UNIVERSIDAD DE TALCA
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA CIVIL EN COMPUTACIÓN

Desarrollo de aplicación prototipo para el proceso de planificación del Departamento de Ciencias de la Computación.

PABLO IGNACIO JARA CONTRERAS

Profesor Guía: BENJAMÍN INGRAM

Memoria para optar al título de
Ingeniero Civil en Computación

Curicó – Chile
Noviembre, 2016

CONSTANCIA

La Dirección del Sistema de Bibliotecas a través de su encargado Biblioteca Campus Curicó certifica que el autor del siguiente trabajo de titulación ha firmado su autorización para la reproducción en forma total o parcial e ilimitada del mismo.

Curicó, 2019

Dedicado a mis padres, que a pesar de todas las dificultades, siempre han estado ahí para apoyarme.

AGRADECIMIENTOS

Quiero agradecer a mis padres por todo el apoyo que me han otorgado y los grandes esfuerzos que han realizado para darme la oportunidad de estudiar y de llegar hasta aquí.

A mis amigos y compañeros, en donde prefiero no escribir nombres, porque es mas que seguro que olvidaré a alguno y no deseo que me cobren sentimientos. A todos ellos con los cuales tuve la oportunidad de compartir una infinidad de experiencias y aprendizajes, en donde pasamos buenos y malos momentos, quienes me apoyaron e hicieron todo este proceso mucho mas grato y me enseñaron que también hay que estudiar de vez en cuando !! Esta etapa sin ellos, sin duda no hubiera sido lo mismo.

A los docentes, los cuales me hicieron trasnochar y calentarme la cabeza durante este laaargo proceso, pero con quienes también compartí divertidos momentos y quienes a la vez, todos, en sus distintas maneras, fueron capaces de traspasar sus conocimientos.

Finalmente, agradecer a mi Profesor Guia, Ben Ingram, por toda su paciencia, apoyo y buena onda, con la cual hizo gratas una gran cantidad de reuniones, su enorme disposición para trabajar y alentarme en todo momento a continuar este proceso, en la cual muchas veces afloje, pero siempre estuvo ahí motivándome a continuar y me demostró que podía contar con su apoyo.

RESUMEN

En la Universidad, el proceso de gestión y planificación de módulos y profesores para un determinado período académico es un proceso bastante tedioso, que debe ser realizado por el Director de Departamento correspondiente. Este proceso puede tomar grandes cantidades de tiempo, esto, debido a que existe una gran cantidad de detalles que hay que tener en consideración para su correcta su elaboración.

En base a esta problemática, surge la idea desarrollar un prototipo funcional que pueda ser utilizado como herramienta para ayudar en esta labor.

A lo largo de este documento se dan a conocer las objetivos principales de la herramienta a desarrollar, la metodología para su construcción, lo cual incluye un análisis del contexto y requerimientos del sistema, diseño de la arquitectura necesaria para el desarrollo del prototipo y finalmente los pasos realizados para su construcción.

TABLA DE CONTENIDOS

página

ÍNDICE DE FIGURAS

página

ÍNDICE DE TABLAS

página

1. Introducción

1.1. Descripción de la propuesta

En la actualidad el proceso de planificación en el departamento de ciencias de la computación es llevado a cabo de manera manual, es decir, se trabaja con diversos documentos (principalmente plantillas excel), el encargado de esta tarea tiene que buscar y/o contactar a cada profesor, ver su carga académica y preguntarle si esta dispuesto a dar un curso por una cierta cantidad de horas. Si esto ocurre, posteriormente viene un proceso complejo que implica balancear las horas de cada profesor, ver que los cursos de distintos planes no se solapen y también calcular no salirse del presupuesto (entre otras tareas). La realización de todas estas actividades es un proceso bastante engorroso y conlleva a una gran cantidad de tiempo. Es por esta razón que surge la idea de implementar un sistema web que facilite la realización de estas tareas y a la vez ayude a disminuir los errores, para de esta forma contar con una herramienta más automatizada para el proceso de planificación.

1.1.1. Contexto del proyecto

El proceso de planificación de cursos y profesores en la universidad se puede resumir de la siguiente manera: Existen varios directores de escuelas, son ellos quienes hacen las solicitudes de cursos al director de departamento de ciencias de la computación. Es este director el responsable de buscar y contratar a dichos profesores para que dicten estos cursos. Al momento de realizar estos contratos se deben de seguir ciertos procedimientos, puesto que existen 2 tipos de profesores: Los que ya han sido

contratados y los profesores nuevos. Si el profesor es nuevo hay que solicitar una serie de antecedentes y documentos.

Si el profesor está dispuesto a trabajar por cierta cantidad de horas, se procede a realizar el contrato, pero hay que tener especial cuidado de no salirse del presupuesto disponible por el departamento. También hay que prestar atención al plan de estudio, puesto que, en la universidad, la malla curricular de una carrera puede ir variando y a esta se le pueden ir agregando, modificando o eliminando cursos. Para conocer de cual malla se está hablando, se le asocia un Plan. Ej: Plan 44, Plan 11, Etc...

En conclusión, lo que se requiere es construir un sistema prototipo web para ayudar a facilitar todo el seguimiento de este proceso, para así agilizar la planificación de cursos y minimizar el riesgo de errores de este proceso.

1.1.2. Trabajo relacionado

En la universidad existe un sistema hecho por el DTI para subir los datos de la planificación una vez que esta se encuentra lista. Hay que mencionar que estos sistemas no tendrán interacción alguna. Pero si facilitará al director de departamento a subir esta información a dicho sistema, puesto que se encontrarán todos los datos necesarios a la vista en la herramienta web.

1.1.3. Definición del problema

Como se indica en el inicio de la propuesta, actualmente, en el departamento de ciencias de la computación no se cuenta con un sistema que ayude a realizar las tareas de planificación. Es por esta razón que surge la idea de implementar una herramienta que apoye la gestión de estas.

Lo que me motiva a realizar este sistema es que el desarrollo de esta herramienta me permitirá poner en práctica los conocimientos adquiridos durante mi transcurso en la carrera, y a la vez, obtener experiencia realizando sistemas web, lo cual, lo veo como un buen desafío, que de llevarlo a cabo puede ayudarme para en un futuro contar con los conocimientos necesarios para desenvolverme en este ámbito.

1.1.4. Propuesta de solución

Se propone implementar un sistema web que ayude a gestionar y agilizar las tareas principales del proceso de planificación del departamento de ciencias de la computación, para que de esta manera los procesos llevados a cabo no sean tan engorrosos, ni demanden tanto tiempo.

Se propone construir esta herramienta en base a los siguientes pasos:

1. Entrevista con el cliente para capturar los requisitos.
2. Definir herramientas a utilizar para la construcción del software
3. Modelamiento e implementación de base de datos a partir de los puntos anteriores.
4. Creación de interfaces.
5. Desarrollo de sistema prototipo web e implementación de las funcionalidades del sistema.

1.1.5. Hipótesis

- Se puede desarrollar un sistema prototipo que ayude a gestionar las tareas concernientes a la planificación de módulos y profesores de un determinado departamento.

1.2. Objetivos

1.2.1. Objetivo general

- Implementar un prototipo web que posea las cualidades para realizar el proceso de planificación del departamento de ciencias de la computación.

1.2.2. Objetivos específicos

- Utilizar un *framework* para agilizar el proceso de desarrollo.
- Permitir a los usuarios del tipo Director de Escuela entregar solicitudes al departamento en forma automática

- Administrar a los profesores y sus cargas académicas.
- Administrar los planes y módulos del departamento de ciencias de computación.
- Permitir evaluar el presupuesto.
- Contar con un sistema capaz de generar contratos a los profesores

1.2.3. Alcances

- La aplicación prototipo estará diseñada para cubrir las necesidades del Director de Departamento Ben Ingram. Por ende, no cubrirán necesariamente las necesidades que pueda requerir un posterior Director de Departamento.
- La aplicación prototipo estará diseñada solo para el Departamento de Ciencias de la Computación.
- El diseño de la aplicación se encuentra ligado para la estructura orgánica del momento.
- La aplicación prototipo será desarrollada solo en version web.

2. Marco teórico

2.1. Introducción

En esta sección se dan a conocer todos los conceptos necesarios para la compresión de este proyecto.

2.2. Estructura orgánica de la Universidad

La estructura Orgánica de la Universidad de Talca se encuentra conformada por las siguientes unidades [3]:

- Rectoría, que estará a cargo del Rector.
- Vicerrectorías a cargo de un Vicerrector.
- Secretaría General a cargo del Secretario General.
- Contraloría a cargo del Contralor.
- Facultades a cargo de un Decano.
- Institutos a cargo de un Director.
- Departamentos Académicos a cargo de un Director.
- Escuelas a cargo de un Director.
- Direcciones, Centros y Programas a cargo de un Director.
- Departamentos Administrativos a cargo de un Jefe de Departamento.
- Sección Administrativa a cargo de un Jefe de Sección.

2.2.1. Organización Académica

La universidad para el desarrollo de sus actividades académicas, se encuentra organizada en Facultades, Institutos, Departamentos, Escuelas, Centros y Programas. Para el correcto entendimiento del desarrollo de este proyecto es necesario dar a conocer las siguientes partes:

- Facultad: Unidad académica que, en conformidad con el Estatuto y las Ordenanzas de la Universidad, agrupan a un cuerpo de personas asociadas con el propósito de enseñar e investigar en áreas afines del conocimiento superior. Una Facultad estará dirigida por un Decano [3].

- Departamento Académico: Unidad en la cual se llevan a cabo las funciones propias de la Universidad. Se organiza en torno a un área del conocimiento cuya dimensión epistemológica ha sido establecida por la Facultad a la que pertenece [3].

  Estará a cargo de un Director de Departamento, el que dependerá jerárquicamente del Decano.

- Escuela: Es la unidad básica de administración de uno o más programas docentes a fines de pre-grado. Estará a cargo de un Director de Escuela, el cual dependerá jerárquicamente del Decano[3].

2.2.2. Facultades de la Universidad de Talca

En su conjunto, la Universidad de Talca cuenta con las siguientes Facultades:

- Facultad de Ciencias Agrarias.
- Facultad de Ciencias Empresariales.
- Facultad de Ciencias Forestales.
- Facultad de Ciencias Jurídicas y Sociales.
- Facultad de Ciencias de la Salud.
- Facultad de Ingeniería.
- Facultad de Psicología.

2.2.3. Facultad de Ingeniería

La Universidad de Talca cuenta con cinco campus ubicados en las ciudades de Talca, Curicó, Santa Cruz (Colchagua), Santiago y Linares. Estos campus engloban distintas facultades que abarcan amplios ámbitos de las ciencias. En el Campus de Curicó se encuentra la Facultad de Ingeniería la cual, le pertenecen los siguientes Departamentos:

- Ciencias de la Computación.
- Ciencias Aplicadas.
- Tecnologías Industriales
- Modelación y Gestión Industrial
- Ingeniería y Gestión de la Construcción

Es en este Campus el que concentra las diversas carreras de ingeniería y entre ellas, la de Ingeniería Civil en Computación.

2.2.4. Departamento de Ciencias de la Computación

El Departamento de Ciencias de la Computación se encuentra compuesto actualmente por 10 académicos junto a su respectivo Director de Escuela y Director de Departamento.

2.2.4.1. Director de Departamento

El Director de Departamento es el responsable de liderar el desarrollo y gestión académica de su unidad. Las funciones del Director son las siguientes [3]:

a) Representar al Departamento en todas las instancias internas y externas a la Universidad, que le corresponda participar en razón de su cargo.

b) Proponer al Decano, un Plan de Desarrollo de su Unidad, que comprenda el período para el cual fue elegido.

c) Programar las acciones docentes, de pre y postgrado, investigación y extensión y dar cuenta anual al Decano de la marcha del Departamento.

d) Convenir el Compromiso de Desempeño anual, con cada uno de los académicos adscritos a la Unidad de proceder a su evaluación

e) Formular y ejecutar el presupuesto

f) Participar en el Consejo de Facultad.

g) Designar representantes o participar en el Consejo de Escuela y en toda otra unidad que los reglamentos de la Universidad así lo estipulen.

2.2.4.2. Director de Escuela

El Director de Escuela es el académico, que con dedicación preferente, está encargado de la gestión del plan de estudios de las Carreras a su cargo. Es designado por el Rector a proposición del Decano y dependerá Jerárquicamente de éste [3].

Son funciones del Director de Escuela:

a) Velar por el correcto cumplimiento del plan de estudios, la calidad de la docencia y los servicios a los estudiantes, de la Carrera que dirige.

b) Solicitar oportunamente al Director de Departamento o Instituto, la asignación de profesores necesarios para cubrir la docencia de la(s) Carrera(s) que dirige.

c) Programar y calendarizar todas las actividades docentes de cada período lectivo.

d) Controlar el cumplimiento de las actividades docentes programadas en cada asignatura

e) Aplicar la normativa vigente en materias de su competencia a los estudiantes de su Escuela

f) Mantener un archivo actualizado de planes de estudios, programas de asignaturas de la Carrera y las carpetas académicas de cada uno de los alumnos.

g) Solicitar a los profesores de la asignatura, el horario de atención de alumnos fijados por cada profesor y comunicarlo.

h) Solicitar a las unidades académicas y administrativas correspondientes, los servicios de apoyo requeridos por los estudiantes.

i) Apoyar las actividades estudiantiles, culturales y recreativas de los estudiantes de la Carrera a su cargo, que hayan sido aprobadas oficialmente por la Universidad

j) Administrar el presupuesto asignado

k) Velar por la aplicación de instrumentos de evaluación de la docencia y requerir los informes correspondientes.

l) Representar a la Escuela en las instancias y comisiones relacionadas con la administración curricular

m) Coordinar el proceso de homologación de asignaturas e informar a Registro Académico, Dirección de Desarrollo Docente y Tesorería, el retiro temporal y postergación de estudios que hagan efectivos los alumnos de la carrera.

n) Coordinar las prácticas estivales, profesionales, internados y actividades extramurales de los alumnos.

ñ) Colaborar con la Facultad en las actividades de titulación tales como memoria, tesis, examen de grado, examen de título

o) Confeccionar los expedientes de título

p) Atender a los estudiantes de la Carrera en materias relativas a su orientación académica y profesional.

q) Colaborar con la promoción de la Carrera en las actividades organizadas por la Vicerrectoría de Asuntos Estudiantiles y la Universidad.

r) Mantener un sistema de información actualizada de los ex-alumnos de la Escuela.

s) Realizar anualmente una evaluación general de la Escuela y presentarla al Decano.

t) Dirigir el Consejo de Escuela.

u) Participar en el Consejo De Docencia.

