Universidad Central De Venezuela

Facultad De Ciencias

Escuela De Computación

Ingeniería del software

DESARROLLO DE UN SISTEMA DE ALQUILER DE CANCHAS DE TENIS USANDO ELEMENTOS DE APLICACIONES WEB ENRIQUECIDAS

Trabajo Especial de Grado
Presentado ante la Ilustre
Universidad Central de Venezuela
Por el bachiller
Andrés E. Blanco R.
C.I. 15.343.131
Para optar al título de
Licenciado en Computación

Tutor: Prof. Wilfredo Rangel

Caracas, Mayo de 2011.

UNIVERSIDAD CENTRAL DE VENEZUELA

FACULTAD DE CIENCIAS

ESCUELA DE COMPUTACIÓN

TRABAJO ESPECIAL DE GRADO

Titulo: Desarrollo de un Sistema de Alquiler de Canchas de Tenis usando elementos de Aplicaciones Web Enriquecidas.

Autor: Andrés Blanco

Palabras Claves: AJAX, Sistema, Web, Alquiler de Canchas, FVT

Tutor: Wilfredo Rangel
Fecha de presentación: 31 de Mayo de 2011.

Resumen

Para el desarrollo del Sistema de alquiler de canchas de Tenis de la F.V.T es indispensable el uso de herramientas que ayuden de manera versátil y eficaz en la construcción de esta aplicación. Adicionalmente, se quiere que esta aplicación tenga una interacción con el usuario de la forma mas fluida posible, para ello se utilizaron elementos de Aplicaciones Web Enriquecidas más específicamente el enfoque AJAX y todas las bondades que este ofrece.

ACTA

Quienes suscriben miembros del Jurado, designado por el Consejo de Escuela de Computación, para examinar el Trabajo Especial de Grado presentado por el bachiller **Andrés E. Blanco R., C.I. 15.343.131** con el título **"Desarrollo de un Sistema de Alquiler de Canchas de Tenis usando elementos de Aplicaciones Web Enriquecidas",** a los fines de optar al título de Licenciado en Computación, dejan constancia lo siguiente:

Leído como fue, dicho trabajo por cada uno de los miembros del Jurado, se fijó el 31 de Mayo de 2011 a las 9:30 a.m., para que su autor lo defendiera en forma pública, se hizo en el aula de Postgrado de la Escuela de Computación, mediante una presentación oral de su contenido. Finalizada la defensa pública del Trabajo Especial de Grado, el Jurado decidió aprobarlo con una nota de _____ puntos, en fe de lo cual se levanta la presente Acta, en Caracas a los 31 días del mes de Mayo del año dos mil once.

Prof. Antonio Leal
Jurado Principal

Prof. Rossana Díaz
Jurado Principal

Prof. Wilfredo Rangel
Tutor

"Le dedico este trabajo especial de grado, ante todo, a Dios, que me ha acompañado a lo largo de mi vida, quien me guía y me protege de todas las cosas malas, por darme salud, fuerza y serenidad.

A mi padres Nelly Ruiz y Jesús Blanco, por ser padres ejemplares y ayudarme a salir adelante para ver hecho realidad mis sueños, sin Uds. no estuviera en esta etapa de mi vida, se merecen la misma alegría que siento por este momento tan grato, este trabajo es para los dos por su sacrificio para conmigo.

A mis hermanos Jesús Blanco Ruiz y Andry Blanco, por soportarme y darme sus manos en los momentos que necesitaba de ayuda y motivación, para Uds. hermanos que también fueron parte de este gran esfuerzo.

A mis Tíos y Tías y a todos mis primos, que siempre estuvieron pendientes de preguntarme como iba y de motivarme para seguir adelante y darme su apoyo.

A mi Abuela Carlota, por sus sabios consejos, a los que ya no están conmigo pero siempre fueron parte importante de mi crecimiento, mis Abuelos Jesús A. Blanco y Clara E. Rodríguez y también a mi Tío Pedro Duque que siempre me lleno de alegría con su apoyo y buenas palabras.

Muy Agradecido con todos Uds. Y ya sé lo que tengo que hacer ahora

Debo seguir respirando

Porque mañana saldrá el sol y

Quién sabe qué traerá la marea"

Andrés Blanco

Agradezco

A Dios ante todo, por permitirme vivir este gran momento y mantener firme mi esperanza y dedicación durante todo el proceso de este Trabajo Especial de Grado.

A toda mi familia, mis Tíos, Tías, mis primos, hermanos y en especial a mis Padres Nelly Ruiz y Jesús Blanco por haberme dado su apoyo incondicional durante toda mi carrera.

A mis tutores y jurados Wilfredo Rangel, Antonio Leal y Rossana Díaz por guiarme y apoyarme para alcanzar todos los objetivos propuestos y compartir sus conocimientos.

A la gente de Matrix CPM Solutions, que confiaron en mí para el desarrollo de este proyecto.

Al la Federación Venezolana de Tenis, quienes me facilitaron todo lo necesario para poder desarrollar este proyecto.

A mi linda novia Geraldyne Cabeza por su apoyo, preocupación y brindarme toda esa energía positiva durante estos últimos meses de mi Trabajo Especial de Grado.

A mis mejores amigos y compañeros de todos los semestres Samuel, Hanson, Héctor, Nino, Leo, Edwin, Chuchu y mi compadre Kendall, por todo su apoyo incondicional.

Índice de Contenidos

Índice de Figuras

Índice de Tablas

Introducción

La importancia de Internet es innegable, especialmente cuando la tendencia es cada vez más hacia las aplicaciones vía Web que agrupan personas con interés en común y donde éstas puedan ser vistas de una forma sencilla y que con respecto a que se asemejen mas en su funcionamiento e interacción a las aplicaciones de escritorio, aunado a que Internet permite la comunicación en tiempo real a un número ilimitado de usuarios. Al momento de desarrollar estas aplicaciones, vía Web, se debe buscar la alta calidad, eficiencia y efectividad del sistema, minimizar los tiempos de respuesta y que la interacción con la aplicación sea lo mas fluida posible. Existe un tipo de aplicación Web llamadas Aplicaciones Web Enriquecidas, o RIA (acrónimo en inglés de Rich Internet Application) que provee una mayor riqueza interactiva que las aplicaciones Web tradicionales, incorporando características muy similares a las que poseen las aplicaciones de escritorio, las cuales proveen una interacción dinámica y rica en elementos de interfaz.

Cada vez son más las instituciones que se unen a la idea de buscar automatizar sus procesos a través de aplicaciones que puedan ser desarrolladas vía Web, que permitan ser visualizadas de una forma sencilla y que puedan tener acceso desde casi cualquier computador. Tal es el caso del F.V.T (Federación Venezolana de Tenis), una institución que requiere de un sistema para la reservación de canchas de Tenis dentro de la Federación. Este sistema debe poder estar al alcance de los diferentes miembros (usuarios-empleados) que conforman la federación.

Por todas las razones mencionadas anteriormente nace el interés en la FVT de desarrollar una aplicación Web, que sea fácil, sencilla y casi inmediata que permita la gestión e información de las canchas alquiladas y las que están libres para su uso basado en los beneficios que ofrecen las aplicaciones RIA (AJAX) y favorecerlos con las comodidades que este enfoque posee. Entre las bondades que obtendría la aplicación está la interacción continua y dinámica entre el usuario y la aplicación a través de la interfaz de usuario; el uso de la API de Persistencia Java permitirá simplificar el desarrollo de la capa de persistencia, realizando el mapeo entre el modelo orientado a objetos y el modelo relacional, procurando mantener el enfoque de desarrollo orientado a objetos y brindar al desarrollador una abstracción de los detalles del

manejo del esquema relacional. La idea es integrar y utilizar una serie de *frameworks* y de esta manera lograr la implementación del Modelo Vista Controlador (MVC).

A continuación se describen los capítulos desarrollados para cumplir los objetivos planteados.

El capítulo 1 se centra en la explicación del enfoque AJAX, además de describir las tecnologías JSF y RichFaces, el *framework* para la persistencia de los datos *Hibernate*. También se definirá la metodología de trabajo PU (Proceso Unificado).
El capítulo 2 describe la adaptación del proceso unificado que es la metodología de trabajo utilizado, junto con sus principales características y distintos aspectos que la conforman como son:

- Fase de Análisis.
- Fase de Diseño.

También se definirá el patrón de diseño MVC (Modelo, Vista, Controlador) junto con la elección de capas y componentes como también los beneficios esperados de la arquitectura. Esto con el propósito de brindar el soporte teórico para el desarrollo de un sistema de alquiler de canchas de tenis.

El capítulo 3 plantean los detalles referentes a la implementación de la solución presentada en este trabajo, las plataformas de hardware y software utilizadas, la integración entre los distintos *frameworks* y librerías utilizados para la solución, así como una descripción de los escenarios encontrados durante el desarrollo de la aplicación.

Para finalizar, se presentan las conclusiones de este trabajo, los resultados obtenidos, así como las bondades y limitaciones de la solución planteada, a fin de cimentar las bases para futuras investigaciones y dar testimonio de la evolución en el desarrollo de aplicaciones Web en el entorno de desarrollo Java.

Planteamiento del Problema

En este punto se conocerán los objetivos tanto generales como objetivos específicos que se desean alcanzar y el problema que se desea atacar con este Trabajo especial de grado los cuales son los siguientes:

Problema

Actualmente en la FVT no se lleva un control adecuado y automatizado de la generación de alquiler, lo cual dificulta los procesos de visualización de información y análisis de gestión. Se debe pasar por dos procesos para reservar las canchas y esto dificulta la rapidez para atender a los clientes, lo cual es ineficiente y fastidioso para el manejo de los usuarios.

El sistema actual no cuenta con una estructura que permita un fácil manejo para la reservación y pago de la cancha, ya que hay que estar de manera manual colocando la cancha a alquilar y la hora.

Objetivo General

Análisis y diseño de un sistema de información para el control y reservación de las canchas de tenis de la FVT utilizando las tecnologías para el desarrollo de aplicaciones Web enriquecidas.

Objetivos Específicos

- Levantamiento de información.
- Análisis y diseño.
- Incorporar arquitectura basada en el Modelo Vista Controlador (MVC).
- Aplicación del esquema de programación AJAX, para brindar una interacción mas fluida con la aplicación.
- Implementación de la capa de presentación haciendo uso de RichFaces.
- Desarrollo de la capa de lógica de negocio, utilizando el framework Spring DAO para lograr la integración exitosa de todos los componentes.
- Desarrollo de la capa de persistencia haciendo uso de Hibernate.
- Pruebas: implementación de cada uno de los escenarios, hasta completar las

funcionalidades del sistema.

- Despliegue: fase de transición la cual consistió en asegurar la aceptación del producto por los usuarios finales, ajustar los errores y defectos encontrados para asegurar que el producto cumpla con los requerimientos del usuario.

Capítulo 1. Antecedentes tecnológicos y metodológicos

Durante mucho tiempo ha existido una brecha entre la experiencia que brindan al usuario las aplicaciones de escritorio y la experiencia que éste obtiene con las aplicaciones Web tradicionales, en cuanto a la mayor riqueza de interacción y rapidez de respuesta de las primeras; algo que parecía estar fuera del alcance de la Web.

Mientras que las aplicaciones de escritorio ofrecen al usuario una gran riqueza en cuanto a interfaz e interacción, y una respuesta casi instantánea ante sus acciones, las aplicaciones Web tradicionales ofrecen una interacción lenta, donde el usuario normalmente, después de cada acción, debe esperar unos segundos por una respuesta, teniendo así un tiempo ocioso. Esta diferencia, básicamente, se debe a la manera en que trabaja el protocolo que hace posible la Web: el protocolo HTTP.

El esquema de funcionamiento que siguen las aplicaciones Web tradicionales se resume de la siguiente manera: para la mayoría de las acciones del usuario se genera una petición HTTP, la cual viaja al servidor; el servidor, al recibir esta petición, y según los parámetros indicados en la misma, realiza algún procesamiento específico para generar la respuesta correcta, la cual es enviada al cliente y es desplegada e interpretada por el navegador.

En la Figura 1 se puede apreciar, gráficamente, el esquema de interacción que siguen las aplicaciones Web tradicionales.

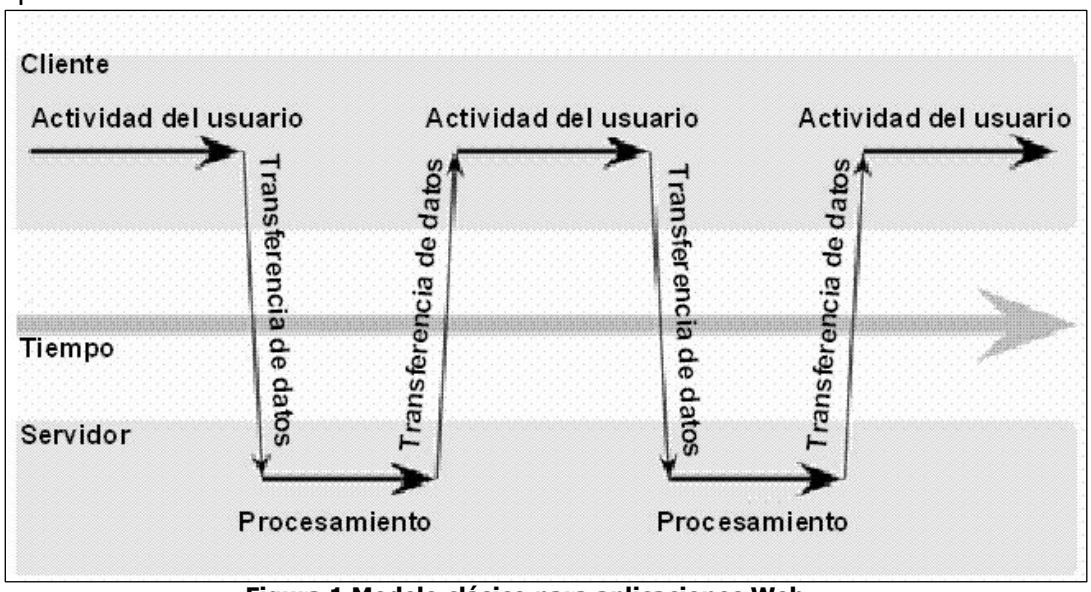


Figura 1 Modelo clásico para aplicaciones Web

Una limitación de este modelo es que mientras la petición viaja hacia el servidor, y éste la procesa, la interacción se detiene; es decir, el usuario se encuentra en espera de los resultados que arrojará su petición, lo cual, entre otros aspectos, produce la brecha que se mencionó anteriormente entre las aplicaciones de escritorio y las aplicaciones Web.

Sin embargo, esta brecha se ha ido cerrando cada vez más gracias a un nuevo esquema de desarrollo, denominado Aplicaciones Web Enriquecidas.

1.1. Aplicaciones de Internet Enriquecidas

La siguiente sección fue extraída del trabajo especial de grado *Reingeniería de una aplicación para el seguimiento de hojas de tiempo utilizando el enfoque AJAX y la tecnología JPA*, la cual aborda detalladamente el tema de las aplicaciones RIA.

RIA (*Rich Internet Application,* o Aplicación de Internet Enriquecida) es un tipo de aplicación Web que provee una mayor riqueza interactiva que las aplicaciones Web tradicionales, incorporando características muy similares a las que poseen las aplicaciones de escritorio, las cuales proveen una interacción dinámica y rica en elementos de interfaz.

El término *Rich Internet Application* fue introducido en marzo de 2002 por Macromedia sin embargo, este mismo concepto ya había sido manejado bajo otros nombres [1]:

- Remote Scripting, por Microsoft, en 1998
- X Internet, por Forrester Research, en octubre de 2000
- Rich Web Clients
- Rich Web Application

Como se mencionó anteriormente, el modelo tradicional de aplicaciones Web tiene una serie de limitaciones, como la poca capacidad multimedia que posee y la recarga continua de páginas. Todo esto se debe a que el cliente en las aplicaciones Web tradicionales sólo se limita a desplegar el contenido HTML. En cambio, las RIAs incorporan un motor como una nueva capa del lado del cliente, que sirve como intermediaria entre la interacción del cliente con el servidor, y tiene la responsabilidad de realizar los cambios sobre la interfaz de usuario.

