error creando grafico.");
            }
            redirect (action:'resultados', params:[respuesta:resp.id])
        }

        def resultados = {
            def resp = Respuesta.get(params.respuesta)
            def enc = Encuesta.get( resp.encuesta.id )
            def blogInstance = Blog.get( enc.blog.id)
            def respuestas = Respuesta.findAllByEncuesta( enc )
            def categorias =
CategoriaEntrada.findAllByBlog(blogInstance)
            def pagina = Pagina.findAllByBlog(blogInstance)
            def entrada = Entrada.findAllByBlogAndEstado(blogInstance,
true)
            def catLink = CategoriaEnlace.findAllByBlog(blogInstance)
            def link = Enlace.findAllByBlog(blogInstance)
            def logo = Logo.findByBlogAndEstado(blogInstance, true)
            def enclis = Encuesta.findAllByBlogAndEstado(blogInstance,
true)
            return ['respuestas':respuestas, enc: enclis, encRes: enc,
blogInstance : blogInstance, catList:categorias,pagList:pagina, entrada:
entrada, catLink:catLink, link: link, logo:logo]
        }
}
```

III.1.4.2.31. Controlador UsuarioBlog

```
class UsuarioBlogController {

  MailService mailService

  def beforeInterceptor = [action:this.&auth, except:["login",
"olvidoClave"]]
   def auth() {
        def user = session.usuario
        if(!user) {
         flash.warning ="Usted necesita iniciar sesion "
            redirect(uri:"/")
             return false
     }
   }

   // the delete, save and update actions only accept POST requests
   static allowedMethods = [save:'POST', update:'POST']

        def login = {
            if (params.usuario && params.pass) {
                    def user =
UsuarioBlog.findByIdUsuarioAndEstado(params.usuario, true)
                    String password = params.pass.encodeAsHash()
                    if (user != null && user.clave == password)  {
                    flash.message = "Bienvenida/o  al ${user.g?.nombre}"
                            session.usuario = user
                            redirect(uri:"/", )
                    } else {
                flash.warning = "Usuario o Clave incorrecto, intente de nuevo"
                            redirect(uri:"/")
                    }
                }else{
              flash.warning = "Ingrese los datos para autentificarse"
                            redirect(uri:"/")
                    }
                }

   def logout = {
        if (session.usuario) {
            def usuario = Usuario.get(session.usuario.id)
            usuario.ultimoAcceso = new Date ()
            session.usuario = null
            flash.message = "Tu session ha sido cerrada, hasta pronto!"
            redirect(uri:"/")
         } else {
        flash.warning ="Usted necesita iniciar sesion"
        redirect(uri:"/") }
    }
```

```
  def olvidoClave = {
            if (params.userid) {
               def user = Usuario.findByIdUsuario(params.userid)
                  if (user){
                        if (user.persona.infAct.email) {
   def PW_POOL = "23456789ABCDEFGHJKLMNPQRSTUVWXYZ"
                              def genPw = ""
8.times {
genPw += PW_POOL[new Random().nextInt(PW_POOL.size() -1)]
                        }
                              user.genPw = genPw
                              user.clave = genPw.encodeAsHash()
                              def msg = """
                        <h1>BLOGPROF</h1>
                        <p>
  Hola ${user.idUsuario}, nosotros cambiamos tu clave
a<b>${genPw}</b>.
Para ingresar a tu cuenta tu debes ingresar esta clave en mayusculas
Una vez hayas iniciado session tu puedes cambiar esta clave de modo
                              que sea mas recordable para ti.
                              </p>
                              <p>
                              blogprof.com
                              </p>
                        """
mailService.send(user.persona.infAct.email, msg, "blogprof.com : Clave
cambiada")
    flash.message = "Una nueva clave a sido generada en tu cuenta email"
                redirect(uri:"/")
                           } else {
      flash.message = "No se puede localizar tu cuenta email"
                              redirect(uri:"/")
                                    }
                        } else {
          flash.warning = "El ID Usuario no se encuentra registrado"
                              redirect(uri:"/")
                              }
                }
          }

   def edit = {
          def userBlog = UsuarioBlog.get(session.usuario.id)
          if (userBlog){
              render(view:'edit')
          } else {
          flash.warning ="Usted necesita iniciar sesion como usuario blog"
          redirect(uri:"/") }
   }

    def update = {
          def userBlog = UsuarioBlog.get(session.usuario.id)
```

```
        if (userBlog){
            if (params.idUsuario && params.claveActual) {

                        String claveA = params.claveActual.encodeAsHash()
                        def usuario = UsuarioBlog.findByClaveAndIdUsuario(
claveA, params.idUsuario )
                        if (usuario!= null )  {
                                String claveN = params.clave.encodeAsHash()
                                usuario.clave = claveN
                                flash.message = "Su clave a sido cambiada."
                                 redirect(controller:'miembro', action:'editA')
                                } else {
                                flash.message = "No coincide el ID usuario o
clave actual ingresada, intente de nuevo."
                                redirect(controller:'miembro', action:'editA')
                                }
            }
        } else {
        flash.warning ="Usted necesita iniciar sesion como usuario blog"
        redirect(uri:"/") }
    }
}
```

III.1.4.3. Vistas

III.1.4.3.1. Vista Blog:show

```
<html>
   <head>
      <title>${blogInstance?.nombre}</title>
            <meta name="layout" content="main_blogs" />
  </head>
   <body >


               <h2 >Entradas publicadas en estos ultimos 7
dias</h2><br><hr>
               <g:each  in="${entradas}" status="i" var="ent">
                        <g:if test="${ent.estado == true}">
                          <g:link controller="entrada" action="show"
params="[titulo:ent.titulo]">
                           <h2><g:message code="${ent.titulo}"
encodeAs="HTML"/></h2> </g:link> <br>
                                    ${ent?.fechaCreacion} publicado por
${ent?.usuarioBlog?.idUsuario} <br>
                        <div class="articles">   <g:message
code="${ent.cita}" encodeAs="HTML"/>  <g:link controller="entrada"
action="show" params="[titulo:ent.titulo]"> Leer mas... </g:link>
                        <br></div><hr><br>
                        </g:if>
               </g:each>
   </body>
</html>
```

III.1.4.3.2. Vista Entrada:listBlog

```
<html>
   <head>
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
      <meta name="layout" content="main" />
      <title>Entradas Publicadas</title>
<script language="JavaScript">
          //configure refresh interval (in seconds)
           var countDownInterval=30;
          //configure width of displayed text, in px (applicable only in NS4)
           var c_reloadwidth=200
          </script>
<ilayer id="c_reload" width="&amp;{c_reloadwidth};"><layer
id="c_reload2" width="&amp;{c_reloadwidth};" left="0"
top="0"></layer></ilayer>
<script>
var countDownTime=countDownInterval+1;
function countDown(){
```

```
countDownTime--;
if (countDownTime <=0){
countDownTime=countDownInterval;
clearTimeout(counter)
window.location.reload()
return
}
if (document.all) //if IE 4+
document.all.countDownText.innerText = countDownTime+" ";
else if (document.getElementById) //else if NS6+
document.getElementById("countDownText").innerHTML=countDownT
ime+" "
else if (document.layers){ //CHANGE TEXT BELOW TO YOUR OWN
document.c_reload.document.c_reload2.document.close()
}
counter=setTimeout("countDown()", 1000);
}
function startit(){
if (document.all||document.getElementById)
document.write('<i id="countDownText">'+countDownTime+' </i>')
countDown()
}

if (document.all||document.getElementById)
startit()
else
window.onload=startit
</script>
   </head>
   <body>
            <div class="body">
           <div class="nav">
          <h2>Entradas Publicadas en el Blog </h2>
           </div>
          <g:if test="${flash.message}">
          <div class="message">${flash.message}</div>
          </g:if>
           <g:if test="${enTotal == 0}">
               <div><br><br><br>
          <center><h3>Aun no ha publicado ninguna entrada en el blog. Las
entradas<br>pueden ser agrupadas en categorias. Cuando cree una
entrada<br>usted tiene la opcion de permitir que le hagan comentarios o
no.<br>Las entradas pueden ser creadas tanto ppor el administrador del
<br>blog como por los demas usuarios que se encuentren habilitados.<br>
                          </h3><br>
                             <g:link controller="entrada" class="create"
action="create">Crear Entrada</g:link></center>
                          <br><br><br>
                          </div>
               </g:if>
               <g:else>
```