2.2.5. Tareas de Planificación

Las tareas de planificación consisten básicamente en asegurar que los cursos solicitados por los Directores de Escuela de las siguientes carreras:

- Ingeniería Civil en Bioinformática
- Ingeniería Civil en Computación
- Ingeniería Civil Industrial
- Ingeniería Civil de Minas
- Ingeniería Civil Mecánica
- Ingeniería Civil Mecatrónica
- Ingeniería Civil en Obras Civiles

sean impartidos, para ello, el Director de Departamento es quien tiene el rol de proveer de los profesores y cursos necesarios, para esto se deben de realizar diversos tipos de tareas que se encuentran relacionadas con los profesores, cursos y carreras.

2.2.5.1. Planificación de Módulos y Profesores

Este proceso consiste básicamente en recibir las solicitudes realizadas por los Directores de Escuela, solicitudes las cuales contienen los módulos que deben ser dictados durante el próximo período de clases, y asignar estos módulos a los profesores pertenecientes al Departamento de Ciencias de la Computación.

En la tarea de asignación, hay que tener especial atención, puesto que los Módulos pertenecen a un Plan y una Carrera puede tener múltiples Planes. Debido a esta relación entre Plan y Módulo es que se da el caso de que un Módulo puede tener el mismo nombre que otro (pertenecen a distintos planes), pero no necesariamente tendrá el mismo contenido y cantidad de créditos. También existe la posibilidad de que 2 Módulos tengan distintos nombres, pero que esta vez si compartan las mismas características (contenido o cantidad de créditos). Si se tiene bien identificado la Carrera Y el Plan al cual pertenecen estos módulos se puede realizar el proceso de asignar esta carga académica a un Profesor.

Los Profesores de Planta o jornada completa de la Universidad deben de cumplir con un determinado número de horas de docencia, cuando la asignación de módulos a estos profesores genera un exceso o sobrecarga en estas horas, es decir los profesores con los cuales se cuenta en el Departamento no da abasto para cubrir con el número de horas requeridas por las solicitudes recibidas, se debe de contratar a los Profesores necesarios para dictar los módulos faltantes. A estos Profesores contratados se les llama Profesor Honorario y se debe de verificar que pueda cumplir con el número de horas que solicita el Módulo, puesto que el contrato que se genera, es un contrato por horas. Al momento de realizar estos contratos hay que tener cuidado de salirse del presupuesto establecido por el Departamento.

La Figura 2.1 representa de manera simple el proceso de Planificación de módulos y profesores.

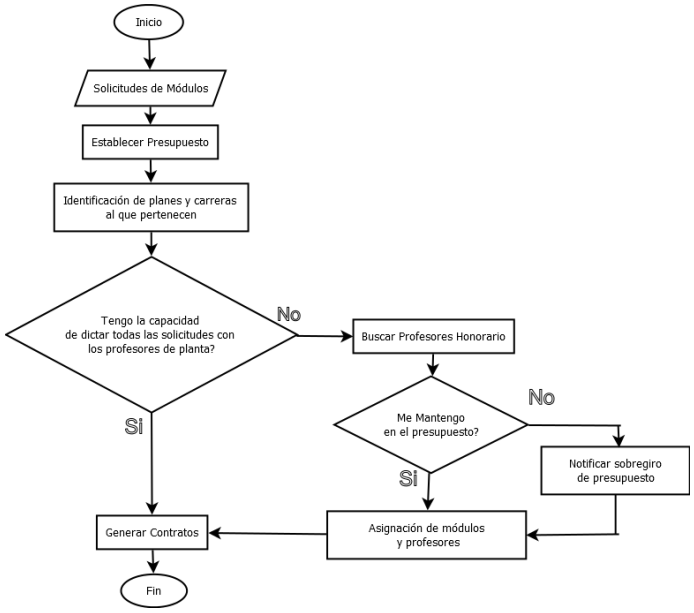

Figura 2.1: Diagrama de Flujo- Planificación Módulos y Profesores.

2.3. Sistema Web

Un sistema web, es un software que se encuentra instalado en un servidor web y al igual que una página web, se puede acceder a él mediante el uso de un navegador[1].

[1]Ejemplos de navegador web: Google Chrome,Firefox, Internet Explorer

La diferencia entre un sistema y una página web, radica en que el sistema tiene funcionalidades que permiten procesar y mostrar información de forma dinámica. En cambio, una página web se limita solamente a mostrar texto e imágenes o algún tipo de información.

La principal ventaja o característica de un sistema web es que son considerados prácticos, debido a que este no necesita ser instalado sobre una plataforma o sistema operativo, y que para acceder a él, sólo se necesita un navegador web, lo cual lo vuelve independiente y ligero.

2.4. Metodologías de desarrollo de software

Las Metodologías de Desarrollo de Software surgen ante la necesidad de utilizar un conjunto de procedimientos, técnicas, y/o herramientas para llevar a cabo el desarrollo de un proyecto de software. Dentro del proceso de la ingeniería de software, existen 2 tipos de metodologías; las Metodologías Tradicionales y las Metodologías Ágiles.

2.4.1. Metodología Tradicional

Las Metodologías Tradicionales se centran principalmente en definir y documentar detalladamente cada uno de los procesos, herramientas, artefactos[2] y tareas a realizar para llevar a cabo el desarrollo de un software, es decir, se pone especial énfasis en lo que es la planificación total del trabajo a realizar. Una vez que se encuentra todo documentado, se procede a iniciar el ciclo de desarrollo del software.

La interacción entre cliente y desarrollador no es muy estrecha a lo largo del proyecto, puesto que una vez que se definen los requerimientos o funcionalidades que tendrá el software, se entregará un informe con estos. Este informe, de estar completo y ambas partes estar de acuerdo, hará alusión a una especie de contrato, el cual definirá el qué es lo que hará y no el software como producto final. Con el contrato ya definido se da inicio al proceso de construcción del software y de aquí en adelante la participación del cliente es prácticamente nula, ya que el cliente

[2]Artefactos: Es un producto resultante del proceso de desarrollo de software (casos de uso, diagramas de actividad, diagramas de clases, especificación de requisitos, etc...)

solamente se reúne con parte del equipo desarrollador mediante reuniones esporádicas para obtener *feedback* por parte de los desarrolladores y viceversa.

Este tipo de metodologías, al centrarse especialmente en lo que es el control de los procesos y la documentación, no son buenas para adaptarse adecuadamente a los cambios, por lo que estos métodos no se recomiendan cuando se trabaja en un entorno donde los requisitos no pueden predecirse o bien pueden variar.

2.4.2. Metodología Ágil

Las Metodologías de desarrollo Ágil están orientadas a proyectos pequeños o de mediana envergadura, en los cuales el equipo de desarrollo no sobrepasa las 10 personas. Estas metodologías destacan por ser iterativas e incremental. Iterativas porque se tiene como finalidad el poder responder y ser flexible ante algún cambio en un requisito del software. E incremental porque se deben de realizar pequeñas entregas, en ciclos cortos de tiempo. En este tipo de metodologías prevalece la cooperación entre desarrollador y cliente, ya que se da más importancia a construir un producto funcional que escribir una documentación exhaustiva.

No se recomienda este tipo de metodología cuando el software a construir es un sistema crítico en los que es necesario contar con un análisis detallado de los requerimientos del sistema.

A continuación se muestra una tabla que muestra las principales diferencias entre las Metodologías Tradicionales y Ágiles [8].

Metodologías Tradicionales	Metodologías Ágiles
Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo	Basadas en heurísticas provenientes de prácticas de producción de código
Cierta resistencia a los cambios	Especialmente preparados para cambios durante el proyecto
Proceso mucho más controlado, con numerosas políticas/normas	Proceso menos controlado, con pocos principios.
El cliente interactúa con el equipo de desarrollo mediante reuniones	El cliente es parte del equipo de desarrollo
Más artefactos	Pocos artefactos
Más roles	Pocos roles
Grupos grandes y posiblemente distribuidos	Grupos pequeños (10 integrantes) y trabajando en el mismo sitio
La arquitectura del software es esencial y se expresa mediante modelos	Menos énfasis en la arquitectura del software
Existe un contrato prefijado	No existe contrato tradicional o al menos es bastante flexible

2.4.3. Historias de usuario

Dentro de las metodologías ágiles, una de las técnicas más utilizadas para especificar cuáles serán los requisitos de software son las Historias de Usuario. Esta técnica consiste en describir brevemente las características que el sistema debe poseer. Wake, W.C[13] propone utilizar un nombre y una descripción. En cambio en el libro de Beck[10] se presenta un ejemplo mucho más completo, el cual propone los siguientes atributos: fecha, tipo de actividad (nueva, corrección, mejora), prueba funcional, número de historia, prioridad técnica y del cliente, referencia a otra historia previa, riesgo, estimación técnica, descripción, notas y una lista de seguimiento con la fecha, estado cosas por terminar y comentarios. El primer paso para comenzar a construir una Historia de Usuario consiste en identificar los distintos roles que habrá en el sistema, posterior a esto se le debe de dar un formato, una manera estándar para construir el enunciado es:
Como *rol* quiero *requerimiento* para poder *beneficio.*

Ejemplo: Como *administrador* quiero *registrar todos los profesores y sus cursos* para *tener un control de los cursos que han dictado.*

Para que una historia de usuario cumpla correctamente con su función, debe tener las siguientes características:

- Independientes: Deben ser atómicas en su definición. Es decir, se debe intentar que no dependa de otras historias para poder completarla.
- Negociables:Deben ser ambiguas en su enunciado para poder debatirlas, dejando su concreción a los criterios de aceptación.
- Valoradas: Deben ser valoradas por el cliente. Para poder saber cuanto aporta al Valor de la aplicación y junto con la estimación convertirse en un criterio de prioridad.
- Estimables: Aunque sea siempre un poco como leer de una bola de cristal, deben poder ser estimadas. Tener su alcance lo suficientemente definido como para poder suponer una medida de trabajo en la que pueda ser completarla.
- Pequeñas. Para poder realizar una estimación con cierta validez y no perder la visión de la Historia de Usuario, se recomienda que sean mayores de dos días y menores de dos semanas.
- Verificables: Este es el gran avance de las Historias de Usuario. Que, junto con el cliente, se acuerdan unos Criterios de Aceptación que verifican si se ha cumplido con las funcionalidades descritas y esperadas.[11]

2.5. Arquitectura Cliente-Servidor

La arquitectura Cliente-Servidor es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes.Un cliente realiza peticiones a otro programa, el servidor, que le da respuesta. Esta comunicación se puede ver reflejada en la siguiente Figura 2.2

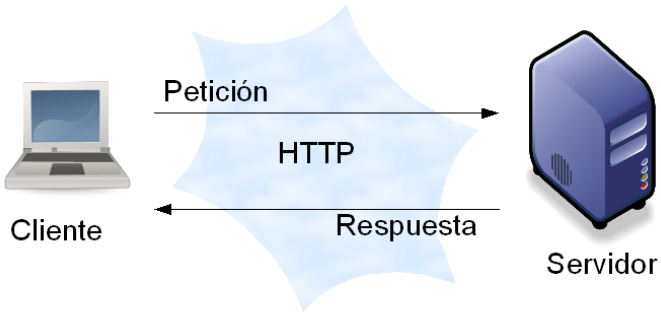

Figura 2.2: Comunicación Cliente-Servidor.

La separación entre cliente y servidor es una separación de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un solo programa. Los tipos específicos de servidores incluyen los servidores web, los servidores de archivo, los servidores del correo, etc. Mientras que sus propósitos varían de unos servicios a otros, la arquitectura básica seguirá siendo la misma[1].

Las características del cliente en una arquitectura Cliente-Servidor son las siguientes:

- Es el que inicia solicitudes o peticiones. Tiene, por tanto, un papel activo en la comunicación (dispositivo maestro o amo).
- Espera y recibe las respuestas del servidor.
- Por lo general, puede conectarse a varios servidores a la vez.
- Normalmente, interactúa directamente con los usuarios finales mediante una interfaz gráfica de usuario
- Al contratar un servicio de red, se debe de tener en cuenta la velocidad de conexión que se le otorga al cliente y el tipo de cable que utiliza.

El receptor de la solicitud enviada por el cliente se conoce como servidor. Sus características son:

- Al iniciarse espera a que le lleguen las solicitudes de los clientes. Desempeñan entonces un papel pasivo en la comunicación (dispositivo esclavo).

- Tras la recepción de una solicitud, la procesan y luego envían la respuesta al cliente.
- Por lo general, aceptan conexiones desde un gran número de clientes (en ciertos casos el número máximo de peticiones puede estar limitado).
- No es frecuente que interactúen directamente con los usuarios finales.

La mayoría de los servicios de Internet son tipo de cliente-servidor. La acción de visitar un sitio web requiere una arquitectura cliente-servidor, ya que el servidor web sirve las páginas web al navegador (cliente). Al leer algún artículo en una página web, el computador y el navegador web del usuario serían considerados un cliente; y los computadores, las bases de datos y los usos que componen la página web serían considerados el servidor. Cuando el navegador web del usuario solicita un artículo particular de alguna página web, el servidor de esta página recopila toda la información a mostrar en su base de datos, la articula en una página web y la envía de nuevo al navegador web del cliente.

2.6. Modelo-Vista-Controlador

El patrón de arquitectura MVC (Modelo Vista Controlador) es un patrón que define la organización independiente del Modelo (Objetos de Negocio), la Vista (interfaz con el usuario u otro sistema) y el Controlador (controlador del *workflow* de la aplicación).

2.6.1. La Capa del Modelo

El modelo representa la parte de la aplicación que implementa la lógica de negocio. Esto significa que es responsable de la recuperación de datos convirtiéndolos en conceptos significativos para la aplicación, así como su procesamiento, validación, asociación y cualquier otra tarea relativa a la manipulación de dichos datos.

A primera vista los objetos del modelo pueden ser considerados como la primera capa de la interacción con cualquier base de datos que podría estar utilizando la aplicación. Pero en general representan los principales conceptos en torno a los cuales se desea implementar un programa.

2.6.2. La Capa de la vista

Las vistas son las que interactúan con los usuarios y están encargadas de la representación de los datos, contenidos en el modelo, al usuario. La relación entre las vistas y el modelo son de muchas a uno, es decir cada vista se asocia a un modelo, pero pueden existir muchas vistas asociadas al mismo modelo.

