FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERIA DE SISTEMAS

DESARROLLO DE UNA PLATAFORMA WEB QUE INTERCONECTE Y ORGANICE GRUPOS UNIVERSITARIOS PERTENECIENTES AL MOVIMIENTO ESTUDIANTIL

Autores:

Ignacio Andrés Fuentes Gimón

Juan Andrés Lagrange Delfino

Tutor: Pablo Giolito

Caracas, Julio del 2010

Derecho de Autor

Quienes suscriben, en condición de autores del trabajo titulado “Desarrollo de una plataforma web que interconecte y organice grupos universitarios pertenecientes al movimiento estudiantil.”, declaramos que: Cedemos a título gratuito, y en forma pura y simple, ilimitada e irrevocable a la Universidad Metropolitana, los derechos de autor de contenido patrimonial que nos corresponden sobre el presente trabajo. Conforme a lo anterior, esta cesión patrimonial sólo comprenderá el derecho para la Universidad de comunicar públicamente la obra, divulgarla, publicarla o reproducirla en la oportunidad que ella así lo estime conveniente, así como, la de salvaguardar nuestros intereses y derechos que nos corresponden como autores de la obra antes señalada. La Universidad en todo momento deberá indicar que la autoría o creación del trabajo corresponde a nuestra persona, salvo los créditos que se deban hacer al tutor o a cualquier tercero que haya colaborado o fuere hecho posible la realización de la presente obra.

_____	_____
Ignacio Andrés Fuentes Gimón	Juan Andrés Lagrange Delfino
C.I. v-17.706.289	C.I. v-18.358.109

En la ciudad de Caracas, a los 7 dias del mes de Junio del año 2010

Aprobación

Considero que el Trabajo Final titulado:

DESARROLLO DE UNA PLATAFORMA WEB QUE INTERCONECTE Y ORGANICE GRUPOS UNIVERSITARIOS PERTENECIENTES AL MOVIMIENTO ESTUDIANTIL.

Elaborado por los ciudadanos:

IGNACIO FUENTES Y JUAN ANDRÉS LAGRANGE

para optar al título de

INGENIERO DE SISTEMAS

reúne los requisitos exigidos por la Escuela de Ingeniería de Sistemas de la Universidad Metropolitana, y tiene méritos suficientes como para ser sometido a la presentación y evaluación exhaustiva por parte del jurado examinador que se designe.

En la ciudad de Caracas, a los 7 días del mes de Junio del año 2010.

_____

Ing. Pablo Giolito

ACTA DE VEREDICTO

Nosotros, los abajo firmantes, constituidos como jurado examinador y reunidos en Caracas, el día 7 de Junio de 2010, con el propósito de evaluar el Trabajo Final titulado

DESARROLLO DE UNA PLATAFORMA WEB QUE INTERCONECTE Y ORGANICE GRUPOS UNIVERSITARIOS PERTENECIENTES AL MOVIMIENTO ESTUDIANTIL.

Presentado por los ciudadanos:

IGNACIO FUENTES Y JUAN ANDRÉS LAGRANGE

para optar al título de

INGENIERO DE SISTEMAS

emitimos el siguiente veredicto:

Reprobado ___ Aprobado ___ Notable ___ Sobresaliente ___

Observaciones:

_____

_____

_____

Ing. Pablo Giolito Ing. Pablo Giolito Ing. Pablo Giolito

Agradecimientos

adscasdasasdcapsdqpacdñqancdañadscasdasasdcapsdqpacdñqancdañasd
ascdadscasdasasdcapsdqpacdñqancdañadscasdasasdcapsdqpacdñqancda
ñadscasdasasdcapsdqpacdñqancdañasdascdadscasdasasdcapsdqpacdñqa
ncdañadscasdasasdcapsdqpacdñqancdañadscasdasasdcapsdqpacdñqancd
añasdascdadscasdasasdcapsdqpacdñqancdañ

DEDICATORIA

adscasdasasdcapsdqpacdñqancdañadscasdasasdcapsdqpacdñqancdañasd
ascdadscasdasasdcapsdqpacdñqancdañadscasdasasdcapsdqpacdñqancda
ñadscasdasasdcapsdqpacdñqancdañasdascdadscasdasasdcapsdqpacdñqa
ncdañadscasdasasdcapsdqpacdñqancdañadscasdasasdcapsdqpacdñqancd
añasdascdadscasdasasdcapsdqpacdñqanc

Tabla de Figuras

Tabla de Contenido

Resumen

DESARROLLO DE UNA PLATAFORMA WEB QUE INTERCONECTE Y ORGANICE GRUPOS UNIVERSITARIOS PERTENECIENTES AL MOVIMIENTO ESTUDIANTIL

Autores: Juan Andrés Lagrange Delfino
Ignacio Andrés Fuentes Gimón

Tutor: Ing. Pablo Giolito Caracas, 7 de junio de 2010.

La presente investigación consiste en el diseño y el desarrollo de una plataforma Web, que sea capaz de responder a las necesidades comunicacionales y de organización de los grupos estudiantiles pertenecientes el Movimiento Estudiantil. Para el desarrollo del proyecto se utilizo la metodología de programación extrema aplicado al Framework de Ruby on Rails. El sistema fue basado en las practicas, estándares y convenciones dictados por Ruby in Rails. En ese sentido se trata de un sistema orientado a objetos bajo el esquema MVC. El almacenamiento de los datos se implemento gracias a MySQL y la persistencia se logró mediante la utilización del ORM de Ruby in Rails, Actve Record. Las vistas de la aplicación se desarrollaron en XHTML junto con complementos de JavaScript para lograr coherencia y simplicidad. El diseño de las vistas se complementó a través de la utilización de CSS y su respectivo Framework Blueprint. Se propuso un esquema de implementación que utilizará Phusion Passenger en conjunto con Apache Web Server.

Introducción

El impacto que ha tenido y siguen teniendo las redes sociales sobre nuestra sociedad es abrumador. Actualmente las redes sociales en Internet crecen 47% cada año. A su vez, dos tercios ó 66% de las personas que navegan en Internet, visitan redes sociales. Esas personas que visitan las redes sociales pasaron 143% más tiempo en esas mismas redes sociales este año que el año pasado.

A través de las redes sociales se ha logrado empoderar al ciudadano común y prueba de ello es la iniciativa convocada a través de Factbook llamada “Un millon de voces contra las Farc”. En esta protesta participaron más de 12 millones de personas en más de 199 ciudades del mundo.

Es por ello que en el presente trabajo de grado se decidió poner en práctica los conocimientos adquiridos en la Universidad Metropolitana y optar por el desarrollo de una red social que empodere al movimiento estudiantil venezolano así como a todas las agrupaciones estudiantiles que la quieran utilizar.

La plataforma desarrollada les proporciona a los líderes de estas agrupaciones una herramienta de comunicación entre sí, así como distintas vías de interacción con personas externas al movimiento con el fin de captar voluntarios y de hacer conocer al público las actividades relacionadas con el movimiento estudiantil.

El presente informe documenta el desarrollo de dicha plataforma y está conformado por diversos capítulos. El tema de investigación, Marco

Teórico, Marco Metodológico, Desarrollo, Conclusiones y Recomendaciones, así como su respectivo Apéndice y Citas bibliográficas.

Capítulo I: Tema de Investigación:

Título del Proyecto:

"Desarrollo de una plataforma web que interconecte y organice grupos universitarios pertenecientes al movimiento estudiantil"

I.1. Planteamiento del problema:

Actualmente no existe una manera en que los distintos grupos universitarios tengan contacto permanente entre ellos. Si bien es cierto que redes sociales como *Facebook* han ayudado a otorgarles presencia y comunicación a las diversas agrupaciones universitarias, no existe una manera institucional de preservarlas en el tiempo. Adicionalmente los datos no le pertenecen a las propias instituciones estudiantiles sino a terceros. Esto dificulta la organización y alcance del movimiento estudiantil actual.

Para ello se propone desarrollar una plataforma logística mediante la cual se puedan establecer redes estudiantiles para la convocatoria y toma de decisiones que conciernen a la población estudiantil nacional. Además de una red comunicacional permanente de los distintos grupos que hacen vida en la universidad, previéndolos de las herramientas tecnológicas necesarias para su óptimo desarrollo. De esta manera se consolida una comunidad estudiantil que permita conocerse y convocar a la comunidad a que se integre junto con ellos para aportar efectivamente al desarrollo del país.

I.2. Delimitación del tema

La presente investigación abarcaría desde el desarrollo de la plataforma web hasta su propuesta de implementación. Dicha plataforma se desarrollará bajo el concepto de “web 2.0” y será desplegada en Internet. Debe contener el sistema de registro y seguimiento de usuarios, registro y seguimiento de agrupaciones estudiantiles, creación de actividades, modulo de carga y descarga de archivos, foro y mensajería interna así como el envío de emails.

I.3. Justificación

En la sociedad moderna venezolana, se observa que ha surgido un fenómeno denominado el “movimiento estudiantil”.

A través de los años se ha visto el crecimiento de este movimiento y el despertar de una sociedad civil que día a día se hace más consciente de los problemas que enfrentan los venezolanos. Clara evidencia de esto es la campaña electoral que se presentó para la votación de la enmienda constitucional del 15 de Febrero del 2009. En esta oportunidad, la participación de los partidos políticos fue mínima en comparación con votaciones previamente efectuadas y se vio como los estudiantes se enfrentaron a toda la maquinaria propagandista del gobierno. Los resultados si bien no eran los esperados por el movimiento estudiantil, están muy lejos de ser despreciables.

Sin embargo estos resultados, a pesar de que se han hecho grandes esfuerzos organizativos, en su mayoría se deben a la resolución de conflictos puntuales y no enmarcados dentro de un proyecto a largo plazo ó según una estrategia bien formulada. Claro ejemplo de esto es la iniciativa “noesno.net”, en donde se levantó una importante base de datos de voluntariado y se logró

organizar al movimiento estudiantil conjunto a la sociedad civil con el propósito de promover la vigilia de los votos de las elecciones del 15 de Febrero del 2009. En este sentido, la iniciativa de “noesno.net”, fue el producto de una disyuntiva particular.

Por consiguiente, se ve la necesidad de contar con una estructura más formal para la resolución de futuros conflictos.

Los problemas organizativos del movimiento estudiantil se extienden más allá de la estrategia de vigilia electoral y se evidencia la falta de organización interna. Para entender un poco más, se debe primero dar una definición de lo que es el movimiento estudiantil.

El movimiento estudiantil se le puede definir por aquellos estudiantes de la sociedad civil venezolana que se encuentran activamente involucrados en la política del país. En este sentido involucraría desde los niños de primaria y bachillerato hasta los jóvenes y adultos de pre-grado y post-grado universitario. Sin embrago los protagonistas de dicho movimiento se han caracterizado por ser los estudiantes de pregrado universitario a nivel nacional.

Actualmente para organizarse cuentan con las estructuras políticas que brinda cada casa de estudio. Es decir, los centros de estudiantes y federaciones de centros de estudiantes de cada universidad eligen a sus propios líderes. Luego dichos líderes buscan la manera para comunicarse entre sí y definir las estrategias a tomar para una situación determinada. Para establecer esta comunicación, se acuden a medios como conocidos y amigos de otras casas de estudio y herramientas tecnológicas como “Facebook”.

Sin embargo, la comunicación entre centros de estudiantes no es tan sencilla, especialmente en el interior del país. Esto se debe principalmente a que los centros estudiantiles cambian cada año y por lo tanto deben todos los años construir o reconstruir alianzas nuevas con otros centros de estudio. A su vez, implica que los estudiantes no tienen experiencia ni memoria a largo plazo y por lo tanto deben reaprender de errores probablemente cometidos por centros estudiantiles anteriores.

I.4. Objetivo general

Desarrollar una red social “web 2.0”, que interconecte y provea herramientas comunicacionales a las distintas agrupaciones universitarias de Venezuela.

I.4.1. Objetivos específicos

- Determinar las necesidades de las agrupaciones universitarias en cuanto a herramientas comunicacionales se refiere.
- Desarrollar el módulo de seguridad basado en roles y usuarios que administre las tareas permitidas por cada usuario y mantenga la información delicada oculta.
- Desarrollar el modulo de carga y descarga de archivos que permita a los usuarios nutrir y administrar los contenidos de la red social.
- Desarrollar una interfaz sencilla y fácil de utilizar que proporcione las herramientas para asistir en la interconexión y comunicación de las agrupaciones estudiantiles.

- Desarrollar las pruebas modulares, de integración y de seguridad pertinentes para garantizar el buen funcionamiento del sistema.
- Desarrollar un plan de implementación del sistema propuesto en web.

Capítulo II: Marco Teórico

Para la creación de la plataforma web se tomaron una serie de decisiones que garantizaran que la aplicación final estuviese acorde con los más modernos estándares de aplicaciones “web 2.0”. Entre las consideraciones tomadas se encuentran las siguientes:

- La aplicación será desarrollada mediante la metodología de programación extrema.
- Se utilizará el Framework Ruby on Rails combinado con su respectivo ORM.
- Se utilizará la arquitectura REST combinada con el patrón de desarrollo MVC.
- Se utilizarán prácticas comunes “WEB 2.0”.
- Para mantener la integridad de la interfaz, se utilizara el *Framework* “*Blueprint*” para manejar el diseño a través de CSS.

II.1. Movimiento Estudiantil:

II.1.1. ¿Qué es el movimiento estudiantil?

El movimiento estudiantil venezolano es un movimiento conformado por los estudiantes venezolanos que buscan lograr cambios en la sociedad. Está conformado en su mayoría por jóvenes universitarios y se caracteriza por ser apartidista.

Se dice que el movimiento estudiantil nació a partir del anuncio de cierre de RCTV (Radio Caracas Televisión) el 27 de Mayo del 2007. A partir de este momento se coincide que la juventud venezolana, hasta su momento apática ante la política, despertó y comenzó a jugar un rol importante dentro de la política nacional.

Muchos líderes y actuales figuras públicas comenzaron su carrera política dentro del movimiento estudiantil. Este es el caso de Yon Goicoechea, Stalin González, Freddy Guevara, entre otros.

II.1.2. Estructura del movimiento estudiantil

Actualmente el movimiento estudiantil no cuenta con una estructura bien definida. Sin embargo se pudiera decir que la directiva está conformada por los presidentes de los centros de estudiantes de las distintas casas de estudio. Cada uno de ellos es electo por un proceso electoral según los estatutos de cada casa de estudio. Luego la estructura y logística es independiente según cada universidad.

Es importante resaltar que el proceso de elección de los líderes estudiantiles se efectúa anualmente, por lo que todos los años cambia la directiva del movimiento.

II.1.3. Agrupaciones jóvenes y universitarias

Dentro de cada universidad existen diversas agrupaciones que cumplen labores extracurriculares. Estas agrupaciones cumplen funciones tan diversas como gustos y *hobbies* existan dentro del estudiantado de cada casa de estudio. Existen agrupaciones formativas, como es el caso de los MUN (Model United Nations), en el cual se forman estudiantes para competir

en simulacros de las naciones a nivel mundial. Otras como FSAE (Formula Society of Automotive Engineers), en las cuales las diferentes agrupaciones compiten por construir el mejor carro de carrera. Los centros de estudiantes de cada carrera son un lugar para aquellos estudiantes con vocación de servicio y ansias de lograr una mejora en su casa de estudio. De igual manera existen los grupos de teatro, así como los clubes de lectura. Todas ellas son agrupaciones universitarias que hacen que la vida dentro de la universidad sea más diversa y su formación sea más rica e integral.

Figura 1: Logos de agrupaciones estudiantiles venezolanas. Fuente: Elaboración Propia

De igual manera existen agrupaciones e iniciativas llevadas por jóvenes que no tienen su sede dentro de una universidad. Tal es el ejemplo de Voto Joven, una organización sin fines de lucro conformada por estudiantes universitarios de las distintas casas de estudio que buscan fomentar la

participación electoral así como la defensa de los valores democráticos. Como este ejemplo, también se conoce la iniciativa de la fundación “soñar despierto”, una fundación liderada por jóvenes universitarios que busca fomentar el desarrollo de la comunidad a través de eventos orientados a la niñez necesitada del país.

II.2. Antecedentes:

II.2.1. Redes Sociales en línea (networking):

Desde la aparición de Internet se ha manejado la idea de que distintos individuos pudiesen relacionarse e interactuar. Los primeros esfuerzos fueron en los años 80, con la aparición de “Usenet”, “Arpanet” y “Listserv”. Estos eran servicios sumamente primitivos que básicamente brindaban información a individuos con intereses comunes. Se pueden comparar con los foros actuales, pero mucho más antiguos.

Las primeras comunidades web en línea que permiten la interacción de los usuarios en tiempo real, vinieron a partir de 1985 con la aparición de sitios web como “The Well”, “Theglobe.com” y “Geocities”. En estas páginas ya se utilizan los famosos perfiles de usuario para compartir información, así como la creación de páginas personales, precursoras a los actuales “Blogs”. Entre otras de las bondades de estas páginas web se encuentra la aparición de las famosas salas de chat. En estas páginas web a distintos usuarios se les permitía interactuar y conversar en tiempo real con otros usuarios alrededor del mundo. Esto rápidamente se volvió sumamente popular e inició una explosión de servicios de chat a nivel mundial.

Figura 2: Logotipos de redes sociales. Fuente: http://shamzu.files.wordpress.com/2009/05/socialnetworks.jpg

La próxima pauta en los llamados servicios de redes sociales fue hecha por paginas tales como “Classmates.com” en 1995. Este sitio web tenía como factor diferenciador la capacidad de poder buscar antiguos compañeros de estudio a los cuales se les podía enviar mensajes y se les permitía compartir información a través de las llamadas “listas de amigos”. A partir de esta novedosa idea, se originó un fenómeno muy parecido al iniciado por los llamados “chats” y explotaron al mercado uno tras otro servicio para las redes sociales. Entre ellos: “Friendster”, “Myspace”, “Bebo”, “Facebook” y “Twitter” son los más utilizados hoy en día.