```
            <div class="nav">
                <span class="menuButton"><g:link class="create"
action="create" >Nueva Entrada</g:link></span>
        </div>
            <div class="list">
              <table>
                <thead>
                  <tr>
           <g:sortableColumn property="titulo" title="Titulo" />
   <g:sortableColumn property="categoria" title="Categoria" />
   <g:sortableColumn property="fechaCreacion" title="Fecha Creacion" />
   <g:sortableColumn property="fechaModificacion" title="Fecha
Modificacion" />
   <g:sortableColumn property="autor" title="Autor" />
   <g:sortableColumn property="estado" title="Estado" />
   <g:sortableColumn property="ver" title="Modificar" />
   <g:sortableColumn property="eliminar" title="Eliminar" />
   <g:sortableColumn property="Ver Comentarios" title="Comentarios" />

                  </tr>
                </thead>
                <tbody>
                <g:each in="${entradaInstanceList}" status="i"
var="entradaInstance">
                    <tr class="${(i % 2) == 0 ? 'odd' : 'even'}">

    <td>${fieldValue(bean:entradaInstance, field:'titulo')}</td>
   <td>${entradaInstance?.catEntrada.nombre}</td>
   <td><g:formatDate format="dd-MMM-yyyy HH:mm"
date="${entradaInstance.fechaCreacion}" /></td>
  <td><g:formatDate format="dd-MMM-yyyy HH:mm"
date="${entradaInstance.fechaModificacion}" /></td>
<td>${entradaInstance?.usuarioBlog?.miembro.nombres}  
          ${entradaInstance?.usuarioBlog?.miembro.apellidos}</td>
     <g:if test="${entradaInstance?.estado == true}">
                        <td>Publicado/<g:link action="deshabilitar"
id="${entradaInstance.id}"> Deshabilitar </g:link></td>
                    </g:if>
                          <g:else>
                            <td>No Publicado/<g:link action="habilitar"
id="${entradaInstance.id}"> Habilitar </g:link></td>
                          </g:else>

                        <td><g:link action="edit" id="${entradaInstance.id}">
Modificar </g:link></td>
                <td> <g:link action="delete" id="${entradaInstance.id}"
onclick="return confirm('Esta seguro que desea eliminar esta
Entrada?');">Eliminar</g:link></td>
                          <td><g:link controller="comentario" action="list"
id="${entradaInstance.id}"> Comentarios </g:link></td>
                    </tr>
```

```
                </g:each>
                </tbody>
              </table>
            </div>
            <div class="paginateButtons">
            <g:paginate total="${enTotal}" />
               </div>

            </g:else>
   </body>
</html>
```

III.1.4.3.3. Vista Album:listBlog

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8"/>
      <meta name="layout" content="main" />
         <g:javascript src="prototype.js" />
         <g:javascript src="effects.js" />
         <g:javascript src="validation.js" />
         <modalbox:modalIncludes />
      <title>BLOGPROF</title>
   </head>
   <body>
    <div class="body">
         <div class="nav">
        <h2>Fotos </h2>
      </div>
         <g:if test="${ paginateCount == 0 }">
         <div id="nota"><br><br><br>
              <center><h3>Para subir y publicar tus fotografias primero
debes<br>
                          crear el Album correspondiente.<br>
                         Todos los usuarios de este blog podran crear sus
propios<br>
                         albums de imagenes y subir imagenes a albums
creados por <br>
                         otros usuario.<br>
                          </h3>
                       <br><modalbox:createLink  class="create"
action="create" width="500" title="Crear Album">Nueva
Album</modalbox:createLink></center>
                       <br><br><br>
                       </div>
         </g:if>
         <g:else>
      <div class="nav">
```

```
                <span class="menuButton"><modalbox:createLink
class="create" action="create" width="500" title="Crear Album">Nueva
Album</modalbox:createLink></span><br>
        </div>
          <FIELDSET class="derecho">
            <div id="tplmuestras" class="adblock">
              <style type="text/css">
              div#tplmuestras.adblock
{width:800px;display:block;align:center;}
              div#tplmuestras.adblock li{width:180px;}
              div#tplmuestras.adblock li
img{width:150px;height:150;border:3px double #eee;}
              div#tplmuestras.adblock li a em{font-style:normal;font-
size:11px;color:#888;font-family:verdana,sans-serif;margin:0 5px 10px
0;text-align:center;text-decoration:none;overflow:hidden; float:left;}
              div#tplmuestras.adblock li em{font-style:normal;}
              div#tplmuestras.adblock li{display:block;font-
size:11px;color:#888;font-family:verdana,sans-serif;margin:0 5px 10px
0;text-align:center;text-decoration:none;overflow:hidden; float:left;}
              div#tplmuestras.adblock img{border:0;clear:right;}
              </style>
        <ul class="adblock">
              <g:each in="${list}" status="i" var="list">
              <li><g:link controller="imagen" action="cargarimgs"
params="[albumId:list.id]"><img src="../images/album.png"
title="${list.descripcion}"> </g:link> <br/>
                        <g:link controller="imagen" action="cargarimgs"
params="[albumId:list.id]">Cargar Fotos</g:link>|
<p>${list.nombre}</p>|<g:link action="delete" id="${list.id}"
onclick="return confirm('Si elimina el album seran eliminadas tambien
todas sus imagenes ?');">Eliminar</g:link>
              </li>
              </g:each>
              </ul>
            </div>
            </FIELDSET>
        </g:else>
      </div>
    </body>
</html>
```

III.1.4.3.4. Vista Afiliado: listMiembros

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8"/>
      <title>${blogInstance?.nombre}</title>
            <meta name="layout" content="main_blogs" />
         <resource:autoComplete skin="default" />
```

```
        <script language="Javascript">
        function popUp(URL) {
        day = new Date();
        id = day.getTime();
        eval("page" + id + " = window.open(URL, '" + id + "',
        'toolbar=0,scrollbars=1,location=0,statusbar=1,menubar=0,resizabl
e=0,width=730,height=500,left = 50,top = 0');");
        }
        </script>
   </head>
   <body>

      <div class="body">
         <div class="list">
            <center><IMG  src=" ../images/feliz_cumple.gif"
border="0"></center> <br>
            <hr>
            <div id="muestras" class="adblock">
              <style type="text/css">
                    div#muestras.adblock
{width:600px;display:block;align:center;}
                    div#muestras.adblock li{width:170px;}
                    div#muestras.adblock li img{border:3px double #eee;}
                    div#muestras.adblock li a em{font-style:normal;font-
size:11px;color:#888;font-family:verdana,sans-serif;margin:0 5px 10px
0;text-align:center;text-decoration:none;overflow:hidden; float:left;}
                    div#muestras.adblock li em{font-style:normal;}
                    div#muestras.adblock li{display:block;font-
size:11px;color:#888;font-family:verdana,sans-serif;margin:0 5px 10px
0;text-align:center;text-decoration:none;overflow:hidden; float:left;}
                    div#muestras.adblock img{border:0;clear:right;}
               </style>
            <ul class="adblock">
                     <g:each in="${miembros}" status="i"
var="miembroIns">
                        <li>
     <g:if test="${miembroIns.foto == false}">
        <img  src="../images/foto.jpg" ></td>
     </g:if>
     <g:else>
        <img  src="${createLink(controller:'fotoPersona', action:'show',
id:miembroIns?.id)}" >
     </g:else>
<br/><h3>${miembroIns.nombres} ${miembroIns.apellidos}</h3>
                      </li>
                        </g:each>
                      </ul>
              </div>
          </div>
          <div class="paginateButtons">
            </div>
```

```
      </div>
    </body>
</html>
```

III.1.4.3.5. Vista Pagina:create

```
<html>
 <head>
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
      <meta name="layout" content="main" />
         <g:javascript src="prototype.js" />
         <g:javascript src="effects.js" />
         <g:javascript src="validation.js" />
         <resource:richTextEditor type="full" />
      <title>BLOGPROF</title>
    </head>
    <body>

      <div class="body">
           <div class="nav">
        <h2>Crear Pagina</h2>
          </div>