Algunos de los beneficios que provee el uso de este entorno son los siguientes:

- Se aprovecha la capacidad de procesamiento del lado del cliente. El motor del lado del cliente puede tener la capacidad de realizar actividades solicitadas por el usuario sin la necesidad de realizar una petición al servidor. Estas actividades podrían ser cálculos matemáticos, cambios en la interfaz de usuario (reposicionar objetos), etc.
- El motor del lado del cliente puede interactuar con el servidor asincrónicamente; por ejemplo, el motor podría solicitar datos al servidor para futuras peticiones del usuario, mejorando así el tiempo de respuesta para dichas peticiones.
- Las RIAs no necesitan hacer recargas continuas de toda la interfaz de usuario; en vista que el cliente es más sofisticado, es posible recargar la sección específica de la página que presente algún cambio.

A pesar de los múltiples beneficios que provee este entorno, las aplicaciones RIA presentan algunas limitaciones:

- Debido a que las RIAs se ejecutan en un sandbox, éstas tienen acceso restringido a los recursos del sistema. Una aplicación podría necesitar algún tipo de recurso particular, por ejemplo, tener acceso al sistema de archivos del cliente; en ciertas RIAs este acceso puede estar restringido por el sandbox, por lo tanto, su utilización para el desarrollo de esta aplicación no sería viable ya que no se podría satisfacer dicho requerimiento.
- Las RIAs son dependientes de un componente tecnológico para su correcto funcionamiento, el cual en muchos casos es el soporte de JavaScript o el uso de algún plugin específico. En algunos casos también son dependientes de la plataforma tecnológica (por ejemplo, el navegador).
- Según la compañía IBM [2], existe una serie de aspectos que son necesarios considerar a la hora de escoger una tecnología o un enfoque para desarrollar RIAs. A continuación se describe cada uno de estos aspectos:

Funcionamiento

Los usuarios esperan que el navegador continúe trabajando de la misma forma como si se estuviera interactuando con una aplicación Web tradicional. Esto quiere decir que el usuario espera que los botones atrás e historial del navegador mantengan el comportamiento que presentan en las aplicaciones Web tradicionales, así como

también que las teclas de acceso rápido (cortar y pegar, buscar contenido en la página, entre otros) se puedan utilizar en la misma.

Reducción de la recarga continua de páginas

Algunas solicitudes por parte del usuario podrían generar cambios en una pequeña parte de la interfaz de usuario; por consiguiente, la tecnología RIA debe facilitar la disminución del número de recargas de la página, permitiendo restringir los cambios únicamente a los sectores que deban ser modificados en la misma.

Elementos de interfaz de usuario disponibles

Se debe tratar de utilizar los componentes de interfaz de usuario (menús, formularios, entre otros) para darle más comodidad al usuario, así como tratara de manejar eventos del lado del cliente e incorporar elementos que ofrezcan una mejor forma de interacción con el usuario.

Complejidad e interoperabilidad

La tecnología para desarrollar la RIA debe ser fácil de aprender y de usar. También debe tener la capacidad de ínter operar con tecnologías Web existentes.

Seguridad

Que provea características o mecanismos de seguridad que permitan mantener cierto control sobre la aplicación. Por ejemplo, sería conveniente estar al tanto de todas las conexiones que se establecen por medio del cliente, para así conocer la forma en la cual interactúa el motor del lado del cliente con el servidor y evitar así conexiones no generadas por acciones por parte del usuario de la aplicación (intrusiones).

Soporte para los paradigmas básicos de la Web

La tecnología debe soportar los paradigmas básicos existentes en la Web, tales como internacionalización, independencia del dispositivo de acceso, independencia del navegador, entre otros.

Utilidades

Verificar qué utilidades se encuentran disponibles para el desarrollo de RIAs. Estas utilidades pueden presentarse como *plugins* para los IDE de desarrollo, pudiéndose mencionar depuradores, herramientas de asistencia para la codificación, entre otras.

1.1.1. Tecnologías Utilizadas por las RIAs

Hasta el momento se han mencionado los beneficios y limitaciones que presenta el entorno RIA, así como los aspectos que deben tomarse en cuenta al momento de escoger una tecnología o un enfoque para desarrollar RIAs. A continuación se mencionarán y describirán algunas de estas tecnologías.

Laszlo (*www.openlaszlo.org/*)

Laszlo es una plataforma de código abierto para el desarrollo de RIAs basada en Flash y XML. Esta tecnología utiliza como motor del lado del cliente el *FlashPlayer* 6.x o una versión superior (éste es incorporado al navegador en forma de *plugin)*; como lenguaje de scripting utiliza LZX el cual es un lenguaje orientado a objetos basado en etiquetas que utiliza la sintaxis de JavaScript y XML para generar archivos Flash de forma dinámica.

El intercambio de datos entre el motor del cliente y el servidor es a través de XML. Los datos transmitidos son parseados en el servidor mediante funciones XPath que es uno de los lenguajes existentes para acceder a los elementos de un archivo XML; posteriormente, el archivo LZX es compilado en el servidor para generar un archivo Flash, que será enviado al cliente para realizar los cambios necesarios en la Interfaz de Usuario.

Mozilla XUL (*www.mozilla.org/projects/xul/*)

XUL (*XML-based User-interface Language*, o lenguaje basado en XML para la Interfaz de Usuario) es un lenguaje de marcado que utiliza la sintaxis de XML para definir componentes de Interfaz de Usuario en los navegadores Netscape y Mozilla. La diferencia entre HTML y XUL es que XUL tiene un conjunto extenso de componentes gráficos como menús, barras de herramientas, cajas de texto, entre otros; estos componentes gráficos son creados sin la necesidad de desarrollar código JavaScript. El motor del lado del cliente puede ser desarrollado utilizando JavaScript para realizar los cambios en la Interfaz de Usuario haciendo uso del DOM.

El intercambio de datos entre el motor del cliente y el servidor se realiza mediante XML.

XForms (*www.w3.org/TR/xforms/*)

Es un nuevo lenguaje de marcado para formularios Web, diseñado para ser el sustituto de los formularios tradicionales HTML. Este lenguaje permite a los desarrolladores de formularios Web distinguir entre el propósito del formulario y su presentación, lo que ofrece una serie de ventajas en términos de:

- Reutilización: Los módulos XForms pueden reutilizarse independientemente de los datos que recogen.
- Independencia de Dispositivo: Gracias a que los controles de la interfaz de usuario son abstractos y sólo se indican sus características genéricas, es posible el despliegue de estos formularios en diferentes dispositivos.

XForms provee todas las funcionalidades de los formularios HTML y además permite:

- Comprobar automáticamente los datos mientras el usuario los introduce.
- Indicar que ciertos campos son obligatorios y que el formulario no podrá ser enviado sin esta información.
- Enviar formularios de datos como XML, ya que XForms esta basado en XML.
- Enviar el mismo formulario a diferentes servidores (por ejemplo, la búsqueda de una palabra se envía a diferentes motores de búsqueda).
- Guardar y restaurar valores en y desde un archivo (por ejemplo un archivo XML).
- Obtener los datos iniciales para un formulario a partir de un archivo externo.
- Forzar valores para que cumplan determinado criterio; por ejemplo, que los valores estén comprendidos en un rango determinado.
- Combinar tecnologías XML existentes (XML Events, XPath, entre otras).
- Facilitar la creación de formularios complejos.

Como motor del lado del cliente XForms necesita la instalación del *plugin* FormsPlayer. Usando XForms también es posible incorporar el uso de JavaScript y DOM para lograr que las aplicaciones establezcan comunicaciones con el servidor de manera asíncrona.

Adobe Flex (*www.adobe.com/es/products/flex/)*

Adobe Flex es una tecnología basada en Flash para la creación de páginas Web animadas e interactivas. De igual forma que Laszlo, los archivos Flash son generados en el servidor y enviados posteriormente al cliente para que sean mostrados. Adobe Flex utiliza como motor del lado del cliente el reproductor Flash 6.x o una versión superior (éste es incorporado al navegador en forma de *plugin*).

Los componentes de la Interfaz de Usaurio son definidos en un lenguaje basado en sintaxis XML llamado MXML, que provee un conjunto amplio de librerías para definir componentes visuales.

Un lenguaje de scripting llamado *ActionScript 2* es embebido en MXML para manejar eventos de usuario o eventos del sistema. Éste es un lenguaje orientado a objetos, similar a JavaScript. Al igual que XForms, Flex también puede separar presentación, modelo de datos y servicios de datos (similar al patron Modelo-Vista-Controlador).

Con Flex, todas las peticiones son enviadas en forma de XML al servidor; estas peticiones son resueltas por el compilador de Flex y generan un archivo SWF, el cual será enviado al cliente.

AJAX

Es uno de los enfoques para desarrollo de aplicaciones web enriquecidas más utilizado en la actualidad. En el siguiente punto se hará un estudio más detallado para conocer en que consiste, su funcionamiento, ventajas, desventajas y características que presenta el mismo.

1.2. AJAX

AJAX (acrónimo para *Asynchronous JavaScript And XML.* JavaScript y XML Asíncronos) es un enfoque de desarrollo basado en un conjunto de tecnologías ya existentes, agrupadas para presentar información e interactuar dinámicamente, de manera asíncrona, con un servidor Web [3].

Entre las tecnologías que agrupa AJAX se destacan las siguientes como las principales:

- HTML y CSS: para la presentación, estructuración y formato del contenido.
- DOM (Document Object Model): Con el modelo de objetos del documento se logra obtener la estructura del documento HTML. Utilizando esta estructura se pueden agregar, eliminar y modificar, de manera dinámica, elementos de la página mediante el uso de la tecnología JavaScript.

- XML: Para el intercambio de datos entre el cliente (navegador Web) y el servidor.
- JavaScript: Mediante esta tecnología del lado del cliente se realizan las peticiones de manera asíncrona y, junto con el manejo del DOM, se logra la interacción dinámica con el usuario.

Con AJAX se busca entonces proveer de una mayor riqueza de interacción entre el usuario y la aplicación en comparación con la que proveen las aplicaciones Web tradicionales. Este enfoque cambia el esquema tradicional de interacción con las aplicaciones Web, procurando una mayor velocidad de respuesta hacia el usuario, permitiéndole mantener la interacción con la aplicación inclusive durante tiempos de procesamiento, con elementos de interfaz más dinámicos e interactivos.

El enfoque AJAX permite el envío de peticiones al servidor en segundo plano, es decir, sin interrumpir la interacción entre el usuario y la aplicación. Esto disminuye el típico tiempo de espera entre peticiones y permite mantener la continuidad en la interacción entre el usuario y la aplicación, lo que le permite al usuario realizar acciones como la modificación de campos de un formulario, despliegue de menús, visualización de datos, entre otros.

Luego de enviar una petición y obtener la información requerida (respuesta) del servidor, las aplicaciones AJAX tienen la capacidad de modificar la vista (estructura de la página) dinámicamente sin necesidad de solicitar una página distinta. Esta información obtenida del servidor está constituida típicamente de datos en formato XML o texto plano, no es necesario recibir en cada petición un documento HTML entero, ya que la aplicación AJAX utiliza estos datos para modificar la página sin necesidad de cambiarla por una nueva.

A continuación se profundiza en el contraste entre las aplicaciones AJAX y las aplicaciones Web tradicionales.

1.2.1. Contraste entre esquema clásico de aplicaciones Web y AJAX

Las aplicaciones Web tradicionales basan su esquema de interacción en el funcionamiento intrínseco del protocolo HTTP; el protocolo HTTP está basado en el paradigma petición-respuesta, donde participan dos actores principales, el cliente y el servidor [3].

El cliente envía peticiones de recursos al servidor, descartando la página desplegada actualmente. El servidor recibe la petición y la procesa, respondiendo seguidamente con una página HTML nueva, la cual es recibida y desplegada por el navegador. Este proceso genera una interrupción en la interacción entre el usuario y la aplicación en cada petición.

A diferencia de las aplicaciones Web clásicas, las aplicaciones AJAX pueden enviar peticiones al servidor sin interrumpir la interacción, manteniendo la página actual en el navegador de tal manera que le permita al usuario seguir interactuando con la aplicación. Esto es posible gracias al uso de peticiones en segundo plano (peticiones asíncronas).

En el esquema clásico de aplicaciones Web, las páginas de respuesta enviadas por el servidor posiblemente contienen pocos cambios respecto a la página anterior, lo que puede producir una sobrecarga innecesaria entre cada petición. Con el uso de peticiones asíncronas, las aplicaciones AJAX pueden solicitar al servidor Web únicamente la información que represente un cambio en la página, sin solicitar datos que ya se encuentren en el cliente (como imágenes, encabezados, menús, etc.). Está característica permite disminuir considerablemente el impacto sobre la interacción del usuario con la aplicación que comúnmente se presenta en las aplicaciones Web tradicionales al momento de enviar peticiones al servidor.

Al obtener como respuesta del servidor los datos que representen un cambio en la aplicación, el cliente puede modificar la vista (estructura de la página) dinámicamente con la información obtenida, sin necesidad de solicitar una página distinta. Gracias a esta característica, AJAX permite un uso más eficiente del ancho de banda, ya que sólo se transmite del servidor al cliente la información necesaria y no páginas completas con información que ya reside en el cliente, como encabezados y pie de páginas, gráficos e imágenes, etc.

Las aplicaciones AJAX incorporan un componente adicional del lado del cliente denominado motor AJAX. El motor AJAX es un componente constituido principalmente por código JavaScript el cual se encarga de todo el procesamiento del lado del cliente y sirve como intermediario entre la interfaz de la aplicación y el servidor como se muestra en la Figura 2.

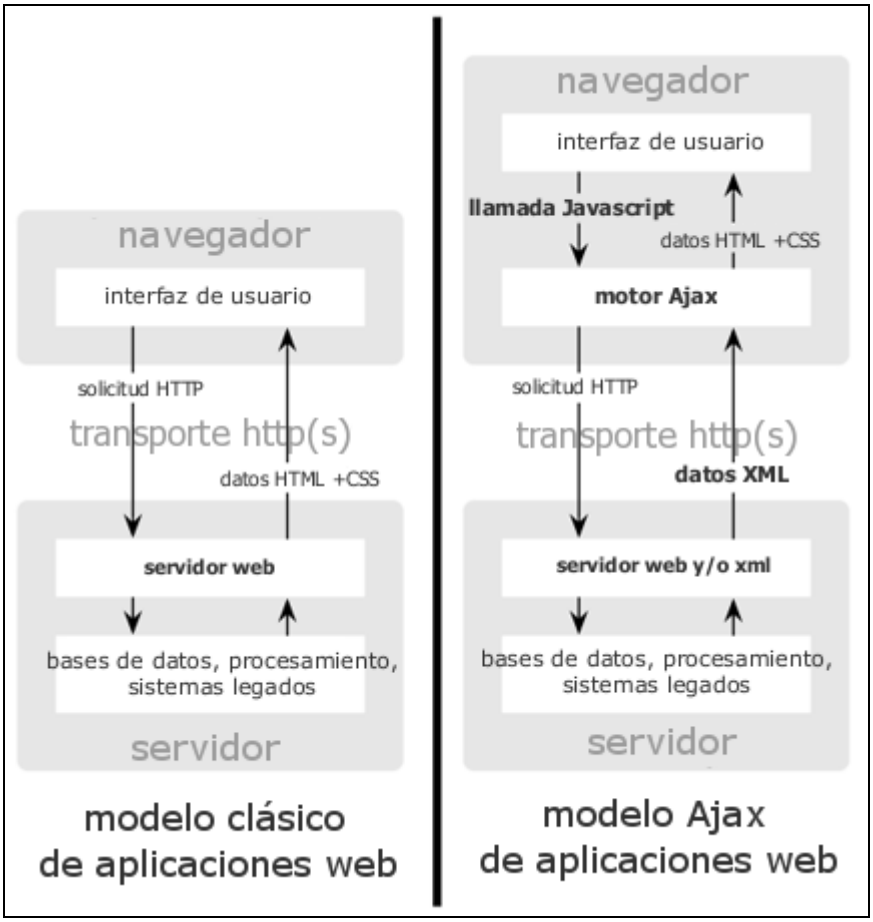


Figura 2 Comparación: Modelo clásico vs. Modelo AJAX

La Figura 2 muestra cómo en el modelo clásico las peticiones al servidor se originan directamente desde la interfaz como consecuencia de las acciones del usuario, en cambio, en el modelo AJAX las acciones del usuario sobre la interfaz son interceptadas por el motor AJAX el cual tiene la posibilidad de darles respuesta directamente o de generar peticiones al servidor (normalmente) en segundo plano.