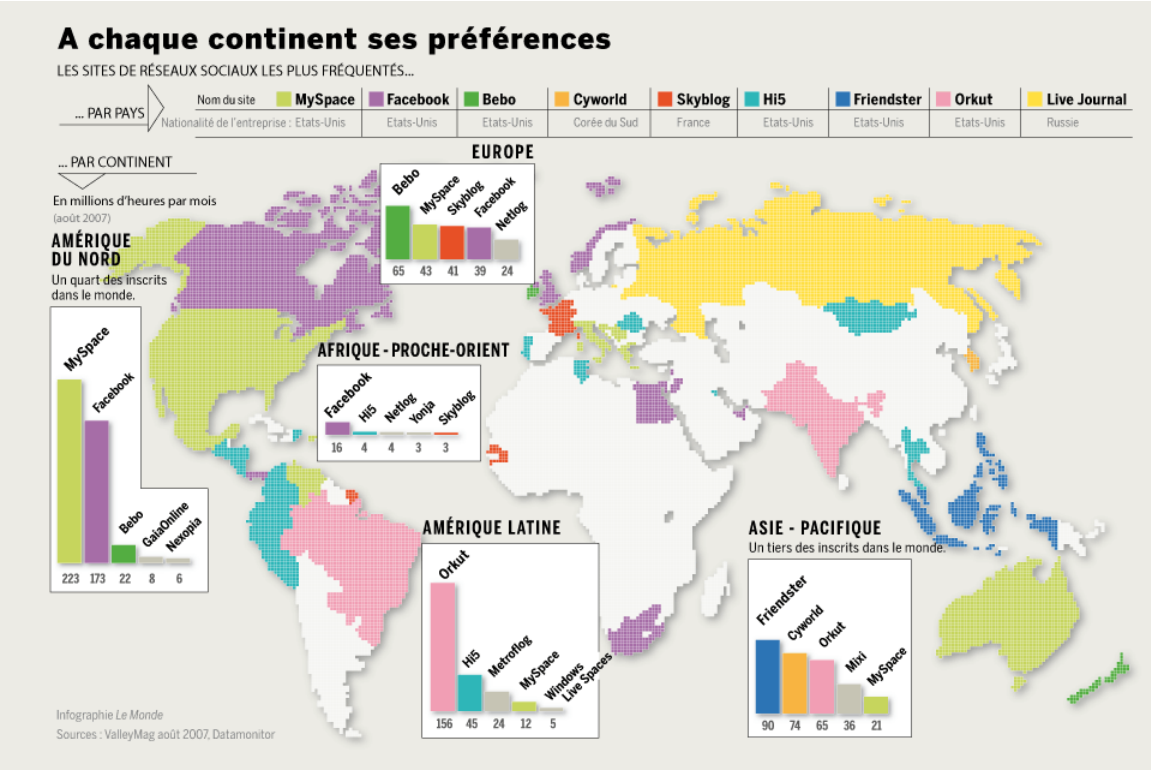

Figura 3: Mapamundi de las redes sociales según el país y según red social para el 2008. Fuente: http://www.adwords-articles.com/blog/wp-content/uploads/2008/02/socialnetworkingmap.gif

II.2.2. Impacto social

Entre los principales debates que se lleva entorno al impacto que tienen las redes sociales en línea, se encuentra la pregunta de si verdaderamente genera capital social. El capital social se define como la capacidad de que una red de individuos con intereses similares pueda amoldar y cambiar el entorno que los rodea. Mientras mayor sea el grupo de individuos, mayor será su capital social, y por tanto su capacidad para generar cambios en su entorno. Una manera sencilla de visualizar el capital social es comparándolo con el llamado “capital humano”. Si bien el capital humano aumenta la productividad del individuo y del colectivo, de la misma manera el capital social y la interconexión entre individuos puede aumentar su capacidad para ejercer cambios.

De esta manera se ha estudiado el impacto de las redes sociales desde el enfoque de la capacidad de la sociedad civil para organizarse. En este sentido existen miles de ejemplos que indican que los servicios de redes sociales han logrado un impacto impresionante sobre su entorno. Uno de los ejemplos más notorios de la actualidad es la campaña electoral creada por el actual presidente de los Estados Unidos, Barack Obama, quien en su campaña electoral lanzó la página web “My.BarackObama.com”. En ella, utilizó herramientas de redes sociales para recaudar fondos y hacer llegar su mensaje a la mayoría de los votantes jóvenes (entre 18 y 29 años). Para muchos analistas políticos esta estrategia fue fundamental para el éxito. Otro ejemplo famoso a nivel de América latina fue la protesta “Un millón de voces contra las FARC”. Esta fue una protesta pacífica organizada en más de 200 ciudades del mundo y lo logró todo a través del portal de Facebook.

II.3. Desarrollo Web:

Desde los mediados de los noventas, el desarrollo web ha sido una de las industrias con mayor crecimiento en el mundo. En 1995 había menos de 1000 compañías de desarrollo web en los Estados Unidos. Para el año 2005 ya había más de 30000 compañías de este campo.

Hoy en día, desarrollo web es un término utilizado para referirse a cualquier desarrollo, sea de aplicaciones en línea, páginas de presentación o animación, tiendas virtuales, scripts de cliente o de servidor, etc. cuyo fin resultante sea un producto que se ejecuta desde un navegador web.

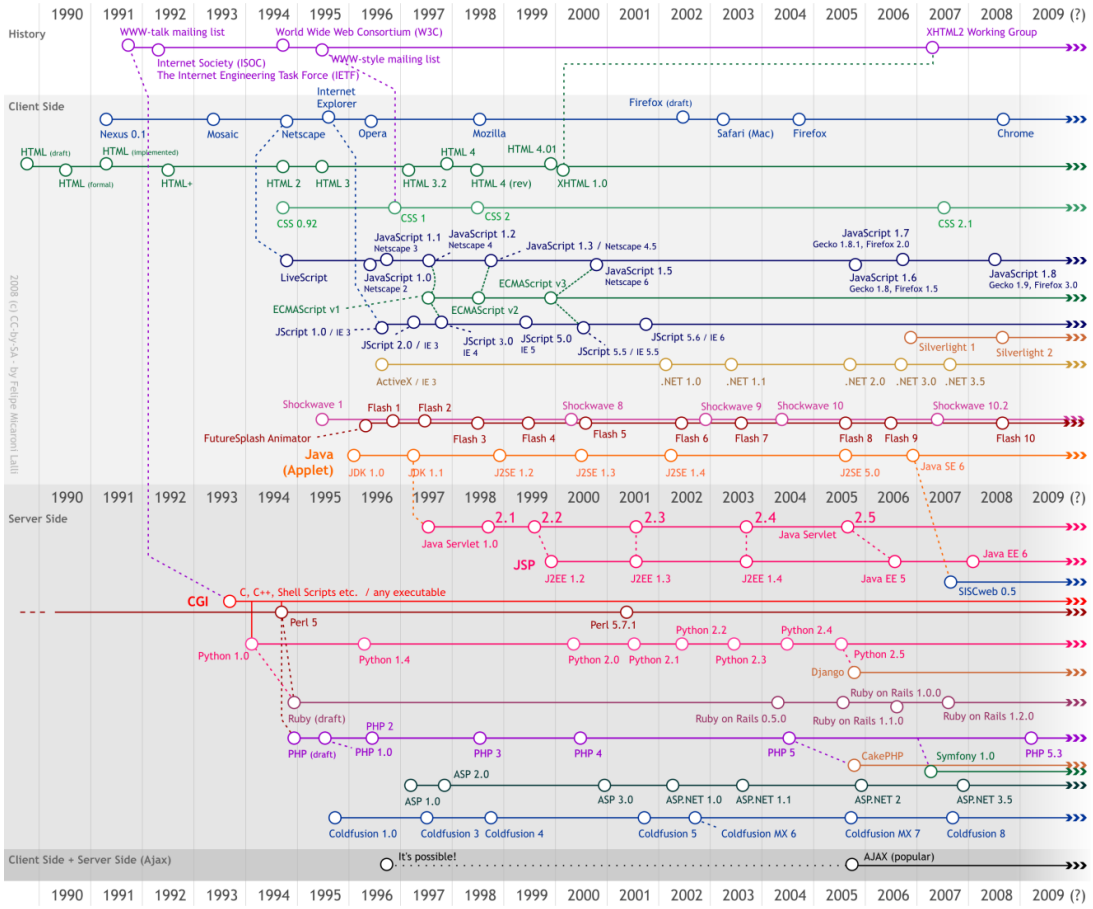

Figura 4: Cronología de eventos y hechos importantes para el desarrollo web. Fuente: http://upload.wikimedia.org/wikipedia/commons/e/e4/Web_development_timeline.png

II.4. HTML

HTML, *HyperText Markup Language* (Lenguaje de Marcado de Hipertexto), es el lenguaje de marcado predominante para la elaboración de páginas web y es usado para describir la estructura de una página web a través de texto. Se caracteriza por estar escrito a través de etiquetas rodeadas por signos de “mayor que” y “menor que” (<,>). Un archivo HTML está principalmente constituido de varios *elementos*.

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML
2  <html>
3      <head>
4          <title>Example</title>
5          <link href="screen.css" rel="sty
6      </head>
7      <body>
8          <h1>
9              <a href="/">Header</a>
10         </h1>
11         <ul id="nav">
12             <li>
13                 <a href="one/">One</a>
14             </li>
15             <li>
16                 <a href="two/">Two</a>
17             </li>
```

Figura 5: Elementos HTML. Fuente: Elaboración Propia

II.4.1. Elementos

Los elementos son la estructura básica de HTML. Tienen dos componentes básicos, que son atributos y contenido. Generalmente, un elemento tiene una etiqueta de inicio (`<algún-elemento>`) y una etiqueta de cierre (`</algún-elemento>`), estando sus atributos contenidos en la etiqueta de inicio y el contenido entre la etiqueta de apertura y de cierre (`<algún-elemento atributo="mivalor">Mi Contenido</algún-elemento>`). Algunos elementos, como `<br>` o `<img>`, no tienen contenido y por lo tanto no llevan etiqueta de cierre.

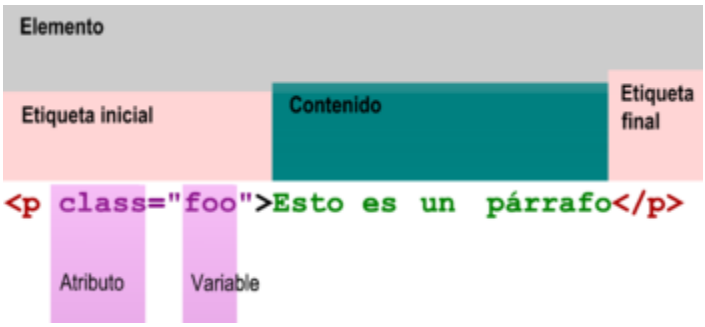

Figura 6: Estructura de un elemento HTML. Fuente: http://es.wikipedia.org/wiki/Archivo:Etiquetas_en_HTML.png

II.5. XHTML

XHTML, *Extensible Hypertext Markup Language*, (lenguaje extensible de marcado de hipertexto), es el lenguaje pensado para sustituir a HTML como estándar para páginas web. XHTML tiene, básicamente, las mismas funcionalidades que HTML pero cumple especificaciones más estrictas.

II.5.1. HTML vs. XHTML

A continuación se presentan algunas de las diferencias primordiales entre XHTML y HTML. Se marca como incorrecto lo que era válido para HTML pero no lo es para XHTML:

- Los elementos deben cerrarse siempre, independientemente de que tengan o no contenido:
    - Incorrecto:
    - Correcto:
</br> o
 o
- Los elementos no vacíos deben cerrarse siempre:
    - Incorrecto: Primer párrafo Segundo párrafo
    - Correcto: Primer párrafo Segundo párrafo
- Los elementos anidados deben tener correspondencia al abrirse y cerrarse.
    - Incorrecto: <h1>Texto</h1>
    - Correcto: <h1>Texto</h1>
- Los valores de los atributos deben siempre ir entre comillas.
    - Incorrecto: <td rowspan=3>
    - Correcto: <td rowspan="3">
- Los nombres de elementos y atributos deben ir en minúsculas.

    - Incorrecto: <A HREF="http://www.domname.com">Domname</A>
    - Correcto: Domname
- No está permitida la minimización de atributos (se usa el nombre del atributo como valor).
    - Incorrecto: <textarea readonly>Solo-lectura</textarea>
    - Correcto: <textarea readonly="readonly">Solo-lectura</textarea>

- El texto no debe ser insertado directamente en el cuerpo.
    - Incorrecto: <body>Texto plano</body>
    - Correcto: <body>Texto plano</body>

II.6. CSS

Cascading Style Sheets (CSS) es un lenguaje de hojas de estilo usado para describir la presentación (formato y estilo) de documentos escritos en lenguaje de marcado (páginas web).

CSS está diseñado principalmente para permitir la separación entre el contenido del documento (escrito en HTML, XHTML o algún lenguaje de marcado) y la presentación del documento. Esta separación provee mayor control y flexibilidad en la especificación de las características de presentación, facilita la implementación de un mismo estilo en distintas páginas y reduce complejidad y repetición en el contenido estructural.

Una hoja de estilo (CSS) está conformada por varias *reglas*. Cada regla tiene uno o más *selectores* y un *bloque de declaración*. Un bloque de declaración es una lista de *declaraciones* entre llaves ({ , }). Cada

declaración está compuesta por una *propiedad*, dos puntos(:) , un *valor* y un punto y coma (;).

En la figura 7, se muestra una imagen de una página web sin alguna hoja de estilo asignada. Luego en la figura 8, se muestra un ejemplo de una sección de una hoja de estilo en código CSS. Por último en la figura 9 se muestra la misma página web ilustrada en la figura 7, pero utilizando la hoja de estilo de la figura 8.

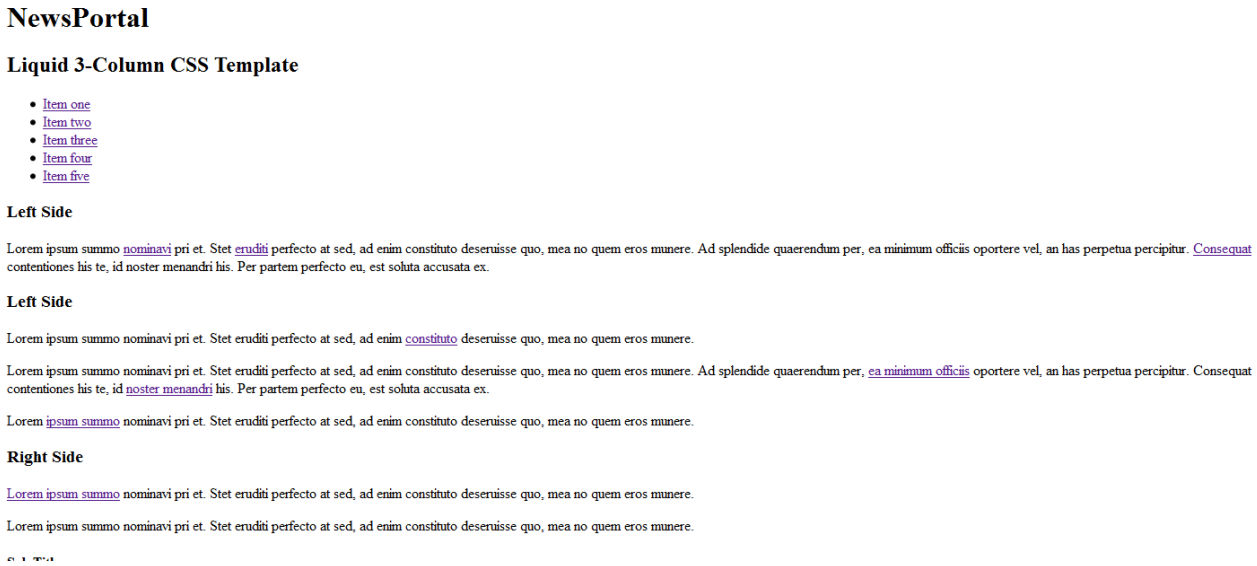

Figura 7: Página web sin hoja de estilo. Fuente: Elaboración propia

```css
body {
  text-align: center;
  margin-top:10px;
  margin-bottom:10px;
  color:#666666;
  background-color: #E0E0E0;
}

A:link {
    COLOR: #0000FF; text-decoration: none;
}
A:visited {
    COLOR: #0000FF; text-decoration: none;
}
A:active {
    COLOR: #0000FF; text-decoration: none;
}
A:hover {
    COLOR: #FF0000; text-decoration: underline;
}