2.6.3. La Capa del Controlador

El controlador es el encargado de interpretar y dar sentido a las instrucciones que realiza el usuario, realizando actuaciones sobre el modelo. Si se realiza algún cambio, comienza a actuar, tanto si la modificación se produce en una vista o en el modelo. Interactúa con el Modelo a través de una referencia al propio Modelo.

El flujo que sigue el patrón MVC es generalmente el que puede apreciar en la Figura 2.3

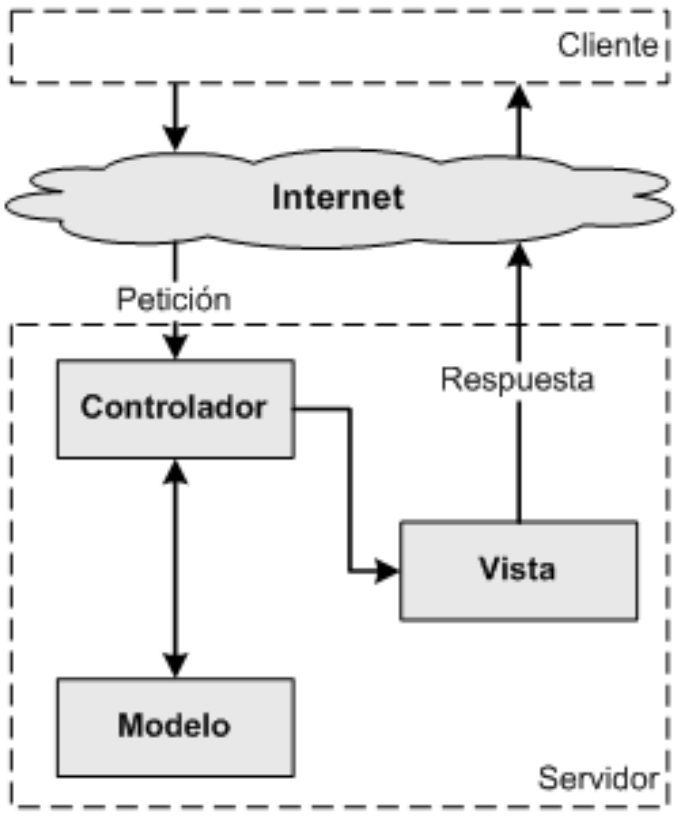

Figura 2.3: Representación patrón MVC.

1. El cliente realiza una acción mediante la interfaz y envía una petición al controlador.

2. el controlador trata el evento de entrada.

3. El controlador notifica al modelo la acción del usuario, lo que puede implicar un cambio del estado del modelo (si no es una mera consulta).

4. se genera una nueva vista. La vista toma los datos del modelo.

5. La interfaz de usuario espera otra interacción del usuario, que comenzará otro nuevo ciclo.

2.7. Framework

Un *Framework* o marco de trabajo, es un conjunto de componentes reutilizables y configurables de software (archivos, clases, librerías, Etc...) que nos sirven como estructura base para llevar a cabo el desarrollo de un software.
El objetivo principal de un *framework* consiste en facilitar y acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas como lo es el uso de patrones. Explicándolo de otra manera podríamos decir que si la tarea de desarrollar un software consiste en armar un rompecabezas, el *framework* vendría siendo partes del rompecabezas ya armadas.

A continuación, se mencionan algunos *frameworks* web que son utilizados hoy en día:

- CakePhp: Es un *framework* rápido para PHP[3], libre, de código abierto. Se trata de una estructura que sirve de base a los programadores para que éstos puedan crear aplicaciones Web [6].

- Ruby on Rails: Es un *framework* de aplicaciones web de código abierto escrito en el lenguaje de programación Ruby. Rails fue creado en 2003 por David Heinemeier Hansson y desde entonces ha sido extendido por el *Rails core team*, más de 2.100 colaboradores y soportado por una extensa y activa comunidad[7].

[3]PHP:*Hypertext Preprocessor* es un lenguaje de código abierto utilizado para el desarrollo web

- Bootstrap: Es un *framework*que permite crear interfaces web con CSS y Javascript que adaptan la interfaz dependiendo del tamaño del dispositivo en el que se visualice de forma nativa, es decir, automáticamente se adapta al tamaño de un ordenador o de una Tablet sin que el usuario tenga que hacer nada, esto se denomina diseño adaptativo o *Responsive Design.*[5]

- Django: Es un *framework* web de código abierto escrito en Python que permite construir aplicaciones web.[2] Fue desarrollado en origen para gestionar varias páginas orientadas a noticias de la *World Company* de Lawrence, Kansas, y fue liberada al público bajo una licencia BSD[4] en julio de 2005;

2.7.1. Resumen

En este capítulo se describieron las partes necesarias para comprender el contexto del proyecto, se vieron conceptos fundamentales tales como; Qué es un Sistema web, la estructura orgánica de la Universidad, se explicó qué es un *framework*, y una Historia de Usuario, entre otros conceptos.

En el siguiente Capítulo daré a conocer el Diseño de la Solución del sistema.

[4]Licencia BSD: es una licencia de software permisiva que permite el uso del código fuente en software no libre

3. Diseño de la solución

3.1. Introducción

Este capítulo contiene el detalle de 2 de las 3 etapas principales del proyecto de ingeniería de software; especificación de requisitos, diseño del software.

3.2. Análisis del problema

El principal inconveniente del proceso de planificación de módulos del departamento de ciencias de la computación es que actualmente no existe ninguna herramienta, ni procedimiento definido para llevarla a cabo. Debido a que el cargo de director de departamento dentro de la facultad de ingeniería va rotando cada 2 años, se hace más difícil llevar a cabo un seguimiento de los profesores, módulos y secciones que han sido dictados en el transcurso del tiempo, puesto que cada director realiza este proceso de planificación a su manera o como mejor le acomode. Este proceso demanda una gran carga de trabajo para el director que se encuentra a cargo y generalmente se comienza a realizar con un mes de anticipación de la finalización del semestre que se está cursando.

Las falencias principales que conlleva la no existencia de una herramienta y que dificultan el proceso de planificación son las siguientes:

- Registro de Profesores: No se cuenta con un registro efectivo de los profesores que trabajan para la facultad. De esta manera es difícil conocer quiénes son los profesores de jornada completa u honorario. Por ende, se dificulta el proceso de asignación de módulos.

- Identificación de Módulos espejo: debido a que dentro de una carrera existen diferentes planes (mallas curriculares), hay módulos que son lo mismo entre sí, pero tienen distinto nombre y distinta cantidad de créditos. A pesar de estas diferencias, estos módulos son dictados como si fueran uno solo. La no identificación de estos módulos espejo puede llevar a errores en la planificación.
- Historial de módulos dictados: Contar con un historial de módulos dictados durante el semestre y a la vez conocer al profesor que ha dictado el módulo, agilizaría enormemente el proceso de planificación.
- Secciones dictadas por módulo: no se cuenta con un historial que permita conocer la cantidad de secciones que ha necesitado tener un módulo en particular.
- Manejo de Contratos: Al tener los registros de contratos en documentos o archivos por separado, trae dificultades al momento de no pasar a llevar el presupuesto disponible.
- Control de presupuesto: El cálculo de presupuesto se lleva a cabo cruzando los distintos documentos de contratos de manera manual. Tener esto de manera automatizada lograría disminuir el índice de errores.

Con el desarrollo de esta aplicación se desea cubrir estas falencias, agilizar el proceso de planificación y disminuir la carga de trabajo del director de departamento.

3.3. Metodología de trabajo

Sabemos que para el proceso de ingeniería de software existen diversas metodologías tradicionales y ágiles (ver 2.4), que de ser aplicadas sirven como un referente de buenas prácticas que ayudan en todo el proceso de construcción del proyecto. Para este proyecto en particular, no se escogerá ninguna metodología en específico, pero si se tomarán los principios y características de las metodologías ágiles (ver 2.4.2), es decir, las etapas de especificación, diseño y construcción del proyecto serán iterativas e incrementales. La razón principal de no escoger una metodología en particular es que será solo un desarrollador quien llevará a cabo el proyecto.

Para el proceso de captura y especificación de requisitos se realizan entrevistas semanales con el director de departamento de la escuela de ingeniería civil en computación, en cada una de estas reuniones se van capturando las necesidades del sistema. Los requisitos capturados son almacenados como historias de usuario y son corroborados en cada reunión, en caso de que un requisito cambie, este será modificado en la misma reunión. Finalmente, una vez que se logra tener una compresión global de lo que se espera del sistema se pasará a la etapa de diseño de software.

Al igual que en la etapa de especificación de requisitos, en la etapa de diseño se dará énfasis en la interacción con el cliente, promoviendo las reuniones semanales. Como indican los principios ágiles; no se producirán documentos a menos que sean necesarios de forma inmediata para tomar una decisión importante. Esta etapa se basará básicamente en escoger la arquitectura y herramientas adecuadas para el desarrollo del sistema y diseñar el modelo relacional de la base de datos.

Finalmente la etapa de construcción y validación de software se llevará a cabo en base a entregas pequeñas y modulares[1].El principio fundamental es producir rápidamente versiones del sistema que sean operativas, aunque no cuenten con toda la funcionalidad del sistema o del módulo. Estas iteraciones tendrán un rango de duración correspondiente a 2 o 3 semanas. En cada iteración el director de departamento estará a cargo de validar y dar la aprobación de la iteración.

3.4. Entorno de desarrollo

Con la finalidad de agilizar el proceso de desarrollo es necesario seleccionar un *framework* que permita al desarrollador el uso de herramientas como librerías y/o funcionalidades ya creadas para así utilizarlas directamente.

3.4.1. Ruby on rails

Ruby on Rails es un *framework* de aplicaciones web escrito en el lenguaje de programación Ruby, que utiliza el paradigma del patrón Modelo Vista Controlador (ver 2.6).

Algunas de las principales características de Ruby on Rails son:

[1]Los módulos del sistema son: Usuario, Modulo,Sección,Profesor y Contrato.

- Permite la creación y utilización de gemas, las cuales corresponden a librerías que pueden ser utilizadas dentro de un proyecto.
- Posee ActiveRecord que es una interfaz para la manipulación de objetos dentro de bases de datos que facilita la creación y el manejo de los objetos en la aplicación y en la base de datos.
- Posee ORM ó *Object-Relational-Mapping* es un sistema que permite corresponder los objetos con una base de datos de tipo relacional. Con ello, cada clase es representada por una tabla, cada atributo es representado por un campo y cada instancia se representa con un registro.
- Centrado en *Convention over configuration*, que se traduce en reglas de estructuración de directorios. Así, si conocemos estas reglas, cuando se necesite tocar algo en el código, sabremos rápidamente dónde localizar este código.

Por estas características y como motivación personal, debido a que no se tiene dominio del lenguaje ni del *framework* se ha decidido a optar por RoR como el *framework* de desarrollo para este proyecto.

3.4.2. Bootstrap

Es una colección de herramientas de software libre para la creación de sitios y aplicaciones web. Contiene plantillas de diseño basadas en HTML y CSS con tipografías, formularios, botones, gráficos, barras de navegación y además componentes de interfaz, así como extensiones opcionales de JavaScript. Este nos simplifica el proceso de creación de diseños web.

Para el desarrollo de este prototipo se utilizará un *template* de uso gratis llamado *Gentellela Admin* [4]. El cual se encuentra basado en Twitter Bootstrap e implementa HTML5, Css y Javascript, este se adaptó a las necesidades de la aplicación.

Figura 3.1: Gentellela Admin Template.

3.5. Requisitos funcionales

3.5.1. Roles

Los roles de las Historias de Usuario determinan los requerimientos de software por parte de los distintos usuarios del sistema. Los roles que participan en el sistema son los siguientes:

- Administrador o Director de departamento
- Director de escuela

3.5.2. Manejo de usuarios

Cuadro 3.1: Historia: Iniciar Sesión

ID	01
Historia	Iniciar Sesión
Como	Usuario (Administrador o Director de escuela)
Quiero	introducir mí email y mí contraseña
Para	tener acceso al sistema y sus funcionalidades

Cuadro 3.2: Historia: Cerrar Sesión

ID	02
Historia	Cerrar Sesión
Como	Usuario (Administrador o Director de Escuela)
Quiero	Presionar sobre la opción cerrar sesión
Para	abandonar el sistema

Cuadro 3.3: Historia: Registro de Usuario

ID	03
Historia	Registro de Usuario
Como	Usuario no registrado
Quiero	registrar una nueva cuenta
Para	tener acceso al sistema como algún tipo de usuario

3.5.3. Historias de usuario Administrador - Director de departamento

Cuadro 3.4: Historia: Activar Usuario

ID	04
Historia	Activar usuario
Como	Administrador
Quiero	Activar las cuentas de usuarios registrados en el sistema del sistema de forma individual
Para	generar un nuevo usuario funcional en el sistema

Cuadro 3.5: Historia: Bloquear Usuario

ID	05
Historia	Bloquear usuario
Como	Administrador
Quiero	Bloquear las cuentas de usuarios registrados en el sistema del sistema de forma individual
Para	que no pueda usar las funcionalidades del sistema en su cuenta de un usuario

Cuadro 3.6: Historia: Eliminar Usuario

ID	06
Historia	Eliminar Usuario
Como	Administrador
Quiero	eliminar los usuarios del sistema de forma individual
Para	sus datos no sean válidos para acceder al sistema

Cuadro 3.7: Historia: Manejo de permisos

ID	07
Historia	Manejo de permisos
Como	Administrador
Quiero	asignar roles a los usuarios con respecto a todo el sistema (Director de escuela, Director de departamento)
Para	cambiar los privilegios que cada usuario posee sobre el sistema

Cuadro 3.8: Historia: Mantener Carrera

ID	08
Historia	Mantener Carrera
Como	Administrador
Quiero	Registrar, Modificar y eliminar carreras
Para	Realizar mantención de las carreras en el sistema

Cuadro 3.9: Historia: Mantener Profesor

ID	09
Historia	Mantener Profesor
Como	Administrador
Quiero	Registrar,Modificar y eliminar profesores
Para	Realizar mantención de los profesores en el sistema

Cuadro 3.10: Historia: Mantener Módulo

ID	10
Historia	Mantener Módulo
Como	Administrador
Quiero	Registrar, Modificar y eliminar módulos
Para	Realizar mantención de los módulos en el sistema

Cuadro 3.11: Historia: Mantener Malla

ID	11
Historia	Mantener Malla
Como	Administrador
Quiero	Registrar, Modificar y eliminar malla
Para	Realizar mantención de las mallas en el sistema