En el modelo clásico de aplicaciones Web el servidor responde directamente con datos en formato HTML y CSS los cuales son desplegados directamente en el cliente. En el modelo AJAX el servidor responde comúnmente con datos en formato XML o texto plano, los cuales son obtenidos por el motor AJAX y son utilizados por éste para realizar los cambios dinámicos sobre la interfaz.

Luego de comparar el esquema de trabajo de las aplicaciones Web tradicionales y las aplicaciones AJAX, en la siguiente sección se profundiza en el funcionamiento del enfoque AJAX describiendo la base de todas las bondades que brinda y las diferencias con respecto al enfoque clásico de aplicaciones Web.

1.2.2. Funcionamiento

El funcionamiento de la Web sobre el protocolo HTTP explica las razones por las cuales las aplicaciones Web tradicionales presentan interrupciones por cada petición al servidor que se realice y porque éstas presentan una menor riqueza de interacción comparadas con las aplicaciones de escritorio. A continuación se describe cómo funcionan las distintas tecnologías que conforman el enfoque AJAX para incrementar la riqueza de interacción de las aplicaciones Web y obtener un mayor acercamiento al esquema de interacción de las aplicaciones de escritorio. Para esto es importante conocer el papel que juegan las tecnologías que conforman este enfoque [3].

Tecnologías

En la sección se identificaron las tecnologías que componen el enfoque AJAX, sin embargo el uso de cada una por su lado no construye una aplicación AJAX, cada tecnología cumple un papel importante y debe funcionar en conjunto con las otras tecnologías.

- Las Hojas de Estilo en Cascada (CSS) proveen un repositorio de atributos visuales predefinidos que pueden ser asignados a ciertos elementos de la página en cualquier momento.
- El Modelo de Objetos del Documento (DOM) provee una estructura de árbol que representa al documento, donde las distintas etiquetas HTML y los cuerpos de las mismas son los nodos del árbol, que a su vez pueden tener nodos hijos según como se encuentre estructurado el documento. El DOM expone la estructura del documento al motor JavaScript, el cual puede modificar sus elementos dinámicamente según se requiera. Esto se hace utilizando la variable global *document* la cual representa la raíz del árbol.
- JavaScript es el lenguaje utilizado para integrar todas las tecnologías, definiendo la lógica de la presentación (del lado del cliente) y el flujo de trabajo de la aplicación. JavaScript permite:
    - Manipular los elementos del DOM para modificar la interfaz.
    - Obtener la respuesta del servidor y usarla para transformar la vista.
    - Usar las hojas de estilos en cascada para aplicar, dinámicamente, diferente estilos a los elementos de la página.

Hasta ahora se han descrito las tecnologías que conforman el enfoque AJAX; existe un elemento que es el que hace posible la comunicación con el servidor de manera asíncrona, el `XMLHttpRequest`.

1.3. ORM

ORM (Object-Relational Mapping, o Mapeo Objeto-Relacional) es una técnica de programación que permite integrar lenguajes de programación orientados a objetos con sistemas de bases de datos relacionales; es decir, permite enlazar, o mapear, los objetos del lenguaje de programación con las tablas de una base de datos, procurando resolver el problema de la diferencia de impedancia (Amber, 2006).

Las herramientas ORM permiten disminuir el trabajo necesario para mapear objetos con las tablas de la base de datos ya que eliminan la necesidad de pasar los datos entre objetos y tablas a través de sentencias SQL embebidas en el código y, en el caso del lenguaje de programación Java, el uso de la API de acceso a base de datos JDBC, haciendo transparente este proceso sin necesidad de codificación extra. Con una herramienta ORM es necesario definir sólo una vez la forma en que las clases se mapean con las tablas de la base de datos y luego, durante el desarrollo y evolución de la aplicación, este enlace es transparente para el programador.

En los puntos 1.3.1 y 1.3.2 veremos las diferencias entre el mundo de Base de Datos Relacionales y el mundo Orientado a Objetos.

1.3.1. Mundo de Base de Datos Relacionales

- Componentes:
    - Tablas
    - Vista
    - Columnas
    - Tipo de Datos (NUMBER, VARCHAR, DATE)
    - Filas/Registros
    - Índices
    - Secuencias
    - Claves Primarias, Alternas, Foráneas
    - Constraints

- Mejores Prácticas:
    - Calidad de Datos
    - Eliminar redundancia de data
    - Integridad de Data
    - Reutilización de Tablas

1.3.2. Mundo de Programación Orientada a Objetos

- Componentes:
    - Tipos de Datos Primitivos
    - Clases
    - Objetivos
    - Paquetes
    - Atributos
    - Encapsulación
    - Composición
    - Asociación
- Mejores Prácticas:
    - Encapsular información como objetos
    - Reutilización de Clases
    - Herencia
    - Polimorfismo

Base de Datos Relacional	Programación Orientada a Objetos
Tablas, Vistas	Clases
Columnas	Atributos
Tipos de Datos	Tipos de Datos Primitivos, String, Date, Calendar
Filas	Objetos
Índices	
Secuencias	
Claves Primarias, Alternas, Foráneas	
Constraints	
SQL	
Transacciones	
	Paquetes
	Encapsulación
	Composición
	Herencia y Polimorfismo
	Asociación

Figura 3 Comparación: Objeto vs Relacional

1.3.3. Hibernate

Hibernate (http://www.hibernate.org/) es un motor de persistencia de código abierto. Permite mapear un modelo de clases a un modelo relacional sin imponer ningún tipo de restricción en ambos diseños. Cuenta con una amplia documentación, tanto a nivel de libros publicados como disponibles gratuitamente en la Web. A nivel comercial está respaldado por JBoss, que proporciona servicios de soporte, consultoría y formación en el mismo.

Desde su versión 1.0, el motor sigue en evoluciòn, incorporando todas las nuevas ideas que se iban incorporando en este campo.

Hoy en día, en su versión 3.2, ya soporta el estándar EJB 3 (su autor es uno de los principales integrantes del JCP que está definiendo esta especificación) por lo que ya se puede elegir desarrollar aplicaciones empleando EJB 3 -que correrán en cualquier contenedor de EJB's que soporte J2EE 5- o aplicaciones independientes. Esto no solo brinda la inversión de tiempo en Hibernate de cara al futuro, sino que, de ser útil, hace totalmente independientes del J2EE. [15]

Hibernate busca reducir la complejidad y el tiempo requerido por el programador que se presenta al manejar datos de una base de datos manualmente. No necesita la implementación de interfaces complejas, trabaja utilizando un archivo XML donde se describen las clases que se almacenarán en la base de datos y las relaciones entre ellas. Además, provee una API para la realización de consultas y actualizaciones.

En tiempo de ejecución, Hibernate lee el archivo XML de mapeo y genera las clases que contendrán los datos almacenados en la base de datos.

Hibernate fue desarrollado en un principio por un equipo de desarrolladores de Java alrededor del mundo liderados por Gavin King, pero tiempo después su desarrollo fue liderado por el grupo JBoss.

Características

- Es de código abierto.
- Para su ejecución no requiere un contenedor específico.
- Permite la portabilidad de código entre distintos manejadores de base de datos.
- Para el acceso a los datos se emplea el esquema orientado a objetos.
- Permite realizar consultas utilizando SQL o su propio lenguaje de consultas embebido HQL.
- Permite el uso de colecciones Java.
- Soporta tres tipos de estrategias de mapeo: tabla por jerarquía de clases, tabla por clase concreta y tabla por clase (abstracta o concreta).
- Soporta el uso de claves compuestas.

La filosofía de Hibernate es que permite trabajar con cualquier objeto java y cualquier modelo de base de datos, además se tener un idioma orientado a Objetos basado en las clases.

En la figura 4 se muestra como Hibernate busca reducir código a través de archivos de Mapping o Annotations, además se puede observar como se propone un alto desempeño trabajando con una Caché Dual consultando la data solo cuando se necesita.

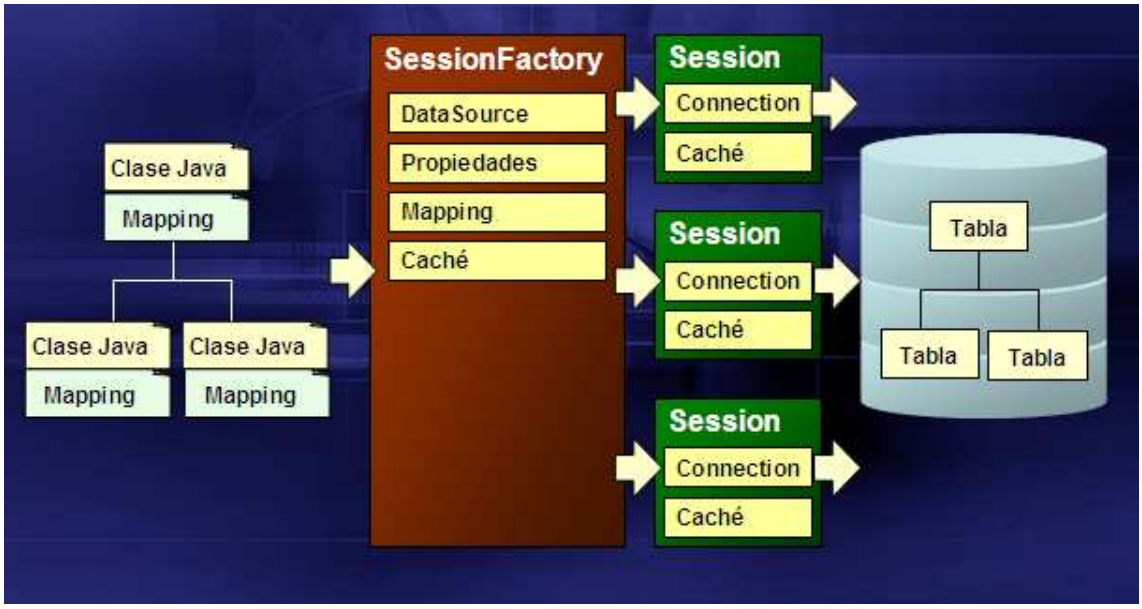


Figura 4 Hibernate

1.4. RichFaces

RichFaces es una biblioteca de componentes para JSF y un avanzado *framework* para la integración de AJAX con facilidad en la capacidad de desarrollo de aplicaciones de negocio. RichFaces componentes vienen listos para su uso, por lo que los desarrolladores pueden ahorrar tiempo aprovechando las características de los componentes y crear aplicaciones Web que proporcionan mejoras en gran medida a los usuarios.

RichFaces aprovecha al máximo los beneficios de JSF *framework* incluyendo, la validación y conversión de instalaciones, junto con la gestión de estática y dinámica los recursos. [15]

RichFaces es una librería de componentes visuales para JSF, escrita en su origen por Exadel y adquirida por Jboss. Además, RichFaces posee un *framework* avanzado para la integración de funcionalidades Ajax en dichos componentes visuales, mediante el soporte de la librería Ajax4JSF.

Características de RichFaces:

- Se integra perfectamente en el ciclo de vida de JSF.
- Incluye funcionalidades Ajax, de modo que nunca vemos el JavaScript y tiene un contenedor Ajax propio.
- Contiene un set de componentes visuales, los más comunes para el desarrollo

de una aplicación web rica (Rich Internet Application), con un número bastante amplio que cubren casi todas nuestras necesidades.

- Es un proyecto open source, activo y con una comunidad también activa.

1.4.1. Ajax4jsf y RichFaces

Son unas bibliotecas open source que se integra totalmente en la arquitectura de JSF y hereda las funcionalidades de sus etiquetas dotándolas con tecnología Ajax de forma limpia y sin añadir código Javascript. Mediante este *framework* podemos variar el ciclo de vida de una petición JSF, recargar determinados componentes de la página sin necesidad de recargarla por completo, realizar peticiones al servidor automáticas, control de cualquier evento de usuario, etc. En definitiva Ajax4jsf y richfaces permite dotar a nuestra aplicación JSF de contenido mucho más profesionales con muy poco esfuerzo.

1.4.2. Funcionamiento del framework

El funcionamiento del *framework* es sencillo. Mediante sus propias etiquetas se generan eventos que envían peticiones al contenedor Ajax. Estos eventos se pueden ejecutar por pulsar un botón, un enlace, una región específica de la pantalla, un cambio de estado de un componente, etc. Esto significa que no nos preocuparemos de crear el código Javascript y el objeto XMLHttpRequest para que envíe la petición al servidor ya que el *framework* lo hará por nosotros.

Algunas Etiquetas de Ajax4jsf y RichFaces

< aj4:support > : Etiqueta que se puede añadir a cualquier otra etiqueta JSF para dotarla de funcionalidad Ajax. Permite al componente generar peticiones asíncronas mediante eventos (onclick, onblur, onchange,...) y actualizar campos de un formulario de forma independiente, sin recargar toda la página.

< aj4:poll > : Realiza cada cierto tiempo una petición al servidor.

< aj4:commandButton > : Botón de envío de formulario similar a de JSF. La principal diferencia es que se puede indicar que únicamente actualice ciertos componentes evitando la recarga de todo el formulario.

< aj4:commandLink > : Comportamiento similar a < aj4:commandButton > pero en un link.

< aj4:htmlCommandLink > : Muy parecida a la etiqueta anterior con pequeñas diferencias en la generación de links y cuando se utilizan etiquetas < f:param >.

< aj4:form > : Similar al < h:form > con la diferencia de que se puede enviar previamente el contenido al contenedor Ajax.

< rich:calendar > : Este componente se utiliza para crear elementos de calendario.

< rich:comboBox > : Este es un componente, que proporciona combo Box editable.

< rich:componentControl > : Este permite llamar a funciones API de JavaScript en los componentes definidos después de los acontecimientos.

< rich:contextMenu > : Este componente se utiliza para la creación de "multileveled context menus" que se activan después de que un usuario define un evento (onmouseover, onclick, etc) sobre cualquier elemento de la página. Demo

< rich:dataFilterSlider > : Un control basado en la acción, Este componente se utiliza para crear un filtrar de los datos de una tabla.

< rich:datascroller > : El componente diseñado para proporcionar la funcionalidad de los cuadros de desplazamiento utilizando Ajax solicitudes.

< rich:columns > : Es un componente, que le permite crear una columnas dinámica.

< rich:columnGroup > : Este componente nos permite combinar las columnas en una fila para organizar.

< rich:dataGrid > : Este componente permite ver los datos como una rejilla que nos deja elegir los datos.

< rich:dataList > : El componente dataList permite prestar los datos de un modo lista. http://livedemo.exadel.com/richfaces-demo/richfaces/dataLists.jsf?c=dataList

1.5.Metodología de Desarrollo

El proceso de desarrollo software es una de las áreas de investigación más importantes para la comunidad de ingeniería del software. Continuamente aparecen nuevos trabajos y propuestas que definen distintas aproximaciones para el proceso de desarrollo de software. Sin embargo, es difícil que satisfagan todas las necesidades de un proyecto específico. Teniendo en cuenta que dos proyectos pueden ser muy diferentes, el proceso aplicado con éxito en uno de ellos puede ser un completo fracaso en el otro. Por eso, el proceso software debe ser adaptado al contexto y características específicas de cada caso. A continuación se explicará más a fondo el

método de desarrollo del Proceso Unificado el cual constituye una metodología estándar ampliamente usada para el análisis, diseño, implementación y documentación de sistemas orientados a objetos.