#page_wrapper {
  margin-left: auto;
  margin-right: auto;
  width: 98%;
  text-align: left;
  background: #FFFFFF;
  border: 8px solid #FFFFFF;
}
```

Figura 8: Código fuente de una hoja de estilo. Fuente: Elaboración propia

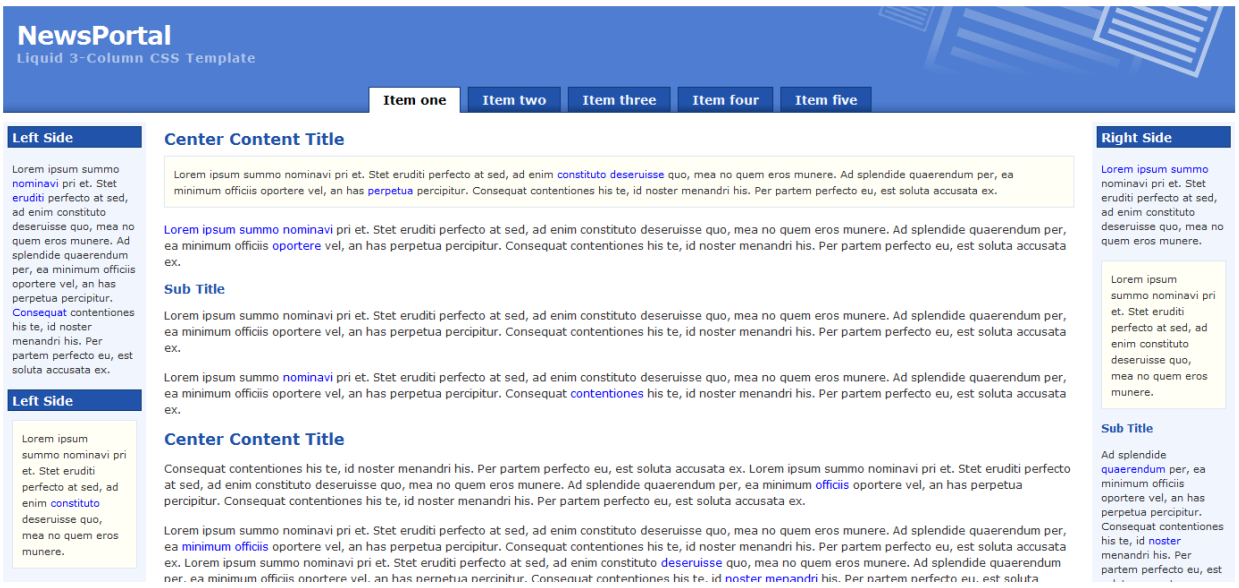

Figura 9: La misma página web de la Figura 7, ahora con la hoja de estilo de la Figura 8. Fuente: Elaboración propia

II.6.1. Blueprint CSS

Blueprint es un Framework de CSS diseñado para simplificar el desarrollo de una interfaz web. Al utilizar este framework, los usuarios pueden usar una serie de clases CSS que pueden ser comunes a través de distintos sitios web dando uniformidad al desarrollo de diseño gráfico de las aplicaciones sin sacrificar creatividad artística. Uno de sus mayores beneficios es que asegura la intercompatibilidad entre navegadores y mantiene una interfaz visualmente similar entre ellos.

Figura 10: Logotipo de Blueprint. Fuente: http://honeyl.public.iastate.edu/blueprint/blueprint-css.gif

II.7. REST:

Es una arquitectura de la ingeniería de software que consiste en el modelo cliente-servidor. Los clientes hacen pedidos al servidor quien a su vez procesa los pedidos y devuelve las respuestas apropiadas. Tanto los pedidos como sus respuestas están construidos bajo el concepto de representación de recursos. Un recurso se refiere a un modelo conceptual de cualquier información útil. La representación de un recurso se refiere normalmente a la captura del estado de un recurso dado en un momento determinado.

II.8. Desarrollo para el lado del servidor:

Como es de esperarse, la mayor parte del esfuerzo en un desarrollo web, es realizado en el servidor.

Para que una página web pueda ser considerada parte de una “aplicación web” tiene que haber sido previamente procesada por un servidor para que su contenido dependa de distintos factores, parámetros y variables a fin de mostrar como resultado una página distinta al usuario que varíe en función de dichos parámetros. A diferencia de una página estática, la cual no tiene necesidad de ser procesada por un servidor sino más bien ser simplemente proveída por éste.

En definitiva, el código que reside en el servidor es ejecutado cuando se recibe un *request* por parte del cliente lo que genera código HTML, XHTML, entre otros, específico para ese cliente, que es mostrado por el navegador.

Es importante destacar, que en el mundo del desarrollo web, existen lenguajes estáticos y lenguajes dinámicos (strongly typed vs. loosely typed). Ambos con ventajas y desventajas.

Los lenguajes estáticos requieren que las variables sean debidamente definidas de un solo tipo durante todo el programa mientras que en un lenguaje dinámico es común encontrarse que una variable en una sección del programa esté definida de un tipo para luego cambiar totalmente su clase.

Los lenguajes estáticos poseen la ventaja de que a la hora de editar (es decir, antes de compilar) ya pueden reportar errores en el código. Esta ventaja no la tienen los lenguajes dinámicos precisamente por esa flexibilidad

que ofrecen a la hora de programar. Los errores, en caso de haberlos, son reportados durante la propia ejecución de los scripts.

Existen múltiples lenguajes de programación del lado del servidor. En el pasado la alternativa de Microsoft era “ASP” mientras que la alternativa de software libre con más uso era PHP. Hoy en día existen muchas más opciones de “server-side”. Microsoft ofrece su plataforma ASP.NET, la cual permite programación en Visual Basic, C++, o C#. Sun Microsystems también tiene su alternativa orientada a objetos para desarrollo web en Java Server Pages.

Por parte de software libre sigue estando PHP en el liderazgo en popularidad pero con el lenguaje Ruby presentándose como una opción que cada vez más es adoptada por los programadores enfocados en el desarrollo rápido de las aplicaciones.

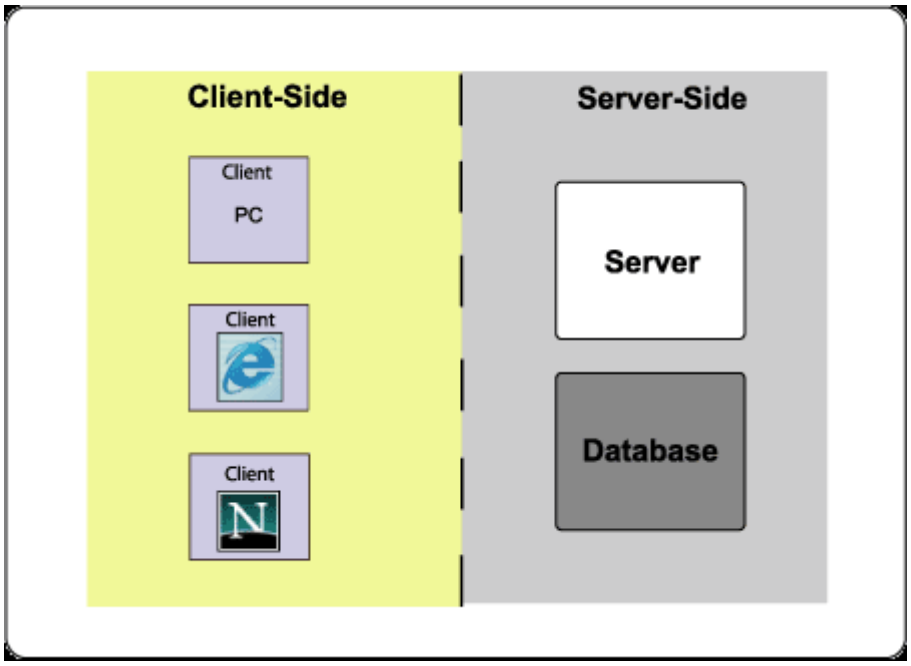

Figura 11: Diferencias entre un servidor y un cliente. Fuente:http://wiki.eeng.dcu.ie/ee557/287-EE/version/default/part/ImageData/data/server-side_intro.gif

II.8.1. Ruby:

Es un lenguaje dinámico de propósito general que se originó en Japón a mediado de los noventa creado por Yukihiro Matsumoto.

Ruby soporta distintos paradigmas de programación incluyendo funcional, orientado a objetos, imperativo y reflectivo. Asimismo, tiene un sistema de manejo de tipos dinámicos y un manejo automático de memoria.

Este lenguaje es, hoy en día, mayormente conocido por ser la base para el famoso *framework* de desarrollo web "Ruby on Rails", hasta el punto que ambos suelen ser intercambiados indiferentemente cuando se habla de ellos.

Figura 12: Logotipo de Ruby. Fuente: http://3columns.net/habitual/docs/images/ruby.png

II.9. Framework de desarrollo web:

Un *framework* es una abstracción en la cual rutinas comunes de código que proveen funcionalidades genéricas son parametrizadas para facilitar la implementación de funcionalidades comunes en sistemas web. El objetivo principal es reducir el tiempo requerido para llevar a cabo funcionalidades que son comunes en distintos sistemas.

Hoy en día existen distintos *frameworks* de desarrollo web en los distintos lenguajes disponibles; como lo son JavaEE, Zend *framework*, ASP.net, Ruby on Rails, etc.

II.9.1 Ruby On Rails

Ruby on Rails es un *framework* de desarrollo web *open source* para el lenguaje Ruby. Este *framework* está diseñado para ser usado con alguna metodología de desarrollo ágil como lo es la *Programación Extrema*.

Figura 13: Logotipo de Ruby on Rails. Fuente: http://3columns.net/habitual/docs/images/rails.png

II.9.1.1 Historia

Ruby on Rails fue creado por David Heinemeier Hansson quién lo ofreció como una distribución de software libre en el 2004. Ruby on Rails empezó a adquirir atención y popularidad a partir del 2006 cuando Apple Computers anunció que las siguientes entregas de su Sistema Operativo Mac OS se lanzarían con Ruby on Rails previamente instalado.

II.9.1.2 Estructura

Ruby on Rails utiliza la arquitectura *Model-View-Controller* para organizar la programación de aplicaciones.

Ruby on Rails provee herramientas que facilitan la implementación de actividades comunes en el desarrollo web. Algunas de estas herramientas son *scaffolding,* que a través de un comando puede crear todo lo necesario (controlador, modelo, vistas y especificación de rutas) para una página web dinámica básica. También incluidos con *Rails* están *WEBrick,* un servidor web sencillo para Ruby, y *Rake*, un módulo de comandos para automatizar la generación de códigos.

Rails también es conocido por utilizar librerías de JavaScript (como *Prototype*, *Script.aculo.us* y *Jquery*) para facilitar la implementación de acciones con *Ajax*.

Rails está separado en varios paquetes. Específicamente *ActiveRecord* (un ORM), *ActiveResource* (que provee servicios web), *ActionPack*, *ActiveSupport* y *ActionMailer*.

Con Ruby on Rails se pretende enfatizar los principios de *Convention over Configuration* (CoC) y el principio DRY (*Dont Repeat Yourself*)

Convention over Configuration significa que el desarrollador sólo debe especificar los aspectos no convencionales de su aplicación. Por ejemplo, si hay una clase “Carro” en los modelos del sistema, la tabla en base de datos correspondiente deberá ser “carros”, es decir, la pluralización del nombre de la clase en cuestión. Si hay una desviación de este paradigma, como lo sería llamar a la tabla “carros_vendidos” entonces será necesario que el

desarrollador escriba código relacionado a estos nuevos nombres. Este principio lleva a menos código y menos repetición.

Don't repeat yourself se refiere a que la información está en un solo sitio. Por ejemplo, al usar el modulo ActiveRecord de Rails, el desarrollador no tendrá que especificar los nombres de las columnas en la definición de las clases. En vez, Rails obtendrá esta información de la base de datos según el nombre que se le dé a la clase (el cuál será la versión singularizada del nombre de la tabla).

II.9.1.3 RubyGems

RubyGems es un manejador de paquetes para el lenguaje de programación Ruby que proporciona:

- Un formato estándar para distribuir programas, *plugins* y librerías hechas en Ruby de una forma auto-contenida en paquetes llamados *gems.*
- Una herramienta diseñada para manejar con facilidad la instalación de *gems*.
- Un servidor para la distribución de las *gems*.

Figura 14: Logotipo de Ruby Gems. Fuente: http://www.maestrosdelweb.com/images/2009/11/rubygems.png

II.10. Desarrollo para el lado del cliente

El desarrollo para el lado del cliente se refiere a la programación de métodos y funciones que se compilan y se ejecutan completamente desde el equipo cliente de alguna aplicación, sin que el servidor tenga ninguna influencia.

En el desarrollo web sólo existe un una herramienta “stand-alone” para la programación del lado del cliente, la cual es JavaScript.

Otras alternativas existen para ejecutar scripts desde el lado del cliente pero estas requieren algún tipo de “plugin” o extensión a los navegadores lo que las hacen menos atractivas a la hora de asegurarse de que la aplicación funcione en el mayor número de computadores posible.

II.10.1. JavaScript:

Es un lenguaje orientado a objetos utilizado primordialmente para aplicaciones web.

El código se ejecuta completamente en el lado del cliente (usuario) y sirve para darle riqueza a la interfaz gráfica y dinamismo a las páginas web sin la necesidad de que el servidor de la aplicación ejecute ninguna instrucción.

El código JavaScript puede formar parte del archivo XHTML.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head><title>simple page</title></head>
  <body>
    <script type="text/javascript">
      document.write('Hello World!');
    </script>
    <noscript>

Your browser either does not support JavaScript, or you have JavaScript turned off.
    </noscript>
  </body>
</html>
```

Figura 15: Ejemplo de código Javascript dentro de un archivo XHTML. Fuente: http://en.wikipedia.org/wiki/JavaScript

Sin embargo, es preferible que se le provea al HTML/XHTML en un archivo aparte para mantener una sólida separación entre lógica y diseño (markup).

```
<script type="text/javascript" src="hello.js"></script>
```

Figura 16: Ejemplo de importación de un archivo Javascript. Fuente: http://en.wikipedia.org/wiki/JavaScript

II.10.1.1. jQuery

jQuery es una librería de Javascript compatible con múltiples navegadores que simplifica la programación del lado del cliente en páginas web en XHTML/HTML.

Salió el mercado en el 2006 y hoy en día, de las 10.000 páginas más visitadas del mundo el 27% de estas utiliza jQuery de alguna manera u otra.

Figura 17: Logotipo jQuery. Fuete: http://benjaminsterling.com/wp-content/uploads/2008/11/jquery_logo_color_onwhite.png

La sintaxis de jQuery facilita seleccionar elementos con los que se va a interactuar del documento, crear animaciones, manejar eventos (como movimiento y presionar botones del *mouse* o del teclado) y comportamientos Ajax.

II.10.1.1.1. Uso jQuery

Para utilizar la librería jQuery, basta con descargarse el archivo y referenciarlo en el XHTML.

```
<script type="text/javascript" src="jQuery.js"></script>
```

Figura 18: Ejemplo de inserción de librerías jQuery en archivo HTML. Fuente: Elaboración propia

Generalmente, accesar y manipular uno o varios elementos del XHTML comienza con llamar la función *$* con un selector de CSS. Por ejemplo:

```
$("div.test").add("p.quote").addClass("blue").slideDown("slow");
```

Figura 19: Ejemplo de manipulación de elementos con jQuery. Fuente: Elaboración propia

En el ejemplo presentado, a través de la función $ se está seleccionando todos los elementos en el XHTML que tengan el atributo clase *test* y todos los <p> que tengan el atributo clase *quote.* A todo lo seleccionado se le agregará la clase *blue* y se le aplicará el efecto slideDown. (Este efecto está definido en las animaciones de jQuery).

Es importante destacar que jQuery también cuenta con un portal de plugins que extienden altamente sus funcionalidades por defecto, algunas desarrolladas por externos o por el mismo equipo de creadores de jQuery.

II.0.1.2. jQuery UI

jQuery es una librería de *widgets* e *interacciónes* que se basa en las funcionalidades más primitivas de jQuery. Su objetivo es facilitar la creación de páginas web altamente interactivas.

Los widgets son controles de interfaz de usuarios. Todos tienen total flexibilidad para personalizar colores y opciones. Entre los widgets más importantes están:

- ***Accordion*:** Este *widget* funciona para fácilmente proporcionar una interfaz de acordeón en una página web.

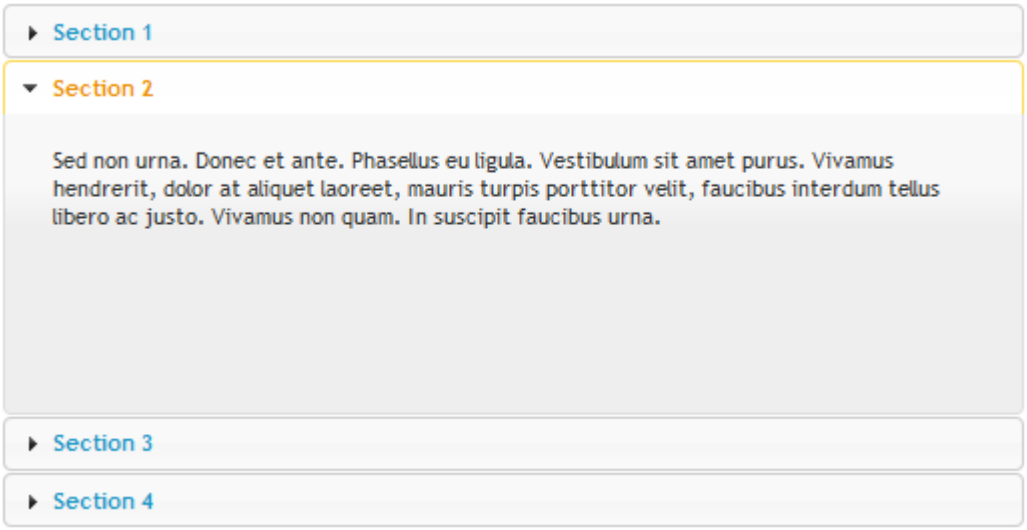

Figura 20: Ejemplo de función acordeón de jQuery UI. Fuente: http://jqueryui.com/demos/accordion/

- **Datepicker:** Este widget convierte un elemento *<input type="text"*>de XHTML en un seleccionador de fechas.

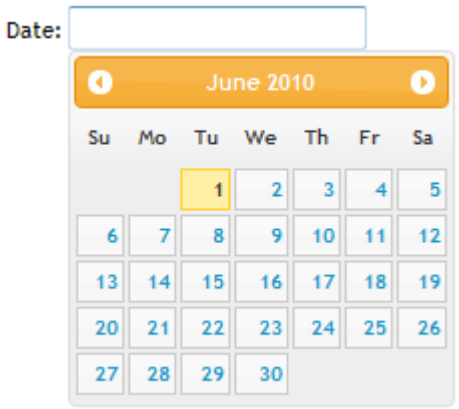

Figura 21: Ejemplo de función para elegir fechas de JQuery UI. Fuente: http://jqueryui.com/demos/datepicker/

- **Tabs:** Este widget funciona para fácilmente proporcionar una interfaz de tabs.

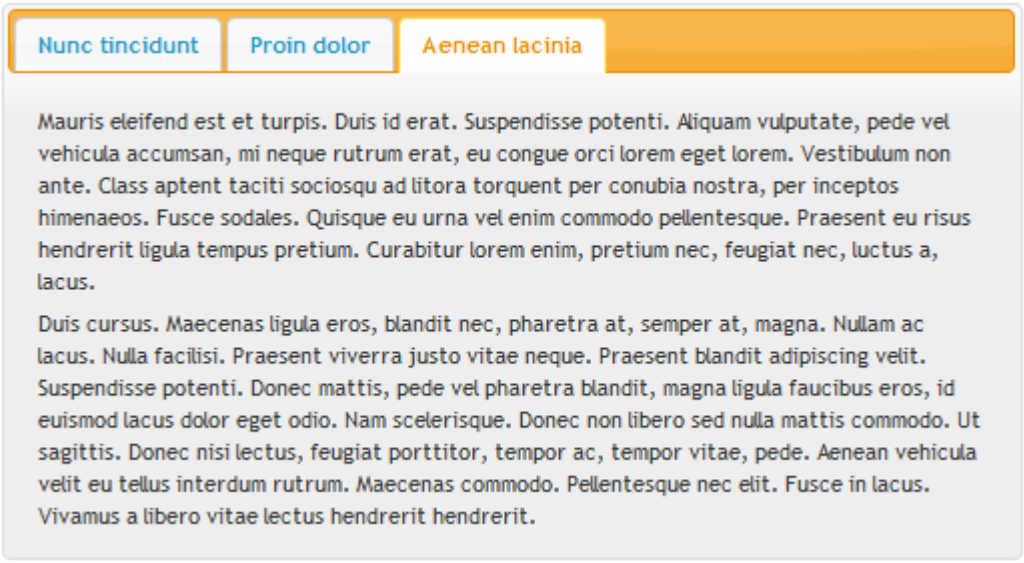

Figura 22: Ejemplo de función de tabs de jQuery UI. Fuente: http://jqueryui.com/demos/tabs/

- **Auto-Complete:** Proporciona la función de autocompletar, mientras se va escribiendo, a un cuadro de texto para fácil selección de algún elemento específico.

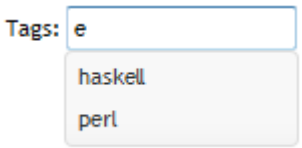

Figura 23: Ejemplo de funcion de autocompletar de jQuery UI. Fuente: http://jqueryui.com/demos/autocomplete/

Además de todos estos widgets, Jquery UI provee interacciones a prácticamente cualquier elemento XHTML para darle dinamismo. Entre estas interacciones están:

- **Draggable**: El elemento se vuelve arrastrable en la página.
- **Droppable**: El elemento además de ser arrastrable puede ser soltado en un lugar que registra que fue soltado (esto es de gran funcionalidad para un carrito de compras en línea).

- **Resizable**: El elemento puede ser fácilmente cambiado de tamaño.
- **Selectable**: El elemento puede ser seleccionado.
- **Sortable**: Una serie de elementos pueden ser ordenados interactivamente

II.11. Base de Datos:

Una base de datos consiste en una recolección organizada de información. Es decir, consiste en clasificar la información según criterios que luego permitan la búsqueda y acceso sencillo de la información.