           <div class="dialog">
             <g:if test="${flash.message}">
         <div class="message">${flash.message}</div>
          </g:if>
          <g:hasErrors bean="${paginaInstance}">
        <div class="errors">
          <g:renderErrors bean="${paginaInstance}" as="list" />
         </div>
          </g:hasErrors>

            <g:if test="${rol.nombreRol == 'ROLE_ADMINBLOG'}">
             <g:uploadForm action="save" name="create" method="post"
enctype="multipart/form-data">
            <fieldset class="derecho">
                 <div class="form-row">
                   <label for="titulo">Titulo :</label>
                   <input type="text" id="titulo" class="required"
size="70" name="titulo"
value="${fieldValue(bean:paginaInstance,field:'titulo')}"/>
                 </div>
                 <div class="form-row-editor" ><center>
               <richui:richTextEditor name="publicacion" id="publicacion"
class="required" value="${pagina?.publicacion}" width="525"
html:title="This editor fields also allows formatted text."/>
                 </center></div>
                     <div class="form-row">
                  <label for="estado">Publicar:</label>
```

```
                            <g:checkBox name="estado"
value="${paginaInstance?.estado}" ></g:checkBox>
                                Si desea publicar esta entrada seleccione esta
casilla
                    </div>
                        <div class="form-row">
                     <label for="fechaCreacion">Fecha Creacion:</label>
                     <g:datePicker name="fechaCreacion"
value="${paginaInstance?.fechaCreacion}" precision="minute"
></g:datePicker>              </div>
                        <div class="form-row">
                          <span class="button"><input class="save"
type="submit" value="Crear" /></span>
                        </div>
                    </fieldset>
                </div>