Cuadro 3.12: Historia: Mantener Sección

ID	12
Historia	Mantener Sección
Como	Administrador
Quiero	Registrar, Modificar y eliminar secciones
Para	Realizar mantención de las secciones en el sistema

Cuadro 3.13: Historia: Mantener Presupuesto

ID	13
Historia	Mantener Presupuesto
Como	Administrador
Quiero	Registrar, Modificar y eliminar un presupuesto
Para	Definir el presupuesto con el cual se contará

Cuadro 3.14: Historia: Recibir Solicitud

ID	14
Historia	Recibir Solicitud
Como	Director de departamento
Quiero	Visualizar las solicitudes de los directores de escuela que me han sido enviadas
Para	Buscar y asignarles profesores a las solicitudes ó rechazarlas

Cuadro 3.15: Historia: Generar Contrato

ID	15
Historia	Generar Contrato a Profesor
Como	Administrador
Quiero	Generar un contrato que tenga todas los módulos, secciones y horas en las cuales será contratado el profesor
Para	Conocer en detalle el contrato

Cuadro 3.16: Historia: Generar Contrato

ID	16
Historia	Asignar Sección a Profesor
Como	Administrador
Quiero	Asignar una o más secciones a un profesor
Para	tener un registro de los módulos en los que trabajará

Cuadro 3.17: Historia: Filtrar Profesor

ID	17
Historia	Filtrar Profesor
Como	Administrador
Quiero	Filtrar a los profesores jornada completa y a honorarios
Para	conocer el tipo al que pertenece cada profesor

Cuadro 3.18: Historia: historial de módulos de profesor

ID	18
Historia	Historial de módulos de Profesor
Como	Administrador
Quiero	Visualizar el historial de los módulos que ha dictado un profesor
Para	conocer el tipo de módulos que dicta

Cuadro 3.19: Historia: Asignar Profesor responsable

ID	19
Historia	Asignar Profesor responsable
Como	Administrador
Quiero	Asignar la responsabilidad de un módulo a un profesor
Para	conocer al profesor responsable

Cuadro 3.20: Historia: Visualizar historial de profesor

ID	20
Historia	Filtrar módulos por malla
Como	Administrador
Quiero	Visualizar todos los módulos que ha dictado un profesor en los distintos años y semestres
Para	Conocer los módulos en los que ha trabajo

Cuadro 3.21: Historia: Asignar módulo Espejo

ID	21
Historia	Asignar módulo espejo
Como	Administrador
Quiero	asignarle a un módulo uno o más espejos
Para	conocer los módulos que son espejos de otros

3.5.4. Historias de usuario Director de escuela

Cuadro 3.22: Historia: Visualizar Módulos

ID	22
Historia	Visualizar Módulos
Como	Director de escuela
Quiero	Visualizar los Módulos pertenecientes a mí escuela
Para	Seleccionar los que deben ser dictados

Cuadro 3.23: Historia: Solicitar Sección

ID	23
Historia	Solicitar Sección
Como	Director de escuela
Quiero	solicitar una o más secciones de un módulo
Para	que éste se dicte

Cuadro 3.24: Historia: Visualizar estado de Solicitud

ID	24
Historia	Visualizar estado de Solicitud
Como	Director de escuela
Quiero	Visualizar el estado de mis solicitudes
Para	Conocer si estas han sido aceptadas o rechazadas

3.6. Product Backlog

Product Backlog, es un artefacto utilizado en las metodologías ágil de trabajo para la gestión de proyectos de desarrollo de software. Y que es, en líneas generales, es una lista ordenada u priorizada de las tareas que componen un proyecto de aplicación.

Con el objetivo de definir el orden en el cual se llevarán a cabo la implementación de las Historias de usuario (ver 3.5), se generado el siguiente *Product Backlog*:

Cuadro 3.25: Product Backlog

Id	Historia	Prioridad	Dependencia
08	Mantener Carrera	1	
11	Mantener Malla	2	08
10	Mantener Módulo	3	11
20	Filtrar Módulos por Malla	4	10
03	Registro de Usuario	5	
01	Iniciar Sesión	6	03
02	Cerrar Sesión	7	01
07	Manejo de permisos	8	03
04	Activar Usuario	9	07
05	Bloquear Usuario	10	07
06	Eliminar Usuario	11	03
12	Mantener Sección	12	10
09	Mantener Profesor	13	
17	Filtrar Profesor	14	09
22	Visualizar Módulos	15	10
23	Solicitar Sección	16	12
24	Visualizar Solicitud	17	23
23	Recibir Solicitud	18	14
21	Asignar Módulo Espejo	19	10
16	Asignar Sección a Profesor	20	09
19	Asignar Profesor responsable	21	16
18	Historial de Módulos de Profesor	22	16
15	Generar Contrato a Profesor	23	16
13	Mantener Presupuesto	24	15

3.7. Arquitectura del sistema

3.7.1. Modelo Entidad-Relación

El modelo Entidad-Relación nos permite conocer La Figura 3.2 nos muestra cómo será el modelo Entidad-Relación del sistema.

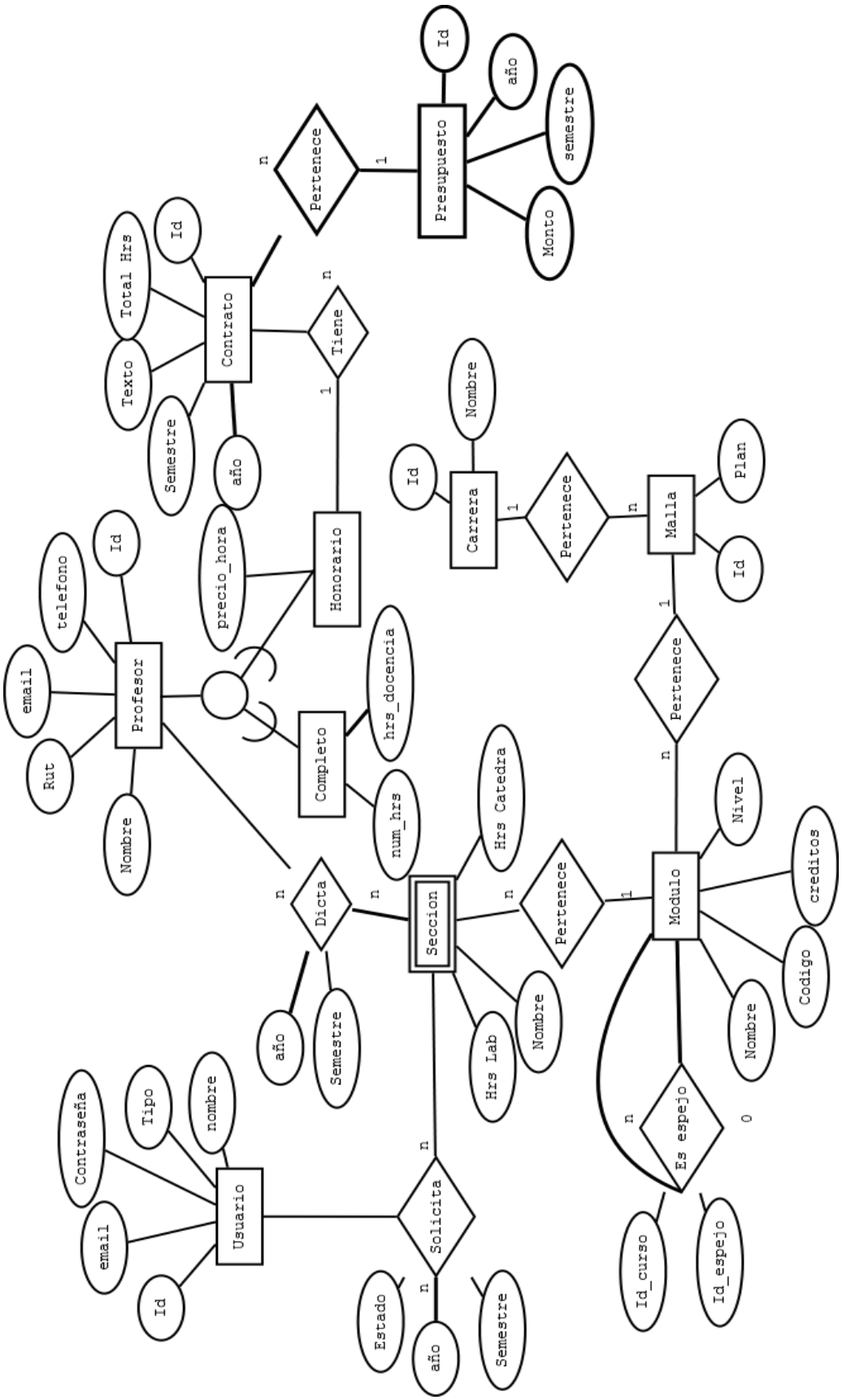

Figura 3.2: Modelo Entidad-Relación del sistema.

3.7.2. Modelo Relacional

Al convertir el modelo Entidad-Relación de la Figura 3.2 a modelo relacional, como resultado las tablas del sistema son las siguientes:

Cuadro 3.26: Tabla: Usuario

Usuario		
Atributo	**Tipo**	**Descripción**
Id	Pk: integer	identificador del usuario
Nombre	text	nombre del usuario
Rut	text	será el nombre del usuario para iniciar sesión en el sistema
Contraseña	text	contraseña para validar el acceso del usuario al sistema
Tipo	integer	Identificará al tipo de usuario (Administrador, director de escuela ó jefe de departamento

Cuadro 3.27: Tabla: Solicitud

Solicitud		
Atributo	**Tipo**	**Descripción**
Id	Pk: integer	identificador de la solicitud
Estado	text	Estado de la solicitud (en proceso, aceptada, rechazada)
Usuario_id	Fk: integer	Llave foránea, para identificar al usuario

Cuadro 3.28: Tabla: Módulo

Modulo		
Atributo	**Tipo**	**Descripción**
Id	Pk: integer	Identificador del módulo
Nombre	text	Nombre del módulo
Codigo	text	Código del módulo
Creditos	integer	Número de créditos del módulo
Nivel	integer	Nivel del módulo respecto a la malla curricular
Malla_id	Fk: integer	Llave foránea para conocer la carrera y plan al cual pertenece el modulo

Cuadro 3.29: Tabla: Sección

Sección		
Atributo	**Tipo**	**Descripción**
Modulo_id	Fk: integer	Para identificar el modulo al cual pertenece la sección
Nombre	text	Nombre de la sección
Hrs_lab	integer	Número de horas de laboratorio que tiene la sección
Hrs_catedra	integer	Número de horas de cátedra que tiene la sección

Cuadro 3.30: Tabla: Malla

Malla		
Atributo	**Tipo**	**Descripción**
Id	Pk: integer	Atributo identificador que pertenece a la malla
Nombre	text	Nombre de la carrera a la cual pertenece (Ej: ICC)
Plan	integer	Nombre del plan de la malla (ej: Plan 44)

Cuadro 3.31: Tabla: Espejo

Espejo		
Atributo	**Tipo**	**Descripción**
Id_modulo	Fk:integer	Identificador del módulo
Id_espejo	Fk: integer	Para identificar el módulo al cual se está referenciando

Cuadro 3.32: Tabla: Profesor

Profesor		
Atributo	**Tipo**	**Descripción**
Id	Pk:integer	Identificador del profesor
Rut	text	Rut del profesor
Nombre	text	Nombre del profesor
Email	text	Email del profesor
Telefono	text	Teléfono de contacto del profesor

Cuadro 3.33: Tabla:Profesor_Completo

Profesor_Completo		
Atributo	**Tipo**	**Descripción**
Id_Profesor	Fk:integer	Llave foránea para identificar al profesor
Num_hrs	integer	Número de horas por las cuales es contratado el profesor
Hrs_docencia	integer	Número de horas que deben ser realizadas como docencia

Cuadro 3.34: Tabla: Profesor_Honorario

Profesor_Honorario		
Atributo	**Tipo**	**Descripción**
Id_Profesor	Fk:integer	Llave foránea para identificar al profesor
Precio_hora	integer	Correspondiente al valor que se pagará por hora trabajada

Cuadro 3.35: Tabla:Contrato

Contrato		
Atributo	**Tipo**	**Descripción**
Id	Pk:integer	Identificador del contrato
Id_Profesor	Fk: integer	Llave foránea correspondiente al profesor al cual pertenece el contrato
Total_hrs	Float	Correspondiente al número de horas totales que se tiene por contrato
Texto	text	Corresponde al texto que tendrá el contrato
Semestre	integer	Semestre en el cual fue hecho el contrato
Año	integer	Año en el cual fue hecho el contrato

Cuadro 3.36: Tabla: Solicitud_Tiene_Sección

Solicitud_Tiene_Seccion		
Atributo	**Tipo**	**Descripción**
Id_Solicitud	Fk:integer	Llave foránea para identificar la solicitud
Id_Seccion	Fk:integer	Llave foránea para identificar a la sección del módulo
Id_Usuario	Fk:integer	Llave foránea para identificar al usuario que solicita la sección

Cuadro 3.37: Tabla: Profesor_Dicta_Sección

Profesor_Dicta_Seccion		
Atributo	**Tipo**	**Descripción**
Id_Profesor	Fk:integer	Llave foránea para identificar al profesor que dicta el curso
Id_Modulo	Fk:integer	Llave foránea para identificar al módulo
Nombre_Sección	Fk:text	Referencia al nombre de la sección
Año	integer	Año en el cual el profesor dictó la sección
Semestre	text	Semestre en el cual el profesor dictó la sección
Responsable	Bool	Para saber si el profesor es el responsable del módulo

Cuadro 3.38: Tabla: Presupuesto

Presupuesto		
Atributo	**Tipo**	**Descripción**
Id	Pk:integer	Llave primaria identificador del presupuesto
Monto	integer	Monto correspondiente al valor total del presupuesto
Semestre	integer	Semestre al cual corresponde el presupuesto
Año	integer	Para saber si el profesor es el responsable del módulo

3.7.3. Resumen

En este capítulo se describieron los procesos llevados a cabo para diseñar la solución del sistema, se explicó la metodología de trabajo a utilizar, se escogieron las herramientas con las cuales se llevará a cabo la construcción del sistema, posteriormente se especificaron las Historias de Usuario y se realizó un diagrama Entidad-Relación en conjunto a sus respectivas tablas de las entidades pertenecientes al sistema.

En el siguiente Capítulo se dan a conocer las partes involucradas en la construcción del proyecto.