En cuanto a desarrollo de software, se utilizan Sistemas de Manejo de Bases de Dato, llamados en ingles *Database Management Systems* (DBMS). Un DBMS es un software que se encarga de la creación, manejo, mantenimiento y uso de las estructuras de datos que se van a almacenar. De esta manera le permiten al usuario almacenar y buscar información de forma estructurada junto con que simplifican el proceso de integridad, concurrencia y persistencia de datos.

Es importante mencionar que los DBMS utilizan *query languages* ó lenguajes de *query* para accesar la base de datos. Uno de los lenguajes más utilizados y con el cual también trabaja Ruby on Rails es SQL.

Existen distintos tipos de base de datos, los principales son RDBMS que son los sistemas de base de datos relacionales y los ODBMS que son los sistemas de bases de datos orientados a objetos. No viene al caso

discutir las diferencias entre ambos sino más bien señalar los puntos que tienen en común:

- Los contenidos de la base de datos se almacenan en una matriz
- En dicha matriz, las columnas representan un atributo del modelo mientras que las filas representan una instancia de ese modelo.
- Cada matriz en su totalidad representa un modelo y por tanto son almacenadas en tablas.
- Las relaciones entre un modelo y otro se manejan a través de apuntadores ó también llamados claves foráneas.

II.11.1. SQL

SQL, conocido como Structured Query Language es un lenguaje de base de datos diseñado para manejar data de las bases de datos relacionales. Fue creado en 1970 por Edgar Codd y es actualmente el lenguaje más utilizado para manejar bases de datos relacionales.

II.11.1.1 MySQL

Es un RDBMS que provee acceso multiusuario a distintas bases de datos. Como su nombre lo indica, utiliza *SQL* como su lenguaje para manejar las bases de datos. Es importante resaltar que MySQL es un proyecto *open source* y se encuentra entre los principales RDBMS hoy en día. Entre sus principales usuarios se encuentra *Google*, *Wikipedia* y *Facebook*.

Figura 24: Logotipo de MySQL. Fuente: http://steve-parker.org/urandom/2009/dec/mysql.gif

II.12. WEB 2.0:

El termino WEB 2.0 se le llama a aquellas aplicaciones web que de manera interactiva promueven la interoperabilidad entre diferentes sistemas, comparten información y son centradas en la colaboración y el usuario. En una aplicación web 2.0, los usuarios son los propios creadores del contenido de la página web. Por contraste a aquellas páginas en las cuales los usuarios se les limita a ser observadores y lectores pasivos de la información que se les suministra.

Es importante resaltar que el término 2.0 no se refiere a una actualización de las especificaciones técnicas de ninguna herramienta utilizada para crear contenido web, sino más bien a un cambio en la manera en la cual tanto los desarrolladores de software como los usuarios finales utilizan la Web. Entre los ejemplos más conocidos de aplicaciones web 2.0 se encuentran los blogs (Blogspot), las redes sociales (Facebook, MySpace), los Wikis (Wikipedia) y las comunidades que comparten archivos (Youtube).

Otra de las funcionalidades de la web 2.0 es que le permite al usuario correr aplicaciones de software completamente a través de su navegador. En

este sentido comparte la carga entre el cliente y el servidor, empleando tecnologías tales como *AJAX*.

II.12.1. AJAX

Ajax (Asynchronous JavaScript y XML) es un grupo de técnicas web interrelacionadas usadas en el lado del cliente para crear aplicaciones web interactivas. Con Ajax, las aplicaciones web pueden buscar datos de un servidor de forma asíncrona sin interferir con el comportamiento y presentación de la página. El uso de técnicas Ajax ha llevado a un crecimiento en las interfaces dinámicas de las páginas web.

Ajax no es una tecnología como tal, sino un grupo de tecnologías. Ajax es una forma de combinar XHTML, CSS y JavaScript para dinámicamente mostrar y permitir al usuario interactuar con la información presentada. JavaScript provee un método de interacción con el servidor de forma asíncrona para intercambiar datos entre el servidor y el navegador, evitando recargas completas de la página sino simplemente segmentos del archivo XHTML.

Capitulo 3: Marco Metodológico

Debido a la naturaleza del proyecto, hemos determinado que se necesita aplicar una metodología flexible que permita la adaptación de la aplicación inclusive en sus últimas etapas del desarrollo. Adicionalmente se contó con la colaboración continua de diversos grupos universitarios que en última instancia serán los futuros usuarios de dicha aplicación. Es por ello que aplicamos una metodología de ingeniería de software que se denomina programación extrema.

III.1. Programación Extrema:

Es una metodología de la ingeniería de software que se caracteriza por mantener un esquema de programación ágil y flexible a los cambios. Considera que los cambios son un hecho inevitable del desarrollo de proyectos y por esto una de sus principales características es la adaptabilidad.

Para comprender esto un poco mejor, debemos definir la programación extrema como la metodología más ágil de desarrollo de software ya que se enfoca principalmente en ser sumamente adaptable, inclusive en las etapas finales del desarrollo. Como se va a trabajar en estrecha relación con las agrupaciones estudiantiles, es posible que sus requerimientos y necesidades varíen según la etapa en que se encuentre el proyecto. Por lo tanto, la flexibilidad ofrecida por la programación extrema nos proporciona una herramienta idónea para el desarrollo de la plataforma web 2.0.

La programación extrema lleva a cabo los mismos procesos que las metodologías clásicas, como lo son el levantamiento de información, el

diseño de la aplicación, las pruebas y la implementación del producto final. Sin embargo, las fases de la metodología clásica, son resueltas por la programación extrema mediante 4 actividades iterativas: planificación, diseño, codificación y pruebas. Estas 4 fases son repetidas cuantas veces sea necesario para cumplir con la meta pautada. Es decir, se descompone el proyecto, en pequeños ciclos ágiles, para poco a poco irá armando el proyecto en su totalidad. En forma resumida, entiéndase la programación extrema como una metodología que levanta un gran diseño original pero que dicho diseño va a ser sujeto a cambios continuos durante su ejecución a través de múltiples ciclos.

III.2. Planificación y levantamiento de la información:

La planificación y levantamiento de información consistió en entrevistar a miembros de diversas agrupaciones estudiantiles para identificar las necesidades comunicacionales de cada uno de ellos. De esta manera se elaboró una lista de requerimientos bastante extensa. A partir de la lista de posibles funcionalidades nos reunimos con las diversas agrupaciones para determinar cuáles de ellas se podrían calificar como de mayor importancia. De esta manera le asignamos un valor numérico a cada una de las funcionalidades y por medio de la votación determinamos cuáles eran las funciones más importantes que debe ofrecer el software.

Debido a que la funcionalidad era sumamente ambiciosa para el plazo de tiempo planteado del proyecto, se decidió reducir aún más las funcionalidades deseadas hasta lograr un consenso de qué funciones específicamente iban a ser desarrolladas.

Luego se fijó una reunión semanal con las agrupaciones con el propósito de que se pudiera mostrar los avances logrados y se pudiera retroalimentar el progreso del proyecto. Determinamos que era esencial la retroalimentación con las agrupaciones estudiantiles, para asegurarse de esta manera que los requerimientos técnicos se estuviesen cumpliendo y que estuviesen satisfechos con la manera de implementar la solución. De esta manera y durante el trascurso del proyecto, se consultó continuamente a las agrupaciones estudiantiles buscando que estas aprobasen y estén conformes con la solución coherente y eficiente de las necesidades planteadas.

III.2.1. Proyectosfp.org

Para lograr organizar mejor el proyecto así como para poder trabajar de manera colaborativa, se utilizó la herramienta web de la Fundación Futuro Presente, “Proyectosfp.org”. Esta herramienta permite descomponer un proyecto en tareas y a su vez en sub-tareas. De igual manera permite asignar metas a las personas involucradas, así como colgar archivos y compartirlos con los miembros del equipo. Fue una herramienta muy importante que logró aumentar la productividad y sincronía.

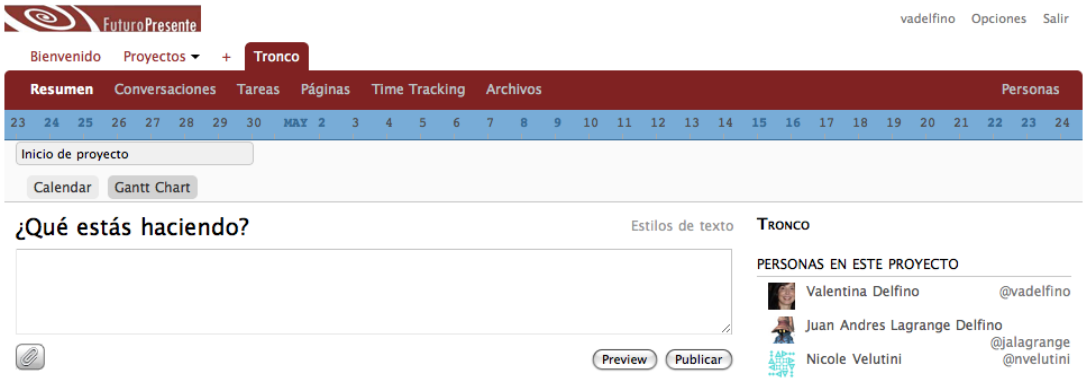

Figura 25: Vista de usuario de ProyectosFP.org. Fuente: http://www.proyectosfp.org, vista de usuario

Continuamos con las fases del marco metodológico y por tanto con la segunda fase de la programación extrema, el diseño.

III.3. Diseño:

El diseño según la planificación extrema se fundamenta en lo que modestamente se denomina “buen diseño” y se basa en poder lograr un diseño y, por lo tanto, un sistema final que tenga un mínimo de dependencias. Es decir, el diseño de la aplicación debe lograrse de una manera tal que cada componente sea lo más independiente posible de cada otro componente. Por lo tanto se diseña y se desarrolla en módulos.

Al desarrollar el sistema de manera modular, se logra efectivamente un desarrollo ágil de cada una de las funcionalidades y se les permite a los usuarios finales la oportunidad de poder interactuar con el sistema mientras éste aún se encuentra en etapa de desarrollo. De esta manera cada módulo puede ser continuamente criticado y mejorado hasta lograr un producto final satisfactorio.

En primer lugar, el diseño consistió en desarrollar el modelo entidad relación. Este modelo es la base de todo el proyecto. Si bien es cierto que el modelo entidad relación sufrió modificaciones durante el desarrollo, la programación extrema establece que esto es un proceso natural del desarrollo. Esto se debe principalmente a que al planificar idóneamente una aplicación no se toman todas las consideraciones y, por esto, en el transcurso del desarrollo se pueden presentar cambios para lograr mayor eficiencia y simplicidad de la misma.

Aparte del modelo entidad relación, en la fase inicial se diseñaron los casos de uso que va a emplear el sistema. De igual manera, los casos de uso iniciales y según la programación extrema, fueron dinámicos y susceptibles a cambios.

Luego se diseñaron las vistas que fuesen coherentes con los casos de uso y modelo entidad relación. Al igual que todo el resto de la aplicación, y según la retroalimentación recibida por las agrupaciones estudiantiles, estas fueron sufriendo cambios paulatinos durante el desarrollo del software. Específicamente en el diseño de la interfaz, se utilizaron modernas técnicas de CSS (Cascading Style Sheets) para lograr una continuidad coherente dentro de las vistas. Al utilizar CSS se hizo mucho más sencillo enmarcar el código XHTML y de esta manera separar la lógica de la aplicación de las vistas. Para el desarrollo de las CSS, se utilizo un Framework para CSS llamado “Blueprint”, este Framework permite estandarizar el diseño de las vistas dentro de un ambiente web.

De igual manera se utilizaron técnicas de AJAX que modernizaron las vistas y le permiten al usuario realizar un mayor número de funciones sin tener que estar continuamente recargando la página.

Es importante mencionar que el diseño de la plataforma se hizo según los principios MVC (Model, View, Controller) para mantener la lógica de la aplicación separada de la interfaz gráfica. A su vez se implementó la arquitectura REST (Representational State Transfer).

Por otra parte, se determinó que para simplificar la persistencia de los datos en la aplicación se utilizará el *ORM* (Object-relational Mapper) proporcionado por el *framework* “Ruby on Rails”. Adicionalmente, en esta

fase se determinaron los requerimientos del servidor para lograr una correcta implementación en cuanto a software y hardware.

Por último se diseñaron las pruebas que debían ser aplicadas para determinar que la aplicación cumpliese con las funcionalidades que se habían acordado con las agrupaciones estudiantiles.

III.3.1 MVC:

Es un patrón utilizado para el desarrollo de software que consiste en separar la lógica de la aplicación en tres componentes. El modelo, la vista y el controlador. De tal manera se logra mantener independencia entre la funcionalidad del diseño gráfico, la lógica de la aplicación y la lógica del acceso de datos, por lo que se logra desarrollar, actualizar y mantener cada uno por separado.

A continuación el diagrama de Modelo-Vista-Controlador:

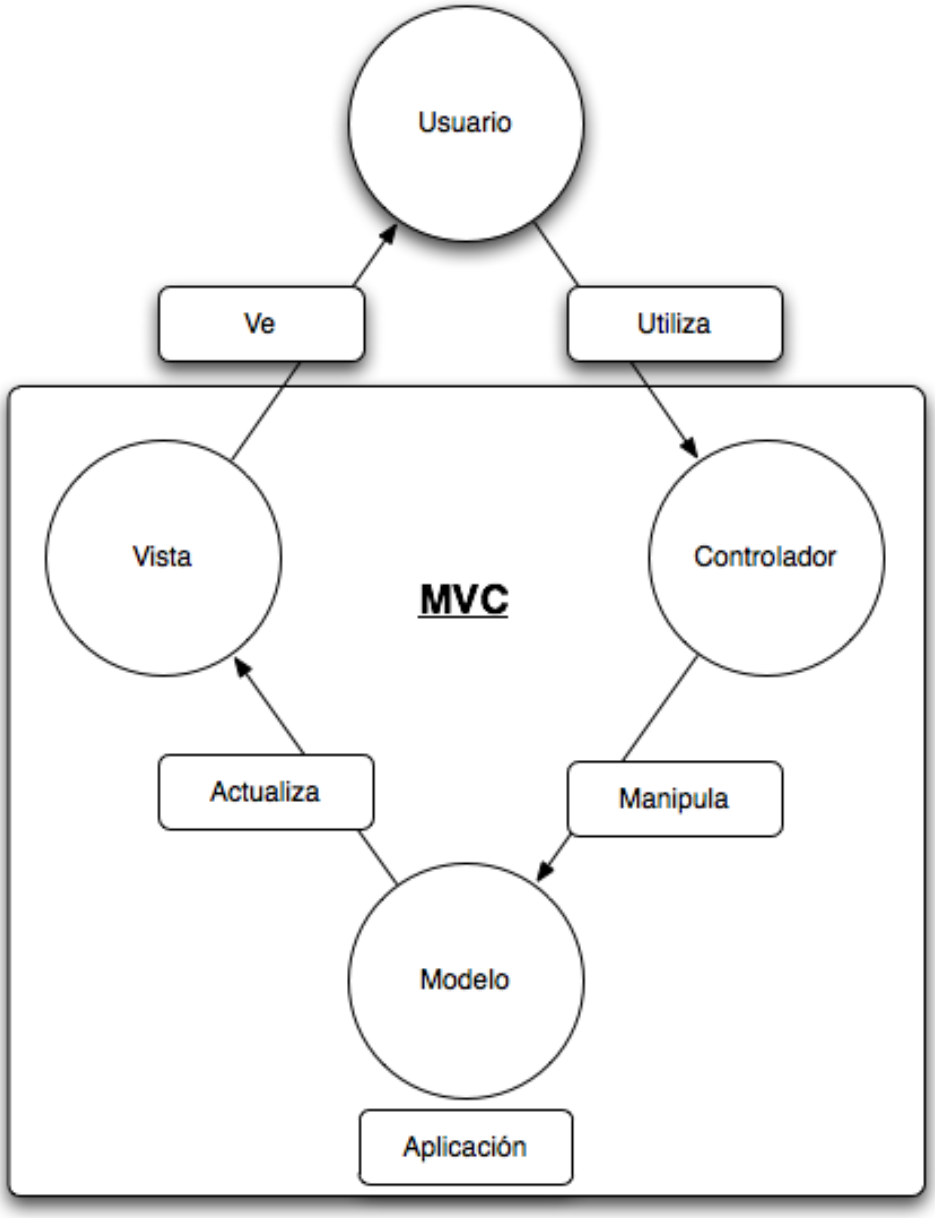

Figura 26: Diagrama de MVC. Fuente: Elaboració propia

El modelo es la representación de la data o información de un objeto con el cual la aplicación opera. Al cambiar el modelo, éste notifica a las respectivas vistas para que se puedan actualizar. Las vistas no son más que la interfaz gráfica que percibe el usuario. Por último el controlador funciona

como intermediario entre el modelo y las vistas. Es decir, el modelo hace llamados al controlador, quien ejecuta la lógica de la aplicación y hace llamados para que la vista respectiva muestre al usuario final la información del modelo que éste está requiriendo. Luego el usuario interactúa con la nueva vista y el ciclo se repite.

III.3.2. ORM:

Object-Relational mapping es una técnica para lograr la persistencia de los datos entre lenguajes de programación orientados a objetos y bases de datos relacionales.

La aplicación está desarrollada utilizando una base de datos *MySQL*, de manera que para trabajar con un lenguaje estructurado como es el *SQL*, se deben modificar las variables para que cumplan con el formato requerido por las tablas normalizadas de *MySQL*. El ORM le permite al programador obviar los formatos de variable que requieren las bases de datos. Es decir, el ORM logra diagramar los modelos de la aplicación y le permite al programador interactuar con la base de datos utilizando el lenguaje de la aplicación que desarrolla.

Por ejemplo, en Ruby on Rails, a través de ActiveRecord, se pueden escribir estas 4 líneas de código:

```
1  usuario = new Usario()
2  usuario.nombre = "Pepe"
3  usuario.apellido = "Perez"
4  usuario.save
```

Figura 27: Ejemplo de creación de un usuario utilizando ORM de Ruby on Rails. Fuente: Elaboración propia

Con ellas se añade un registro a la tabla usuario con 2 campos; “nombre” y “apellido” con valores “Pepe” y “Perez”.

```
INSERT INTO 'usuarios' (nombre, apellido) VALUES ('Pepe','Perez');
```

Figura 28: Query de SQL equivalente a realizar el código de la figura 27. Fuente Elaboración propia

El ORM de Rails llamado Active Record, se encargó de traducir las 4 líneas de código en los el *query* de SQL. Es decir en ningún momento el programador se preocupa en el desarrollo de los queries.

Actualmente la utilización de los ORM ha sido fundamental para mantener la persistencia de los datos y disminuir los tiempos y costos de desarrollo de software.

III.3.2.1. Active Record

Se define como el ORM que utiliza Ruby on Rails. Una de las ventajas que tiene ActiveRecord es que además de manejar los modelos a través de sus distintos campos (mapeando cada campo de la tabla a la que se le está aplicando el patrón ORM a un atributo en la clase modelo que le corresponde a dicha tabla), se caracteriza por tener un complejo sistema de manejo de las relaciones entre modelos. Para ilustrar esto mejor, se deben comprender las reglas que rigen a este ORM.