1.5.1. El Proceso Unificado

El Proceso Unificado de Desarrollo Software o simplemente Proceso Unificado es un marco de desarrollo de software que se caracteriza por estar dirigido por casos de uso, centrado en la arquitectura y por ser iterativo e incremental. El refinamiento más conocido y documentado del Proceso Unificado es el Proceso Unificado de Rational o simplemente RUP. El Proceso Unificado no es simplemente un proceso, sino un marco de trabajo extensible que puede ser adaptado a organizaciones o proyectos específicos. El nombre Proceso Unificado se usa para describir el proceso genérico que incluye aquellos elementos que son comunes a la mayoría de los refinamientos existentes [5].

Este provee un enfoque disciplinado en la asignación de tareas y responsabilidades dentro de una organización de desarrollo. Su meta es asegurar la producción de software de muy alta calidad que satisfaga las necesidades de los usuarios finales, dentro de un calendario y presupuesto predecible.

El Proceso Unificado es un proceso de software genérico que puede ser utilizado para una gran cantidad de tipos de sistemas de software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de competencia y diferentes tamaños de proyectos.

El Proceso Unificado tiene dos dimensiones (Figura 5):

- Un eje horizontal que representa el tiempo y muestra los aspectos del ciclo de vida del proceso a lo largo de su desenvolvimiento.
- Un eje vertical que representa las disciplinas, las cuales agrupan actividades de una manera lógica de acuerdo a su naturaleza.

La primera dimensión representa el aspecto dinámico del proceso conforme se va desarrollando, se expresa en términos de fases, iteraciones e hitos.

La segunda dimensión representa el aspecto estático del proceso: cómo es descrito en términos de componentes del proceso, disciplinas, actividades, flujos de trabajo, artefactos y roles. [5]

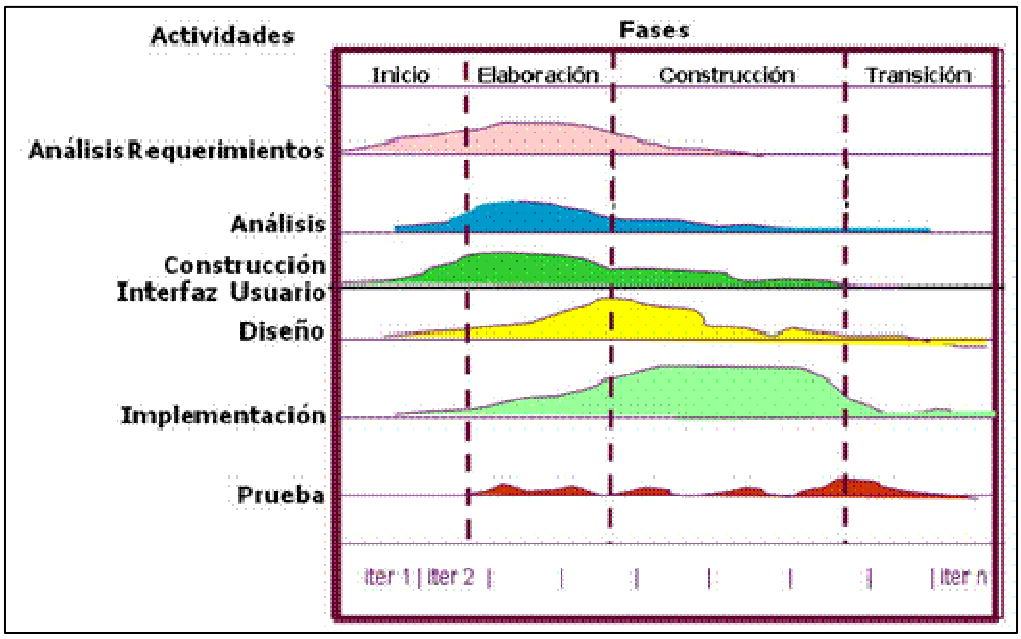


Figura 5 Esfuerzo en actividades según fase del proyecto

El Proceso Unificado se basa en componentes (component-based), lo que significa que el sistema en construcción está hecho de componentes de software interconectados por medio de interfaces bien definidas (well-defined interfaces).

El Proceso Unificado usa el Lenguaje de Modelado Unificado (UML) en la preparación de todos los planos del sistema. De hecho, UML es una parte integral del Proceso Unificado, fueron desarrollados a la par.

Los aspectos distintivos del Proceso Unificado están capturados en tres conceptos clave: dirigido por casos de uso (use-case driven), centrado en la arquitectura (architecture-centric), iterativo e incremental. Esto es lo que hace único al Proceso Unificado.

1.5.2. Características

Las características principales del proceso unificado son las siguientes

Iterativo e Incremental

El Proceso Unificado es un marco de desarrollo iterativo e incremental compuesto de cuatro fases denominadas Inicio, Elaboración, Construcción y Transición. Cada una de estas fases es a su vez dividida en una serie de iteraciones. Estas iteraciones ofrecen como resultado un incremento del producto desarrollado que añade o mejora las funcionalidades del sistema en desarrollo.

Cada una de estas iteraciones se divide a su vez en una serie de disciplinas que recuerdan a las definidas en el ciclo de vida clásico o en cascada: Análisis de requisitos, Diseño, Implementación y Prueba. Aunque todas las iteraciones suelen incluir trabajo en casi todas las disciplinas, el grado de esfuerzo dentro de cada una de ellas varía a lo largo del proyecto. [21]

Dirigido por los casos de uso

En el Proceso Unificado los casos de uso se utilizan para capturar los requisitos funcionales y para definir los contenidos de las iteraciones. La idea es que cada iteración tome un conjunto de casos de uso o escenarios y desarrolle todo el camino a través de las distintas disciplinas: diseño, implementación, prueba, etc. el proceso dirigido por casos de uso es el RUP.

Centrado en la arquitectura

El Proceso Unificado asume que no existe un modelo único que cubra todos los aspectos del sistema. Por dicho motivo existen múltiples modelos y vistas que definen la arquitectura de software de un sistema. La analogía con la construcción es clara, cuando construyes un edificio existen diversos planos que incluyen los distintos servicios del mismo: electricidad, fontanería, etc. [21]

Enfocado en los riesgos

El Proceso Unificado requiere que el equipo del proyecto se centre en identificar los riesgos críticos en una etapa temprana del ciclo de vida. Los resultados de cada iteración, en especial los de la fase de Elaboración, deben ser seleccionados en un orden que asegure que los riesgos principales son considerados primero. [21]

1.5.3. Etapas del Proceso Unificado

El proceso unificado podría dividirse en dos importantes etapas como son las siguientes:

Etapa de ingeniería

Esta etapa agrupa las fases de concepción y de elaboración, lo que básicamente le da por objetivos la conceptualización del sistema y el diseño inicial de la solución del problema.

Se inicia el proceso de administración de los requerimientos con la identificación y especificación de casos de usos, así como el proceso de aseguramiento de la calidad a través de los casos de prueba.

Se identifican los riesgos y se establece su plan de manejo, para determinar en qué orden y en cuántas iteraciones se desarrollarán los artefactos de software que son la solución a los casos de uso.

Se identifican los recursos necesarios, tanto económicos como humanos, acordes con las necesidades del proyecto. Se da comienzo al proceso de estimación y planificación inicial a un nivel macro para todo el proyecto y posteriormente se realiza una estimación detallada de tiempos y recursos de las fases de concepción y elaboración.

- **Fase de concepción**

  Esta fase tiene como propósito definir y acordar el alcance del proyecto con los patrocinadores, identificar los riesgos asociados al proyecto, proponer una visión muy general de la arquitectura de software y producir el plan de las fases y el de iteraciones.

- **Fase de elaboración**

  Los casos de uso seleccionados para desarrollarse en esta fase permiten definir la arquitectura del sistema, se realiza la especificación de los casos de uso seleccionados y el primer análisis del dominio del problema, se diseña la solución preliminar del problema y comienza la ejecución del plan de manejo de riesgos, según las prioridades definidas en él. Al final de la fase se determina la viabilidad de continuar el proyecto y si se decide proseguir, dado que la mayor parte de los riesgos han sido mitigados, se escriben los planes de trabajo de las etapas de construcción y transición y se detalla el plan de trabajo de la primera iteración de la fase de construcción.

Etapa de desarrollo

En esta etapa se realiza un proceso de refinamiento de las estimaciones de tiempos y recursos para las fases de construcción y transición, se define un plan de mantenimiento para los productos entregados en la etapa de ingeniería, se implementan los casos de uso pendientes y se entrega el producto al cliente, garantizando la capacitación y el soporte adecuados.

- **Fase de construcción**

  El propósito de esta fase es completar la funcionalidad del sistema, para ello se deben clarificar los requerimientos pendientes, administrar el cambio de los artefactos construidos, ejecutar el plan de administración de recursos y mejoras en el proceso de desarrollo para el proyecto.

- **Fase de transición**

  El propósito de esta fase es asegurar que el software esté disponible para los usuarios finales, ajustar los errores y defectos encontrados, capacitar a los usuarios y proveer el soporte técnico necesario. Se debe verificar que el producto cumpla con las especificaciones entregadas por las personas involucradas en el proyecto al inicio del mismo

1.5.4. Principios fundamentales del Proceso Unificado

PU está basado en 5 principios clave que son:

Adaptar el proceso

El proceso deberá adaptarse a las características propias del proyecto u organización. El tamaño del mismo, así como su tipo o las regulaciones que lo condicionen, influirán en su diseño específico. También se deberá tener en cuenta el alcance del proyecto.

Balancear prioridades

Los requerimientos de los diversos participantes pueden ser diferentes, contradictorios o disputarse recursos limitados. Debe encontrarse un balance que satisfaga los deseos de todos. Debido a este balanceo se podrán corregir desacuerdos que surjan en el futuro.

Demostrar valor iterativamente

Los proyectos se entregan, aunque sea de modo interno, en etapas iteradas. En cada iteración se analiza la opinión de los inversores, la estabilidad y calidad del producto, y se refina la dirección del proyecto así como también los riesgos involucrados

Elevar el nivel de abstracción

Este principio dominante motiva el uso de conceptos reutilizables tales como patrón de software, lenguajes, marcos de referencia (*frameworks*) por nombrar algunos. Esto evita que los ingenieros de software vayan directamente de los requisitos a la codificación de software a la medida del cliente. Éstas se pueden acompañar por las representaciones visuales de la arquitectura, por ejemplo con el lenguaje UML.

1.5.5. Adaptación al Contexto del Proceso Unificado

El Proceso Unificado no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada proyecto, por ende este se puede adaptar, a un tipo de proyecto específico.

El uso de diagramas para apoyar el diseño de la aplicación así como el uso de elementos externos que den soporte a un mejor entendimiento del sistema a desarrollar queda a conveniencia del grupo de desarrolladores.

También es importante presentar los prototipos de interfaces gráficas de usuario diseñadas para la aplicación final. Presentar prototipos de interfaces de usuario que se negociaron con el cliente como candidatos a ser incluidos hasta la segunda iteración de la fase de construcción.

Una adaptación particular de esta metodología es llamada **Proceso Unificado Ágil** la cual será utilizada para el desarrollo del sistema que constituirá este Trabajo Especial de Grado y será explicada con mayor detalles en capitulo 2 Análisis y diseño.

1.5.6. Proceso Unificado ágil

El Proceso Unificado Ágil es una versión simplificada del UP, la cual describe en una forma simple, fácil de entender y brinda un enfoque de desarrollo de software utilizando técnicas ágiles y conceptos del UP.

Esta metodología describe de una manera simple y sencilla el desarrollo de software utilizando técnicas y conceptos ágiles conservando características primordiales de UP. Plantea un proceso de desarrollo de software para la construcción de sistemas orientados a objetos, el cual fomenta buenas prácticas, destacando el desarrollo

iterativo. En este enfoque, el desarrollo se organiza en una serie de mini-proyectos cortos, de duración fija (por ejemplo, cuatro semanas) llamados iteraciones; el resultado de cada uno es un sistema que puede ser probado, integrado y ejecutable. Cada iteración incluye sus propias disciplinas, por ejemplo análisis, diseño, implementación y pruebas, organizadas en cuatro fases: inicio, elaboración, construcción y transición. Siguiendo las consideraciones de Modelado Ágil (Agile Modeling) (Ambler, 2001), los artefactos a producir, como modelos, diagramas y documentos, dependen de las necesidades del equipo de desarrollo. La frase que define que esta metodología es "apenas lo suficientemente bueno utilizada para referirse a los modelos y documentos asociados al proyecto.

Entre las disciplinas que se destacan en esta metodología tenemos:

- Requerimientos: Entender el negocio de la organización, el problema de dominio que se abordan en el proyecto.
- Análisis y diseño: determinar una solución viable para resolver el problema de dominio, en la que se identifican las tecnologías a utilizar y la realización los diagramas.
- Implementación: Transformar el modelo(s) en código ejecutable y realizar un nivel básico de pruebas individuales.
- Prueba: Realizar una evaluación objetiva para garantizar la calidad. Esto incluye la búsqueda de defectos, validar que el sistema funciona tal como está establecido, y verificar que se cumplan los requisitos.
- Despliegue: Realizar un plan para la presentación del sistema y ejecutarlo para hacer que el sistema se encuentre a disposición de los usuarios finales.

Capítulo 2. Análisis y diseño

En este capítulo se presenta un análisis sobre el proceso de alquiler de canchas de tenis, proponiendo una solución para el desarrollo de una aplicación Web enriquecida. Dicha solución está basada en el uso del enfoque de desarrollo AJAX, que es integrado por el *Framework* RichFaces. Mediante este *framework* podemos recargar determinados componentes de la página sin necesidad de recargarla por completo, realizar peticiones al servidor automáticas, control de cualquier

evento de usuario, etc. En definitiva Ajax4jsf y richfaces permite dotar a nuestra aplicación JSF de contenido mucho más profesionales con muy poco esfuerzo. También el uso de Hibernate para la persistencia de los datos con la intención de obtener una mayor riqueza en la interacción del usuario con la aplicación.

En la sección 2.1 se plantea la metodología utilizada para el desarrollo del sistema, y cuales son las fases de desarrollo que se llevaran a cabo, en la sección 2.2 se realiza la captura de los requerimientos para la solución a desarrollar, proporcionando una visión global del escenario. La sección 2.3 se dedica al análisis de los requerimientos capturados para cubrir el escenario planteado. Para culminar el capítulo, la sección 2.4 cubre lo referente al diseño de la solución, base fundamental para la implementación.

2.1. Metodología utilizada

El objetivo de esta sección consiste en definir la metodología utilizada para el desarrollo e implementación de El Sistema de Alquiler de Canchas de tenis.

2.1.1. Proceso Unificado Ágil

En el proceso unificado Ágil se plantea que dado el tipo de aplicación, se puede instanciar esta metodología para ajustar el proceso a las exigencias de los desarrolladores. En este trabajo se realiza los diferentes modelos del análisis de requerimientos, a partir de estos se desarrolló el prototipo de interfaz y posteriormente, se procedió a implementar cada uno de los escenarios, adaptándose en esta etapa a un proceso de desarrollo ágil. Se establecieron las siguientes etapas [4]:

Etapa de Ingeniería, se inició el proceso de captura de los requerimientos para el sistema de Alquiler de Canchas de tenis, una etapa de análisis donde se identificaron los Actores y los Casos de Uso. Se realizó un refinamiento de los casos de uso principales y se desarrolló el Modelo de Datos a ser utilizado, seguido de una etapa de diseño en la que se identificaron las tecnologías a utilizar y se realizaron los diagramas de componentes y de despliegue, determinando luego en cuál orden y en cuántas iteraciones se desarrollaron los artefactos de software que son la solución de los Casos de Uso.

Etapa de Desarrollo, según el plan establecido en la etapa anterior se empezó con la implementación de cada uno de los escenarios, hasta completar las funcionalidades del sistema (**Prueba**). Por último, se comenzó con una fase de transición la cual consistió

en asegurar la aceptación del producto por los usuarios finales, ajustar los errores y defectos encontrados para asegurar que el producto cumpla con los requerimientos del usuario (**Despliegue**).

2.2. Captura de requerimientos

La F.V.T tiene la necesidad de llevar un registro de las canchas alquiladas y de las personas que poseen las mismas, teniendo en cuenta el costo por reservación y el descuento que se le hacen a ciertos afiliados. Este proceso es realizado por un sistema ya viejo que dificulta la rapidez para atender al cliente ya que hay que colocar de manera manual la cancha, el día y la hora de la reservación.