4. Implementación de la solución

4.1. Introducción

En este capítulo se inicia la codificación del Sistema Prototipo para la planificación de módulos y profesores del departamento de ciencias de la computación.

Gracias a la facilidad de implementación que ofrece Rails, es posible poner en marcha el sistema de forma inmediata. En capítulos posteriores, conforme se desarrollan los tópicos concernientes a Rails, cada uno es ejemplificado con un módulo del sistema.

4.2. Creación de la aplicación

Para generar una aplicación con Rails, se escribe el siguiente comando:

Listing 4.1: Crear aplicación con Rails

```
$ rails new {nombre de la aplicación}
```

Para el caso del sistema SPMP[1], el comando se ejecuta de la siguiente forma:

Listing 4.2: Crear aplicación Memoria

```
$ rails new Memoria
```

Esto va a generar un directorio llamado memoria, y dentro de éste se encuentran todas los archivos y directorios necesarios para la aplicación. La estructura de directorios obtenida se puede ver en la siguiente Figura 4.1

[1]SPMP: Sistema prototipo de Módulos y Profesores

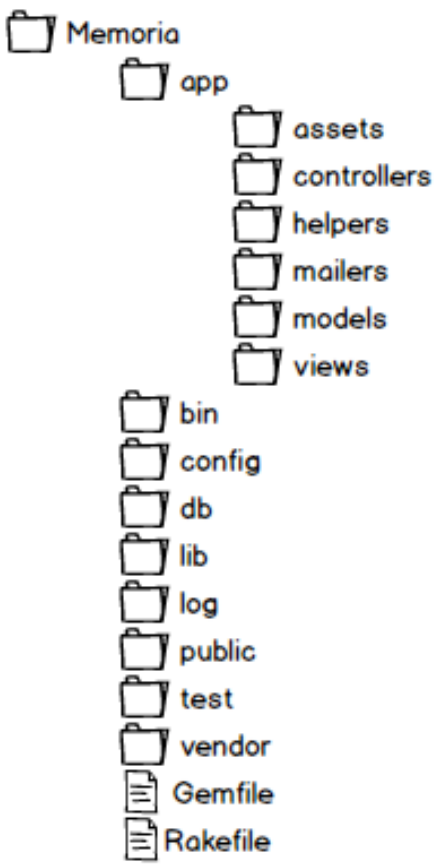

Figura 4.1: Estructura de directorios de proyecto rails.

- app/controllers: contiene los archivos fuente correspondientes a los controladores de la aplicación.
- app/assets: contiene todas las imágenes, sonidos, tipografías, hojas de estilos y archivos javascript que se utilizaran en el proyecto.
- app/helpers: contiene *helpers*, que son las clases utilitarias que contienen métodos que se utilizan para ayudar a las clases del modelo, vista y controlador.
- app/mailers: contiene *mailers*, son similares a los controladores y contienen funciones específicas para enviar *emails* dentro de la aplicación
- app/models: contiene los modelos, son aquellos en donde se mantiene toda la lógica de negocios de nuestra aplicación web. Como los accesos a datos, validaciones o clases y métodos de importancia para el sistema.

- app/views: contiene los archivos relacionados con las vistas. Los archivos tienen una extensión html.erb, lo cual nos permite combinar el lenguaje de Ruby con HTML.
- bin: contiene los scripts rails para comenzar la aplicación, y puede contener otros scripts utilizados para configurar, desplegar y ejecutar la aplicación.
- config: contiene archivos de configuración para la aplicación, rutas, base de datos, entre otros.
- db: contiene el actual esquema de datos, como también las migraciones de la base de datos.
- lib: es en donde van librerías específicas que serán utilizadas en la aplicación
- log: contiene un archivo de *logs* para la aplicación. Rails crea automáticamente los archivos para cada ambiente.
- public: contiene los ficheros estáticos y los ficheros assets compilados. Los archivos que se encuentran dentro de este directorio son accesibles directamente desde https://nombreaplicacion.com/nombredearchivo.
- test: contiene test unitarios, ficheros con datos de prueba y otros tipos de test.
- vendor: contiene el código de terceros, archivos como javascripts, librerías, hojas de estilo.
- Gemfile & RakeFile: Estos archivos permiten especificar las gemas y/o dependencias que son necesarias para hacer andar la aplicación Rails.

Para poner en marcha la aplicación, se escribe el siguiente comando:

Listing 4.3: Iniciar Aplicación Rails

```
$ rails server
```

Si todo marcha bien, al final de su ejecución, Rails muestra la siguiente salida:

Listing 4.4: Salida lanzamiento de aplicación

```
=> Booting WEBrick
=> Rails 4.2.5 application starting in development
    on http://localhost:8080
=> Run `rails server -h` for more startup options
=> Ctrl-C to shutdown server
```

Para visualizar la aplicación, se debe abrir un navegador web y dirigirse a la dirección http://localhost:8080/memoria. El navegador muestra la página de la Figura 4.2

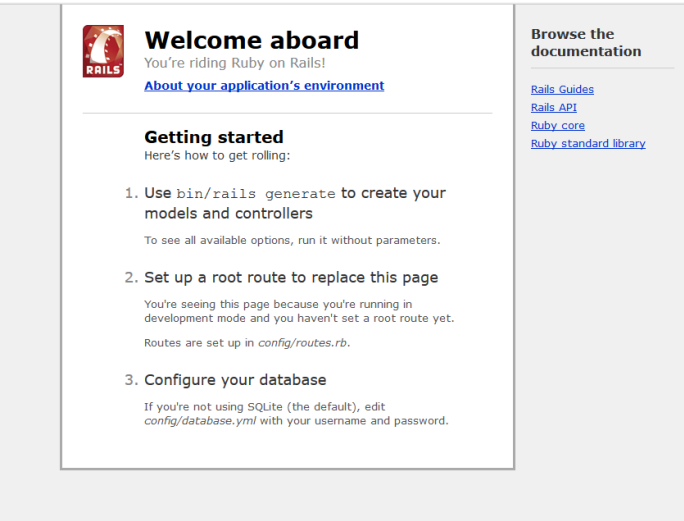

Figura 4.2: Página inicial de la aplicación.

Rails ha generado y desplegado de forma automática toda la infraestructura necesaria para una aplicación web. No hay necesidad de instalar un servidor de aplicaciones como ocurriría con otros *framework* de desarrollo web y no hay necesidad de escribir HTML para la página inicial. Rails automatiza todos esos pasos.

4.2.1. Creando la lógica del negocio: Modelos

Los modelos en Rails al igual que en los otros *framework* MVC son aquellos en donde se mantiene toda la lógica de negocios de nuestra aplicación web. Los modelos desarrollan la información importante de la aplicación, como los accesos a datos.

Para el desarrollo de la aplicación se han generado modelos para todas las entidades obtenidas en la fase de análisis y diseño ver Figura 3.2

Los modelos en Rails usan un nombre en singular, y sus correspondientes tablas de base de datos usan un nombre en plural. Rails provee un generador para crear modelos, dicha codificación se realiza mediante archivos .rb. Para la creación de un modelo, se utiliza el comando:

Listing 4.5: Generar modelo Carrera

```
$ rails generate model Carrera nombre:text
```

La ejecución del comando genera un modelo Carrera, junto con un atributo nombre de tipo *text*. Esos atributos son automáticamente añadidos a la tabla carreras en la base de datos y mapeados al modelo Carrera.

La lista de modelos creados durante el desarrollo de la aplicación fue:

- Carrera
- Malla
- Modulo
- Seccion
- Espejo
- Usuario
- UsuarioSolicitaSeccion
- Profesor
- ProfesorHonorario
- ProfesorCompleto
- ProfesorDictaSeccion
- Contrato
- Presupuesto

4.2.2. Asociando Modelos

Para cubrir la necesidad relacional existente entre las entidades o modelos, Rails nos provee de ciertos métodos pertenecientes a *ActiveRecord*, el cual se encarga de la capa de persistencia o más bien del mapeo-relacional entre objetos.

Algunos de los métodos utilizados durante el desarrollo de la aplicación para el mapeo relacional fueron los siguientes:

- belongs_to: Esta asociación establece una relación de uno a uno con otro modelo, de forma que cada instancia del modelo "pertenece.ª una instancia del otro modelo.

- has_many: Esta asociación indica el enlace de uno a muchos con otro modelo. Por lo general quiere decir que cada objeto de un modelo puede tener muchos objetos relacionados a él (o ninguno)

En el siguiente modelo implementado en la aplicación podemos apreciar cómo se hace uso de estas relaciones.

Listing 4.6: Relaciones has_many - belongs_to

```
class Malla < ActiveRecord::Base
    belongs_to :carrera, dependent: :delete
    has_many :modulos
end
```

Con esta implementación podemos notar que se cubren las necesidades del diagrama Entidad-Relación de que una Malla pertenece a una Carrera y que a la vez una Malla tiene muchos Módulos. Hay que mencionar que con esto se logra una relación unidireccional, es decir, significa que el modelo Malla puede acceder a su atributo Carrera pero Carrera no puede acceder al Modelo Malla que la contiene. Para lograr que la relación sea bidireccional, basta con agregar la relación *has_many* en el modelo Carrera

Listing 4.7: Relación modelo Carrera

```
class Carrera < ActiveRecord::Base
    has_many :mallas
end
```

Con ello se establece que el modelo Malla pertenece a la clase Carrera. El resultado de esto es que se puede asociar una instancia de Malla a una instancia de la clase Carrera, y cuando se ejecuten acciones de guardar y borrar en ésta última, éstas se ven reflejadas en Malla. En otras palabras, las acciones de guardar y borrar se hacen en cascada.

Configuraciones similares fueron aplicadas a todos los modelos mencionados en la sección anterior.

4.3. Controladores

Como se menciona en el capítulo 2.6 los controladores son quienes están a cargo de recibir las peticiones de los usuarios y disparar las acciones para poder devolver la respuesta a los usuarios. Por cada modelo que iba siendo generado, también era necesario generar su propio controlador. Estos controladores se implementan mediante clases en la carpeta app/controllers. El proceso para generar un controlador se puede ejecutar bajo el siguiente comando:

Listing 4.8: Generar controlador con Rails

```
$ rails generate controller{nombre}{acciones}
```

En nuestro caso para generar el controlador para la entidad Carrera se hizo de la siguiente forma:

Listing 4.9: Generar controlador carrera

```
$ rails generate controller carrera index
```

La ejecución de este comando no tan solo generará el controlador, sino que también Rails creará automáticamente su respectivo *helper* y carpeta de vistas o *views* asociadas.

Cuadro 4.1: Directorio/Archivos generados por el comando generate controller

Archivo/Carpeta	Propósito
app/controllers/carrera_controller.rb	El controlador de Carreras.
app/views/carreras/	Donde se guardan las vistas del controlador.
app/helpers/carreras_helper.rb	El helper de la vista.

Ya ejecutado el comando, el controlador se ve de la siguiente manera:

Listing 4.10: Controlador Carrera

```
class CarrerasController < ApplicationController
  def index
    @carreras = Carrera.all
  end
end
```

De esta manera, ahora existe una clase llamada CarrerasController que hereda de la clase ApplicationController y dentro de esta se encuentra (en este caso) un método llamado índex que contiene una acción particular, dicha acción es ir a la base de datos y buscar todas las carreras, para posteriormente mostrarlas a través de la vista.

4.3.1. Métodos comunes en todos los controladores

En toda aplicación web con conexión a base de datos, las operaciones CRUD, que significa *Create Read Update Delete* (En español crear, leer, modificar y eliminar) son las principales y primeras en codificarse ya que son las más utilizadas. Todas las operaciones CRUD tienen asociadas 2 acciones en su respectivo controlador

4.3.1.1. Create

Como se mencionó en el párrafo anterior, la operación *create* se encuentra compuesta de 2 operaciones:

- *new*: está encargada de mostrar un formulario web vacío con los respectivos campos disponibles para crear nuevo registro en la base de datos
- *create*: procesa los campos obtenidos desde el formulario y guarda el nuevo registro en la base de datos en caso de no existir errores.

4.3.1.2. Read

Esta operación difiere a las demás, puesto que no envía información a través de formularios.

- *list*: Acción encargada de retornar una lista de registros asociados a una entidad.
- *show*: Acción encargada de mostrar en detalle un único record.

4.3.1.3. Update

Esta operación es bastante similar a la operación *create*, la única diferencia es que trabaja con registros que ya se encuentran en la base de datos.

- *edit*: Acción encargada de mostrar un formulario para editar los valores de un registro
- *update*: Acción que procesa los valores obtenidos desde el formulario para posteriormente guardar los cambios en la base de datos

4.3.1.4. Delete

Esta operación está encargada de eliminar registros en la base de datos.

- *delete*: Acción encargada de mostrar información acerca del registro que será eliminado
- *destroy*: Acción que procesa la destrucción del registro en la base de datos

4.3.2. Controladores implementados

El listado de controladores implementados en la aplicación para manejar las solicitudes realizadas por los usuarios desde las vistas, para interactuar con los modelos son:

- application_controller: controlador principal de la aplicación, encargado de validar el inicio de sesión para ingresar al sistema y de derivar a la vista correspondiente según el tipo de usuario.
- registracion_controller: controlador encargado del inicio de sesión, registro y edición de usuario.
- administracion_usuarios_controller: controlador encargado del bloqueo, activación y eliminación de cuentas de usuarios.
- carreras_controller: controlador encargado de la administración de las carreras en la aplicación.

- mallas_controller: controlador encargado de la administración de las mallas en la aplicación.
- modulos_controller: controlador encargado de la administración de los módulos en la aplicación.
- espejo_controller: controlador encargado de la asociación de módulos espejos en la aplicación
- profesores_controller: controlador encargado de la administración de los distintos tipos de profesores en la aplicación.
- profesor_dicta_seccion_controller: controlador encargado de la administración de los profesores que dictan secciones.
- secciones_controller: controlador encargado de la administración de las secciones de cada módulo de la aplicación.
- solicitar_seccion_controller: controlador encargado de la administración de las solicitudes de secciones realizadas por los usuarios en la aplicación.
- presupuesto_controller: controlador encargado de la administración de los presupuestos anuales del departamento en la aplicación.
- contrato_controller: controlador encargado de la generación de los contratos en la aplicación.

4.3.3. Gemas utilizadas

Para agilizar el proceso de desarrollo de esta aplicación utilicé diferentes gemas. La inclusión de ellas en el proyecto se encuentra definida dentro del archivo Gemfile que se encuentra en el directorio raíz, como se puede apreciar en la Figura 4.1 correspondiente al directorio del proyecto.