En general, una fila de la tabla de la base de datos, se refiere a una instancia de un modelo específico. Por tanto, al cargar un objeto éste lo hace de esa fila específica de la base de datos. De igual manera, al actualizar un objeto, la fila correspondiente en la base de datos, será actualizada.

El manejo de relaciones se lleva a través de claves foráneas. De tal manera que Active Record reconoce las relaciones existentes y permite hacer modificaciones a objetos relacionados, así como permite crear nuevas relaciones y por tanto nuevas claves foráneas. Inclusive para relaciones N-N, Active Record utiliza tablas en paralelo que simulan estas relaciones y por tanto se asegura la persistencia continua de los datos.

III.4. Codificación

Esta es la tercera fase cíclica de la programación extrema y es durante la codificación que se procedió a trasladar el diseño a la implementación de la lógica de desenvolvimiento del sistema de manera modular. Es decir, se desarrollaron los módulos de seguridad, usuarios, actividades, carga y descarga, interfaz, entre otros.

Es importante resaltar que antes de comenzar a programar, se desarrollaron las pruebas modulares que debían de ser aplicadas en cada etapa del desarrollo para así determinar que la aplicación cumpliera con las funcionalidades para las cuales fue diseñada. Una vez desarrolladas las pruebas, se crea un ambiente en el cual el principal objetivo es desarrollar módulos que cumplan con dichas pruebas. Por último se desarrollaron las pruebas de integración que avalan la integridad del sistema.

Durante la codificación, se programaron todos los casos de uso así como la respectiva base de datos. Entre los beneficios que ofrece la programación extrema en la fase de codificación, se encuentra la cercanía con el cliente, en este caso las agrupaciones estudiantiles, quienes revisaron en repetidas ocasiones la aplicación desarrollada para notificar cambios

según se presentaron nuevas oportunidades para emplear funcionalidades más útiles y sencillas para el usuario.

Para trabajar de manera más eficiente, se contó con herramientas como *Textmate*, *Navicat*, *Notepad*++ y *Firebug*. De igual manera debemos mencionar que para trabajar en sincronía y controlar las versiones utilizamos el sistema Git junto con su servicio web de almacenamiento de proyectos, Github.

La manera en que se efectúo la codificación es aquella propuesta por Ruby on Rails. Una de las virtudes de trabajar con Ruby on Rails es que prácticamente garantiza que se esté utilizando la gran mayoría de los factores de diseño previamente mencionados. Por principio, utiliza MVC, REST y contiene su propio ORM. De manera que la elección de este sistema de programar agiliza sustancialmente el proceso de desarrollo.

III.4.1. REST

REST significa “Representational State Transfer” y es una arquitectura de software para sistemas de *hypermedia*, tales como la web.

La *hypermedia* se refiere al medio no lineal de información comúnmente encontrado en las páginas web de hoy en día. Comprende la utilización de imágenes, audio, video e hipervínculos para crear páginas interactivas con el usuario.

La arquitectura REST consiste en utilizar clientes y servidores. El cliente, inicia pedidos o *requests* al servidor. El servidor a su vez procesa los llamados y retorna una respuesta acorde al pedido del cliente. Estos pedidos

y respuestas se construyen a partir de la representación de estados de un recurso. En el caso específico de Ruby on Rails, los recursos son básicamente aquello que se tiene almacenado en base de datos, los cuales serán contemplados por el usuario a través de la representación de un objeto.

Adicionalmente, la arquitectura de servicio web REST propone que las operaciones se limiten a utilizar 4 de los 8 métodos definidos HTTP: *Get, Put*, *Post*, *Delete*. A partir de estos 4 métodos, se crean una serie de operaciones con las cuales se puede desarrollar cualquier sistema.

III.4.2. Archivos utilizados

El sistema de programación que propone Ruby on Rails implica la integración de varios lenguajes. Entre ellos JavaScript, CSS, Ruby y XHTML. Es por ello que en la codificación se utilizan varios tipos de archivo:

Archivos .JS: Son los archivos de JavaScript y se utilizan tanto para modificar algún aspecto del HTML como para enviar *requests* del lado del cliente al servidor y recibir las respuestas sin tener que recargar la página.

CSS: Son los archivos que le dan estilo y diseño a la página web. Es decir, en los archivos CSS se definen los colores, las dimensiones y el lugar físico en donde se van a ubicar los elementos de la interfaz grafica.

ERB: Son archivos que contienen texto simple junto con código de Ruby. El servidor los interpreta y a partir de su interpretación, genera el código XHTML que es enviado al navegador del cliente.

Dentro del esquema MVC, estos 3 tipos de archivos están encargados exclusivamente de las vistas.

RB: Son archivos de código Ruby. En el esquema MVC, son estos los archivos es donde se definen los modelos y controladores. Es decir, en estos archivos se define toda la lógica del sistema.

III.5 Pruebas

Siguiendo las especificaciones de la programación extrema, se realizaron pruebas constantemente, muchas veces incluso antes de la propia realización de la funcionalidad a implementar. Asimismo, dado que es una plataforma web, se deberán efectuar también las debidas pruebas de seguridad digital. Adicionalmente cabe resaltar que la programación extrema hace especial énfasis en las pruebas de aceptación, es decir en que los requerimientos de los usuarios finales hayan sido cubiertos de acuerdo con la funcionalidad de la plataforma. Es importante mencionar que se harán todas las pruebas modulares y de integración del sistema. Se comprobará que todos los módulos funcionen por sí solos y a su vez los módulos trabajen entre sí.

Por último se deben hacer las pruebas según la plataforma utilizada. Dado que es un desarrollo web, los distintos navegadores utilizan diferentes *layout engines* para generar las vistas del usuario final. El *layout engine* es el modulo del navegador encargado de determinar la posición final, colores, letras, tamaños y diseño de los distintos componentes con respecto a la interfaz grafica. Al realizar estas pruebas lo que se desea es lograr la máxima compatibilidad entre los distintos navegadores.

Capítulo IV: Desarrollo y Resultados

IV.1. Levantamiento de información

Luego de sostener diversas reuniones con miembros de distintos grupos estudiantiles logramos recopilar cuáles eran sus necesidades en cuanto a las comunicaciones se refiere. Se determinó que existen necesidades comunicacionales internas, inter-organizativas y externas.

IV.1.1. Necesidades de comunicación interna:

Encontramos que existen graves fallas comunicacionales internas de cada organización. En la mayoría de las organizaciones utilizan el correo electrónico para mantener referencia de las minutas y acuerdos logrados. De hecho utilizan el correo electrónico para mantener registro de prácticamente todo lo que quieren mantener registro. El problema con esto es que no es centralizado para todos. Es decir, si por equivocación se le olvida enviar el correo a uno de los integrantes del grupo estudiantil, este integrante no tendrá la posibilidad de visualizar cuáles fueron los acuerdos logrados o las pautas convenidas.

En este sentido se determinó que una de las necesidades que tienen las agrupaciones estudiantiles en cuanto a comunicación interna es la centralización de mensajes entre integrantes. Es decir, necesitan que en el sistema se publiquen las comunicaciones entre unos y otros miembros del grupo.

Otra de las necesidades que presentaron es que debido a no contar con un lugar centralizado de información, la información de contacto depende de la relación personal. Es decir el hecho de que un miembro conozca la información actualizada de contacto de otro miembro del grupo, depende de si éste se la facilito personalmente. En este sentido necesitan un lugar en donde se pueda consultar la información de contacto de alguno de sus integrantes. De tal manera que si la información de contacto llegase a cambiar, todos los miembros tengan un lugar accesible en donde puedan consultar esos datos.

Por último es importante resaltar que todas las agrupaciones insistieron de manera enfática que esta información es bastante sensible y por tanto el sistema debe proveerlos de un sólido sistema de roles, así como de un buen nivel de seguridad informática.

IV.1.2. Necesidades Inter-organizativas:

Es sorprendente darse cuenta de que con frecuencia existen diferentes agrupaciones estudiantiles que hacen actividades muy similares y que sin embargo desconocen de la existencia las unas de las otras. En otros casos se encontró que las agrupaciones estaban conscientes de que existían otras agrupaciones con intereses similares, sin embargo no contaban con una manera de ser contactadas.

Es por ello que como una necesidad prioritaria se determinó que la aplicación debe proveer a las agrupaciones estudiantiles una vía por la cual puedan ser contactados de manera sencilla.

De manera similar, se determinó que por la misma falta de comunicación entre unas y otras organizaciones, éstas pierden la posibilidad de realizar sinergias que fortalezcan las actividades que cada una emprende. Es por ello que se decidió que otra de las necesidades detectadas es la capacidad de coordinar proyectos y actividades en conjunto.

IV.1.3. Necesidades Externas

A través de las distintas conversaciones sostenidas, se evidenció una creciente preocupación por lograr comunicar el mensaje de cada una de estas agrupaciones estudiantiles a la población objetivo de las mismas. En algunos casos nos dimos cuenta de que algunas agrupaciones estarían satisfechas con el simple hecho de lograr tener una presencia web por la cual pudiesen publicar su información de contacto. En otros casos, tenían necesidades más ambiciosas tales como campañas publicitarias a través de medios electrónicos. En concreto, todas las agrupaciones estudiantiles deseaban poder comunicar sus logros, convocatorias, datos de contacto y objetivos en un portal web.

Por ello se determinó que el sistema a desarrollar debe proveer a las agrupaciones estudiantiles herramientas con las cuales puedan convocar y comunicar sus propuestas a la comunidad en general.

Por último se determinó que las agrupaciones también requerían de una manera ordenada y documentada de recibir *feedback*. En este sentido se decidió que la aplicación debe proveer un sistema para que las agrupaciones reciban comentarios sobre sus actividades y sobre ellas mismas en su página de contacto.

IV.2. Diseño y Planificación

Basado en las conversaciones sostenidas y las necesidades levantadas, se descompusieron estas necesidades en módulos y éstos a su vez en funcionalidades. A partir de los módulos y las funcionalidades, se desarrollaron los respectivos casos de uso así como el diagrama entidad relación que, si bien es cierto no es definitivo en términos de la metodología desarrollada, se utilizó como guía para el desarrollo.

IV.2.1. Módulos a desarrollar

Como producto de las necesidades que plantearon las agrupaciones estudiantiles se decidió desarrollar los siguientes módulos:

IV.2.1.1. Módulo de Seguridad

Tomando en cuenta la alta complejidad que se observó en las necesidades de seguridad que requieren los grupos estudiantiles, se contempló un módulo de seguridad que agrupe a los usuarios en distintos tipos de roles y que permita y deniegue distintas acciones según sean los privilegios que se le asignen a estos roles.

En forma concreta se requiere que el modulo permita:

- Crear Usuarios
- Editar Usuarios
- Listar Usuarios
- Ver Detalle Específico de Usuarios

- Resetear Clave de usuarios y envío de email con clave nueva provisional
- Asignar Usuarios a Roles
- Desasignar Usuarios de Roles
- Eliminar Usuarios.
- Establecer privilegios para roles.

IV.2.1.2. Módulo de Universidades

El módulo de las universidades es el que se encarga de todo lo referido a las universidades. Es decir, debe realizar las funcionalidades básicas de las universidades, tales como: crear, editar, actualizar y eliminar. Pero a su vez debe permitir relacionarse con los demás módulos como: actividades, estudiantes y agrupaciones estudiantiles. La idea es que en la interfaz del usuario sea sencillo conseguir los estudiantes o los grupos estudiantiles de una casa de estudio específica. A su vez debe relacionarse con el módulo de roles y usuarios debido a que no debe permitírsele a todos los usuarios del sistema privilegios sobre todas las funciones de las universidades. En concreto, las funciones que debe contener este módulo son:

- Crear una universidad
- Editar y actualizar una universidad
- Añadir un grupo estudiantil a la universidad
- Añadir un estudiante a la universidad
- Listar las universidades
- Ver los detalles específicos de una universidad
- Ver los grupos estudiantiles asociados a la universidad
- Ver los estudiantes pertenecientes a la universidad
- Eliminar una universidad

IV.2.1.3. Módulo de estudiantes

El modulo de estudiantes es el que se encarga de las funciones relacionadas con el modelo de estudiante. Debe contener las funcionalidades básicas tales como: crear, editar, actualizar y eliminar. Los estudiantes a su vez deben pertenecer a una universidad y según el caso a una agrupación estudiantil. En concreto las funcionalidades de este módulo son:

- Crear un estudiante
- Editar y actualizar un estudiante
- Ver los detalles de un estudiante
- Listar los estudiantes
- Listar todos los estudiantes del sistema
- Listar los estudiantes de una universidad
- Eliminar un estudiante

IV.2.1.4. Módulo de Grupos estudiantiles

Este módulo está encargado de lo referido a las agrupaciones estudiantiles. Por definición, una agrupación estudiantil debe contener estudiantes. A su vez una agrupación estudiantil debe tener la capacidad de realizar y convocar actividades. Por lo tanto aparte de las funcionalidades básicas tales como: crear, editar, actualizar y eliminar, es de suma importancia que las agrupaciones estudiantiles permitan la afiliación de estudiantes y la creación de actividades. Es importante recordar que al igual que los demás módulos, las funciones de las agrupaciones estudiantiles estarán limitadas por el rol y el usuario que se encuentre utilizando el sistema. Tomando esto en cuenta, las funciones para este módulo son las siguientes:

- Crear una agrupación estudiantil
- Editar y actualizar una agrupación estudiantil
- Ver los detalles de una agrupación estudiantil
- Listar las agrupaciones estudiantiles
- Listar las agrupaciones del sistema
- Listar las agrupaciones estudiantiles por universidad
- Eliminar una agrupación estudiantil

IV.2.1.5. Módulo de Actividades

Tomando en cuenta las diversas necesidades comunicacionales externas que las agrupaciones del movimiento estudiantil presentan, se decidió desarrollar el módulo de Actividades.

Este módulo tiene la peculiaridad de que, junto con el módulo de universidades, les permite a usuarios externos (no registrados) interactuar con el sistema. Cualquier persona con conexión a internet podrá acceder a la página de lista de actividades, ver alguna en específica, comentar, ver los comentarios y visualizar los datos de contacto de las agrupaciones estudiantiles.

También es importante destacar que como se pretende que las Actividades sean el factor atractivo para que personas externas a las agrupaciones ingresen a la página, este módulo debe contar con una interfaz de creación de actividades que facilite la edición sencilla y atractiva de elementos web. Es decir, que al momento de publicar una actividad los usuarios del sistema puedan modificar el tipo, tamaño, formato y color de letra así como las imágenes, videos, sonidos, banners, etc. que desean ver publicados en su actividad.

Para lograr esta funcionalidad, se requería de un WYSIWYG editor (*What you see is what you get*), que es simplemente una herramienta que le permite a cualquier persona editar el contenido de una página web sin necesidad de tener conocimientos de lenguaje XHTML.
Editores WYSIWYG existen varios, y el que se decidió utilizar para este sistema fue TinyMCE por su facilidad de integración con aplicaciones en Ruby on Rails

En forma concisa, se requiere que el módulo contemple:

- Crear Actividades
- Editar Actividades
- Listar Actividades
- Eliminar Actividades
- Ver Actividad
- Agregar Comentarios

IV.2.1.6. Módulo de Carga y Descarga de Archivos

Tomando en cuenta que una de las necesidades del módulo de Actividades es que estas puedan ser presentadas con fotos, sonidos y/videos, se necesitará de un módulo de carga y descarga de archivos que proporcione estas funcionalidades.

Básicamente, se requiere que el módulo contemple:

- Cargar Multimedia(fotos, sonidos, videos, pdfs, archivos Office)
- Descargar Multimedia
- Eliminar Multimedia

IV.2.1.7. Módulo de Mensajería

Para poder cumplir con las necesidades de comunicación interna, se decidió implementar un modulo de mensajería entre usuarios del sistema. Es decir, este módulo debe encargarse del envío y recepción de mensajes entre los distintos usuarios del sistema con el propósito de establecer una comunicación centralizada entre miembros de una agrupación estudiantil. De la misma manera, miembros de distintas agrupaciones estudiantiles podrán enviar mensajes entre ellos con el propósito de lograr crear un canal para la comunicación inter-organizativa, sin comprometer la confidencialidad de la información de contacto de sus integrantes. De nuevo es importante resaltar la importancia que tiene el módulo de seguridad y roles dentro del éxito de implementación del modulo de mensajería. En concreto, las funciones que tiene este módulo son las siguientes:

- Enviar mensaje
- Ver mensaje
- Ver mensajes enviados
- Ver mensajes recibidos
- Eliminar un mensaje

IV.2.2. Modelado de funciones en Casos de Uso

Para facilitar el desarrollo de manera modular se descompusieron las funciones deseadas en casos de uso. Se debe señalar que los casos de uso no se utilizaron de manera rígida si no por el contrario como una herramienta flexible que actuara de soporte al desarrollo. Es decir se utilizaron los casos de uso para discutir y comprender las funcionalidades, actores, roles y lógica del sistema en general.

IV.2.2.1. Actores

Existen 4 actores que van a interactuar con el sistema, los cuales fueron definidos según su importancia en el esquema jerárquico:

- Administrador: El administrador es la persona encargada del sistema. Éste va a contar con todos los privilegios para realizar todas las funciones que el sistema le brinda, sin excepción. Son los únicos que pueden crear usuarios Master y son los únicos que pueden crear y manipular la información y el perfil de las universidades.

- Master: El master debe ser un usuario creado por el administrador. El perfil del master es el del representante de una agrupación estudiantil. Es decir, al máximo representante de una agrupación estudiantil, se le genera un usuario con el cual podrá personalizar el perfil de su grupo estudiantil, crear y editar actividades e intercambiar mensajes con otros miembros de agrupaciones estudiantiles. El privilegio preponderante que tiene sobre los Junior es que puede crear usuarios Junior. Es importante resaltar que todas las funciones que realice estarán vinculadas y limitadas a su grupo estudiantil.

- Junior: El usuario tipo Junior tiene el perfil de pertenecer al equipo del usuario Master. Es decir, típicamente éste será un miembro de la agrupación estudiantil en el cual el usuario principal ó Master puede delegar funciones. Este usuario podrá realizar prácticamente las mismas funciones del usuario Master, con la excepción de que no puede crear otros usuarios.

- Guest: Guest es el usuario que no tiene usuario. Es decir, es cualquier persona que decida visitar el portal web para conseguir información de una actividad, universidad o grupo estudiantil específico. A este usuario se le limitan sus funciones a visualizar la información pública tal como las fechas y convocatorias a actividades, la información de contacto de las agrupaciones estudiantiles y universidades y su única interacción es a través de los comentarios.

A continuación, se muestra un diagrama de los casos de uso para los actores principales del sistema. Es importante recordar que el Administrador tiene acceso a todas las funciones, pero por motivos de simplificación del diagrama, no se muestra.