En la búsqueda por automatizar este proceso y haciendo uso de las tecnologías actuales, como la red, las aplicaciones Web y el surgimiento de los nuevos esquemas de interacción que provee el entorno RIA se puede desarrollar una aplicación que solucione este problema.

Dados estos factores y con base en un análisis constituido por una serie de entrevistas con la comunidad y posibles usuarios finales de la aplicación surgen un conjunto de requerimientos, que serán desarrollados bajo la plataforma planteada en la propuesta de TEG.

2.2.1. Requerimientos

Luego de una serie de reuniones con el personal de la F.V.T y un estudio del proceso de creación y aprobación de los planes de evaluación dentro de la federación se hizo un levantamiento de los siguientes requerimientos para el sistema.

- Se requiere de una funcionalidad en cual el personal administrativo pueda acceder a los servicios con cierto privilegios como lo son:
    - Agregar una Cancha nueva al sistema.
    - Modificar los días feriados.
    - Editar los precios de las canchas.
    - Editar el descuento para los afiliados.
- Visualizar por fecha las canchas disponibles.
- Buscar a las personas afiliadas, mostrar precios y calcular el descuento si se da el caso.
- Imprimir la hoja con la reservación de la persona.
- Llevar una Agenda en donde:

    - Se pueda Buscar por fecha las reservaciones.
    - Buscar por persona las reservaciones.
    - Hacer un cierre de caja.
- Opciones como usuario Administrativo:
    - Agregar una Cancha nueva al sistema.
    - Modificar los días feriados.
    - Editar los precios de las canchas.
    - Editar el descuento para los afiliados.

Una vez conocidos los requerimientos que debe cumplir el sistema para llevar a cabo el proceso de creación de planes de evaluación y entrega de notas se pasa a la fase de análisis.

2.3. Fase de Análisis

Luego de plantear el escenario global y las funcionalidades básicas de la solución que se desarrollará, se presenta un análisis más profundo del escenario, planteando los sub-escenarios presentes y cómo se enfocará cada requerimiento. Para realizar este análisis se hará uso del modelo funcional y del modelo estructural que provee el lenguaje de modelado unificado (UML), mediante la realización de los siguientes pasos:

- Construcción de diagramas de Casos de Uso.
- Construcción de Modelo de Datos.

Para la construcción de los diagramas se utilizó la versión 2.0 de UML para aprovechar las nuevas características que esta versión provee [8].

2.3.1. Modelo de Casos de Uso

En esta sección se presentan los casos de uso del Sistema de Alquiler de Canchas, en términos de los puntos de interacción del usuario con la aplicación; para ello se precisarán los actores, es decir, los tipos de usuarios que utilizan el sistema, luego se presenta cada uno de los casos clasificados por niveles y seguido la descripción particular de los casos de uso presentes en el mismo.

Actores

En el diagrama de casos de uso se identifican los siguientes actores:

Usuario-Empleado: usuario encargado de interactuar con todo el sistema de reservación y alquiler de canchas de tenis.

Administrador: usuario que se encarga del manejo de la aplicación, con tareas como agregar cambiar feriados, editar canchas, editar descuentos y editar precios.

Modulo Afiliación: usuario que se encarga de ingresar o afiliar nuevas personas en el sistema.

Diagrama de casos de Uso

En la figura 6 se muestra el diagrama de casos de uso correspondiente al levantamiento de requerimientos realizado para la aplicación que se desea desarrollar.

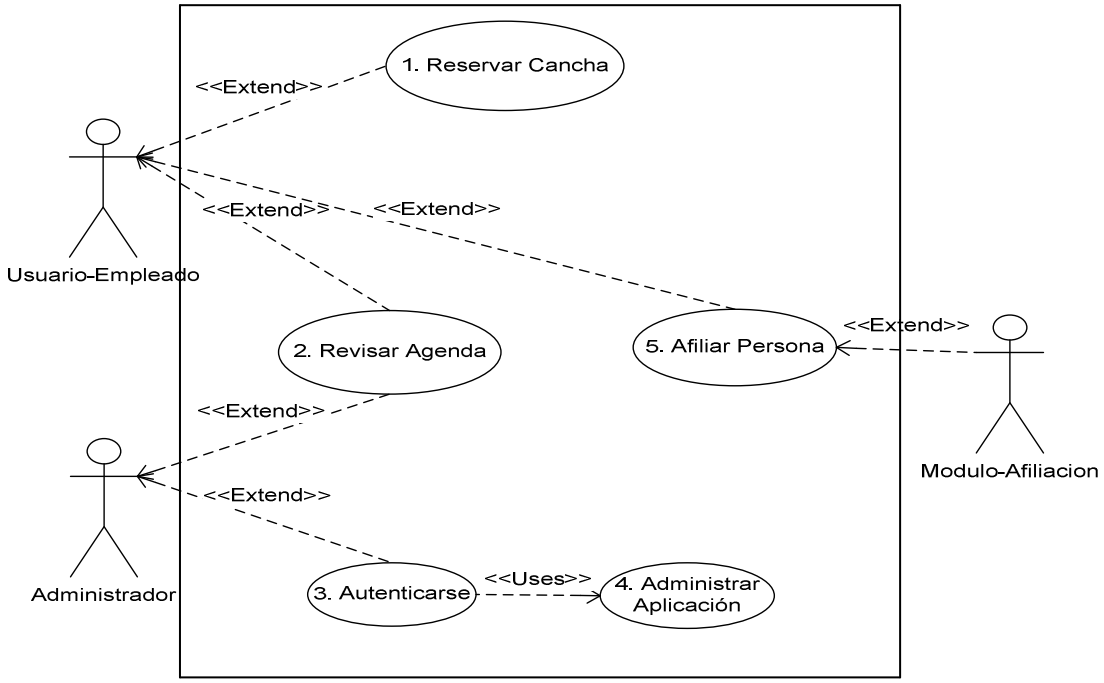


Figura 6 Diagrama de Casos de Uso del Sistema

Descripción de los casos de uso

Especificación del Caso de Uso: Reservar Cancha

1. **IDENTIFICACIÓN DEL CASO DE USO**

Caso de Uso	Reservar Cancha
Actor	Usuario-Empleado y Administrador
Descripción	El usuario Usuario-Empleado o Administrador puede reservar canchas para una persona determinada que se encuentra afiliada al sistema.

Precondiciones	Ninguna
Condición Final Exitosa	Se muestran las opciones para reservar la o las canchas disponibles

2. **IDENTIFICACIÓN DE FLUJOS DE EVENTOS EN EL CASO DE USO**

FLUJO DE EVENTOS BASICOS	FEB-001	Seleccionar Fecha	El Usuario-Empleado o Administrador debe seleccionar la fecha en la cual va a ser reservada la cancha.
	FEB-002	Gestionar Canchas	El usuario Usuario-Empleado o Administrador debe seleccionar la cancha la cual se reservara.
	FEB-003	Gestionar Personas	El usuario Usuario-Empleado o Administrador debe buscar a la persona que va a reservar la cancha en el listado que muestra el sistema.
	FEB-004	Guardar Reservación	El usuario Usuario-Empleado o Administrador guarda la reservación para modificarla o eliminar la misma en algún momento.

FLUJO DE EVENTOS ALTERNATIVO	FEA-001	Afiliar Persona	Si el flujo FEB-003, indica que se quiere afiliar una persona nueva al sistema, se va al modulo Afiliación para ingresar a la persona en la base de datos.

Especificación del Caso de Uso: Revisar Agenda

1. **IDENTIFICACIÓN DEL CASO DE USO**

Caso de Uso	Revisar Agenda
Actor	Usuario-Empleado y Administrador
Descripción	El usuario Usuario-Empleado o Administrador puede revisar la agenda para visualizar las canchas en el sistema y cancelar reservaciones.
Precondiciones	Ninguna
Condición Final Exitosa	Se muestra un menú con las opciones para revisar la Agenda.

2. **IDENTIFICACIÓN DE FLUJOS DE EVENTOS EN EL CASO DE USO**

FLUJO DE EVENTOS BASICOS	FEB-001	Buscar Reservación	Usuario-Empleado o Administrador pueden buscar por fecha o por persona las reservaciones.
	FEB-002	Ver Cierre Caja	Usuario-Empleado o Administrador pueden visualizar el cierre de caja de cualquier día y ver cuanto se totalizo.

FLUJO DE EVENTOS ALTERNATIVO	FEA-001	Cancelar Reservación	Si el flujo FEB-003, indica que se puede eliminar reservaciones que se encuentren en el sistema.

Especificación del Caso de Uso: Afilar Persona

1. **IDENTIFICACIÓN DEL CASO DE USO**

Caso de Uso	Afiliar Persona
Actor	Usuario-Empleado, Administrador y Modulo-Afiliación
Descripción	El usuario Usuario-Empleado, Administrador o Modulo-Afiliación pueden ingresar nuevas personas o afiliarlas en el sistema.
Precondiciones	Ninguna

Condición Final Exitosa	Se muestra un botón que nos lleva directamente al modulo Afiliación y de esta manera poder ingresar a la nueva persona.

2. **IDENTIFICACIÓN DE FLUJOS DE EVENTOS EN EL CASO DE USO**

FLUJO DE EVENTOS BASICOS	FEB-001	Afiliar a Persona	El usuario puede dirigirse al modulo de Afiliación y de esta manera ingresar una nueva persona al sistema para posteriormente reservarle una cancha.

Especificación del Caso de Uso: Autenticarse

1. **IDENTIFICACIÓN DEL CASO DE USO**

Caso de Uso	Autenticarse
Actor	Administrador
Descripción	El usuario debe registrarse en el modulo de Administración para poder realizar tareas Administrativas.
Precondiciones	El Usuario administrador debe estar autenticado
Condición Final Exitosa	Puede ingresar al modulo administrativo

2. **IDENTIFICACIÓN DE FLUJOS DE EVENTOS EN EL CASO DE USO**

FLUJO DE EVENTOS BASICOS	FEB-001	Ingresar login y pasword	El usuario debe colocar correctamente su login y pasword para poder ingresar en el modulo administrativo.

FLUJO DE EVENTOS ALTERNATIVO	FEA-001	Cancelar Reservación	Si el flujo FEB-001, indica que no se autenticado bien, se

			muestra un mensaje de alerta con login y pasword incorrectos.

Especificación del Caso de Uso: Administrar Aplicación

1. **IDENTIFICACIÓN DEL CASO DE USO**

Caso de Uso	Administrar Aplicación
Actor	Administrador
Descripción	El usuario puede administrar todos los componentes principales que constituyen la aplicación.
Precondiciones	El Usuario administrador debe estar autenticado
Condición Final Exitosa	Se agregan. Editan o eliminan los componentes de la aplicación

2. **IDENTIFICACIÓN DE FLUJOS DE EVENTOS EN EL CASO DE USO**

FLUJO DE EVENTOS BASICOS	FEB-001	Cambiar Feriado	El usuario puede editar o cambiar los días feriados para que el sistema los reconozca y aplicar el precio justo para estos días.
	FEB-002	Editar Canchas	El usuario puede editar o cambiar las canchas existentes en el sistema.
	FEB-003	Editar Descuento	El usuario puede editar o cambiar los descuentos que tienen las diferentes canchas por afiliado.
	FEB-004	Editar Precios	El usuario puede editar o cambiar los precios de las canchas.

	FEB-005	Editar Administrador	El usuario puede editar o cambiar sus datos de su perfil como administrador y agregar un nuevo usuario administrador. También puede eliminar su cuenta como administrador.

2.3.2. Modelo de Datos

En esta sección se muestra el modelo de datos de la aplicación y las relaciones de entre cada tabla resultado de la identificación de actores y flujo de información de los casos de uso.

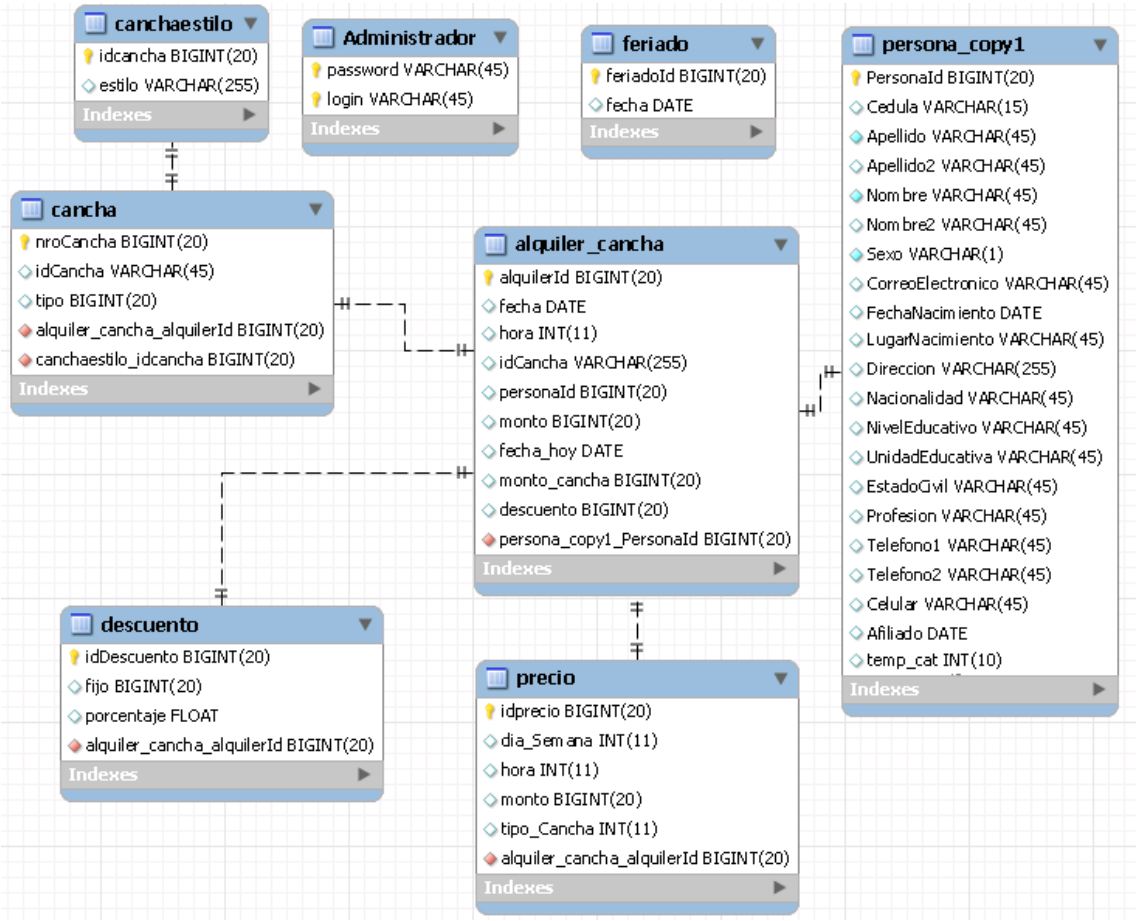


Figura 7 Modelo de Datos del Sistema

En la figura 7 se puede observar las tablas correspondientes a cada uno de los componentes de la aplicación y las cuales van a ser accedidas usando el *framework* de persistencia hibernate, el cual será explicado mas adelante en el capitulo de implementación.

2.3.3. Diccionario de Datos

A continuación se presenta cada una de las tablas que componen la Base de Datos del Sistema de Alquiler de Canchas de Tenis de la Federación Venezolana de Tenis.