- Rails: La gema *Rails*[9] es utilizada para declarar la versión de Rails con la que se desarrolló la aplicación. Es recomendable declarar la versión, ya que posibles actualizaciones de esta pueden llevar a problemas de compatibilidad
- Devise: la gema *devise*[12] permite la creación y autenticación de usuarios. Posee diferentes niveles de configuración para robustecer la seguridad de acceso al sistema y métodos de validación, control y recuperación de cuentas de usuario.
- Spring: La gema *spring*[13] es un pre-cargador de aplicaciones en Rails. Esta gema permite mantener la aplicación en estado de ejecución en el fondo mientras se realizan modificaciones en su código fuente, evitando tener que reiniciar cada vez que se genere un cambio.
- jquery-rails: La gema *jquery-rails*[14] es utilizada para para cargar las librerias jquery de javascript.

4.4. Añadiendo presentación: vistas

Como se menciona en la sección 2.6, las vistas son las interfaces que están encargadas de la interacción con el usuario y representación de los datos. Todas las vistas se encuentran dentro del directorio app/views

4.4.1. Vista de ingreso

Para poder ingresar a la interfaz principal,primero es requerido iniciar sesión, puesto que la aplicación no permite el ingreso de usuarios no registrados en el sistema.

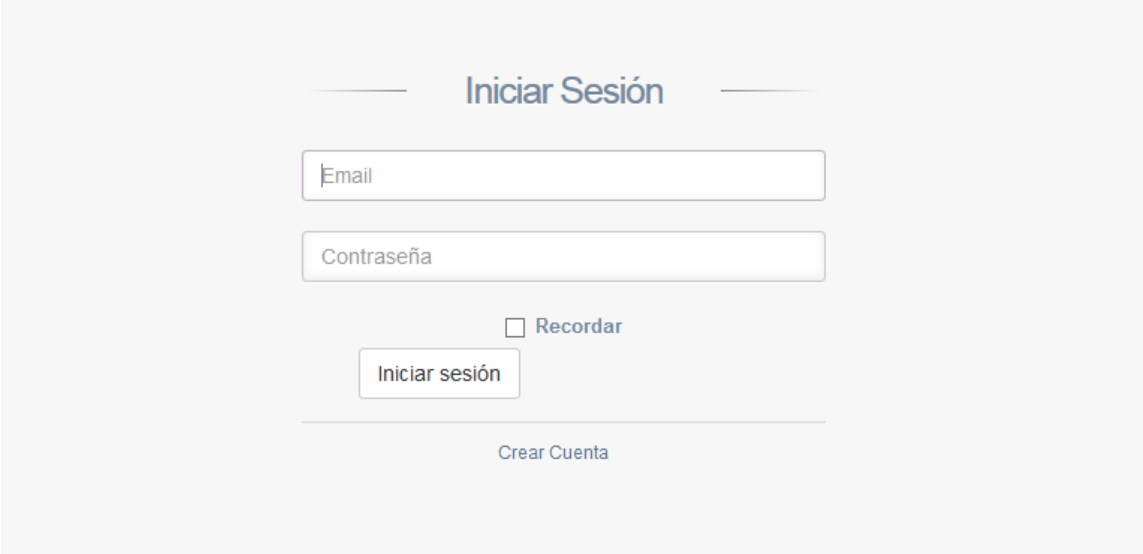

Figura 4.3: vista inicio de sesión.

Para iniciar sesión es necesario ingresar el *email* con el cual es registrada la cuenta de acceso y su respectiva contraseña.

4.4.2. Vista Registro de usuario

Esta vista se encuentra encargada de registrar nuevas cuentas de usuario para poder ingresar al sistema.

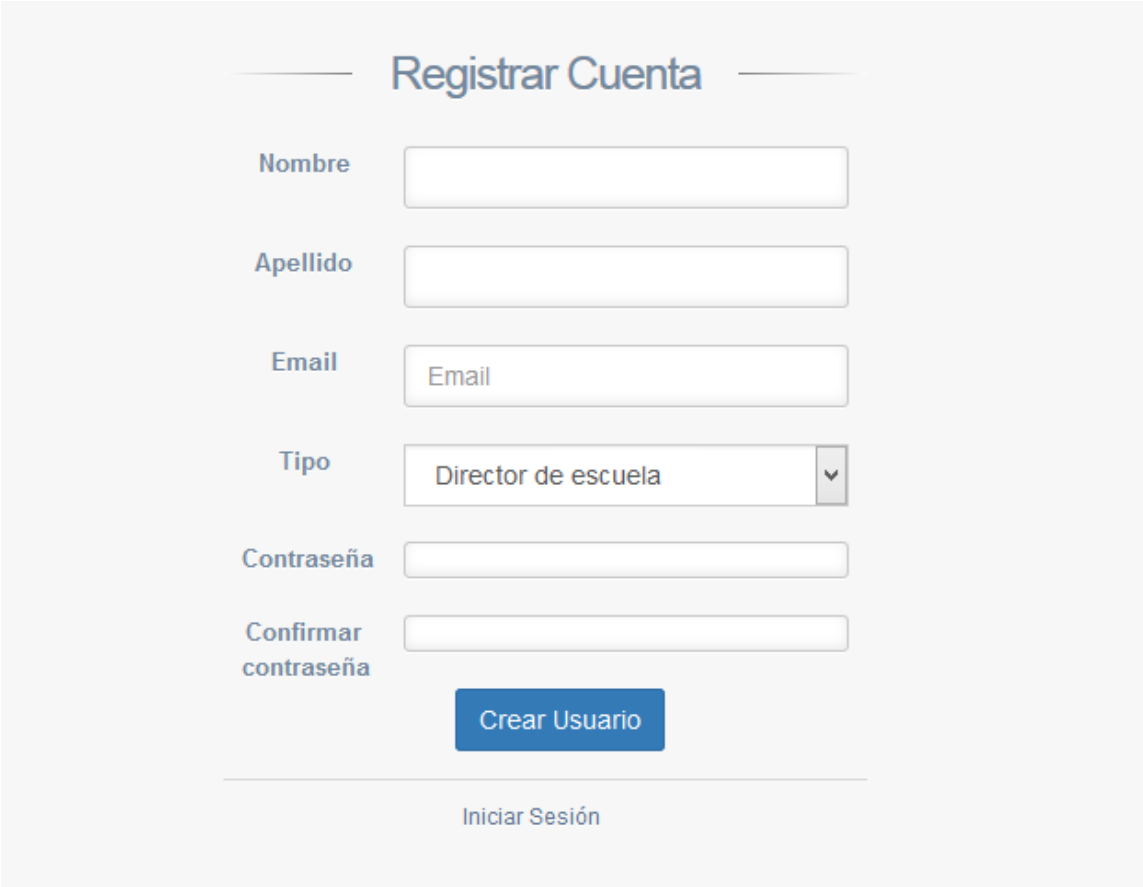

Figura 4.4: vista registro de usuario.

Los campos requeridos para registrar una cuenta en el sistema son: Nombre, Apellido, *Email*, Contraseña y un reingreso de la Contraseña (en caso de que esta haya sido mal escrita). En cuanto al Tipo de cuenta, esta puede ser Director de escuela o Administrador.

4.4.3. Vista Principal

La interfaz principal de la aplicación está compuesta por 3 vistas parciales[2]. Cada una de estas vistas tiene distintos propósitos:

- Vista Parcial 1: Esta vista parcial tiene como propósito desplegar una barra de menú lateral, con la cual, dependiendo del tipo de usuario, mostrará las distintas operaciones con las que podrá navegar dentro de la aplicación.
- Vista Parcial 2: Esta vista se encarga de desplegar las opciones para que el usuario pueda editar su perfil de usuario y cerrar sesión
- Vista Parcial 3: Esta vista es la encargada de desplegar el contenido dentro de la aplicación (desplegar el contenido de módulos, carreras, profesores, etc).

La Figura 4.5 muestra como está compuesto el diseño de la vista principal.

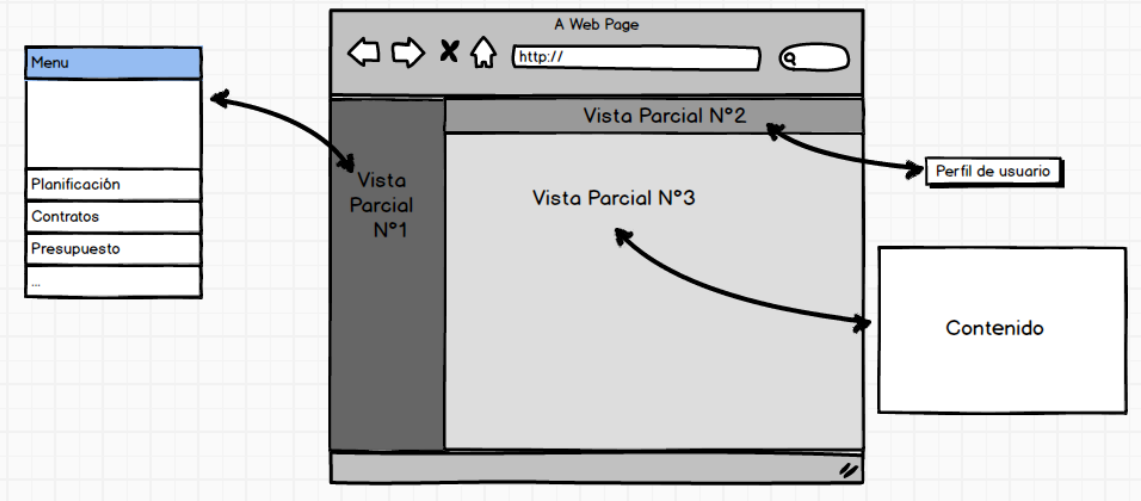

Figura 4.5: Diseño vista principal.

[2]Vista Parcial: Es una vista que puede ser utilizada como parte de otra

Una vez que se cuenta con un usuario válido para ingresar al sistema, la vista principal es desplegada y esta se ve como muestra la Figura 4.6

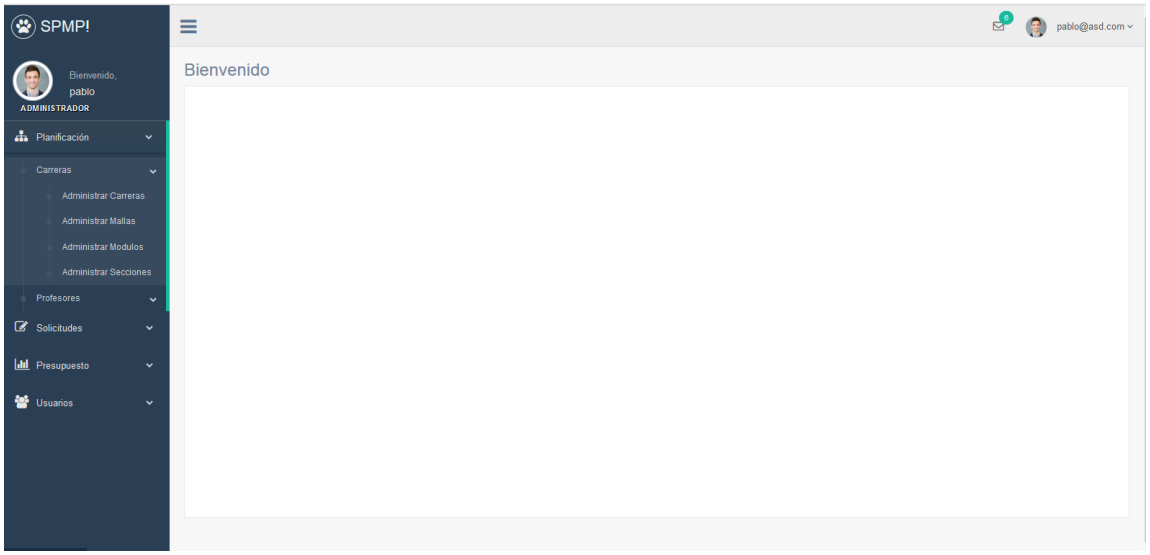

Figura 4.6: Diseño vista principal de usuario administrador.

4.4.4. Navegación

Una vez que el usuario logra iniciar sesión en el sistema, este puede realizar distintas acciones dependiendo del tipo de usuario al que pertenezca.

4.4.4.1. Navegación de Usuario Administrador

La Figura 4.7 muestra el mapa de navegación al cual puede acceder un usuario de tipo administrador.

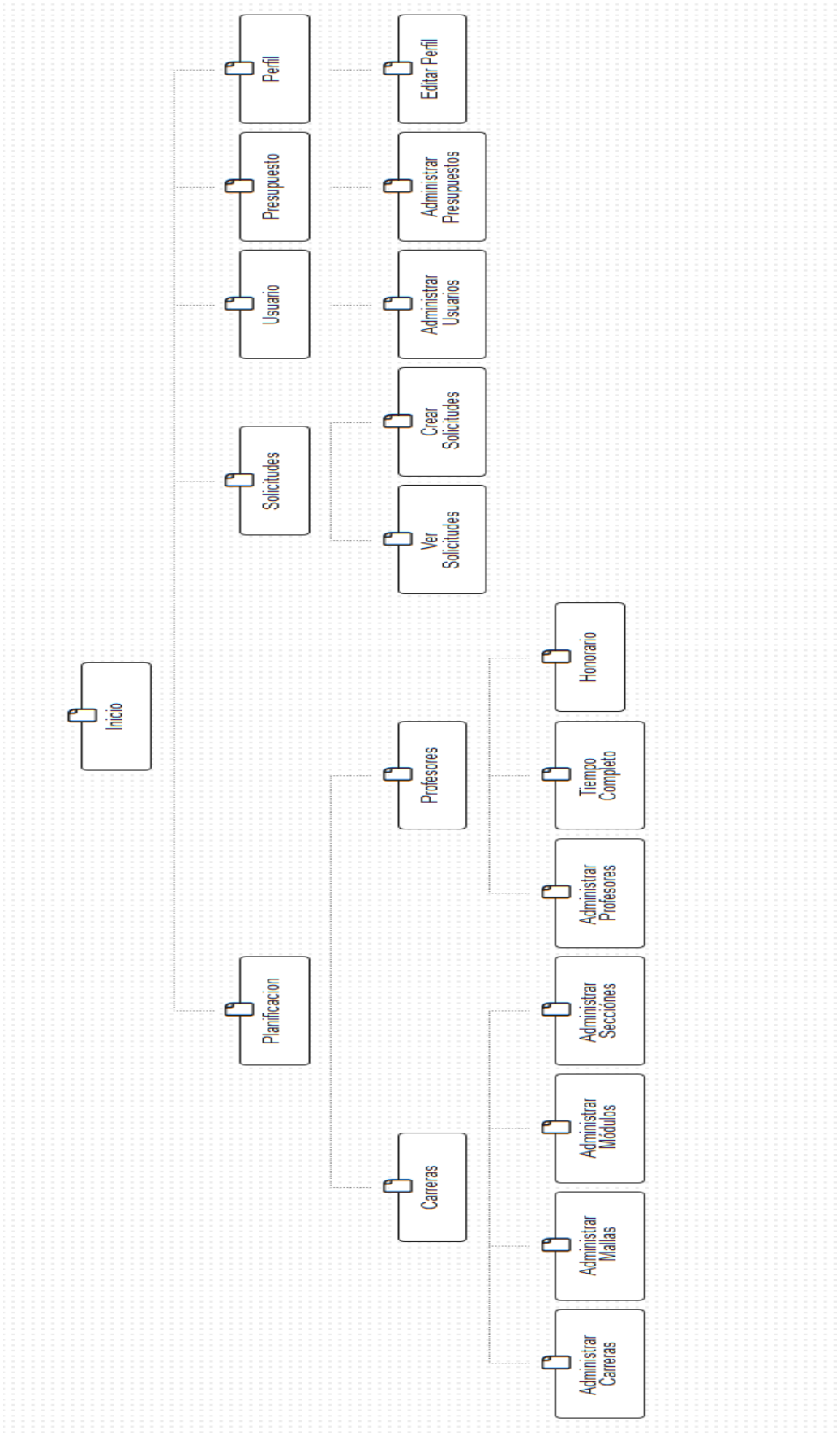

Figura 4.7: Mapa de navegación de Usuario Administrador.