        </g:uploadForm>
          <g:javascript>
              function formCallback(result, form) {
                window.status = "valiation callback for form '" + form.id + "':
result = " + result;
                        }
                 var valid = new Validation('create', {immediate : true,
onFormValidate : formCallback});
          </g:javascript>
          </g:if>
          <g:else>
          <div id="nota"><br><br><br>
              <center><h3>Las paginas son solo creadas por el
Administrador del Blog
                            </h3><br>
            <br><br><br>
          </div>
          </g:else>
       </div>
    </body>
</html>
```

III.1.5. Plataforma BLOGPROF

III.1.5.1. Pantalla : Principal

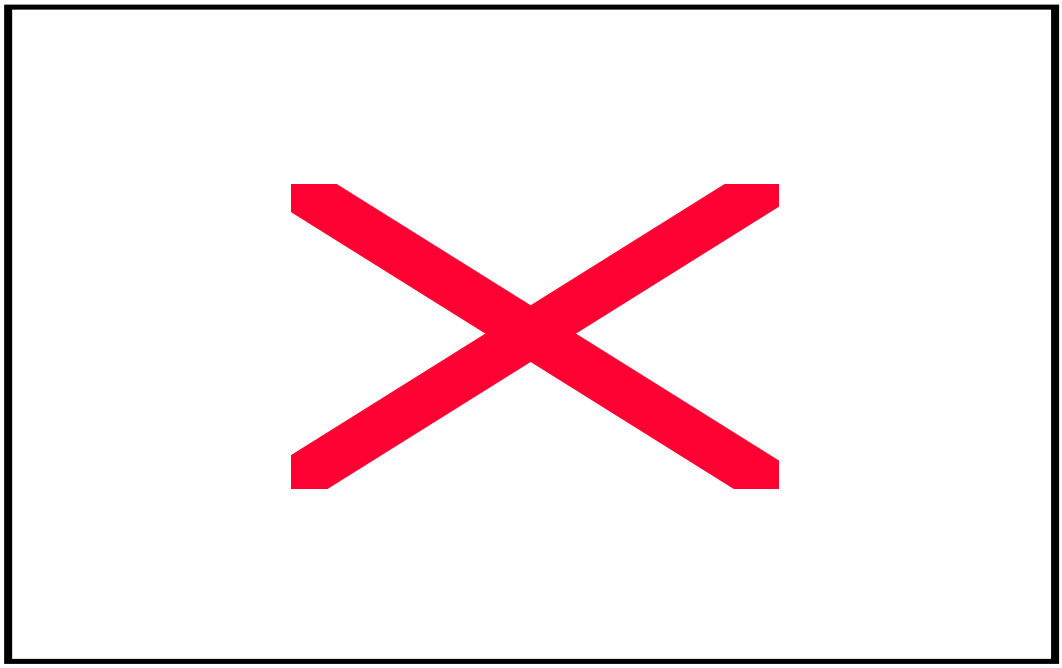

Figura 65. Pantalla Principal

III.1.5.2. Pantalla: Afiliados Registrados

Figura 66. Pantalla Miembros Registrado

III.1.5.3. Pantalla: Crear Entrada

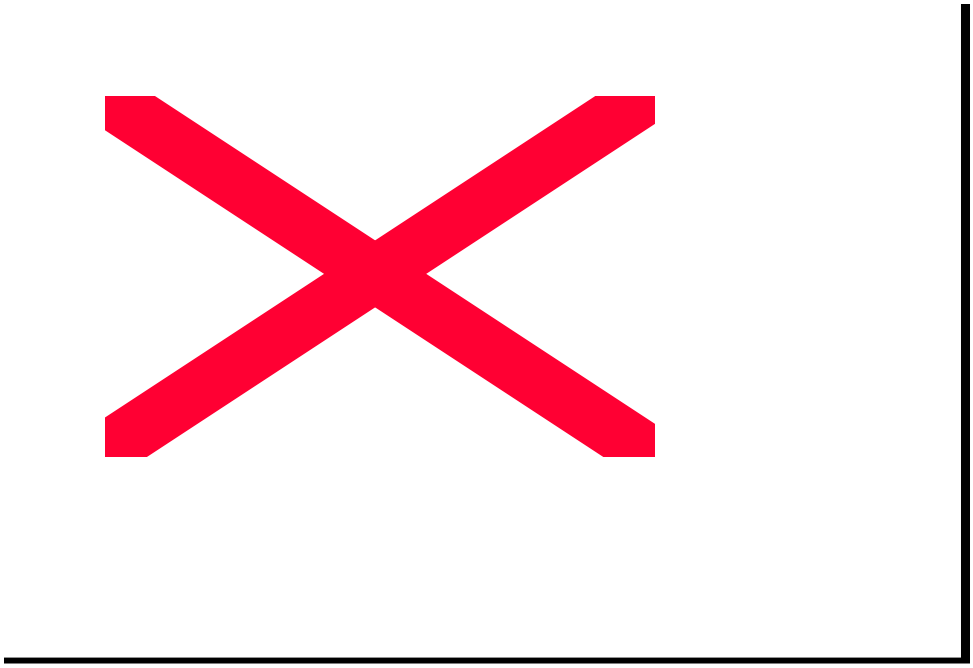

Figura 67. Pantalla Crear Entrada

III.1.5.4. Pantalla: Cambiar Plantilla

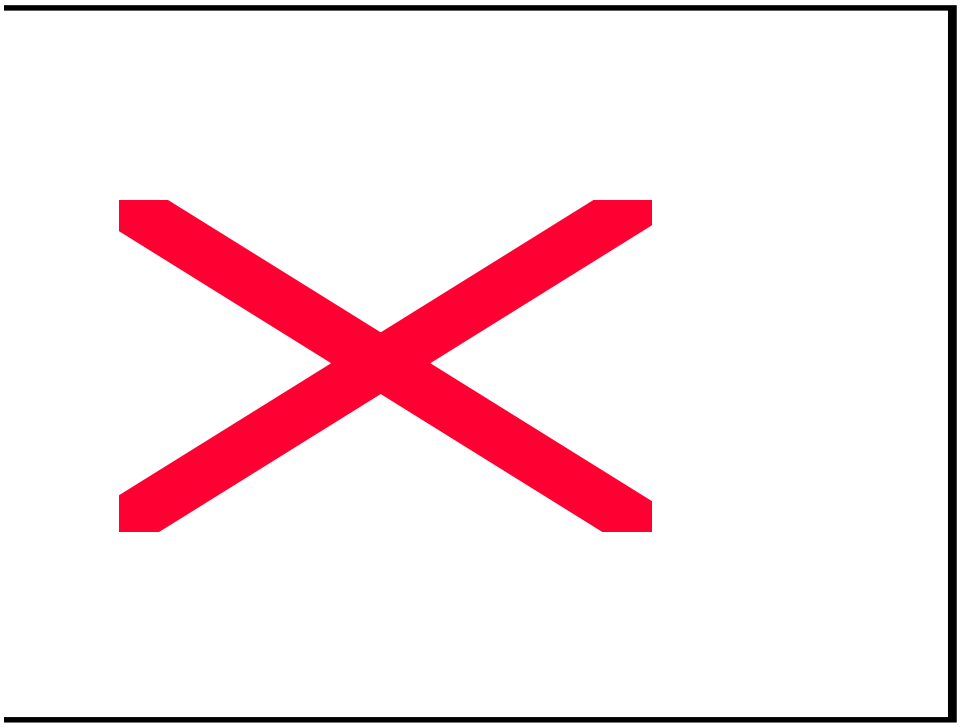

Figura 68. Pantalla Cambiar Plantilla

III.1.5.5. Pantalla: Subir Foto

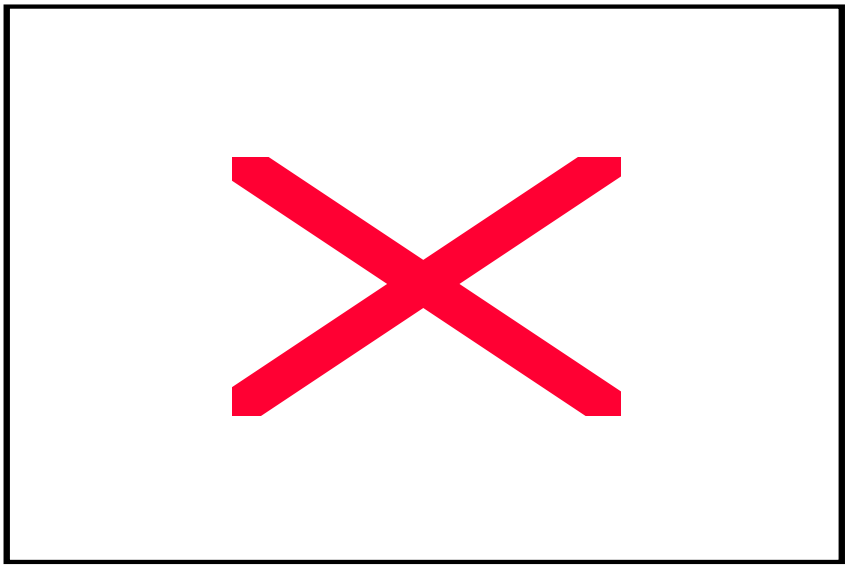

Figura 69. Pantalla Subir Foto

III.1.5.6. Pantalla: Registrar Formacion Academica

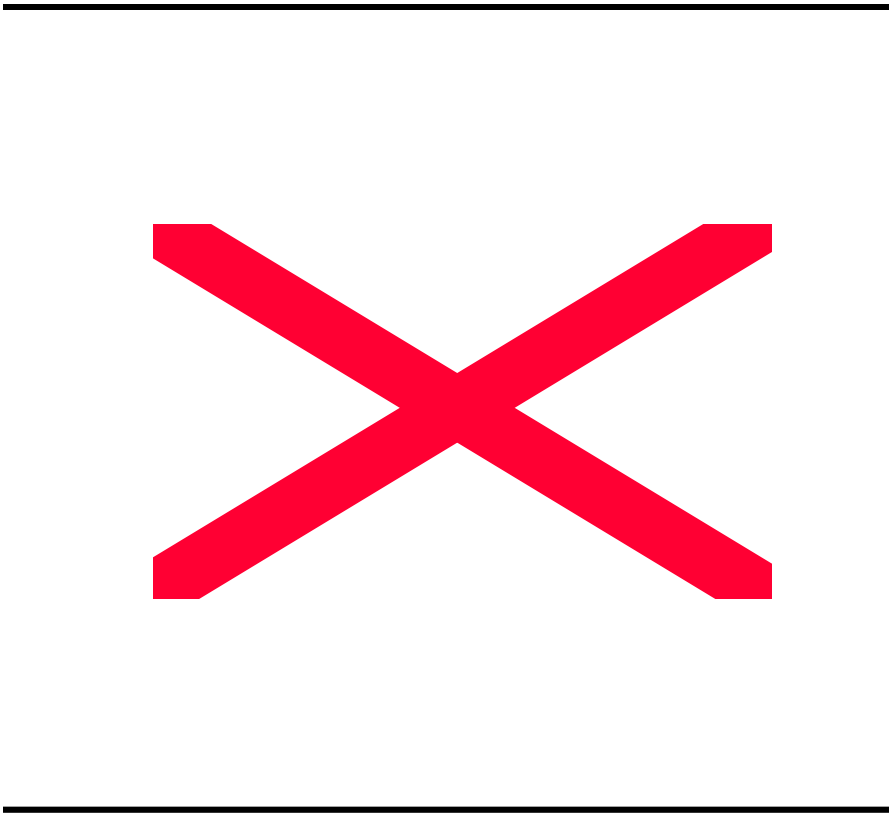

Figura 70. Pantalla Registrar Formacion Academica

III.1.5.7. Pantalla: Ver Fotos en Blog

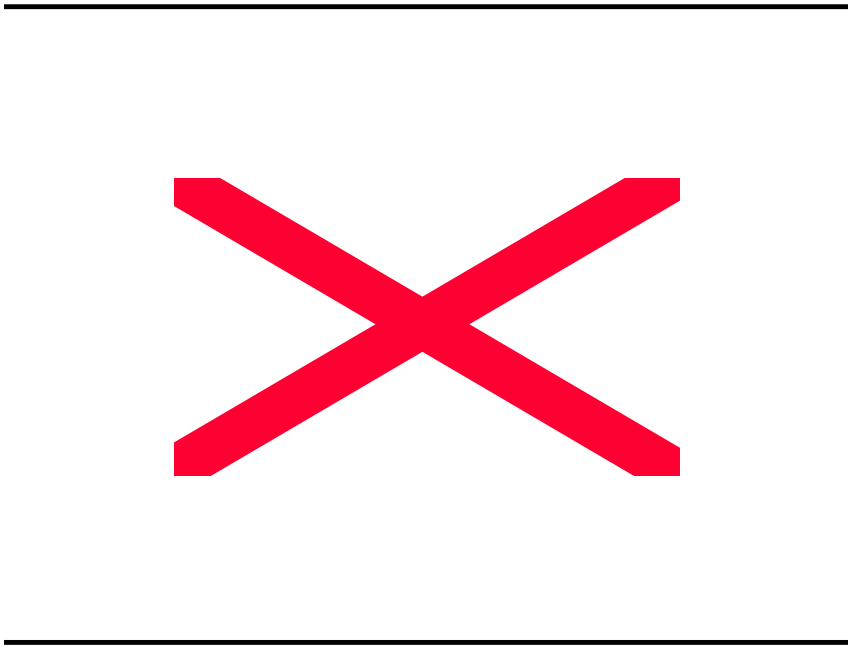

Figura 71. Pantalla Ver Fotos en Blog

III.1.5.8. Pantalla: Ver Directorio en Blog

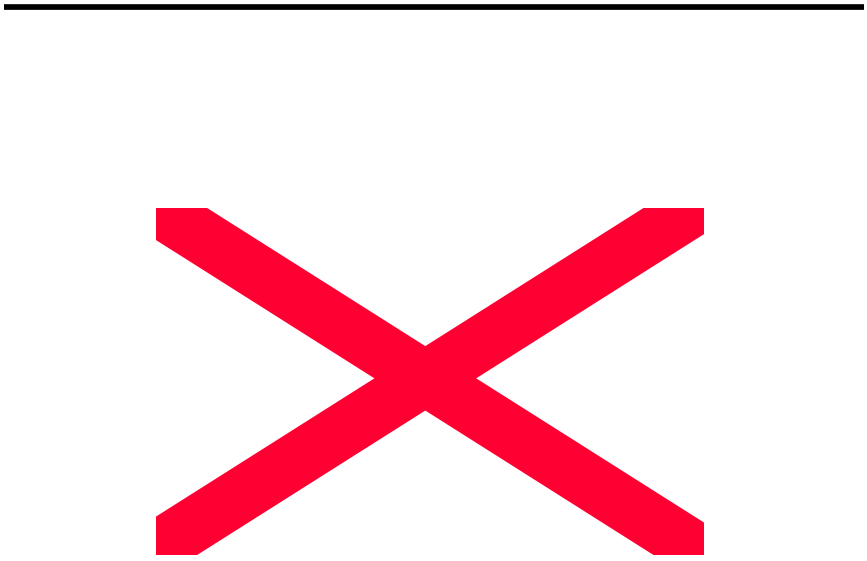

Figura 72. Pantalla Ver Directorio en Blog

III.1.5.9. Pantalla: Listar Archivos en Blog

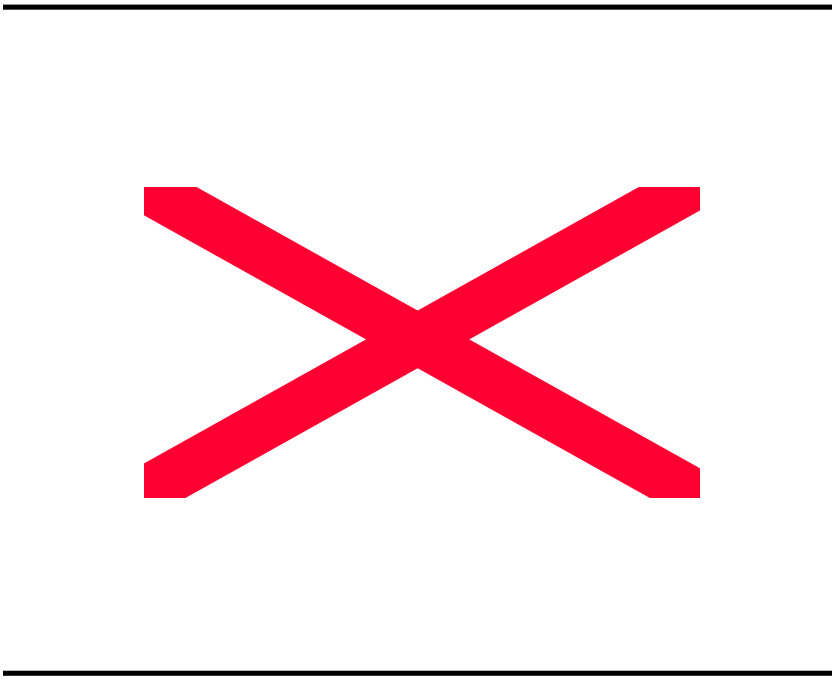

Figura 73. Pantalla Listar Archivos en Blog

III.1.5.10. Pantalla: Ver Blog

Figura 74. Pantalla Ver Blog

III.1.5.11. Pantalla: Comentar Entrada

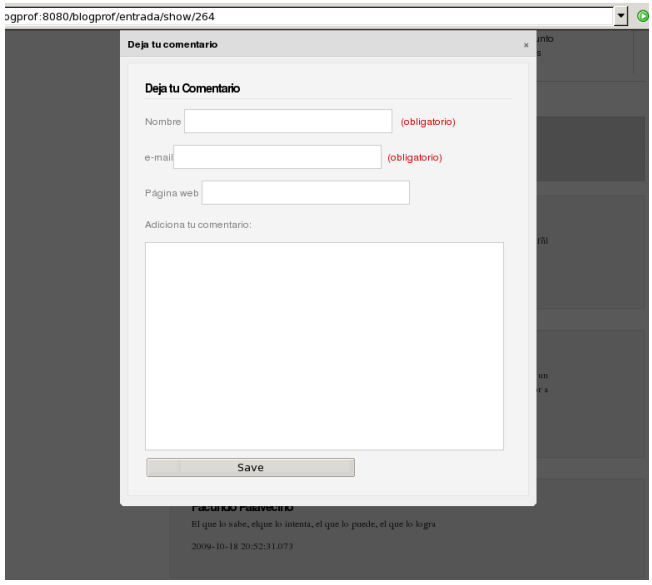

Figura 75. Pantalla Comentar Entrada

III.1.5.12. Pantalla: Ver CV de Afiliado

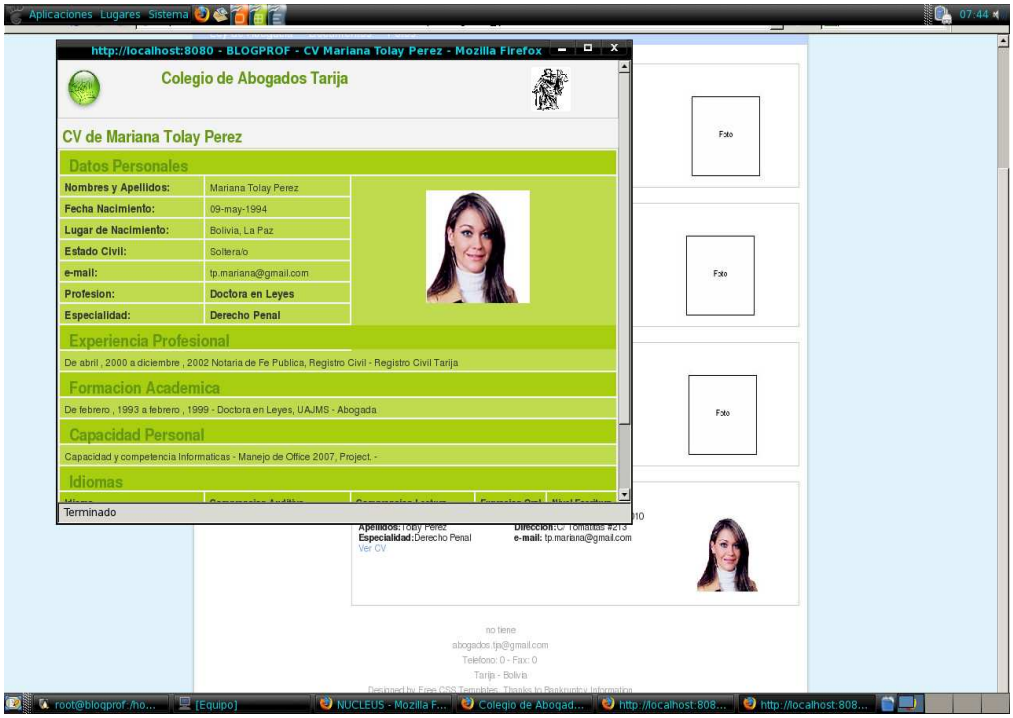

Figura 76. Pantalla Ver CV de Afiliado

III.1.5.13. Pantalla: Resultados de Encuesta

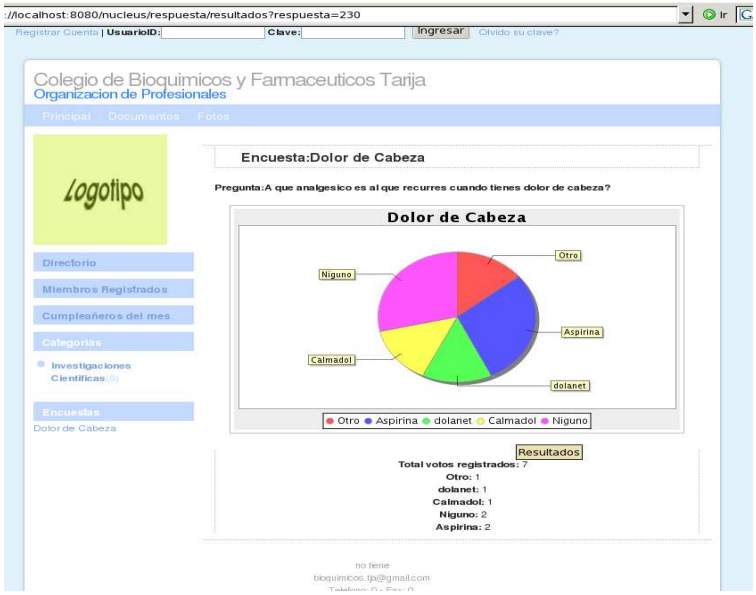

Figura 77. Pantalla Resultados de Encuesta

III.1.5.14. Pantalla: Gestionar Directorio

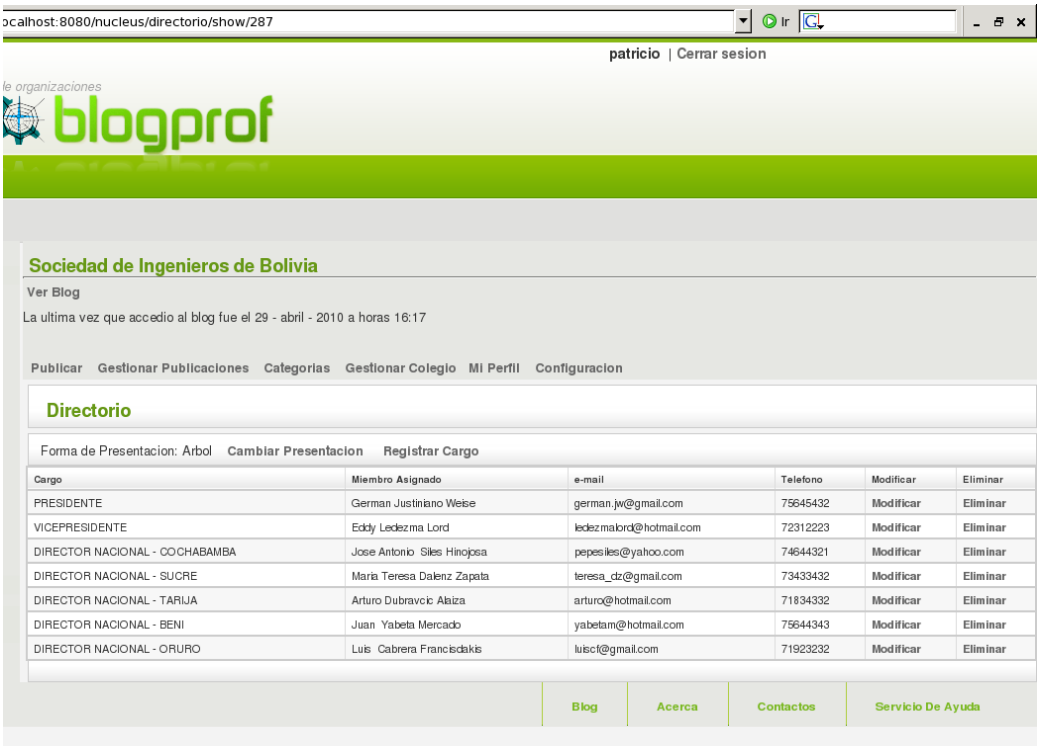

Figura 78. Gestionar Directorio

III.1.6. Estudio de Hosting

III.1.6.1. Introduccion

Hosting (**alojamiento o también conocido como hospedaje web, alojamiento web, web site hosting, web hosting o webhosting**) es un negocio que consiste en alojar, servir, y mantener archivos para uno o más sitios web. Más importante que el espacio del ordenador que se proporciona para los archivos del sitio web es la conexión rápida a Internet.

La mayoría de los **servicios de hosting** ofrecen conexiones que para una persona individual resultarían muy costosas. Usar un servicio de hosting permite que muchas compañías compartan el coste de una conexión rápida a Internet para el acceso a los archivos de sus sitios web.

Geocities ofrece a sus visitantes registrados espacio para un sitio web gratis, aunque este tipo de hosting web gratuito es muy básico.

Algunas compañías de hosting describen sus servicios como **hosting virtual**. Hosting virtual generalmente implica que sus servicios serán transparentes y que cada sitio web tendrá su propio alojamiento de dominio y sus propias direcciones de email. En la mayoría de los casos, el hosting y el hosting virtual son sinónimos. Algunas compañías de hosting permiten a sus usuarios tener su propio servidor virtual, con la apariencia de que el usuario está controlando un servidor dedicado enteramente a su sitio web.

Los **alojamientos dedicados** consisten en el uso exclusivo de todo un servidor por un único cliente, mientras que en los **alojamientos compartidos** varios clientes comparten un servidor. Solamente los sitios web con mucho tráfico requieren el hosting dedicado. Muchas compañías compran sus propios servidores y los colocan en un sitio que proporcione acceso rápido a Internet. Esta práctica se llama **colocación**.

III.1.6.2. Servicios de hosting

Hay una amplia variedad de servicios de hosting. El más básico es el **hosting de archivos (alojamiento web u hospedaje web)**, donde se pueden alojar las páginas de los sitios web y otros archivos vía ftp o una interfaz web. Los archivos se muestran en la web tal cual o sin mucho procesado. Muchos proveedores de Internet ofrecen este servicio de forma gratuita