Nombre de la tabla:	persona

Nombre del campo	Tipo de dato	Longitud	Null	Clave primaria
PersonaId	BIGINT	20	Not_null	PK
Cédula	VARCHAR	15	Not_null	
Apellido	varchar	45	Not_null	
Apellido2	varchar	45		
Nombre	varchar	45	Not null	
Nombre2	Varchar	45		
Sexo	Varchar	1	Not_null	
CorreoElectronico	varchar	45		
FechaNacimiento	Date			
LugarNacimiento	varchar	1	Not_null	FK
Direción	varchar	255		
Nacionalidad	varchar	45		
NivelEducativo	varchar	45		
UnidadEducativa	Varchar	45		
EstadoCivil	Varchar	45		
Profesión	Varchar	45		
Afiliado	Date			
Pasaporte	Varchar	45		
EntidadEstatalId	Int			FK
CorreoTrabajo	Varchar	45		
CodigoCatSingle	Int		Not_null	FK
CodigoCatDoble	Int		Not_null	FK

Tabla 1 Tabla persona

Nombre de la tabla:		administrador		
Nombre del campo	**Tipo de dato**	**Longitud**	**Null**	**Clave primaria**
adminId	BIGINT	20	Not_null	PK
apellido	varchar	45		

nombre	varchar	45		
login	Varchar	45		
password	Varchar	45		

Tabla 2 Tabla administrador

Nombre de la tabla:		alquiler_cancha		
Nombre del campo	**Tipo de dato**	**Longitud**	**Null**	**Clave primaria**
alquilerId	BIGINT	20	Not_null	PK
fecha	date			
idCancha	varchar	45		
personaId	BIGINT	25		
monto	BIGINT	25		
fecha_hoy	date			
monto_cancha	BIGINT	20		
descuento	BIGINT	20		

Tabla 3 Tabla alquiler_cancha

Nombre de la tabla:		cancha		
Nombre del campo	**Tipo de dato**	**Longitud**	**Null**	**Clave primaria**
nroCancha	BIGINT	20	Not_null	PK
idCancha	varchar	45		
tipo	BIGINT	20		

Tabla 4 Tabla cancha

Nombre de la tabla:		canchaestilo		
Nombre del campo	**Tipo de dato**	**Longitud**	**Null**	**Clave primaria**
idcancha	BIGINT	20	Not_null	PK
estilo	varchar	45		

Tabla 5 Tabla canchaestilo

Nombre de la tabla:		descuento		
Nombre del campo	**Tipo de dato**	**Longitud**	**Null**	**Clave primaria**
idDescuento	BIGINT	20	Not_null	PK
fijo	BIGINT	20		
porcentaje	float			

Tabla 6 Tabla descuento

Nombre de la tabla:		feriado		
Nombre del campo	**Tipo de dato**	**Longitud**	**Null**	**Clave primaria**
feriadoId	BIGINT	20	Not_null	PK
fecha	date			

Tabla 7 Tabla feriado

Nombre de la tabla:		precio		
Nombre del campo	**Tipo de dato**	**Longitud**	**Null**	**Clave primaria**
idprecio	BIGINT	20	Not_null	PK
dia_Semana	INTEGER			
hora	INTEGER			
monto	BIGINT	20		
tipo_Cancha	INTEGER			

Tabla 8 Tabla precio

Nombre de la tabla:		reservación		
Nombre del campo	**Tipo de dato**	**Longitud**	**Null**	**Clave primaria**
idCancha	varchar	200	Not_null	PK
hora	varchar	200		

Tabla 9 Tabla reservación

2.4. Fase de Diseño

Una vez hecho el levantamiento de requerimientos y realizado el análisis correspondiente, que permitió determinar los casos de uso y el flujo de tareas, se logra conocer con mayor profundidad qué necesidades se desean cubrir, qué funcionalidades

debe tener la aplicación y la composición de éstas.

El objetivo principal de la arquitectura es separar, de la forma más limpia posible, las distintas capas de desarrollo, con especial atención a permitir un modelo de dominio limpio y a la facilidad de mantenimiento y evolución de las aplicaciones. El modelo de diseño en 3 capas permite definir el software minimizando el desacoplo de las clases de diseño y por lo tanto ofreciendo una mayor flexibilidad a cambios y correcciones en el software. A continuación una breve explicación de las capas:

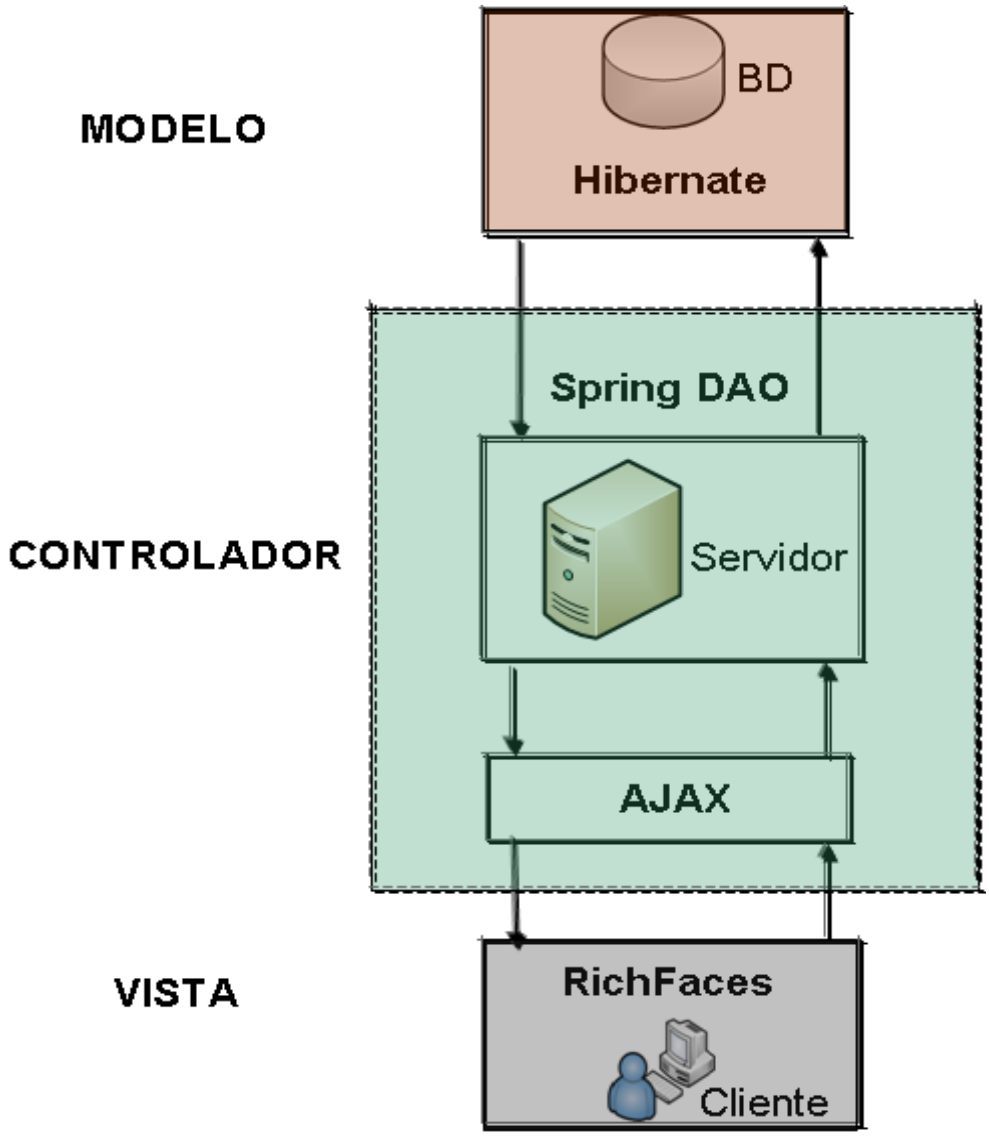


Figura 8 Arquitectura del Sistema

2.4.1. Capa de Presentación (Vista)

Son todas las clases especializadas en la construcción de la parte gráfica e interfaz hacia el usuario. La presentación puede ser desde una página en modo carácter (consola) hasta aplicaciones visuales tanto a nivel de cliente como de interfases Web.

En la actualidad existen herramientas que facilitan la construcción de este tipo de interfases.

2.4.2. Capa de lógica de negocios (Controlador)

Esta capa contiene la lógica de negocio, o por lo menos así lo habla la mayoría de la gente, y significa que aquí están todos los objetos que participan en el dominio del problema que se desea implementar. Esta es la parte medular del desarrollo del software donde se programan las clases en Java. La lógica de datos asegura la integridad de estos y permite derivar nuevos datos; por ejemplo, no permitiendo comprar un número de unidades negativo, calculando si hoy es el cumpleaños del usuario o los totales, impuestos o importes en un carrito de la compra.

2.4.3. Capa de Persistencia (Modelo)

Es un mecanismo que permite el almacenamiento de los datos (persistente). Al igual que la capa de presentación es posible encontrar *frameworks* que facilitan el trabajo de persistencia de la capa lógica, es decir, que los objetos lógicos no tienen que saber como son almacenados ya que el trabajo de transformación y almacenaje de los datos son delegados a la capa de persistencia. Incluso existen aplicaciones que permiten hacer consultas, al igual que SQL, entre objetos. Entre las aplicaciones utilizadas se encuentra Persistencia4 (Desarrollado en el DISC), IBATIS, e Hibernate. Estos *framework* son desarrollados actualmente para Java.

2.5. Elección de capas y componentes

Otros elementos importantes han sido la facilidad del despliegue y el empleo de las mejores tecnologías disponibles en la actualidad, en contraposición al continuismo con opciones que se consideran anticuadas actualmente.

Se desea una arquitectura que permita trabajar en capas y que sirviese tanto para las aplicaciones en la Intranet como en Internet, así como disponer de la flexibilidad necesaria para poder emplear un cliente ligero (navegador Web, Wap) o un cliente pesado (Swing, SWT, etc). Es fundamental no tener que reescribir ningún código y que las capas comunes fuesen reutilizadas sin cambios en ambos casos.

Para lograr esto se eligió el patrón MVC (Modelo-Vista-Controlador) que permite una separación limpia entre las distintas capas de una aplicación.

Para la capa de presentación (la vista) se buscaba un *framework* que proporcionara

una mayor facilidad en la elaboración de páginas, mapeo entre los formularios y sus clases en el servidor, la validación, conversión, gestión de errores, y de ser posible, que facilitase también el incluir componentes complejos (menús, árboles, ajax, etc) de una forma sencilla y sobre todo fácil de mantener. Para esta capa se ha elegido JavaServer Faces y RichFaces.

En la capa de negocio y persistencia, se optó por una solución basada en servicios (no necesariamente servicios web, aunque permitiendo su integración de forma limpia) que trabajaban contra un modelo de dominio limpio. La persistencia de las clases se sustenta en DAOs (Objetos de Acceso a Datos), manteniendo aislada la capa de persistencia de la capa de negocio. Tanto los servicios como los DAOs así como el propio modelo son realmente POJOs (clases simples de Java), con la simplicidad que conllevan y sin dependencias reales con ningún *framework* concreto. Para realizar esta integración se ha elegido Spring DAO.

Para la capa de persistencia se pensó en utilizar alguna herramienta ya existente, que permitiese realizar el mapeo objeto-relacional de una forma cómoda pero potente, sin tener que implementarlo directamente mediante JDBC (Conector de base de datos de Java). Esto último conllevaría, por ejemplo, un esfuerzo importante en un caso de cambio de base de datos (como ha ocurrido), en la gestión de la caché, la utilización de carga perezosa, etc. La herramienta elegida finalmente fue Hibernate.

2.6. Beneficios esperados de la arquitectura diseñada

La primera ventaja se deriva de la modularidad del diseño. Cada una de las partes empleadas (JSF para la vista, Spring para la integración, Hibernate para la persistencia) es intercambiable de forma sencilla y limpia por otras soluciones disponibles. Por ejemplo, para la vista se emplea Java-Server Faces, pero nada impide emplear también una aplicación de escritorio mediante Swing o SWT sin tener que tocar ni una sola línea de código de las restantes capas. Es más, nada impediría que se pudiese disponer de una aplicación con una parte de la capa de presentación en JSF y otra parte, para otro tipo de usuarios, en Swing, ambas funcionando a la vez y compartiendo todo el resto del código (lógica de negocio, persistencia, integración, etc).

De igual forma, si se desean cambiar elementos de la capa de persistencia empleando otro *framework* para el mapeo diferente de Hibernate –o sencillamente no utilizar ninguno- tan sólo serían necesarios cambios en esa capa.

De la misma manera se podrían sustituir cualquiera de las otras capas. El diseño se ha hecho reduciendo al mínimo posible las dependencias entre ellas.

2.7.Diagramas para el diseño de la Aplicación

A continuación se procede a realizar el diagrama de diseño lógico que permite especificar cómo se dará solución a los requerimientos encontrados y cómo se realizarán las distintas tareas que conforman las funcionalidades requeridas, con lo cual se obtiene la arquitectura para la implementación de la aplicación.

Así como Para la comprensión de los diagramas de diseño es necesario conocer que, basándose en esta investigación, se decidió hacer uso de los siguientes *frameworks*:

- Capa de presentación:
    - RichFaces (http://www.jboss.org/richfaces)
- Capa de persistencia:
    - Hibernate (http://www.hibernate.org)
- Capa de persistencia:
    - Spring DAO (http://www.springsource.org/)

Los diagramas se realizaron utilizando el Lenguaje de Modelado Unificado (UML) en su versión 2.0. Para el diseño se modelan los siguientes diagramas:

- Diagrama de componentes
- Diagrama de despliegue

2.7.1. Diagrama de componentes

Antes de realizar la implementación, es de suma importancia planificar desde un alto nivel las partes que conformarán el sistema, para establecer la arquitectura y las dependencias de éste. El diagrama de componentes UML permite modelar los componentes del sistema, para organizarlo en piezas manejables, re-usables y reemplazables.

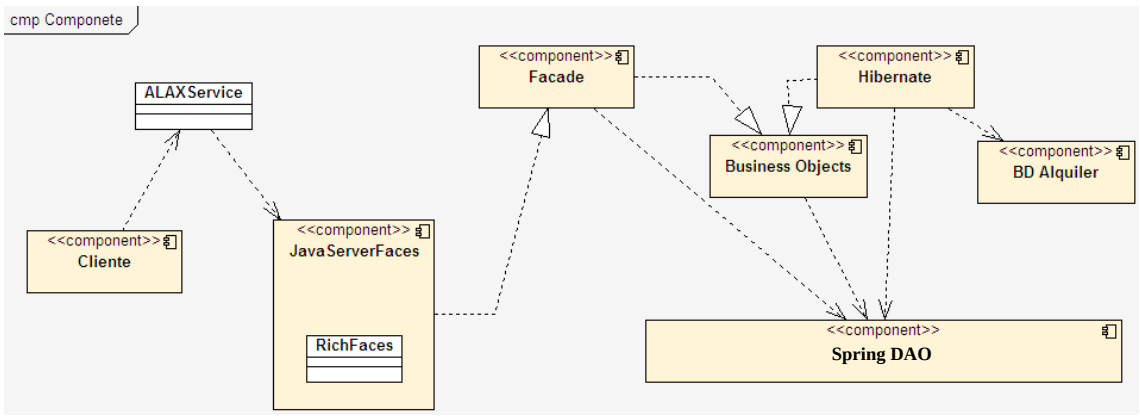


Figura 9 Diagrama de Componentes del Sistema

La figura 9 muestra el diagrama de componentes elaborado para el desarrollo de la aplicación. En este diagrama se observa el uso de la librería RICHFACES como *framework* para el uso de AJAX, así como el uso el *framework* Hibernate para el manejo de persistencia.

Del lado del servidor se encuentra el *framework* Spring (Spring DAO), el cual controla la instanciación y localización de los objetos. Además de los *frameworks* descritos, el diagrama muestra tres patrones de diseño que permiten modularizar la aplicación con base en componentes funcionales:

- **Front Controller:** Provee un punto de acceso centralizado para la manipulación de peticiones por parte de la capa de presentación, y es implementado por el servlet provisto por RICHFACES.
- **Facade:** Provee una capa intermedia (fachada) para los objetos del negocio, separando la lógica de procesamiento de peticiones (Front Controller) de la lógica del negocio. El componente Facade invoca al objeto del negocio correspondiente con el objeto de realizar la acción requerida (lógica del negocio); estos objetos de negocios implementaran el patrón Business Objects.
- **Business Objects:** La idea es proveer una separación entre la lógica y los datos de negocio, usando un modelo de objetos con una lógica sofisticada, validaciones y reglas de negocio bien definidas. Son POJOs (Plain Old Java Objects) que permiten encapsular y manejar los datos de negocio y su comportamiento, con la ayuda de algún mecanismo la persistencia de estos datos.