4.4.4.1.1. Menú Planificación

El menú de Planificación nos permite acceder a las opciones de administración, orientadas a Carreras y Profesores.

Una vez seleccionada la opción de Carrera, se puede acceder a las siguientes vistas:

- Administrar Carreras: Esta opción del menú contiene todo lo relacionado con Crear, Ver, Editar y Eliminar una determinada Carrera.
- Administrar Mallas: Esta opción del menú contiene todo lo relacionado con Crear, Ver, Editar y Eliminar una determinada Malla.
- Administrar Módulos: Esta opción del menú contiene todo lo relacionado con Crear, Ver, Editar, Asignar módulo espejo y eliminar un determinado Módulo.
- Administrar Secciones: Esta opción del menú contiene todo lo relacionado con Crear, Ver, Editar y eliminar una determinada Sección.

Dentro de la opción Profesores, se puede acceder a las siguientes vistas:

- Administrar Profesores: Esta opción del menú contiene todo lo relacionado con Crear, Ver, Editar y Eliminar un Profesor del Tipo Honorario o Tiempo Completo.
- Tiempo Completo: Esta opción del menú permite visualizar a los profesores que son de tiempo completo y asignarles secciones.
- Honorario: Esta opción del menú permite visualizar a los profesores que son a Honorario, asignarles secciones y generar contrato.

4.4.4.1.2. Menú Solicitudes

El menú de Solicitudes permite acceder a las vistas:

- Ver Solicitudes: Esta opción del menú permite Ver, Aceptar, Eliminar ó Rechazar todas las solicitudes de secciones que han sido realizadas en el sistema
- Crear Solicitud: Esta opción del menú contiene todo lo relacionado con Crear, Ver, Editar y Eliminar una solicitud.

4.4.4.1.3. Menú Presupuesto

El Menú de Presupuesto permite acceder a la siguiente vista:

- Administración de Presupuestos: Esta opción del menú contiene todo lo relacionado con Crear, Ver, Editar y eliminar un determinado Presupuesto.

4.4.4.1.4. Menú Usuarios

El Menú de Usuarios permite acceder a la siguiente vista:

- Administración de Usuarios: Esta opción del menú permite activar, bloquear ó eliminar cuentas de usuarios.

4.4.4.1.5. Menú Perfil

Esta opción del menú se encuentra en la vista parcial n°2 ver Figura 4.5 y permite realizar la operación de cerrar sesión para salir del sistema y también acceder a la opción:

- Editar Perfil: Esta opción del menú permite editar campos correspondientes a la cuenta del usuario.

4.4.4.2. Navegación de Usuario Director de Escuela

Las vistas a las cuales puede acceder un usuario Director de Escuela son bastante más limitadas, La Figura 4.8 muestra el mapa de navegación al cual puede acceder este tipo de usuario.

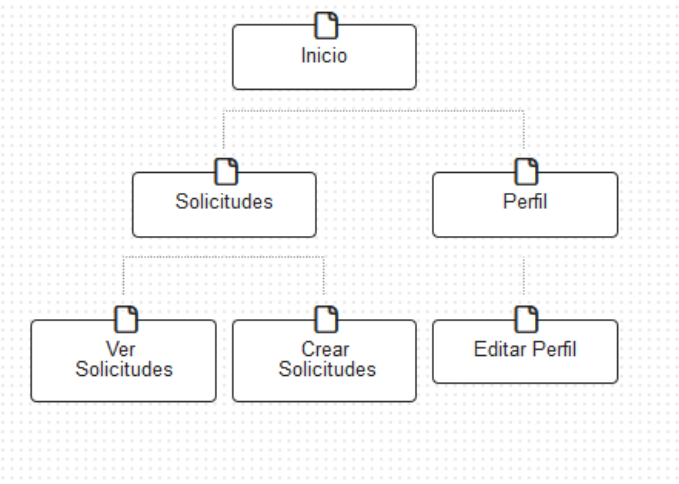

Figura 4.8: Mapa de navegación de Usuario Director de Escuela.

4.4.4.2.1. Menú Solicitudes

El menú de Solicitudes permite acceder a las vistas:

- Ver Solicitudes: Esta opción del menú permite Ver y Eliminar todas las solicitudes que han sido realizadas por el usuario.
- Crear Solicitud: Esta opción del menú, como su nombre lo dice, al usuario permite crear solicitudes.

4.4.4.2.2. Menú Perfil

Al igual que un Usuario Administrador, esta opción del menú permite utilizar la operación cerrar sesión para salir del sistema y también acceder a la opción de Editar Perfil, para modificar los campos relacionados con su cuenta de usuario.

4.4.5. Resumen

En este capítulo se describieron los procesos llevados a cabo para la construcción del sistema. Se explicó desde cuales son los pasos necesarios para crear una apli-

cación usando el *framework* Rails, se mostró la totalidad de los Modelos creados, con ejemplos de cómo fueron relacionados. Posterior a esto se dedicó una sección a los Controladores y se vieron sus principales operaciones. Por último, se presentó el cuales son las vistas del sistema.

En el siguiente Capítulo se dan a conocer las conclusiones obtenidas en el desarrollo del proyecto, en donde se consideran los objetivos, la experiencia obtenida durante el trabajo y el trabajo a futuro.

5. Conclusión

En este Capítulo se describen las conclusiones obtenidas en cuanto a los objetivos y experiencia obtenida al término de la construcción del sistema de este proyecto, posteriormente se muestra una visión del posible trabajo a futuro.

5.1. Objetivos

Para esta Sección son considerados los objetivos de este proyecto mencionados la Sección 1.2 del Capítulo 1.

5.1.1. Utilizar un framework para agilizar el proceso de desarrollo.

La utilización de Ruby on Rails como *framework* fue de gran de ayuda para agilizar la construcción del software. Si bien no se tenía conocimiento o experiencia en el de uso de esta herramienta, considero que el tiempo utilizado como curva de aprendizaje para la utilización de esta compensa con creces el tiempo que se hubiera necesitado para construir este sistema sin un *framework* de desarrollo.

5.1.2. Permitir a los usuarios del tipo Director de Escuela entregar solicitudes al departamento en forma automática

El sistema permite a todos los usuarios que sean Director de Escuela solicitar las secciones que requieran necesarias para un determinado período académico. Para realizar esta operación la cuenta de usuario debe estar activada o aprobada por el administrador.

5.1.3. Administrar a los profesores y sus cargas académicas

El usuario Administrador puede mantener las cargas académicas de todos los profesores que se encuentren registrados en el sistema, si alguno no se encuentra registrado, solo debe crearlo y guardar sus datos para su posterior utilización.

5.1.4. Administrar los planes y módulos del departamento de ciencias de computación.

Durante la fase de diseño del *software* se tuvo en consideración este objetivo, por esta razón es que en la fase de modelado se crearon las entidades Malla y Carrera para así conocer a que plan y carrera pertenece cada módulo dentro del sistema.

5.1.5. Permitir evaluar el presupuesto.

El sistema cuenta con un menú para crear presupuestos para un determinado período y ver cuánto se ha gastado en cada período.

5.1.6. Contar con un sistema capaz de generar contratos a los profesores

El sistema es capaz de generar un documento pdf el cual contiene el número de horas totales de las secciones que dicta un determinado profesor.

5.2. Experiencia

Considero que la oportunidad de construir este proyecto fue un buen desafío, puesto que me hizo aplicar en gran parte los conocimientos adquiridos durante los años estudiados de la carrera. El hecho de hacerme cargo de las decisiones en las etapas fundamentales del proceso de desarrollo de software, llámese Análisis, Diseño y Construcción, fue una experiencia interesante, en la cual aprendí bastante.

Utilizar los principios de las metodologías ágiles para la construcción del proyecto fue de gran ayuda, ya que trabajar en ciclos iterativos me permitió corregir y darme cuenta de errores realizados durante la etapa de diseño del software. Utilizar Ruby on Rails en la implementación del sistema me permitió agilizar el proceso de codificación y fue una experiencia que me mantuvo bastante entusiasmado, ya que aprender acerca del uso de este *framework* era un objetivo personal.

Finalmente, a punto de observación, una de las ventajas de desarrollar prototipos funcionales es que permiten descubrir las necesidades de los usuarios y permiten un rápido desarrollo. Para convertir este prototipo en un sistema final es necesario seguir con el proceso iterativo, realizar una actualización de los requisitos y seguir refinándolo progresivamente. Debido al tiempo transcurrido desde que se comenzó este proyecto al día de hoy, la Universidad se encuentra en un período de transición de sus procesos, esto, a causa de la implementación de la Gratuidad en la Educación Superior, por ende, existe una dificultad en mantener los requisitos de este sistema actualizados.

5.3. Trabajo Futuro

Desde mi punto de vista, la construcción de este prototipo, como prueba de concepto, es un punta pie inicial para la elaboración de una aplicación que puede ser más robusta, en la cual participen un mayor número de *stakeholders*[1], en donde es necesario definir claramente una serie de pasos a seguir para realizar una correcta planificación, también se necesita conocer las necesidades que todos los Directores de Departamento y Escuela puedan tener, de esta manera se abarcaría un mayor número de usuarios.

Sin embargo, es bien sabido que todo proyecto siempre puede mejorar. Por esta razón los siguientes puntos son considerados como mejora y trabajo a futuro:

- Escalar la aplicación: implementar nuevos módulos, cuyas funcionalidades puedan ser útiles para el proceso de planificación. Un ejemplo de esto puede ser establecer una relación entre las cuentas de usuario y las carreras. De esta manera, le otorgamos mayor usabilidad al usuario, puesto que el contenido que vería se encontraría netamente ligado a su área de interés.

- Actualización de Requisitos: actualizar e implementar los Requisitos a las necesidades actuales del Departamento.

- Pruebas de funcionamiento: realizar un ciclo de pruebas exhaustivo, simulando un gran número de planificaciones, para así determinar el comportamiento, encontrar y corregir errores de implementación.

[1]Todos los actores, que de verse involucrados que pueden afectar o se ven afectados por el desarrollo de la aplicación

Bibliografía

[1] Conceptos generales de la arquitectura de aplicaciones web.

[2] Django. http://django.es, Consultado el 25 de mayo de 2015.

[3] estructura orgánica. http://transparencia.utalca.cl/docs/estructura_organica.pdf, Consultado el 15 de octubre de 2016.

[4] Gentelella. https://colorlib.com/polygon/gentelella/, Consultado el 20 de mayo de 2016.

[5] Getbootstrap. http://getbootstrap.com/, Consultado el 25 de mayo de 2015.

[6] ¿qué es cakephp y por qué hay que utilizarlo? http://book.cakephp.org, Consultado el 25 de mayo de 2015.

[7] ¿ruby on rails: El desarrollo web que no molesta? http://www.rubyonrails.org.es/, Consultado el 25 de mayo de 2015.

[8] Valdeverde Rebaza Jorge Carlos. Amaro Calderon, Sarah Damaris. *Metologias Agiles*, pages 9–10. 2007.

[9] David Heinemeier Hansson. rails. http://www.rubygems.org/gems/rails, Consultado el 17 de noviembre de 2016.

[10] Beck. K. Extreme programming explained. embrace change, 2000.

[11] Juan Quijano. Historias de usuario, una forma natural de análisis funcional. http://www.genbetadev.com/metodologias-de-programacion/historias-de-usuario-una-forma-natural-de-analisis-funcional, Consultado el 18 de mayo de 2015.

[12] Carlos Antoónio José Valim. devise. http://www.rubygems.org/gems/devise, Consultado el 18 de junio de 2016.

[13] Wake. W.C. Extreme programming explored, 2002.

ANEXOS

A. Capturas del Sistema

A continuación se adjuntan algunas vistas del sistema.

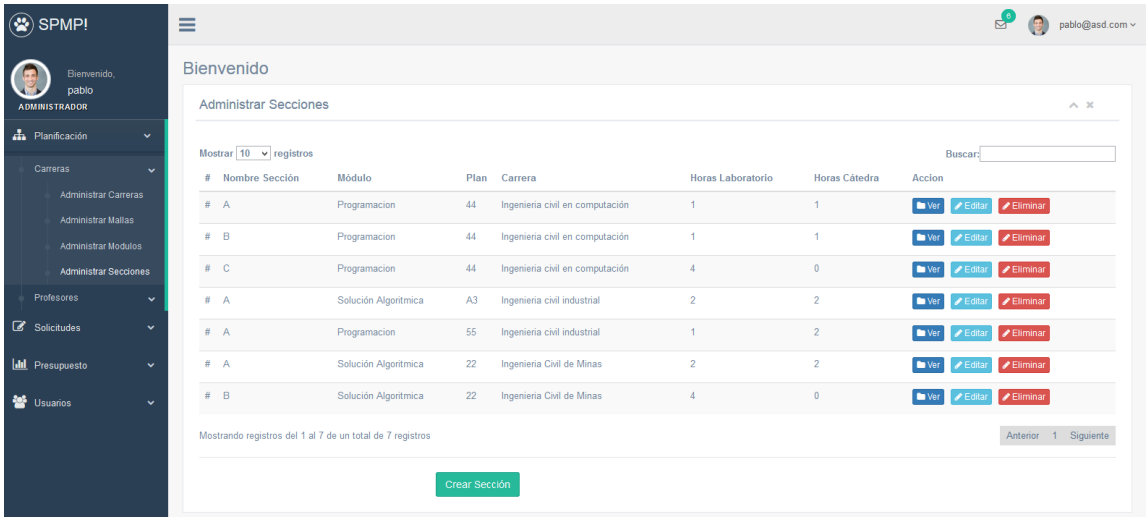

Figura A.1: Vista Administración de Secciones.