a sus clientes. El alojamiento web es normalmente gratuito, patrocinado por anunciantes, o barato.

Normalmente el alojamiento web sólo es suficiente para páginas web personales. Un sitio web complejo necesita un paquete más avanzado que proporcione soporte para **bases de datos** y plataformas de desarrollo de **aplicaciones** (ej. PHP, Java, y ASP. NET). Estas permiten que los clientes escriban o instalen scripts para aplicaciones como foros y formularios. Para el comercio electrónico también se requiere SSL.

El proveedor de hosting puede también proporcionar un **interfaz** web (ej. panel de control) para manejar el servidor web e instalar los scripts así como otros servicios como e-mail.

III.1.6.3. Tipos de hosting

III.1.6.3.1. Hosting gratuito

El hosting gratuito es extremadamente limitado comparado al hosting de pago. Los proveedores de alojamiento gratuito normalmente requieren sus propios anuncios en el sitio alojado de forma gratuita y tienen límites muy grandes de espacio y de tráfico. No obstante, la mayoría de la gente empieza en la web con hosting gratuito.

III.1.6.3.2. Hosting de Imagenes

Alojando solamente algunos formatos de imágenes. Este tipo de alojamiento normalmente es gratuito y la mayoría requieren que el usuario se registre. La mayoría de los proveedores de alojamiento de imágenes permiten el hotlinking, de modo que el usuario pueda subir imágenes al servidor del proveedor para ahorrar espacio y ancho de banda.

III.1.6.3.3. Hosting compartido

El alojamiento compartido es cuando un mismo servidor aloja a varios cientos de sitios web de clientes distintos. Un problema en uno de los sitios en el servidor puede traer abajo al resto de los sitios. El hosting compartido también tiene algunas restricciones con respecto a qué ser puede hacer exactamente, aunque estas restricciones no son en ninguna manera tan restrictivas como en el hosting gratuito.

III.1.6.3.4. Hosting Dedicado

Con el alojamiento dedicado, uno consigue un servidor sólo para él. No tienen ninguna restricción, a excepción de las diseñadas para mantener la integridad del proveedor (por ejemplo, prohibiendo sitios con contenido para adultos debido al riesgo del aumento de ataques por los hackers y las cuestiones legales). A menos que se pague una tarifa a parte al proveedor, el usuario tiene que hacerlo todo por sí mismo. Esto puede ser costoso, pues la contratación del servidor dedicado en sí es generalmente más costosa comparada al alojamiento compartido.

III.1.6.3.5. Hosting VPS

Virtual Private Servers (VPSs): Un VPS **es el paso intermedio entre un servidor dedicado y el alojamiento Web compartido**, pero más cercano al concepto de servidor dedicado. Se trata de un "servidor virtual" integrado en un servidor físico que aloja otros servidores virtuales o VPS. Cada VPS (Servidor Privado Virtual) está completamente separado de los demás servidores virtuales y dispone de un uso reservado de capacidad de almacenamiento, memoria RAM y procesador, por lo que comparten única y exclusívamente ubicación física, pero no comparten ni recursos (CPU o RAM) ni software (cada VPS, es en sí mismo un servidor independiente desde el punto de vista del sistema operativo y del software).

Cada VPS esta **completamente aislado** de los demás VPS disponiendo de un uso reservado de capacidad de almacenamiento, teniendo además su propia configuración de sistema, aplicaciones, librerías, panel de control, etc y su funcionamiento no interfiere con los demás VPS instalados en el mismo servidor o máquina física en cuestión.

Los VPSs o Servidores Privados Virtuales, se basan en la fragmentación de un servidor en múltiples entornos dedicados, compartiendo entre ellos exclusivamente el hardware, pero no el software. A efectos operativos es exactamente **igual que un servidor dedicado**, pero compartiendo única y exclusivamente recursos de hardware con otros VPS.

Cada VPS dispone de sus propios **recursos reservados**, de forma que aunque un VPS exceda su uso de RAM o procesador, **no se verá nunca afectado por el rendimiento de los demás** VPS.

III.1.6.4. Rating Hosting

Según su precio, rentablçilidad y soporte, se seleccionaron los siguientes Web hosting como las 10 mejores opciones a la hora de subir un proyecto Web.

Referencia: http://www.webhosting-top10.com/ **Fecha : 20 de abril del 2010**

Empresa proveedora	Rating	Disco	Ancho de Banda	Paquete Web Hosting	Precio