El diagrama de componentes nos brinda una imagen de la aplicación basada en las distintas piezas que la componen; sin embargo, es de relevancia tener una visión del

sistema, donde se observen los entornos sobre los cuales se soportan estos componentes. Con este fin se modela el diagrama de despliegue, mostrado en la siguiente sección.

2.7.2. Diagrama de despliegue

El diagrama de despliegue que se presenta a continuación (Figura 10), brinda una visión física del sistema, mostrando cómo se encuentra el software con respecto al hardware y como las piezas del sistema se comunican. En este diagrama se muestran tres nodos principales: el cliente, el servidor Web y el servidor de base de datos.

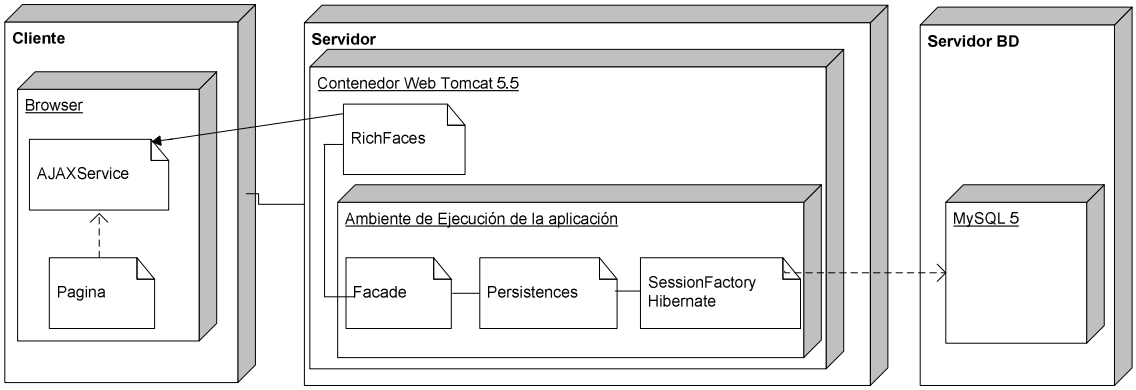


Figura 10 Diagrama de Despliegue del Sistema

Del lado del cliente se presenta el navegador, en el cual residen las páginas de la aplicación y la clase AJAXService, generada por RICHFACES para encapsular las peticiones asíncronas.

Del lado del servidor se utiliza como contenedor Web Apache Tomcat 5.5, un contenedor Web de código abierto desarrollado bajo el auspicio de la Apache Software Foundation. En este contenedor reside el *framework* RICHFACES, las entidades de la aplicación, el componente Facade, los objetos del negocio (Business Objects), y el SessionFactory que provee el *framework* Hibernate.

En el tercer nodo se ejecuta el manejador de base de datos MySQL 5, en el cual residirá la base de datos de la aplicación. Culminado el diseño de la aplicación y la descripción de sus distintos componentes, a continuación el capítulo concerniente a la implementación de la solución, donde se muestra el proceso de desarrollo de los distintos componentes y se materializa todo lo presentado en el diseño de la solución.

Capítulo 3. Implementación

Con base en el análisis y diseño presentado en el capítulo 2 y siguiendo con la metodología propuesta, en este capítulo se presenta la implementación de la solución para los requerimientos planteados.

El capítulo está dividido en 3 secciones que presentan tanto la implementación de la lógica del lado del cliente como la implementación de la lógica del lado del servidor.

La sección 3.1 presenta la plataforma de hardware y software utilizada para el desarrollo. En la sección 3.2 se describe la implementación concerniente al lado del cliente, describiendo los artefactos de software creados en las diferentes iteraciones.

3.1. Plataforma de Hardware y Software

En esta sección se describe la plataforma de hardware y software utilizada durante la implementación de la solución de este trabajo especial de grado.

3.1.1. Plataforma de Hardware

Para el desarrollo de la aplicación se utilizaron dos equipos con la siguiente configuración:

- Procesador Intel Centrino Duo IV 3.2 GHZ
- 1 GB de memoria RAM
- Espacio en disco utilizado para la aplicación y software de desarrollo y ejecución: 1 GB

3.1.2. Plataforma de software

A continuación se describen todos los componentes que conforman la plataforma de software utilizada para el desarrollo de la aplicación de Alquiler de Canchas.

Sistema Operativo

Para el desarrollo de la aplicación se utilizó sobre Windows XP Professional; sin embargo, se han realizado varias pruebas sobre Windows Vista. La aplicación puede ejecutar sobre cualquier sistema operativo que soporte la maquina virtual Java versión

5 y el resto de los componentes, tales como el SMBDR[1], MySQL 5 y el contenedor Web Tomcat 5.5.

Máquina virtual Java

La aplicación fue desarrollada utilizando el lenguaje de programación Java; por lo tanto, es necesario tener una JVM que soporte la aplicación. La versión a utilizar de la JVM debe ser de la 5 en adelante, ya que las anotaciones (utilizadas por el *framework* RichFaces y Hibernate) son soportadas sólo a partir de Java 5. Además de las anotaciones se aprovecharon ciertas bondades de Java 5 como los cambios en la API de Collections, utilizando colecciones restringidas, y la forma de iterar sobre las colecciones.

Contenedor Web

Una pieza fundamental para la ejecución de la aplicación es el contenedor Web, que brindará todos los servicios sobre los que se apoyará la aplicación para su acceso a través de la Web.

Como contenedor Web se utilizó Apache Tomcat en su versión 5.5.

Sistema manejador de base de datos

Como manejador de base de datos se escogió MySQL 5, ya que es un manejador Open Source muy utilizado que cuenta con una amplia documentación, además de su eficiencia y rapidez.

3.2. Implementación del lado del cliente

Esta sección describe los componentes utilizados para la implementación de la lógica del lado del cliente. Para esta implementación se utilizó el *framework* RichFaces. La sección comienza describiendo cómo se realizó la instalación y qué utilidad se dio a este *framework*; seguidamente, se describen las páginas (html) que se desarrollaron, y qué papel juegan en la aplicación.

3.2.1. RichFaces

Por medio de este *framework* se agregaron componentes visuales para JSF incluyendo funcionalidades AJAX mediante un soporte de librería Ajax4JSF.

La versión utilizada fue la 3.1

[1] Siglas para Sistema Manejador de Bases de Datos Relacionales

Instalación

Este *framework* se encuentra disponible en (http://www.jboss.org/richfaces). Para incorporarlo como parte de la aplicación o para añadir un componente visual de RichFaces tenemos que incluir el espacio de nombres correspondiente al nodo raíz de nuestra página xhtml, así crearemos una página index.html incluyendo lo siguiente:

```
01 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
02 <html xmlns="http://www.w3.org/1999/xhtml"
03   xmlns:f="http://java.sun.com/jsf/core"
04   xmlns:h="http://java.sun.com/jsf/html"
05   xmlns:ui="http://java.sun.com/jsf/facelets"
06   xmlns:rich="http://richfaces.org/rich">
07     <f:view>
08     <!-- aquí incluiremos nuestros componentes JSF -->
09     </f:view>
10 </html>
```

Añadiendo el soporte para RichFaces

Configurar el descriptor de despliegue de nuestra aplicación web, deberíamos incluir lo siguiente en el archivo web.xml:

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
03     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
04     xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
   http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
05     <display-name>RichFaces Demo</display-name>
06     <!-- configuración propia de JSF -->
07     <context-param>
08         <param-name>javax.faces.CONFIG_FILES</param-name>
09         <param-value>/WEB-INF/faces-config.xml</param-value>
10     </context-param>
11     <context-param>
12         <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
13         <param-value>server</param-value>
14     </context-param>
15     <servlet>
16         <servlet-name>Faces Servlet</servlet-name>
17         <servlet-class>javax.faces.webapp.FacesServlet</servlet-
   class>
18         <load-on-startup>1</load-on-startup>
19     </servlet>
20     <servlet-mapping>
21         <servlet-name>Faces Servlet</servlet-name>
22         <url-pattern>*.jsf</url-pattern>
23     </servlet-mapping>
24     <!-- sufijo de las páginas que incluyen árboles de componentes
   basados en facelets -->
25     <context-param>
26         <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
27         <param-value>.xhtml</param-value>
28     </context-param>
29     <!-- filtro de Ajax4JSF -->
30     <filter>
31         <display-name>Ajax4jsf Filter</display-name>
32         <filter-name>ajax4jsf</filter-name>
33         <filter-class>org.ajax4jsf.Filter</filter-class>
34     </filter>
35     <filter-mapping>
36         <filter-name>ajax4jsf</filter-name>
37         <servlet-name>Faces Servlet</servlet-name>
38         <dispatcher>REQUEST</dispatcher>
39         <dispatcher>FORWARD</dispatcher>
40         <dispatcher>INCLUDE</dispatcher>
41         <dispatcher>ERROR</dispatcher>
42     </filter-mapping>
43 </web-app>
```

Por último, necesitamos configurar el gestor de vistas para facelets en el archivo faces-config.xml:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:ui="http://java.sun.com/jsf/facelets"
  xmlns:rich="http://richfaces.org/rich">
    <f:view>
    <!-- aquí incluiremos nuestros componentes JSF -->
    </f:view>
</html>
```

Ejemplo Añadir un Calendario:

```
...
  <rich:panel>
    <f:facet name="header">
      <h:outputText value="RichFaces calendar" />
    </f:facet>
    <h:form>
        <h:panelGrid columns="2">
            <h:panelGroup>
                <h:outputLabel value="Fecha" />
            </h:panelGroup>
            <rich:calendar datePattern="d/M/yyyy HH:mm" />
        </h:panelGrid>
    </h:form>
  </rich:panel>
...
```

3.2.2. Prototipo

Gracias al enfoque de desarrollo AJAX una página puede manejar diferentes vistas, dada su capacidad para modificarse dinámicamente y hacer peticiones asíncronas. Este caso se presenta en la solución implementada.

Una vez descrita y analizada la arquitectura del sistema, se diseñó un modelo de prototipo ejecutable basado en el levantamiento de información realizado, permitiendo

definir las clases de la arquitectura completa del sistema.

En esta fase se creó el código fuente necesario para el funcionamiento de la aplicación Web. El producto entregable en esta etapa fue un prototipo I.

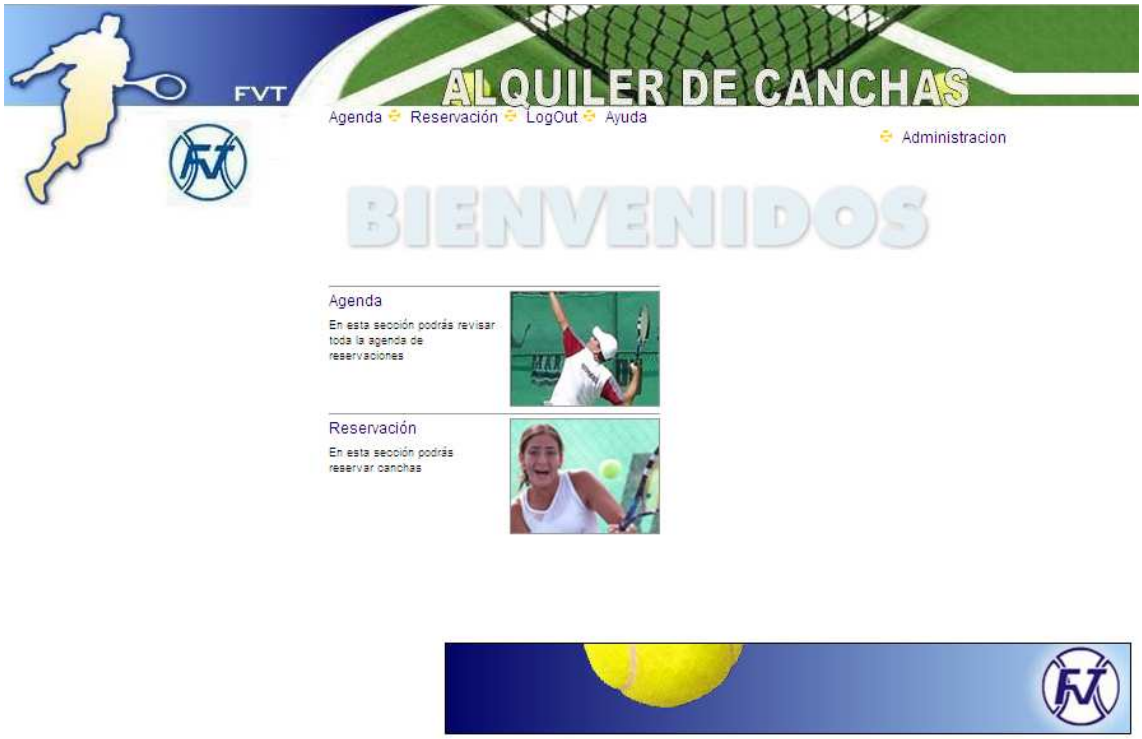
Durante esta fase del desarrollo se elaboraron los formularios de la interfaz gráfica, las clases de conexión a bases de datos, las librerías y funciones requeridas para llevar a cabo las operaciones del sistema, así como el flujo de trabajo para el servicio de transferencia de datos utilizando las herramientas de lenguaje de programación Web Java EE versión 5 con Hibernate + Spring + JSF (Facelets + RichFaces) establecidas a raíz de la elaboración del prototipo. Durante el proceso se establecieron especificaciones, estándares y particularidades de formato para la construcción de rutinas, funciones y programas en general. De igual manera, para mensajes y validaciones implícitas en cada funcionalidad del sistema.

El prototipo fue evaluado y rediseñado de acuerdo a las presentaciones preliminares elaboradas hasta alcanzar los requerimientos del sistema mencionados en la fase de Inicio, para realizar la entrega de un nuevo prototipo operacional.

Cada página Web es un programa aparte que a su vez está compuesto por otros programas que interactúan entre sí para darle funcionalidad a la aplicación. En cada interfaz se agregan las opciones respectivas que le permitirán al usuario escoger las opciones que desee y además acceder al apartado del sistema que crea conveniente en su momento. También la posibilidad de consultas que permite la aplicación.

Artefacto: Inicio.jsf 1ra. Iteración, Caso de Uso 1

En esta página se desarrollaron todos los elementos de interfaz de la sección que le permite al usuario de tipo empleado la reservación y control de las canchas alquiladas. Por un lado RichFaces es utilizado para realizar las peticiones asíncronas y para modificar la vista cuando se ejecute la funcionalidad de alquiler de canchas. Como se mencionó al comienzo de la sección gracias al uso del enfoque AJAX esta misma página puede manejar distintos modos de vista.



Artefacto: Fecha.jsf 2da. Iteración
En esta página se puede escoger la fecha en la cual se va a reservar la o las canchas. Muestra un mensaje de advertencia si se escoge una fecha inválida.


Artefacto: Canchas.jsf 3ra. Iteración

En esta página se pueden visualizar todas las canchas existentes en el sistema, se podrá observar cuales son las que están disponibles para la fecha, nos muestra una numeración para saber en qué paso estamos y cuantos nos falta para reservar la o las canchas, en caso de algún error podemos devolvernos al paso anterior. Nos muestra a un lado cuales son las canchas que se han alquilado con su respectiva hora.

Artefacto: DetallePersona.jsf 4ta. Iteración

En esta página se pueden visualizar todas las personas que están en el sistema que se han previamente afiliado. Hay un formulario de búsqueda para conseguir a las personas por cedula, nombre, apellido, etc. Si se desea agregar un nuevo afiliado existe el botón afiliar que nos lleva al modulo de Afiliación que nos permite agregar una nueva persona al sistema. Para pasar a la siguiente pagina el usuario-empleado debe dar click a cualquier nombre para saber a quien se le hará la reservación.

Artefacto: Monto.jsf 5ta. Iteración

En esta página ya se puede observar las canchas que se reservaron y la persona que las reservó con los diferentes precios y descuento si así lo amerita. Una vez verificada la reservación se confirma para luego quedar registrada en el sistema y poder imprimirla.