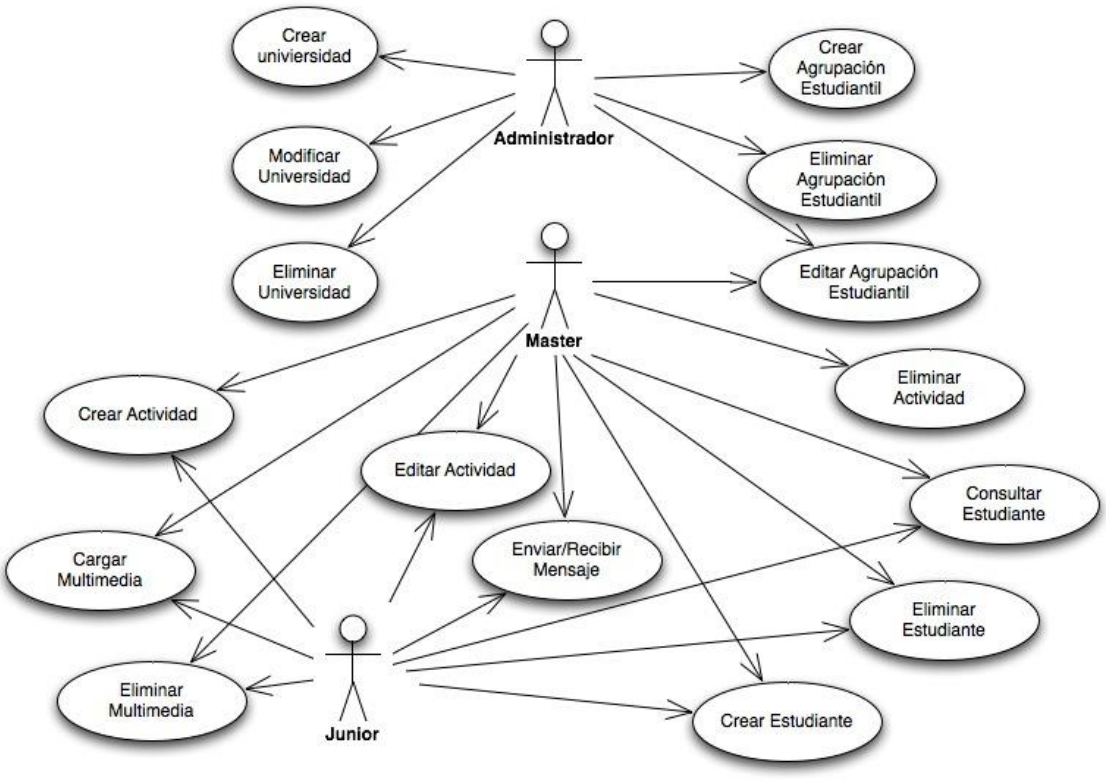

Figura 29: Diagrama de casos de uso para usuarios tipo Administrador, Master y Junior. Fuente: Elaboración propia

La siguiente figura muestra un segundo diagrama de caso de uso utilizado solamente para el actor *Guest* esto se debe a que éste tiene funcionalidades limitadas y para simplificar el diagrama anterior, decidió hacerse en 2 figuras.

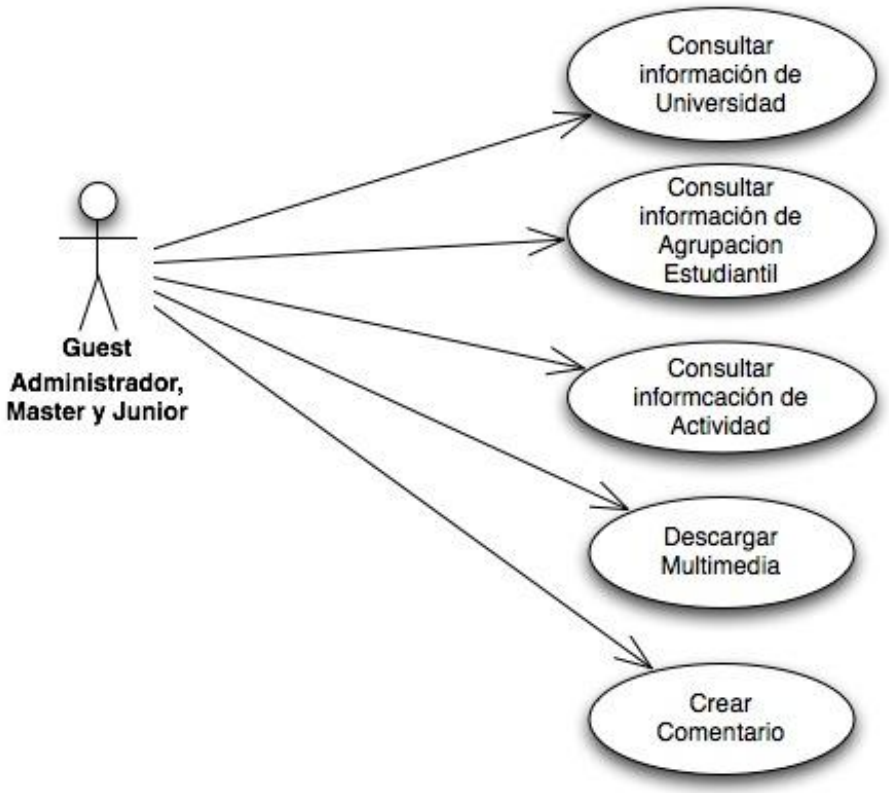

Figura 30: Diagrama de casos de uso generales y comunes a todos los tipos de usuario. Fuente: Elaboración propia

IV.2.3. Casos de Uso

El modelado de las funciones anteriormente citadas fue llevado a casos de uso como el siguiente:

Nombre	Consultar Usuario
Actor	Administrador, Master, Junior
Descripción	Permite consultar los datos de un usuario del sistema
Precondición	El usuario debe existir en el sistema
Flujo normal	1. Se selecciona el usuario a consultar 2. Se accede a los datos del usuario
Comentarios	Según el rol del usuario haciendo la consulta, este podrá acceder a más o menos información.
Poscondición	N/A

Figura 31: Caso de uso "Consultar Usuario". Fuente: Elaboración Propia

Los demás casos de uso están ubicados en el Apéndice A.

IV.2.4. Planificación de tareas

Para trabajar de manera organizada y planificar las tareas, utilizamos el sistema web de la organización Futuro Presente. Este nos permitió dividir el proyecto en tareas pequeñas, así como mantener enviar y recibir las actualizaciones sobre los archivos con los que estábamos trabajando.

IV. 2.5. Diagrama Entidad relación

A continuación se presenta el esquema inicial con el cual se desarrollo el sistema.

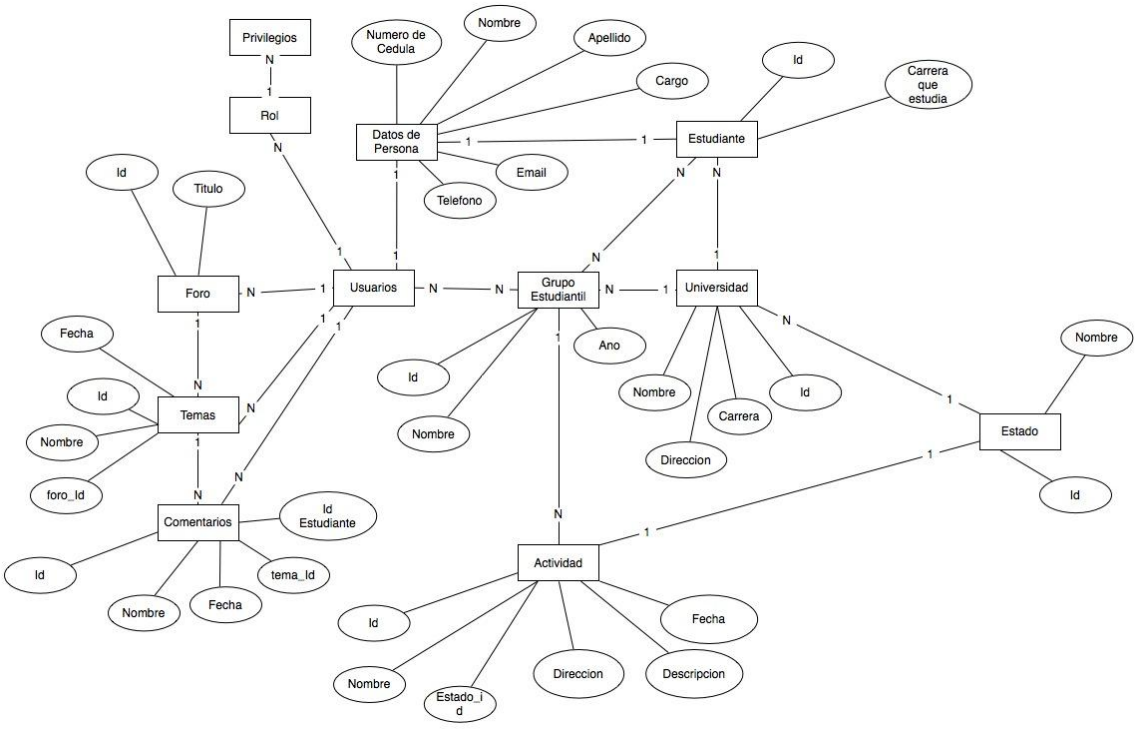

Figura 32: Diagrama Entidad Relación. Fuente: Elaboración Propia

IV.3. Desarrollo de la aplicación

Como se ha mencionado anteriormente, el trabajar con el framework de Ruby on Rails simplifica la fase de codificación. Entre sus principales beneficios es que proporciona la estructura para trabajar continuamente bajo la metodología MVC. Un ejemplo clásico de las facilidades que proporciona ruby on rails es el comando “scaffold”.

El comando scaffold es un comando que con solo proporcionarle el nombre y los atributos de un modelo, crea todos los archivos necesarios para que el modelo funcione de manera básica dentro de la aplicación. Es decir, al emplear este comando a un recurso, como puede ser “estudiante”,

éste tendrá implementadas las funciones básicas: Crear, Editar, Mostrar y Eliminar (CRUD).

A su vez, crea los llamados archivos de migración. Estos archivos proporcionan una manera agnóstica de interactuar con la base de datos. Es decir, si se mantienen los estándares, la aplicación no dependerá de un servidor de base de datos especifico, sea este MySQL, PostgreSQL, Oracle, etc.

Ruby on Rails logra esto debido a que en su buena práctica, dicta que cualquier modificación que se desea hacer a la base de datos, debe hacerse mediante un archivo de migración. Luego en la configuración de la aplicación, se especifica qué manejador se utilizara para la base de datos y Ruby on Rails se encarga de traducir los archivos de migración para que se pueda utilizar el manejador deseado. Esto lo logra a través de otro de sus más poderos comandos que es el *rake db:migrate*, el cual ejecuta secuencialmente las migraciones de bases de datos que aún no hayan sido ejecutadas.

Adicionalmente, se debe mencionar que Ruby on Rails cuenta con los llamados *gems* que proporcionan librerías de funciones que ayudan a desarrollar elementos específicos de una aplicación.

A través de la utilización de todas las bondades que nos brinda Ruby on Rails, se procedió a desarrollar la aplicación.

IV.3.2.Herramientas de Desarrollo

Para el desarrollo de la aplicación se contó con múltiples herramientas que facilitaban y automatizaban distintas tareas relevantes al desarrollo.

IV.3.2.1. Textmate:

Este editor de texto para Mac OS fue el que se utilizó para realizar la mayor parte de la programación porque para las aplicaciones en Ruby on Rails viene incorporado con chequeo de sintaxis a medida que se edita. Además de esto tiene un manejador simple de archivos (lo cual es de suma importancia para proyectos MVC) y una interfaz de pestañas para tener más de un archivo abierto simultáneamente.

IV.3.2.2. Notepad++:

Como la aplicación fue desarrollada por dos programadores, uno con Mac y el otro con Windows, también se utilizó el poderoso editor de texto Notepad++ para Windows. Así como Textmate, Notepad++ tiene funcionalidades de manejo de archivos, chequeo de sintaxis, manejo de carga y descargas de archivos a servidores FTP, etc.

IV.3.2.3. Navicat:

Para realizar monitoreos de base de datos MySQL, se utilizó la aplicación Navicat. Es importante destacar que a través de esta aplicación no se realizaron ningunos cambios en las estructuras de las tablas ya que a

través de ruby migrations se mantuvieron organizadas y documentadas cada modificación de base de datos.

El uso de Navicat se vió limitado al ingreso de data cruda cuando se quería probar alguna funcionalidad y en la fase de pruebas para verificar que los datos estuviesen siendo almacenados en base de datos cuando correspondiese.

IV.3.2.4. Firebug:

Como se le dió gran importancia al diseño de la interfaz gráfica de la aplicación así como a que el lenguaje de marcado sea limpio y válido, se utilizó un plugin del navegador Firefox llamado Firebug.

Este plugin permitió chequear y modificar el código XHTML en vivo con el fin de asegurarse de que el servidor emitiese el XHTML que se esperaba.

Por otra parte, para las interacciones AJAX, este plugin, a través de su controlador por Consola de los *requests,* fue de gran utilidad al momento de implementar este tipo de interacciones.

Por último, Firebug fue utilizado para *debuggear* el código javascript ya que provee una interfaz de *breakpoints* que ayuda a identificar errores para la lógica en el lado del cliente.

IV.3.3. Desarrollo de los modelos

Se basó en los casos de uso y en el modelo entidad relación para iniciar la fase de desarrollo de los modelos. Una vez que se tenía planificado cómo se iban a hacer, se empezó a utilizar las herramientas que Ruby on Rails provee y en este sentido, el primer paso fue correr el comando *scaffold* para cada uno de los modelos deseados.

Al utilizar este comando, Ruby on Rails crea un archivo de modelo por defecto bastante sencillo. Básicamente es la definición de una nueva clase:

```
class GrupoEstudiantil < ActiveRecord::Base

end
```

Figura 33: Declaración de un modelo en Ruby on Rails. Fuente: Elaboración Propia

Este archivo luego va a ser configurado según las relaciones que pueda tener con otros modelos para de esta manera ir programando la lógica del sistema. De igual manera, en este archivo deben especificarse las reglas de validación que mantendrán la integridad de la aplicación, como ejemplo se encuentra el grupo estudiantil:

```
class GrupoEstudiantil < ActiveRecord::Base
  belongs_to :universidad
  has_many :actividades
  has_many :representantes
  has_many :users, :through => :representantes

  validates_presence_of :usario, :nombre, :universidad

end
```

Figura 34: Definición de un modelo con sus respectivas validaciones y relaciones en Ruby on Rails. Fuente: Elaboración Propia

En esta figura podemos observar como el modelo “grupo estudiantil” tiene diversas reglas de relación tales como: “pertenece a una universidad”, puede “tener varias actividades y representantes” así como diversas validaciones tales como la exigencia de la presencia de un usuario, de un nombre y de una universidad al momento de su creación o modificación.

Es importante tomar en consideración que los atributos de cada modelo no están definidos en su archivo de configuración ya que como se mencionó antes, Ruby on Rails cuenta con un ORM que le permite tomar los atributos directamente de la base de datos. En este sentido, cada modelo tiene una tabla específica en la base de datos y cada una de las columnas de esta tabla corresponde a cada uno de los atributos del modelo en cuestión. De tal manera que si se mantienen las convenciones que dicta Ruby on Rails, en ningún momento será necesario interactuar directamente con la base de datos para modificar un modelo u atributo y por el contrario se le delegan estas funciones a el ORM.

IV.3.4. Desarrollo de los Controladores

Los controladores de Ruby on Rails se desarrollan de una manera muy similar a sus modelos. En este sentido, una vez que se deciden cuáles son los modelos y las funciones a implementar, al correr la función de scaffold, ésta, aparte de generar los archivos correspondientes al modelo, automáticamente genera los controladores y a su vez las vistas para este modelo.

Por defecto, al generar el controlador de un modelo, este viene con 7 funciones que con el desarrollo serán modificadas:

- Index: Se encarga de buscar todos los objetos de un modelo específico en la base de datos y se los entrega a la vista. Por ejemplo: Buscar todos los estudiantes.

- Show: Se encarga de buscar un objeto específico y de darle todos sus atributos a la vista. Por ejemplo: Ver los detalles de Pepito Pérez, en este caso el controlador le entregaría a la vista, todos sus atributos, tales como la edad, sexo, nombre y apellidos de Pepito.
- New: Se encarga de crear una instancia vacía de un modelo y entregársela a la vista para ser creada (sin guardar en la base de datos).
- Edit: Es similar a Show, pero con la diferencia de que una vez conseguido el objeto, éste se lo entrega a la vista para ser editado y no a la vista para ser mostrado (Sin guardar las modificaciones en la base de datos).
- Create: Recibe un objeto y se encarga de guardarlo en la base de datos.
- Update: Recibe un objeto modificado y se encarga de actualizarlo en la base de datos.
- Destroy: Elimina un objeto especifico de la base de datos.

A continuación se muestra una figura con un ejemplo de 2 de los métodos que se crean por defecto, *Index* y *Show*. En ella se puede observar como en el método *index*, se buscan todas las universidades, mientras que en el método *show*, se busca solo la universidad que contenga un *id* especifico. También se puede observar como este controlador luego hace un llamado a su respectiva vista.

```ruby
def index
  @universidades = Universidad.all

  respond_to do |format|
    format.html # index.html.erb
    format.xml  { render :xml => @universidades }
  end
end

# GET /universidades/1
# GET /universidades/1.xml
def show
  @universidad = Universidad.find(params[:id])

  respond_to do |format|
    format.html # show.html.erb
    format.xml  { render :xml => @universidad }
  end
end
```

Figura 35: Método "index" y "show" del controlador de "Universidades". Fuente: Elaboración propia

Estos controladores generados por defecto son luego modificados para implementar la lógica específica de la aplicación en desarrollo.

IV.3.5. Desarrollo de las Vistas

Como ya se ha mencionado, el desarrollo de las vistas se asemeja en gran medida al desarrollo de los modelos y controladores en el sentido de que Ruby on Rails provee de una sólida base con la cual comenzar.

Al igual que en los modelos y controladores, las vistas también son autogeneradas por el comando *scaffold*. Es decir una vez más Ruby on Rails

provee una vista sencilla para cada modelo y para cada método de estos modelos.

Por defecto genera 4 vistas:

- New: Muestra un formulario con el cual crear el nuevo objeto
- Show: Muestra el detalle del objeto creado
- Edit: Muestra un formulario con el cual editar un objeto previamente creado.
- Index: Muestra una lista de todas las instancias de un modelo en especifico.

Por ejemplo la vista por defecto para crear una universidad es la siguiente:

Figura 36: Vista básica del método "Crear Universidad". Fuente: Elaboración propia

Es importante tomar en cuenta que esta vista es creada completamente por defecto a partir del comando de scaffold.

Luego de tener las vistas básicas comienza el proceso de diseño para lograr un portal web más atractivo y fácil de utilizar. Esto se logra a través de la utilización de los frameworks de Blueprint para CSS y de jQuery UI.

En la figura 37 se puede apreciar el diseño final una vez aplicado todos los elementos de CSS, blueprint, jQuery UI, etc.