globat.com WEB HOSTING MADE EASY	**98%**	ilimitado	ilimitado	SQL, PHP, CGI, FrontPage, Perl, dominio libre, cuentas email, etc. **# hosting profesional, bajo precio**	**$4.44**
hostmonster	**95%**	ilimitado	ilimitado	SQL, PHP, CGI, FrontPage, Perl, JSP free domain, email accounts, etc. **# rapido, buen servicio**	**$5.95**
FASTDOMAIN HOSTING	**95%**	ilimitado	ilimitado	SQL, PHP, CGI, FrontPage, Perl, free domain, email accounts, etc. **# rápido, buen servicio, rentable**	**$3.95**
Host Gator we eat up the competition	**92%**	ilimitado	ilimitado	SQL, PHP, CGI, FrontPage, Perl, JSP pop3 email accounts, etc. **# hosting de buenísima calidad**	**$4.95**
ix WEB HOSTING	**90%**	ilimitado	ilimitado	PHP, CGI, FrontPage, Perl, ruby on Rails free domain, email accounts, etc. **# bueno, hosting barato**	**$3.95**
bluehost	**86%**	ilimitado	ilimitado	SQL, PHP, CGI, FrontPage, Perl, free domain, email accounts, etc.	**$6.95**

				# hosting rentable, rapido	
STARTLOGIC	**80%**	ilimitado	ilimitado	SQL, PHP, CGI, FrontPage, Perl, ASP free domain, email accounts, etc. **# excelente soporte**	**$4.95**
iPowerWeb Hosting Over 700,000 Websites!	**72%**	1,500 GB	15,000 GB	SQL, PHP, CGI, FrontPage, Perl, free domain, email accounts, etc. **# reliable, established company**	**$4.95**
lunarpages web hosting	**66%**	ilimitado	ilimitado	SQL, PHP, CGI, FrontPage, Perl, free domain, email accounts, etc. **# nice performance, free software**	**$6.95**
JumpLaunch	**65%**	ilimitado	ilimitado	PHP, CGI, FrontPage, Perl, free domain, email accounts, etc. **# nice and cheap hosting**	**$4.95**

Tabla 92. Rating Hosting

III.1.6.5. Hosting VPS

Virtual Private Server comprende la tecnología actual mas avanzada en virtualizacion de servidores. Para una aplicación grails es una excelente opción de hosting es necesaria una buena capacidad de memoria RAM, como minimo 512 MB.

Sistema Operativo	Ancho de Banda	Disco	Memoria RAM	Dominios	Precio
Linux	**2 TB**	**40 GB**	**1 gb**	**Ilimitados**	**$ 70 / mes**
Linux	**3 TB**	**60 GB**	**2 GB**	**Ilimitados**	**$ 132 / mes**
Linux	**4 TB**	**80 GB**	**3 GB**	**ilimitados**	**$ 180 / mes**
Windows	**600 gb**	**20 GB**	**512 MB**	**Solo 100**	**$ 95 / mes**
Windows	**900 gb**	**30 GB**	**512 MB**	**Solo 200**	**$124 / mes**

Tabla 93. Cotizacion Hosting VPS

III.2 Componente Capacitacion y Socializacion

III.2.1. Capacitacion

En la actualidad la capacitación de los recursos humanos en la respuesta al necesidad que tienen las empresas o instituciones de contar con un personal calificado y productivo.
La obsolescencia también es una de las razones por la cual, las instrucciones se preocupan por capacitar a sus recursos humanos, pues esta procura actualizar sus conocimientos con las nuevas técnicas y métodos de trabajo que garantizan eficiencia.

Los procesos de capacitación son recibidos en muchas ocasiones como una oportunidad de crecimiento y de aprendizaje con el fin de, no sólo de mejorar la tarea y el desempeño para el cual hemos sido contratados, sino también para crecer como personas, para interiorizar contenidos que quizá no tenga aplicación inmediata pero que dan temple y seguridad para las oportunidades futuras.

III.2.1.1. Importancia de la Capacitacion

A mayor desarrollo tecnológico en la sociedad, mayor necesidad de talento, o sea, de personas competentes técnica y emocionalmente capaces de crear, innovar, crear valor, afrontar retos en los negocios, elaborar bienes y servicios de calidad y contribuyan a que la organización aprenda a mantenerse en un mercado globalizado, la tendencia es que las organizaciones se conviertan en comunidades de aprendizaje que lo generen, lo conserven y lo traduzcan en acciones de valor agregado, la sobrevivencia en el mundo global y competitivo depende, en estos momentos, de la inversión que hagan las empresas en intangibles, como innovación tecnológica, organización flexible y desarrollo de capital humano. Así, la utilización del conocimiento apropiado se convierte en la principal fuente de ventaja competitiva para una organización en la época actual.

III.2.1.2. Proceso de Capacitacion

Este componente hace referencia a la capacitación que se realizo a los profesionales de los distintos Colegios de nuestra ciudad. Para esto se hizo uso de la Metodologia DICE
Al culminar esta capacitación, se entrego certificados que acreditaron la asistencia y la compresión de los temas tratados, estos fueron repartidos a las personas que asistieron a esta capacitación y es usado como un medio de verificación del cumplimiento de este componente.