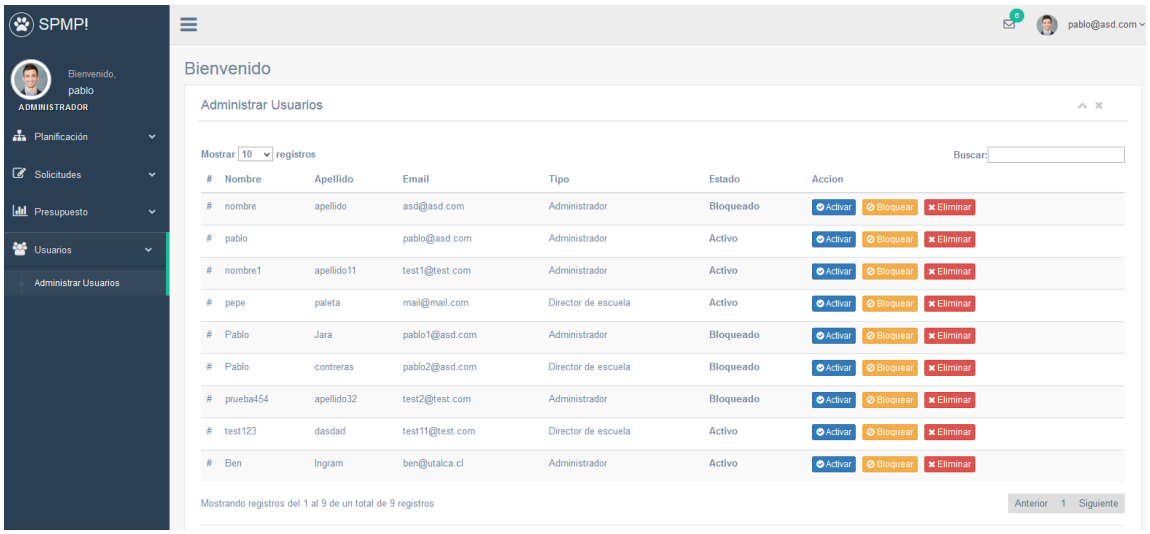

Figura A.2: Vista Administración de Usuarios.

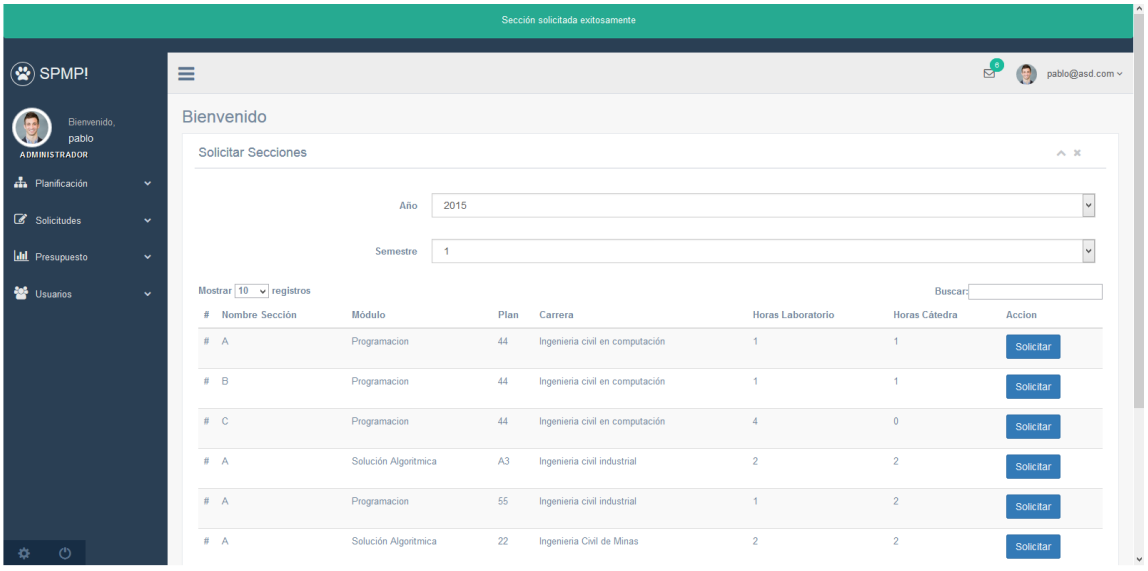

Figura A.3: Vista Solicitar Sección.

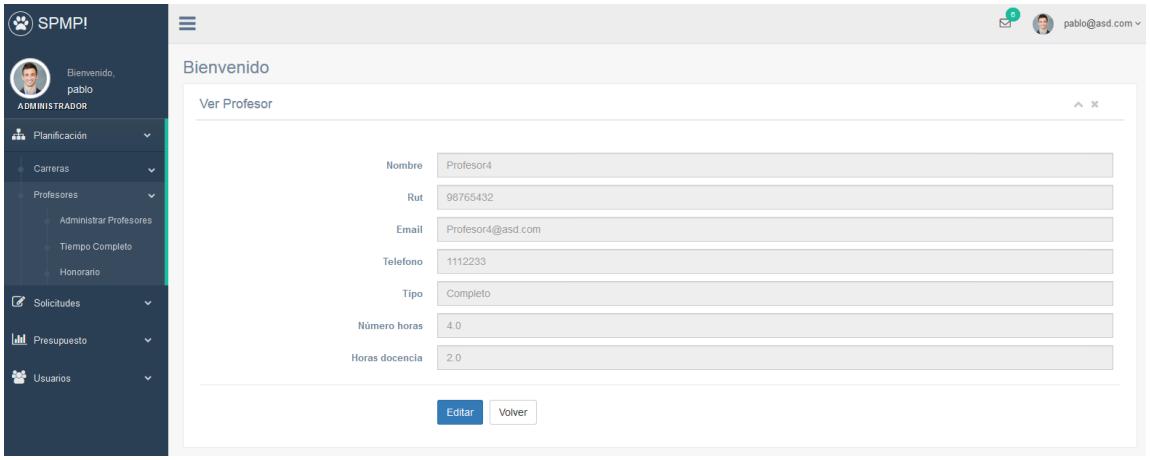

Figura A.4: Vista Profesor Jornada Completa.

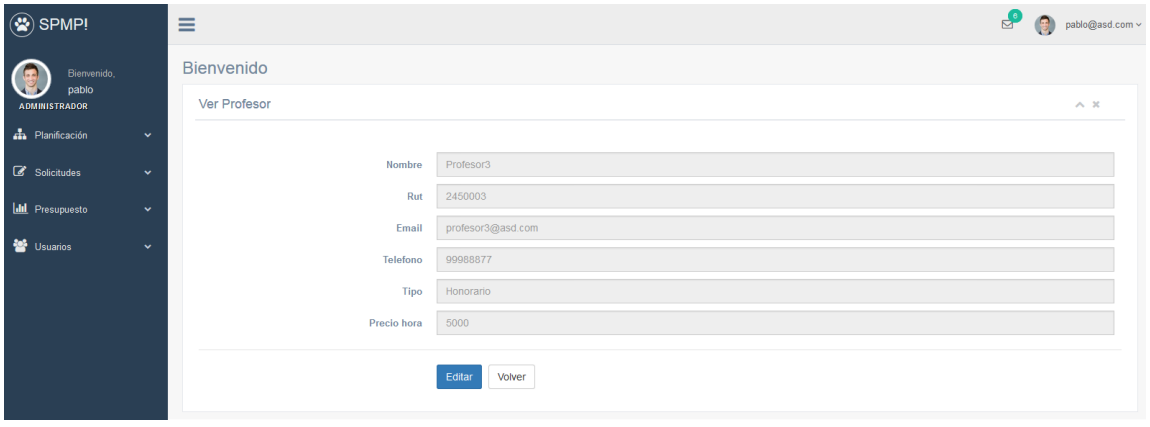

Figura A.5: Vista Profesor Honorario.

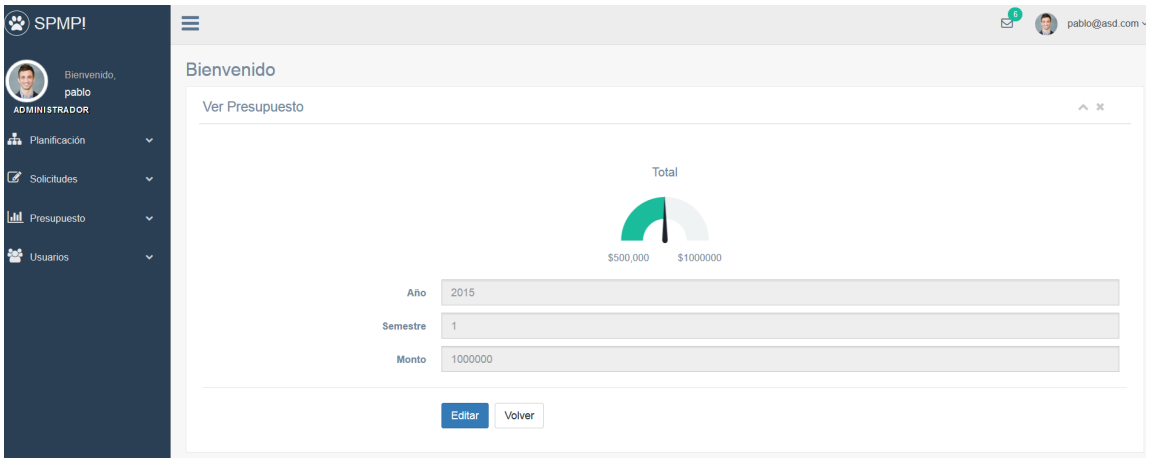

Figura A.6: Vista Presupuesto.

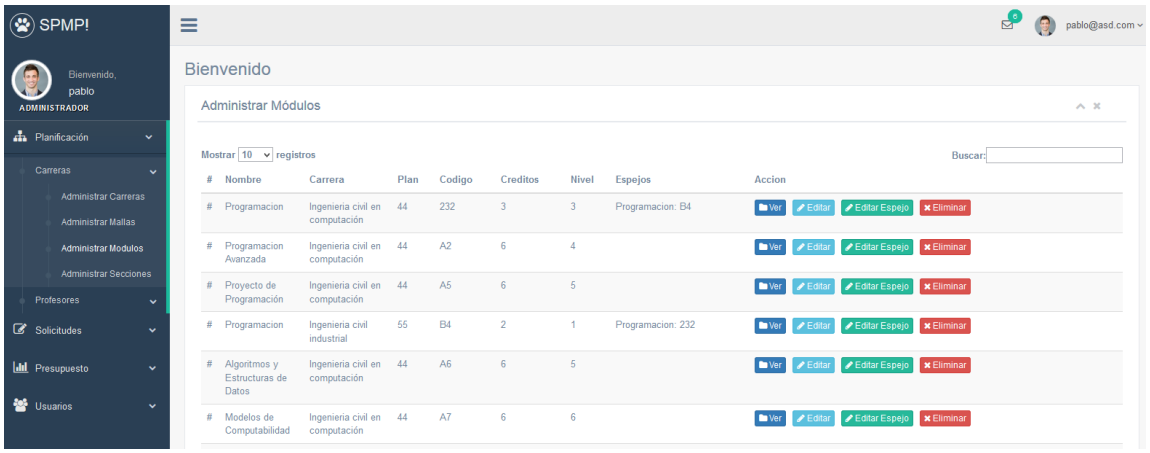

Figura A.7: Vista Administrar Módulos.

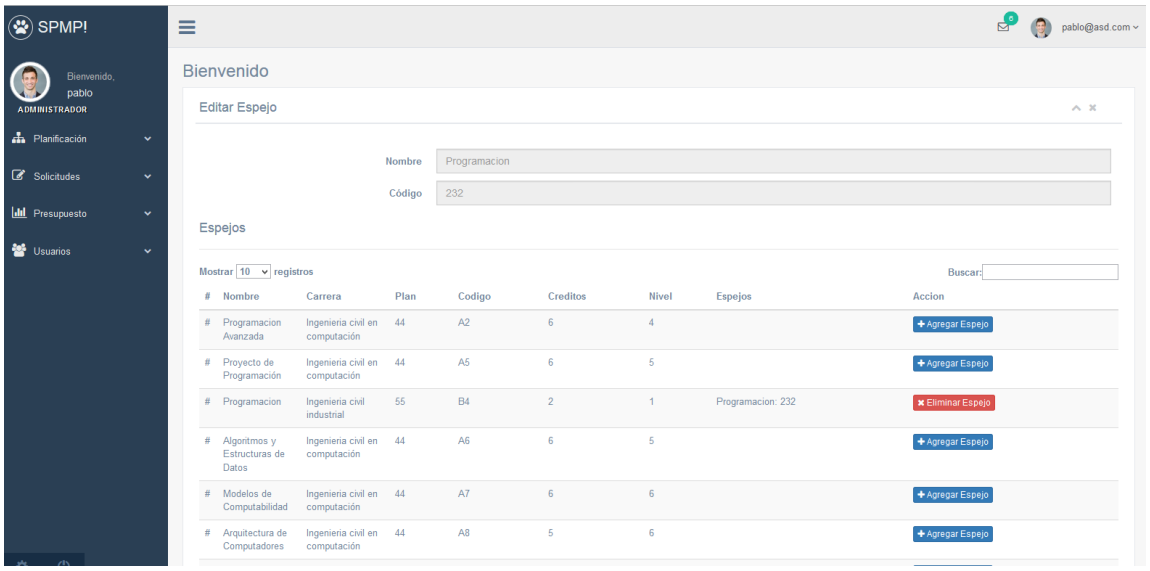

Figura A.8: Vista Editar Espejo Módulo.