Figura 37: Vista de la aplicación sobre una actividad. Fuente: Elaboración propia

IV.3.6. Desarrollo del módulo de seguridad y usuarios.

Éste fue el primer módulo que se decidió planear e implementar debido a que todas las funcionalidades de la aplicación estarían totalmente condicionadas por una buena gestión de seguridad y por consiguiente son dependientes de este módulo.

En primera instancia es importante destacar que en el módulo de seguridad y usuarios hay dos componentes independientes para la aplicación.

Uno es la autenticación, en el cual se definen todas las funciones y condiciones de registro, login, logout, y cambios de clave. Es decir, autenticación se encarga de verificar si el usuario existe en el sistema a través de login y clave.

Para Ruby on Rails existen distintas *gems* que facilitan la implementación de un componente de autenticación. Una de estas es *Authlogic,* la cual proporciona todas estas funcionalidades. Por esto, se procedió a instalar este gem y a crear los archivos de migración de base de datos necesarios.

Para este módulo no se necesitó utilizar el comando scaffold (para el recurso usuarios) ya que *Authlogic* proporciona las funcionalidades CRUD para los usuarios.

Por otra parte, se necesita de un segundo componente de autorización que decida si algún usuario específico se le permite o se le deniega realizar alguna acción en el sistema. Para ello, *Declarative Authorization* es una de las *gems* que proporciona configuración de seguridad de la forma más organizada.

Básicamente consiste en deshabilitar todos los permisos de todos los usuarios para realizar cualquier función. Estando todos los usuarios, sin importar su rol, sin capacidad de realizar ninguna acción del sistema. Estas acciones se refieren a los métodos en los controladores. Ellos funcionarán como los puntos de control antes dar o negar permiso para realizar una acción.

Los permisos para realizar acciones (métodos en los controladores), entonces, se van otorgando en un solo archivo de configuración *"authorization_rules.rb"* en el que a un rol se le abren los permisos para realizar acciones de algún controlador y por ende, todos los usuarios que pertenezcan a ese rol podrán realizar dichas acciones.

De esta marea se trabaja bajo la metodología de autorización de white-listing, es decir, absolutamente nadie puede hacer nada, al menos que se especifique lo contrario.

```
role :admin do
has_permission_on :actividades,:grupos_estudiantiles,:mensajes,:users, :universidades,
 :to=>[:index,:new, :show, :create, :edit, :update, :destroy]
end
role :master do
has_permission_on :users, :to=>[:index,:new, :show, :create, :edit, :update, :destroy]
has_permission_on :grupos_estudiantiles, :to=>[:index, :show]
end
```

Figura 38: Parte del código de "authorization_rules.rb", en donde se declaran las funciones a las cuales puede acceder cada usuario. Fuente: Elaboración propia

IV.3.7. Desarrollo del módulo de carga y descarga de archivos

Para el desarrollo del módulo de carga y descarga se necesitaba de una forma eficiente, segura y sencilla de subir y bajar archivos del servidor.

Se utilizó un gem de ruby on rails llamado "rails-tiny-mce" que incorpora un módulo de subida de archivos a *tinyMCE* en el momento de crear nuevas actividades.

Uno de los requisitos más importantes es que los archivos no pudiesen ser accesibles a menos de que se tengan los privilegios necesarios para descargar.

La funcionalidad que se quería lograr era que al subir un archivo (por ejemplo: *mifoto.jpg*) al servidor, la dirección (url) de acceso al archivo tenga la estructura de su dirección absoluta en internet (por ejemplo *http://www.dominiodelportal.com/uploads/mifoto.jpg*). En un servidor clásico acceder a un archivo de esta manera mantiene este acceso fuera de ningún tipo de sistema sino que se limita a ser una descarga de archivos por http a través del navegador. El problema con esta solución es que cualquier persona con acceso a ese *link* podría descargarse ese archivo.

Ruby on Rails viene con un archivo de configuración de rutas: *routes.rb.* El modelo de enrutamiento de Ruby permite enmascarar que un acceso a "uploads/mifoto.jpg" sea realmente procesado por un controlador de cargas y descargas en el que se realicen distintos tipos de validación antes de otorgar el permiso de descargar el archivo. Es decir, que a pesar de que parezca que se está accediendo a un archivo de data directamente, lo que realmente está ocurriendo es simplemente una ejecución más de un método en un controlador del sistema que luego muestra o permite descargar el archivo solicitado.

IV.4. Pruebas

Ruby on Rails tiene una funcionalidad de la cual no se ha hablado hasta ahora y es que contiene por defecto 3 ambientes funcionales: *Development*, *Test* y *Production*. Hasta ahora todo el desarrollo se ha hecho en el ambiente por defecto que es el de Development o desarrollo. Sin embargo cada uno de

estos ambientes se utiliza para un propósito distinto. Para entender esto mejor, es mejor citar un caso.

Por ejemplo, en el ambiente de desarrollo, los cache están apagados por defecto. Esto permite que al hacer modificaciones a la aplicación, los cambios sean visibles inmediatamente y no haya que preocuparse por borrar los caches y recargar la página. Otra de las diferencias es que en el modo de desarrollo, el servidor para mandar emails se encuentra apagado por defecto, de tal manera que le permite al programador generar correos, revisarlos a través de la consola, pero sin probar el envío como tal. Así como éstas, existen algunas otras funcionalidades de cada uno de los ambientes que va a favorecer o ayudar en la fase de pruebas, desarrollo o implementación.

Rails también provee un ambiente de prueba, en el cual se ejecutan secuencialmente cada una de las pruebas definidas en la carpeta *tests* de la aplicación rails. Es decir, incluso para la fase de pruebas rails incentiva el uso de automatización de las pruebas a través de código. El uso de estas pruebas, además de que ayuda a verificar que el sistema esté cumpliendo con lo esperado, puede ser de gran utilidad en la etapa de desarrollo porque ayuda a verificar que al momento de hacer una modificación en la lógica (ya sea por evitar repetición o por mejoras en el desempeño) la funcionalidad siga intacta.

Para implementar una prueba automatizada en el sistema se siguieron tres pasos básicos.

- Arreglar: Obtener todo lo que se utilizará para la prueba.
- Actuar: Ejecutar los métodos que van a ser probados.

- Verificar: Luego de que se ejecuten todas las acciones de la prueba, se verifique si distintas condiciones se cumplen que hacen que la prueba pase o falle

La prueba será aprobada en el caso de que cada una de estas verificaciones pase.

Como ejemplo de una de las pruebas que se realizaron durante el desarrollo:

Supóngase que en la base de datos de estudiantes se tienen 3 campos:

- nombre
- inicial_segundo
- apellido

El campo inicial_segundo no es requerido, por lo que en la aplicación algunos estudiantes lo tienen y otros no. En algún momento en la aplicación se nota que en varios lugares se está teniendo que concatenar estos tres campos para mostrar en la vista todos juntos.

```
<% for estudiante in @estudiantes%>
    <a href="<% estudiante_path(estudiante)%>">
        <%= estudiante.primer_nombre%>
        <%= if estudiante.inicial_segundo? %>
            <%= estudiante.inicial_segundo%>.
        <% end %>
            <%= estudiante.apellido%>
    </a>
<% end %>
```

Figura 39: Código antes de mejorar la sintaxis por código más eficiente. Fuente: Elaboración propia.

Es por esto que se decidió implementar un método en el modelo estudiante que se encargara de devolver el nombre completo del estudiante tomando en cuenta si tiene o no asignado una inicial de segundo nombre y le agregase el punto en caso tal que lo tenga.

Luego de que se implementó ese método se procede a crear una prueba que verifique que el método esté devolviendo el nombre completo en los 3 casos posibles.

1. Que el estudiante no tenga incial_segundo.
2. Que el estudiante sí tenga inicial_segundo.
3. Que el estudiante tenga inicial_segundo pero que esta sea igual a un string vacío.

```
def test_nombre_completo
  assert_equal 'Juan Lagrange', full_name('Juan',nil,'Lagrange'),"Sin segunda inicial"
  assert_equal 'Juan A. Lagrange'full_name('Juan','A','Lagrange'), "Con segunda inicial"
  assert_equal 'Juan Lagrange'full_name('Juan','','Lagrange'),"Inicial segunda vacia"
end

def full_name(nombre,inicial_segundo,apellido)
  Estudiante.new(:nombre => nombre, :inicial_segundo => inicial_segundo, :apellido => apellido ).nombre_completo
end
```

Figura 40: Código de una prueba. Fuente: Elaboración propia

En estas pruebas programadas se puede verificar desde la funcionalidad de cada controlador hasta aspectos sumamente específicos como por ejemplo, si el navegador del usuario no tiene javascript habilitado el sistema no deje de funcionar.

Es importante destacar que a pesar de que las pruebas automatizadas de Ruby on Rails son de extrema utilidad, estas no invalidan la necesidad de realizar pruebas humanas a las aplicaciones para cerciorarse de que los casos de usos estén siguiendo los flujos correctamente.

Las pruebas fueron clasificadas en 3 tipos:

- Pruebas Modulares: Pruebas específicas de cada módulo.
- Pruebas de Integración: Pruebas de interoperabilidad entre un módulo con otro
- Pruebas de Seguridad: Pruebas minuciosas para garantizar la privacidad necesaria de los datos en el sistema.

Para los 3 tipos de pruebas se realizaron tanto pruebas automatizadas como pruebas humanas tratando de emular casos de vida real y casos totalmente extremos, con el fin de asegurar la robusticidad del sistema.

IV.5. Plan de implementación

Existen 2 principales maneras de llevar a producción una aplicación hecha en Ruby on Rails. Estas son Mongrel y Phusion Passenger.

Mongrel es un servidor web *open source* que tiene como beneficio que es relativamente fácil de instalar y que no depende de ningún otro servidor instalado para poder correr aplicaciones en ruby. Sin embargo, se recomienda de que sea instalado junto con *Apache WEB Server* para repartir la carga. Esta manera de desplegar una aplicación ruby es ideal para aplicaciones que tengan un bajo tráfico en general. Esto se debe a que es un servidor *single threaded* y solo puede manejar un *request* a la vez. Sin embargo, en aplicaciones con mayor tráfico, se recomienda implementar la segunda opción, *Phusion Passenger*.

Phusion Passenger es un módulo para Apache web Server y como tal debe ser desplegado junto con él. Hoy en día constituye la principal manera de desplegar aplicaciones de Ruby on Rails y su éxito se debe en gran medida a que a diferencia de *Mongrel*, *Phusion Passenger* es un servidor *multithreaded*. Es por esto que se decidió crear el plan de implementación en base a *Phusion Passenger*.

En primera instancia debe instalarse o contar con la instalación de Apache Web Server. Luego se deben correr 2 comandos en la consola:

- gem install passenger

Este comando instala el gem passenger en la aplicación.

- passenger-install-apache2-module

Este comando instala el módulo de Phusion Passenger para Apache. En otras palabras éste es el módulo que permite a Phusion Passenger comunicarse y utilizar los recursos de Apache web Server.

Una vez instalado, debe copiarse la carpeta en donde se encuentra la aplicación desarrollada en ruby a la carpeta *home* en donde está corriendo el servidor apache. Luego, debe modificarse el archivo “environment.rb” en la carpeta de configuración de la aplicación. Debe modificarse para colocar como nuevo ambiente, el ambiente de producción. Adicionalmente debe modificarse el archivo llamado database.yml y colocar en él las nuevas variables de la base de datos de producción. Por ejemplo, en la figura 41, se muestra una vista del archivo database.yml en donde se puede observar las distintas configuraciones según el ambiente en que se esté trabajando.

```
development:
  adapter: mysql
  encoding: utf8
  reconnect: false
  database: unet_development
  pool: 5
  username: root
  password: 12345678
  socket: /tmp/mysql.sock

  production:
    adapter: mysql
    encoding: utf8
    reconnect: false
    database: unet_production
    pool: 5
    username: root
    password:
    socket: /tmp/mysql.sock
```

Figura 41: Código de "database.yml", archivo de configuración de las bases de dato de los distintos ambientes ofrecidos por Ruby on Rails. Fuente: Elaboración propia

Una vez que se tienen los datos correctos de ambiente, se debe correr a través de la consola, el comando de ruby anteriormente explicado que es *rake:db migrate.* Este comando como fue mencionado anteriormente, permite

importar todo el esqueleto de la base de datos de la aplicación desarrollada localmente hasta el nuevo servidor de base de datos donde será alojado.

El último paso para implementar la aplicación en el nuevo servidor web utilizando Phusion Passenger, es el de crear un archivo llamado htaccess que contenga 2 líneas de código:

- RailsBaseURI
- PassengerAppRoot /home/nombre_de_aplicacion_ruby_on_rails

El propósito de este archivo es el de indicarle al servidor cuál va a ser su directorio de *home* de donde puede tomar y correr la aplicación desarrollada.

Una vez seguido todos estos pasos, la aplicación se encontraría totalmente implementada en un ambiente web.

Capitulo V: Conclusiones

Luego de la realización de este proyecto se llegó a las siguientes conclusiones:

- Las necesidades que presenta el movimiento estudiantil venezolano son diversas y complejas, pero el desarrollo de un sistema como éste proporciona una herramienta apropiada para mejorar la comunicación y la organización entre las distintas agrupaciones estudiantiles y con el público en general.

- Ruby on Rails, además de proponer una estructura organizada en donde se respeta la separación de las responsabilidades en un programa, proporciona una plataforma de desarrollo sumamente eficaz que automatiza muchas de las actividades comunes que se tienen que hacer para cualquier proyecto web.

- En vista de los constantes cambios en la situación política y social de Venezuela, nuevas necesidades del movimiento estudiantil son propensas a surgir, y haber realizado la plataforma en el framework Ruby on Rails le proporciona al sistema la escalabilidad necesaria para implementar nuevos requerimientos de manera rápida y sencilla.

Capítulo VI: Recomendaciones

En el mundo del desarrollo web, siempre es recomendable que los administradores de un sistema estén alerta ante surgimientos de nuevas herramientas que potencialmente podrían mejorar la aplicación. En especial, en el mundo de Ruby on Rails, constantemente están saliendo nuevas actualizaciones y *gems* que proveen nuevas funcionalidades o mejoran funcionalidades ya presentes en un sistema, por lo que, en pro de evitar una futura obsolescencia del sistema, se recomienda siempre estar en la búsqueda y en la investigación de posibles actualizaciones.

Se recomienda, además, que antes de embarcar en la implementación de una actualización, se revise si ésta no rompería con los distintos paradigmas estructurales con los que el sistema fue diseñado.

Referencias Bibliográficas:

- Fowler, C. (2006). Rails Recipes (Pragmatic Programmers). NY: Pragmatic Bookshelf.
- *Ruby on Rails Guides* (25/03/2008), [en línea] Recuperado el 20 de marzo del 2010 de http://guides.rubyonrails.org/
- Declarative Authorization (16/11/2009), [en linea].Recuperado el 5 de Enero del 2010 de http://railscasts.com/episodes/188-declarative-authorization
- Wells, D. (2009) Extreme Programming: A gentle introduction, [en línea]. Recuperado el 18 de marzo de 2010 de http://www.extremeprogramming.org
- Guthrie, S. (2009) ASP.NET MVC Framework, [en línea]. Recuperado el 11 de febrero de 2010 de http://weblogs.asp.net/scottgu/archive/2007/10/14/asp-net-mvc-framework.aspx.
- Ambler, S. (2008) Mapping Objects to Relational Databases: O/R Mapping In Detail, [en línea]. Recuperado el 15 de febrero de 2010 de http://www.agiledata.org/essays/mappingObjects.html
- Wells, D. (2009) Agile Software Development: A gentle introduction, [en línea]. Recuperado el 18 de marzo de 2010 de http://www. http://www.agile-process.org.org
- Pressman, R. (2005). Ingeniería del Software (6ta Edición ed.). Mexico DF: Mc Graw-Hill.
- Rodriguez, A. (06/11/2008) RESTful Web services: The basics, [en línea]. Recuperado el 21 de marzo de 2010 de

https://www.ibm.com/developerworks/webservices/library/ws-restful/

- *Jquery UI Demos and Documentation* (2010), [en línea] Recuperado el 10 de diciembre del 2009 de http://jqueryui.com/demos/
- Storimer, J. (22/12/2008) A Closer Look At the Blueprint CSS Framework, [en línea]. Recuperado el 11 de marzo de 2010 de http://net.tutsplus.com/tutorials/html-css-techniques/a-closer-look-at-the-blueprint-css-framework/
- Crockford, D. (15/10/2008) A Functional Javascript, [en línea/]. Recuperado el 11 de marzo de 2010 de http://www.blinkx.com/watch-video/douglas-crockford-on-functional-javascript/xscZz8XhfuNQ_aaVuyUB2A
- Bos, B. (15/10/2008Cascading Style Sheets, [en línea/]. Recuperado el 11 de noviembre de 2009 de http://www.w3.org/Style/CSS/
- *Fundación Un millon de voces*, [en línea] Recuperado el 30 de marzo del 2010 de http://www.millondevoces.org/contenido/
- *Facebook and Twitter Post Large Year over Year Gains in Unique UsersRuby* (04/04/2010), [en línea] Recuperado el 5 de Junio del 2010 de http://blog.nielsen.com/nielsenwire/global/facebook-and-twitter-post-large-year-over-year-gains-in-unique-users/
- *Average Time Spent by Social Network Visitors Increases 100% to 6 Hours+ in March* (07/04/2010), [en línea] Recuperado el 5 de Junio del 2010 de http://socialmediaatwork.com/2010/05/07/average-time-spent-by-social-network-visitors-increases-100-to-6-hours-in-march/
- *Time Spent on Social Networks up 82% Around the World* (24/02/2010), [en línea] Recuperado el 5 de Junio del 2010 de

http://www.briansolis.com/2010/02/time-spent-on-social-networks-up-82-around-the-wrold/

Apéndice A: Casos de Uso

Nombre	**Crear Usuario**
Actor	Administrador, Master.
Descripción	Permite la creación de un nuevo usuario
Precondición	N/A
Flujo normal	1. Se accede al formulario de creación 2. Se llenan los datos requeridos 3. Se validan los datos y se procede a la creación
Flujo alternativo	N/A
Poscondición	El Usuario queda guardado en base de datos

Figura 42: Caso de uso "Crear Usuario". Fuente: Elaboración propia

Nombre	**Consultar Usuario**
Actor	Administrador, Master, Junior
Descripción	Permite consultar los datos de un usuario del sistema
Precondición	El usuario debe existir en el sistema
Flujo normal	3. Se selecciona el usuario a consultar 4. Se accede a los datos del usuario
Comentarios	Según el rol del usuario haciendo la consulta, este podrá acceder a más o menos información.
Poscondición	N/A

Figura 43: Caso de uso "Consultar Usuario". Fuente: Elaboración propia

Nombre	Editar Usuario
Actor	Administrador, Master, Junior
Descripción	Permite la edición de los datos de un usuario
Precondición	El usuario debe existir en el sistema
Flujo normal	1. Se selecciona el usuario a editar 2. Se accede al formulario de editar 3. Se modifican los datos 4. Se validan los datos y se procede a la actualización
Comentarios	Según el rol del usuario haciendo la edición, este podrá editar distintos usuarios.
Poscondición	Los nuevos datos del usuario quedan guardados en base de datos