III.2.1.3. Metodologia DICE

DICE es una metodología dirigida especialmente para los responsables de capacitación en las organizaciones, que permite fortalecer las competencias de quienes tienen esa responsabilidad. Esta metodología integra cuatro elementos importantes que son Diagnostico, Intervencion, Comprobacion, Evaluacion.

Como bien se menciono anteriormente esta metodología es para la capacitación del personal de organizaciones o empresas, a si que se trato de adecuar esta metodología para efectuar la capacitación de los profesiones de los distintos Colegios.

III.2.1.3.1. Diagnostico

Es el análisis de necesidades de capacitación, permite al responsable de capacitación encontrar no solamente los temas sobre los cuales se debe trabajar, sino también permite definir la profundidad requerida para cada uno de los temas, establecer la intensidad horaria de cada intervención, definir los grupos de participantes, es decir nos permite estructurar la capacitacion.

En base a un estudio que se realizo a distintos profesionales afiliados y no afiliados de nuestra ciudad se pudo constatar que no existe un buen conocimientos sobre el uso de herramientas Sociales por lo consiguiente se definió la siguiente estructura para la capacitación:

Estructura de la capacitación

Titulo del Seminarion o Taller

"Tendencias Actuales en la Web"

Objetivos

El objetivo central de la capacitación es capacitar a los profesionales de nuestra ciudad pertenecientes a los Colegios de profesionales en el uso de Herramientas Sociales Web 2.0.

Permitir a los participantes de la capacitación conocer nuevas tecnologías que faciliten su interaccion social.

Socializar la herramienta social BLOGPROF

Temas a desarrollar

Tema I: Web 2.0, beneficios del Software Social

- Introducción
- Conceptos Básicos
- Evolución de la Web
- ¿Qué es la Web 2.0?
- Inteligencia Colectiva
- Características de la Web 2.0
- La importancia de la Web 2.0 en las organizaciones

Tema II: BLOGNET, herramienta de administración en los Colegios de Profesionales

- Introducción al blogging
- Comparando blogs con otras herramientas de comunicación
- BLOGNET, Gestor de blogs para Colegios de Profesionales
- Ventajas de BLOGNET para los Colegios de Profesionales
- Presentación de BLOGNET

Tema III: Otras Herramientas Sociales

- Redes Sociales
- Wikis
- Sindicación de Contenidos
- API's publicas
- Otras aplicaciones

Tiempos de Ejecucion: 24 horas

Dirigido a

La capacitación estuvo dirigida a profesionales pertenecientes a los distintos Colegios de Profesionales de nuestra ciudad, para lo cual se hizo llegar una carta de invitacion a cada uno de los Colegios (ver en Anexos) y se lanzo un comunicado por la radio. Como bien se dijo la capacitación estuvo dirigida a profesionales pero no se limito la participación de otras personas.

III.2.1.3.2. Intervencion

La intervención entonces es la ejecución de los programas de capacitación que abarca diferentes escenarios de acción, En el tiempo de intervención es preciso tener en cuenta también ladisponibilidad de los participantes.

En el proceso de intervención o desarrollo de la capacitación se tomo en cuenta la disponibilidad de tiempo de los asistentes, como es sabido que la mayoría trabaja se realizo el taller de capacitación en las noches, para que puedan asistir después de su trabajo.

Para la capacitación se hizo uso de una de las aulas del edificio de la carrera de Informatica, un datadiplay, y diapositivas (var contenido tematico de la capacitación en Anexos).

III.2.1.3.3. Comprobacion

Al final del taller de capacitación se comprobó que hubo una buena captación de la capacitación, porque todos los participantes tenían mas interés en abrir su blog tener su red social, se pudo evidenciar de esa manera su motivación por las herramientas sociales.

III.2.1.3.4. Evaluacion

Con el ánimo de conocer la captación del evento de capacitación, se solicito que cada uno de los participantes cree su blog o su su perfil haciendo uso de cualquiera de las herramientas enseñadas en la capacitación.

III.2.1.4. Medios de Verificacion

Los medios de verificación de este componente son las cartas de invitaciones que se hizo llegar a cada uno de los Colegios de Profesionales, y los certificados de asistencia que se les entrego a las personas que asistieron al taller de capacitación. En Anexos se tiene las cartas de invitación que se hizo llegar a cada uno de los colegios y un certificado como muestra para verificar que se entregaron certificados.