Figura 44: Caso de uso "Editar Usuario". Fuente: Elaboración propia

Nombre	Eliminar Usuario
Actor	Administrador, Master, Junior
Descripción	Permite eliminar a un usuario del sistema
Precondición	El usuario debe existir en el sistema
Flujo normal	1. Se selecciona el usuario a eliminar 2. Se presiona sobre eliminar
Descripción	Según el rol del usuario haciendo la edición, este podrá eliminar distintos usuarios.
Poscondición	El Usuario queda eliminado del sistema

Figura 45: Caso de uso "Eliminar Usuario". Fuente: Elaboración propia

Nombre	Resetear Clave de usuario
Actor	Administrador, Master, Junior
Descripción	Permite modificar la claverde un usuario del sistema
Precondición	El usuario debe existir en el sistema
Flujo normal	1. Se selecciona la opción de “olvidé mi clave” 2. Se llenan los datos requeridos 3. Se resetea la clave y se envía un email con una clave provisional
Descripción	
Poscondición	La Clave del usuario es modificada

Figura 46: Caso de uso "Resetear clave de Usuario". Fuente: Elaboración propia

Nombre	Crear Universidad
Actor	Administrador
Descripción	Permite la creación de una universidad con sus respectivos datos
Precondición	NA
Flujo normal	1. Se accede al formulario de creación 2. Se llenan los datos requeridos 3. Se validan los datos y se procede a la creación
Flujo alternativo	NA
Poscondición	La universidad queda guardada

Figura 47: Caso de uso "Crear Universidad". Fuente: Elaboración propia

Nombre	Consultar Universidad
Actor	Administrador, Master, Junior, Guest
Descripción	Permite visualizar los datos de una universidad
Precondición	Que la universidad ya haya sido creada
Flujo normal	1. Se selecciona la universidad a ser consultada 2. Se accede a la vista de la universidad
Comentario	Según el rol del usuario haciendo la consulta, este podrá acceder a más o menos información
Poscondición	NA

Figura 48: Caso de uso "Consultar Universidad". Fuente: Elaboración propia

Nombre	Editar Universidad
Actor	Administrador
Descripción	Permite editar los datos de una universidad
Precondición	Que la universidad ha ser editada ya haya sido creada
Flujo normal	1. Se selecciona la universidad a editar 2. Se accede al formulario de editar 3. Se modifican los datos 4. Se validan los datos y se procede a la actualización
Flujo alternativo	NA
Poscondición	La universidad queda actualizada

Figura 49: Caso de uso "Editar Universidad". Fuente: Elaboración propia

Nombre	Eliminar Universidad
Actor	Administrador
Descripción	Permite eliminar una universidad del sistema
Precondición	Debe existir la universidad que se desea eliminar
Flujo normal	1. Se selecciona la universidad 2. Se presiona sobre eliminar 3. Al aparecer una ventana de confirmación, presionar confirmar
Comentario	Al eliminar una universidad, quedan eliminados a su vez todas las agrupaciones, estudiantes y actividades relacionadas con esa universidad
Poscondición	La universidad queda eliminada

Figura 50: Caso de uso "Eliminar Universidad". Fuente: Elaboración propia

Nombre	Crear Estudiante
Actor	Administrador, Master, Junior
Descripción	Permite la creación de un estudiante con sus respectivos datos
Precondición	NA
Flujo normal	1. Se accede al formulario de creación 2. Se llenan los datos requeridos 3. Se validan los datos y se procede a la creación
Flujo alternativo	
Poscondición	El estudiante queda guardado

Figura 51: Caso de uso "Crear Estudiante". Fuente: Elaboración propia

Nombre	Consultar Estudiante
Actor	Administrador, Master, Junior
Descripción	Permite visualizar los datos de un estudiante
Precondición	Que el estudiante ya haya sido creado
Flujo normal	1. Se selecciona el estudiante ha ser consultado 2. Se accede a la vista del estudiante
Comentario	NA
Poscondición	NA

Figura 52: Caso de uso "Consultar Estudiante". Fuente: Elaboración propia

Nombre	Editar Estudiante
Actor	Administrador, Master, Junior
Descripción	Permite editar los datos de un estudiante
Precondición	Que el estudiante ya haya sido creado
Flujo normal	1. Se selecciona el estudiante a editar 2. Se accede al formulario de editar 3. Se modifican los datos 4. Se validan los datos y se procede a la actualización
Flujo alternativo	NA
Poscondición	El estudiante queda actualizado

Figura 53: Caso de uso "Editar Estudiante". Fuente: Elaboración propia

Nombre	Eliminar Estudiante
Actor	Administrador, Master, Junior
Descripción	Permite eliminar un estudiante del sistema
Precondición	Debe existir el estudiante que se desea eliminar
Flujo normal	1. Se selecciona la universidad 2. Se presiona sobre eliminar 3. Al aparecer una ventana de confirmación, presionar confirmar
Flujo alternativo	
Poscondición	El estudiante queda eliminado

Figura 54: Caso de uso "Eliminar Estudiante". Fuente: Elaboración propia

Nombre	Crear Agrupación Estudiantil
Actor	Administrador
Descripción	Permite la creación de una nueva agrupación estudiantil con sus respectivos datos
Precondición	NA
Flujo normal	1. Se accede al formulario de creación 2. Se llenan los datos requeridos 3. Se validan los datos y se procede a la creación
Flujo alternativo	
Poscondición	La agrupación estudiantil queda guardada

Figura 55: Caso de uso "Crear Agrupación Estudiantil". Fuente: Elaboración propia

Nombre	Consultar Agrupación estudiantil
Actor	Administrador, Master, Junior, Guest
Descripción	Permite visualizar los datos de una agrupación estudiantil
Precondición	Que la agrupación estudiantil ya haya sido creado
Flujo normal	1. Se selecciona la agrupación estudiantil ha ser consultada 2. Se accede a la vista de la agrupación estudiantil
Comentario	NA
Poscondición	NA

Figura 56: Caso de uso "Consultar Agrupación Estudiantil". Fuente: Elaboración propia

Nombre	Editar Agrupación Estudiantil
Actor	Administrador, Master
Descripción	Permite editar los datos de una agrupación estudiantil
Precondición	Que la agrupación estudiantil ya haya sido creado
Flujo normal	1. Se selecciona la agrupación a editar 2. Se accede al formulario de editar 3. Se modifican los datos 4. Se validan los datos y se procede a la actualización
Flujo alternativo	NA
Poscondición	La agrupación estudiantil queda actualizada

Figura 57: Caso de uso "Editar Agrupación Estudiantil". Fuente: Elaboración propia

Nombre	Eliminar Agrupación Estudiantil
Actor	Administrador
Descripción	Permite eliminar una agrupación estudiantil del sistema
Precondición	Debe existir la agrupación estudiantil que se desea eliminar
Flujo normal	1. Se selecciona la agrupación estudiantil 2. Se presiona sobre eliminar 3. Al aparecer una ventana de confirmación, presionar confirmar
Comentario	Al eliminar una agrupación estudiantil, quedan eliminadas todas las actividades relacionadas con esta agrupación.
Poscondición	La agrupación estudiantil queda eliminada

Figura 58: Caso de uso "Eliminar Agrupación Estudiantil". Fuente: Elaboración propia

Nombre	Crear Actividad
Actor	Administrador, Master, Junior
Descripción	Permite la creación de una nueva actividad con sus respectivos datos
Precondición	NA
Flujo normal	1. Se accede al formulario de creación 2. Se llenan los datos requeridos 3. Se validan los datos y se procede a la creación
Flujo alternativo	
Poscondición	La actividad queda guardada

Figura 59: Caso de uso "Crear Actividad". Fuente: Elaboración propia

Nombre	Consultar Actividad
Actor	Administrador, Master, Junior, Guest
Descripción	Permite visualizar los datos de una actividad
Precondición	Que la actividad ya haya sido creada
Flujo normal	1. Se selecciona la agrupación estudiantil ha ser consultada 2. Se accede a la vista de la agrupación estudiantil
Comentario	NA
Poscondición	NA

Figura 60: Caso de uso "Consultar Actividad". Fuente: Elaboración propia

Nombre	Editar Actividad
Actor	Administrador, Master, Junior
Descripción	Permite editar los datos de una actividad
Precondición	Que la actividad ya haya sido creado
Flujo normal	1. Se selecciona la actividad a editar 2. Se accede al formulario de editar 3. Se modifican los datos 4. Se validan los datos y se procede a la actualización
Flujo alternativo	NA
Poscondición	La actividad queda actualizada

Figura 61: Caso de uso "Editar Actividad". Fuente: Elaboración propia

Nombre	Eliminar Actividad
Actor	Administrador
Descripción	Permite eliminar una actividad del sistema
Precondición	Debe existir la actividad que se desea eliminar
Flujo normal	1. Se selecciona la actividad 2. Se presiona sobre eliminar 3. Al aparecer una ventana de confirmación, presionar confirmar
Comentario	
Poscondición	La actividad queda eliminada

Figura 62: Caso de uso "Eliminar Actividad". Fuente: Elaboración propia

Nombre	Cargar Multimedia
Actor	Administrador, Master, Junior
Descripción	Permite cargar archivos de diversos formatos al sistema
Precondición	NA
Flujo normal	1. Se selecciona la opción de cargar archivo 2. Se selecciona el archivo a cargar 3. Se presiona sobre cargar 4. Se valida el archivo según tipo y tamaño 5. Se guarda el archivo en el sistema
Comentario	Este archivo podrá ser utilizado para nutrir el diseño del sistema
Poscondición	El archivo queda almacenado

Figura 63: Caso de uso "Cargar Multimedia". Fuente: Elaboración propia

Nombre	Descargar Multimedia
Actor	Administrador, Master, Junior, Guest
Descripción	Permite descargar archivos desde el sistema
Precondición	NA
Flujo normal	1. Se presiona sobre el archivo que se desea descargar
Comentario	
Poscondición	El usuario debe haber descargado el archivo a su computadora.

Figura 64: Caso de uso "Descargar Multimedia". Fuente: Elaboración propia

Nombre	Eliminar Multimedia
Actor	Administrador, Master, Junior
Descripción	Permite eliminar los archivos guardados en el sistema
Precondición	NA
Flujo normal	1. Se selecciona el archivo que se desea eliminar 2. Se presiona sobre eliminar archivo 3. Al aparecer una ventana de confirmación, presionar confirmar
Comentario	
Poscondición	El archivo queda eliminado del sistema

Figura 65: Caso de uso "Eliminar Multimedia". Fuente: Elaboración propia

Nombre	Enviar Mensaje
Actor	Administrador, Master, Junior
Descripción	Permite el envío de mensajes entre usuarios del sistema
Precondición	N/A
Flujo normal	1. Se accede al formulario de nuevo mensaje 2. Se llenan los datos requeridos 3. Se selecciona el destinatario 4. Se validan los datos y se procede al envio
Flujo alternativo	N/A
Poscondición	El receptor tiene un mensaje nuevo en su bandeja de entrada

Figura 66: Caso de uso "Enviar Mensaje". Fuente: Elaboración propia

Nombre	Consultar bandeja de entrada
Actor	Administrador, Master, Junior
Descripción	Permite la visualización de los mensajes recibidos
Precondición	N/A
Flujo normal	1. Se accede a la bandeja de entrada
Flujo alternativo	N/A
Poscondición	

Figura 67: Caso de uso "Consultar Bandeja de Entrada". Fuente: Elaboración propia

Nombre	Consultar bandeja de salida
Actor	Administrador, Master, Junior
Descripción	Permite la visualización de los mensajes enviados
Precondición	N/A
Flujo normal	1. Se accede a la bandeja de salida
Flujo alternativo	N/A
Poscondición	

Figura 68: Caso de uso "Consultar Bandeja de Salida". Fuente: Elaboración propia

Nombre	Leer Mensaje
Actor	Administrador, Master, Junior
Descripción	Permite la lectura de mensajes
Precondición	N/A
Flujo normal	1. Se accede a la bandeja de entrada o de salida 2. Se selecciónа el mensaje a leer. 3. Se procede a la vista del mensaje
Flujo alternativo	N/A
Poscondición	El mensaje se marca como leído.

Figura 69: Caso de uso "Leer Mensaje". Fuente: Elaboración propia

Apéndice B: Firebug

Firebug es una extensión de Mozilla Firefox que provee herramientas de desarrollo web. Entre sus principales funcionalidades, permite revisar, editar y monitorear el HTML, XHTML, CSS y JavaScript de una página web. Es una herramienta open source y puede ser instalada en cualquier sistema operativo.

Para demostrar su funcionamiento, en la figura 70 se muestra la página de platum.unimet.edu.ve desplegada utilizando firebug. En ella logramos ver que se está apuntando sobre el elemento superior izquierdo de la página (donde se encuentra el logotipo de la Universidad Metropolitana). En la parte inferior izquierda podemos ver desplegado el código HTML que está recibiendo el navegador y en la parte inferior derecha se logra distinguir la hoja de estilo ó CSS de la página. De esta manera también se pueden modificar los elementos y el CSS para probar los diseños de la página en cuestión.

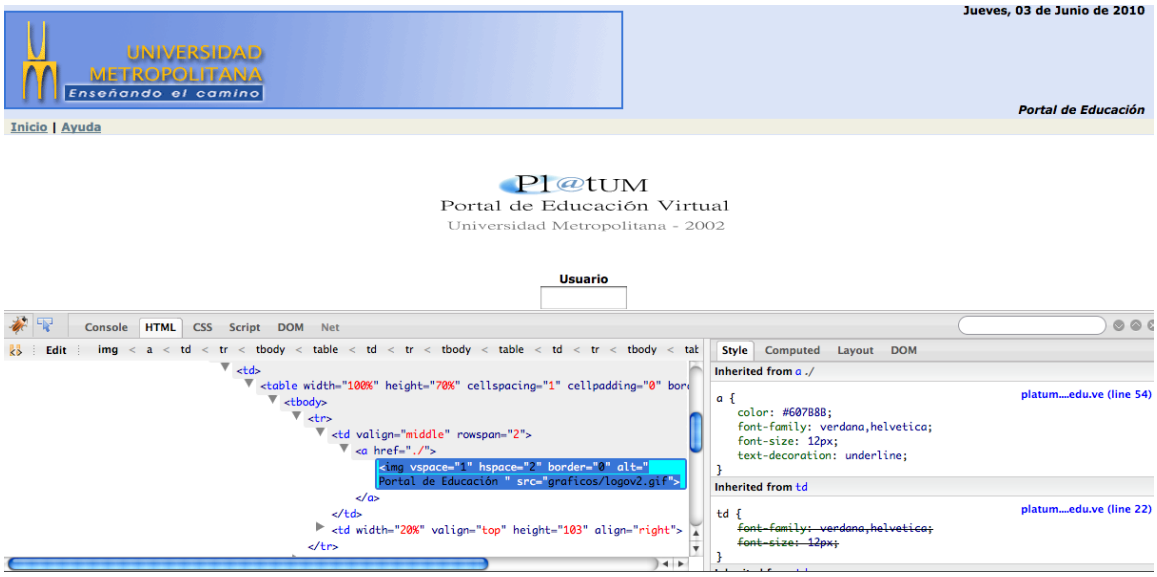

Figura 70: Vista del portal de "Platum" de la Universidad Metropolitana vista a través de Firebug. Fuente: http://platum.unimet.edu.ve/

En la figura 71, se modificó el atributo “bgcolor” del elemento “body” de la tabla principal de la página web de platum.

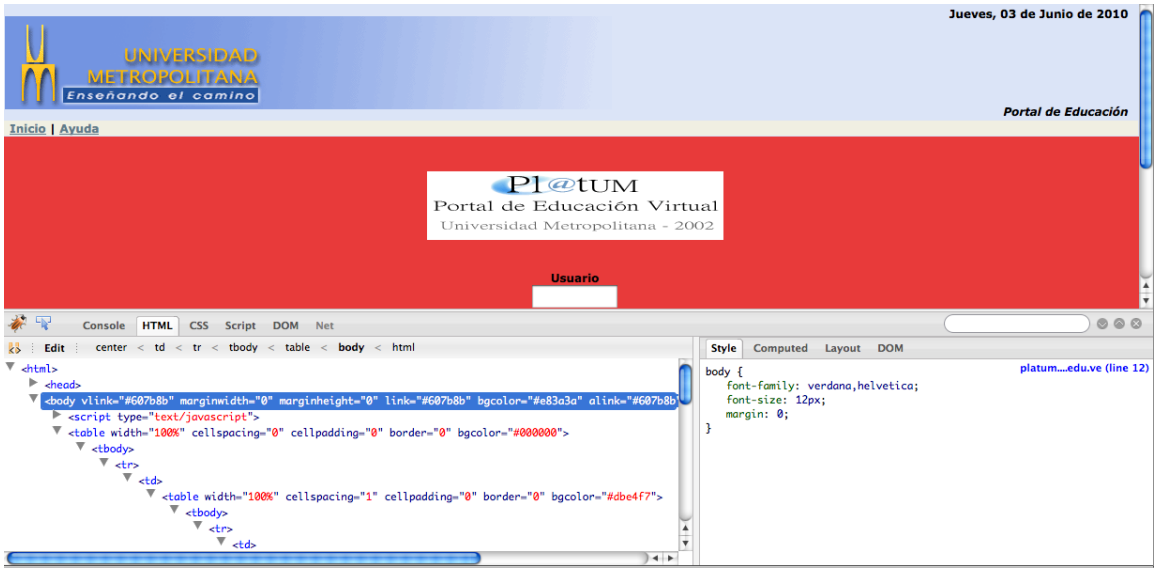

Figura 71: Vista del portal de "Platum" de la Universidad Metropolitana visto y modificado a través de Firebug. Fuente: http://platum.unimet.edu.ve/

Apéndice C: Apache

El servidor HTTP Apache, es un servidor web de uso masivo. Es reconocido por haber sido una pieza clave en el crecimiento de inicial del Internet. En el 2009 se convirtió en el primer software de servidor web que sobrepasó los 100 millones de usuarios.

Figura 72: Logotipo de Apache Web Server. Fuente: http://www.webdesignblog.biz/wp-content/uploads/2009/09/apache-http-server-logo1.jpg

Apache es desarrollado y mantenido por una comunidad abierta de desarrolladores bajo el auspicio de la *Apache Software Foundation.* La aplicación está disponible para Unix, Linux, Windows, Mac OS, etc.

Apache es utilizado principalmente para servir páginas web estáticas y dinámicas en Internet, soporta una variedad de funcionalidades en su forma más básica pero también muchas a través de módulos compilados los cuales extienden las funcionalidades básicas. Estas pueden variar desde programación del lado de servidor para un lenguaje específico hasta esquemas de autenticación.

Generalmente, programadores desarrollando aplicaciones web tienen una instalación local de Apache para ir realizando pruebas de de su código a medida que se va desarrollando.