III.2.2. Socializacion

La socialización es el proceso por el cual una cultura, sociedad y organización condiciona el comportamiento de sus miembros.

La socialización organizacional, en palabras de Schein, es la forma de "ponerse al tanto", el proceso de adoctrinamiento y adiestramiento, en el cual se señala lo que es importante en una organización o en alguna parte de la misma.

La socialización es un proceso, que a pesar de su continua presencia, resulta fácil pasarlo por alto. Sin embargo, puede hacer o deshacer una carrera y los planes del personal en una organización. La rapidez y eficacia de la socialización determinan la lealtad, el compromiso, la productividad de los empleados, así como su permanencia o salida. La estabilidad y eficacia de las organizaciones dependerán de la habilidad que tengan éstas para socializar a sus componentes.

La socialización de la plataforma BLOGPROF y otras herramientas sociales se realizo a travez del curso de capacitación.

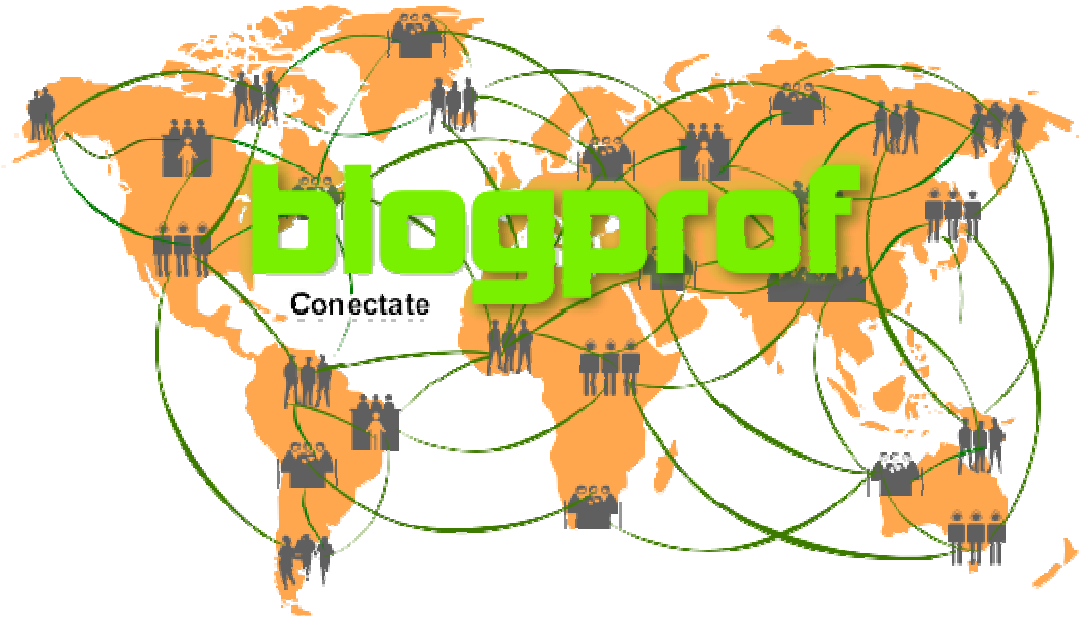

Figura 79. BLOGPROF

IV. CAPITULO IV : CONCLUSIONES Y RECOMENDACIONES

IV.1 Conclusiones

El trabajo esta en su etapa final de desarrollo, resta la incorporación de los procedimientos y herramientas que posibilitarán al usuario el ingreso de consultas avanzadas. Estas consultas le permitirán extraer del almacén de datos la información que responde al criterio de selección, formato y presentación, elegido por el mismo. Esta posibilidad, incrementará la independencia del sujeto que requiere la información, logrando mayor eficiencia en el proceso de la toma de decisiones.

Las etapas cumplidas permitieron, en el aspecto profesional, incrementar los conocimientos sobre las Web 2.0, reconocer y apreciar las ventajas y mejorar el aprovechamiento de nuevas herramientas software para el tratamiento de los datos, disponibles en el mercado.

A nivel de aplicación tecnológica, se pretende contribuir a facilitar el acceso a la información para la toma de decisiones en un ambiente de organización gubernamental, con escasa infraestructura de recursos informáticos y poco desarrollo en la sistematización de los datos.

La Web 2.0 deja la creación y edición de los contenidos en las manos de los usuarios finales, el arquitecto de información ya no tiene el control por sobre la organización detallada de la información. En este contexto, el arquitecto de información se enfoca en establecer las reglas de participación y las interfaces en tal forma que motiven a los usuarios a formar parte activa de estos sistemas. los usuarios siempre encontrarán formas de extender las bases del sistema para crear nuevas formas de participación.

Las aplicaciones que la Web 2.0 ofrece al usuario son infinitas, desde *weblogs*, sitios como Flickr o Myspace donde se comparten juegos, vídeos, fotografías; se difunden noticias mediante *podcasting* (archivos de sonido que se descargan y se pueden escuchar y reproducir en cualquier momento); se coeditan de forma participativa enciclopedias en línea, como Wikipedia; se añaden textos o *tags* (la llamada folksonomia); hasta las fotografías, música, etc., que se cuelgan en la Red, para que otros las puedan buscar en sitios *web* como Del.icio.us, etc. La lista puede no acaba nunca, ya que cada día nacen nuevas aplicaciones.

Esta enorme actividad está cambiando el mundo de las telecomunicaciones, de los medios de comunicación, del marketing, del software, del entretenimiento, etc.

IV.2 Recomendaciones

Ante el actual *boom* de implementaciones de nuevas ideas colaborativas y experimentos de comunicación, no pocas veces se nos pide realizar un proyecto "**que sea Web 2.0**". Pues bien, para cumplir con lo básico de

esa especificación tan de moda pero que pocos entienden, se propone como recomendación tomar en cuenta los siguientes pautas:

1. Centrarse en un mercado concreto pero con aspiraciones internacionales
2. Tu proyecto debe estar en constante producción y debes ir añadiendo funciones que atrapen usuarios que quieras como mercado.
3. Focalizacion del producto, como se indico en el punto 1 debes centrarte en un mercado concreto
4. Al realizar tu proyecto, debes pensar como usuario, como quisieras que funcione, que te gustaría que te permitiera hacer la aplicación.