Artefacto: ImprimirHoja.jsf 6ta. Iteración

En esta página se puede imprimir la reservación ya confirmada por el usuario-empleado o volver a la página de fecha para así continuar reservando canchas. Cuando se le da al botón imprimir se visualiza a través de JavaScript otra ventana para así no interrumpir el proceso de reservación indicando la impresión de la hoja de reservación.

Artefacto: Agenda.jsf 7ma. Iteración, Caso de Uso 2
En esta página se desarrollaron todos los elementos de interfaz de la sección que le permite al usuario de tipo empleado revisar todas las reservaciones que existen en el sistema, haciendo una búsqueda por fecha o por personas. También existe una opción de cierre de caja que permite llevar un control de lo que se ha hecho por día.

Artefacto: ReservacionFecha.jsf 8va. Iteración

En esta página se puede observar todas las canchas reservadas en una fecha específica y en más detalles se puede ver quién es la persona que tiene reservada esa cancha.

AGENDA

Datos de la Reservación

Detalle	Fecha	Hora Militar	Cancha	Monto	Cancelar Reservación
Mas Detalles	25-may-2009	13	Cancha08	BsF. 40	Cancelar
Mas Detalles	25-may-2009	13	Cancha09	BsF. 40	Cancelar
Mas Detalles	25-may-2009	14	Cancha11	BsF. 40	Cancelar
Mas Detalles	25-may-2009	21	Cancha11	BsF. 45	Cancelar
Mas Detalles	25-may-2009	22	Cancha06	BsF. 45	Cancelar
Mas Detalles	25-may-2009	8	Cancha06	BsF. 40	Cancelar
Mas Detalles	25-may-2009	7	Cancha06	BsF. 40	Cancelar
Mas Detalles	25-may-2009	9	STADIUM	BsF. 0	Cancelar

Artefacto: BuscarPersona.jsf 9ma. Iteración

En esta página se puede observar todas las personas que están en el sistema y una vez buscada la persona que se desee se pueden visualizar todas las canchas que ha reservado en diferentes fechas.

Artefacto: DetalleReservacion.jsf 10ma. Iteración

En esta página se puede observar todas las canchas reservadas por una persona, y también cancelar una reservación (si es el caso) para así habilitar la cancha para otra persona.

Artefacto: CierreCaja.jsf 11va. Iteración

En esta página se puede observar todas las reservaciones que van por día y de esta manera llevar un control de lo que hay en caja.

Artefacto: IngresarNuevaPersona.jsf 12va. Iteración, Caso de Uso 3

En esta página se muestra del modulo Afiliación el formulario para ingresar una nueva persona al sistema.

Artefacto: Admin.jsf 13va. Iteración, Caso de Uso 4
En esta página se muestra el menú de todas las operaciones que se pueden hacer como usuario administrador.


Artefacto: Feriado.jsf 14va. Iteración

En esta página el usuario administrador tiene la opción de agregar feriados o de editarlos y de esta manera se sabrá que precio obtendrá la cancha ese día.

Artefacto: AgregarCanchas.jsf 15va. Iteración

En esta página el usuario administrador puede agregar nuevas canchas al sistema o eliminar una ya existente.

Artefacto: Descuento.jsf 16va. Iteración

En esta página el usuario administrador puede editar los descuentos dependiendo de los días de la semana o feriados.

Descuentos

Horario	Descuento	Editar Fecha
Lunes a Viernes	20	Editar
Sabado, Domingo y Feriados	25	Editar

Artefacto: Precio.jsf 17va. Iteración

En esta página el usuario administrador puede editar los precios de todas las canchas por día o por tipo de cancha.

Precios

Dia de la semana	Cancha	7am	8am	9am	10am	11am	12pm	1pm	2pm	3pm	4pm	5pm	6pm	7pm	8pm	9pm	10pm
Fines y Feriados	Ambas	50	50	50	50	50	50	50	50	50	50	50	0	0	0	0	0
Lunes	Normal	40	40	40	40	40	40	40	40	40	0	0	0	0	0	45	45
Martes	Normal	40	40	40	40	40	40	40	40	40	0	0	0	0	0	45	45
Miercoles	Normal	40	40	40	40	40	40	40	40	40	0	0	0	0	0	45	45
Jueves	Normal	40	40	40	40	40	40	40	40	40	0	0	0	0	0	45	45
Viernes	Normal	40	40	40	40	40	40	40	40	40	0	0	0	45	45	45	45
Lunes	STADIUM	45	45	45	45	45	45	45	45	45	0	0	0	0	0	50	50
Martes	STADIUM	45	45	45	45	45	45	45	45	45	0	0	0	0	0	50	50
Miercoles	STADIUM	45	45	45	45	45	45	45	45	45	0	0	0	0	0	50	50
Jueves	STADIUM	45	45	45	45	45	45	45	45	45	0	0	0	0	0	50	50
Viernes	STADIUM	45	45	45	45	45	45	45	45	45	0	0	0	50	50	50	50

Artefacto: FormularioAdmin.jsf 18va. Iteración
En esta página el usuario administrador editar sus datos en el sistema como cambiar su clave o agregar un nuevo administrador en el sistema.

3.3. Implementación del lado del servidor

La aplicación maneja del lado del servidor varias capas que fueron divididas por paquetes, según su nivel. La ruta principal de los paquetes es com.ges. Así, según su nivel en la arquitectura multicapa y el papel que juegan sus clases, los paquetes definidos son los siguientes:

actions: Contiene la clase que sirve como fachada y que contiene los métodos que son invocados desde el cliente.

dao: Contiene las clases que manejan todos los accesos a los objetos persistentes.

util: Contiene todas las utilidades externas a la lógica de programación del sistema, como el SessionFactory de Hibernate, funciones para cambiar de formato fechas, etc.

beans: Contiene las clases persistentes de la aplicación que son mapeadas a la base de datos utilizando los archivos de mapeo del *framework* de persistencia Hibernate ubicados en el mismo paquete.

A continuación se describen las clases de mayor importancia contenidas en cada paquete.

3.3.1. Paquete actions

Los métodos de mayor importancia se listan en la siguiente tabla.

Método	Descripción
`String registrarSeleccion()`	Se registran las canchas que se están seleccionando para hacer la reservación. Se guardan los diferentes valores (IdCancha, Fecha y Hora) en la clase Reservación
`String buscarCanchasFecha()`	Busca cuales son las canchas que están disponibles para una fecha especifica.
`void` `cargarValores()`	Encargada de pintar el tablero con las canchas del sistema, según los valores obtenidos indica cuales son las canchas que se pueden alquilar según la fecha.
`String mostrarPersona()`	Se encarga de mostrar a las personas dentro de la Base de Datos que son las que pueden reservar una cancha.

String GuardarCancha()	Si en la Federación de Tenis deciden construir una cancha nueva, se debe agregar al sistema, esta función es la encargada de guardar una cancha nueva al sistema.
long getDescuento()	Esta Función se encarga de calcular el descuento que se le dará a un afiliado tomando en cuanta el día de la semana o si es feriado.
String eliminarReservacionLista()	Elimina de una lista la reservación que se indico por error.
void logInValidateEvent	Recibe un objeto usuario de la vista y hace una llamada al objeto de negocio correspondiente para que verifique su existencia en la base de datos.
String EditarFecha()	Se encarga de cambiar los días feriados del sistema.

3.3.2. Paquete dao

El paquete `dao` contiene todas las clases que implementan la lógica del negocio por cada clase del paquete `beans` se crea una clase dao que define los métodos que manipulan directamente estos Objetos, de tal manera que la clase `actions` para acceder a los Objetos del paquete `beans` y hacer operaciones sobre estos debe invocar al objeto correspondiente que se encuentra en este paquete `dao` con el objeto de separar la lógica del negocio de los objetos de control logrando un mayor desacoplamiento de las capas.

Las Clases contenidas en el paquete `dao` son las siguientes:

- `AlquilerDAO`: Define los métodos relacionados para el alquiler de la cancha.
- `CanchaDAO`: Define los métodos correspondientes al control de usuarios y consulta de la existencia de estos.
- `DescuentoDAO`: Presenta los métodos relacionados con los descuentos que se le aplican a los afiliados
- `FeriadoDAO`: Define los métodos correspondientes a los días Feriados en el

sistema.

- `PersonaDAO`: Presenta los métodos relacionados para buscar las personas que están dentro de la base de datos y desean reservar una cancha.
- `PrecioDAO`: Presenta los métodos relacionados buscar las precios de las diferentes canchas.

A continuación se presentan los métodos más importantes de cada una de las clases presentes en el paquete `dao` a fin de entender más a fondo su importancia dentro del sistema

`AlquilerDAO`

Método	Descripción
`Alquiler_cancha getAlquiler(Long alquilerId)`	Devuelve el objeto Alquiler_cancha correspondiente al id de la cancha.
`List<Alquiler_cancha> getlistaCierre(Date fecha_hoy)`	Devuelve la lista de las canchas alquiladas según la fecha.
`Object getlistaMonto(Date fecha_hoy)`	Devuelve el objeto correspondiente al monto total de las canchas alquiladas según la fecha.

`CanchaDAO`

Método	Descripción
`Cancha getCancha(String IdCancha)`	Se obtiene la cancha según el id
`List<Cancha> getlistarCancha()`	Retorna la lista de las canchas.

`DescuentoDAO`

Método	Descripción
`List<Descuento> getListaDescuento(long idDescuento)`	Devuelve la lista de los descuentos según el id.
`List<Descuento> getlista()`	Retorna la lista de los descuentos.

`FeriadoDAO`

Método	Descripción
`List<Feriado> getlistaFeriadoLista(Date fecha)`	Retorna la lista de los feriados.

`PersonaDAO`

Método	Descripción
`Persona getPersona(long personaId)`	Retorna la persona según el id.
`List<Persona> getlistaPersona()`	Retorna la lista de las personas dentro de la base de datos.

List<Persona> busquedaPersonaInicial(FiltroPersonas filtroPersonas)	Retorna una lista de las personas según el filtro de busqueda.

PrecioDAO

Método	Descripción
List<Precio> getListaMonto(Integer hora, Integer tipo_Cancha, Integer dia_Semana	Retorna la lista según los valores pasados por parametro.
List<Precio> getListaPrecio(Integer hora)	Retorna la lista de los precios según la hora.

Todas las clases contenidas en el paquete dao también implementan tres funciones adicionales cada uno, una de inserción o guardado, una consulta y una de eliminación de objetos, llamadas `save` y `delete` respectivamente, a las cuales no se le hicieron referencia en los cuadros anteriores por ser estas unas funciones generales y comunes para cada clase.

El paquete `dao` a su vez realiza todas las consultas, inserciones y actualizaciones en la base de datos a través del *framework* de persistencia Hibernate. Para esto hace uso de las entidades que se encuentran en el paquete `beans` sus archivos de mapeo y del Objeto `HibernateUtil` contenido en el paquete `util`.

Es importante destacar que gracias al uso de Hibernate el desarrollador no tuvo que preocuparse por detalles del modelo de datos luego de la primera configuración; inclusive los querys son basados en el modelo orientado a objetos y no es necesario hacer recorrido a los resultados para asignarle los valores a las entidades, éstas ya vienen con la información requerida.

3.3.3. Paquete Beans

El paquete beans contiene todas las entidades persistentes de la aplicación y sus correspondientes archivos de mapeo hbm utilizados por Hibernate para manejar la persistencia. Cada una de las clases presentes en este paquete esta asociada a cada una de las tablas presentes en el modelo de datos representado en la figura del capitulo 2 de analisis y diseño.

A continuación se muestran las figuras correspondientes a la tabla, el código de la clase y el contenido del archivo hbm a fin de ilustrar como se encuentrar relacionados estos entre si.

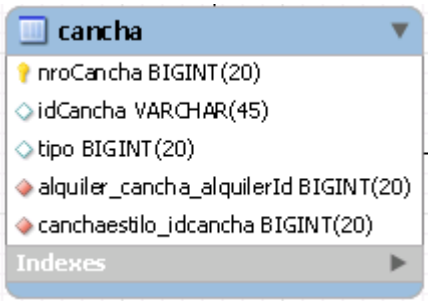


Tabla cancha en el Modelo de datos

```
<hibernate-mapping>

      <class name="com.gerelca.fvtenis.alquiler.dal.beans.Cancha"
schema="alquiler" table="cancha">
            <id name="nroCancha" type="long" column="nroCancha"
length="19">
                  <generator class="native">
                  </generator>
            </id>
            <property name="idCancha" type="string"
column="idCancha" length="255"/>
            <property name="tipo" type="long" column="tipo"
length="19"/>
      </class>
      <query name="cancha.lista">
      <![CDATA[from Cancha as cancha]]>
    </query>

    <query name="traer.cancha.nroCancha">
      <![CDATA[select cancha from Cancha as cancha ORDER BY
nroCancha]]>
    </query>
    <query name="traer.idCancha">
      <![CDATA[select cancha from Cancha as cancha where
      cancha.idCancha=:IdCancha]]>
    </query>
       </hibernate-mapping>
```

Se puede observar que las clases contenidas en el paquete persistente son clases normales con una serie de atributos con la única peculiaridad que el tipo de datos corresponde con el de su homónimo en la base de datos, es en el archivo de mapeo (hbm) donde se le indica a `Hibernate` que esta clase esta relacionada con la tabla actividad de la base de datos y se le indica cual cuales campos van a ser mapeados en la base de datos, como buena practica de programación y para mantener el orden y la legibilidad se le suelen colocar los mismos nombres pero esto no tiene por que

corresponder, mientras se le indique de forma correcta en el archivo de mapeo que campo mapea a cual y que clase mapea a que tabla.

De esta forma quedan integradas todas las piezas que conforman la estructura del sistema desarrollado para este trabajo especial de grado.

3.3.4. Integración Spring DAO

Código CanchaDAO.java.

```
      public class CanchaDAO extends BaseDAO{
      public Cancha getCancha(String IdCancha)
      {
            return (Cancha)
getSesionHibernate().getSessionFactory().getCurrentSession().get(Ca
ncha.class, IdCancha);
      }
      public Object getTraerCancha(String IdCancha)
      {
            Query
q=getSesionHibernate().getSessionFactory().getCurrentSession().getN
amedQuery("traer.idCancha");
            q.setString("IdCancha",IdCancha );
            return q.uniqueResult();
            }
      public Cancha getCargarCancha(long NroCancha)
      {
            return (Cancha)
getSesionHibernate().getSessionFactory().getCurrentSession().get(Ca
ncha.class, NroCancha);
      }
      @SuppressWarnings("unchecked")
      public List<Cancha> getlistaCancha()
      {
            return (List<Cancha>)
getSesionHibernate().getSessionFactory().getCurrentSession().getNam
edQuery("traer.cancha.nroCancha").list();
            }
      public com.gerelca.fvtenis.alquiler.dal.beans.Cancha
ingresarcancha(Cancha cancha) {

            this.getSesionHibernate().saveOrUpdate(cancha);
            return null;
      }
      public com.gerelca.fvtenis.alquiler.dal.beans.Cancha
borrarcancha(Cancha cancha) {

            this.getSesionHibernate().delete(cancha);
            return null;
      }}
```

Para que Spring se haga cargo del control de los DAOS fue necesario configurar el archivo de contexto applicationContextBeans.xml como se muestra en el siguiente fragmento de código.

```
<!DOCTYPE beans PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://www.springframework.org/dtd/spring-beans.dtd">

      <beans>
            <bean id="hibernateUtil"
class="com.gerelca.fvtenis.alquiler.jsf.utils.HibernateUtil" />
            <bean id="hibernatesessionFactory"
class="com.gerelca.fvtenis.alquiler.jsf.utils.HibernateUtil"
factory-method="getSessionFactory"/>

            <bean id="baseDAO"
class="com.gerelca.fvtenis.alquiler.dal.dao.BaseDAO">
                  <property name="sessionFactory"
ref="hibernatesessionFactory"></property>
            </bean>

            <bean id="generalDAO" parent="baseDAO"
class="com.gerelca.fvtenis.alquiler.dal.dao.GeneralDAO" />
            <bean id="personaDAO" parent="baseDAO"
class="com.gerelca.fvtenis.alquiler.dal.dao.PersonaDAO" />
            <bean id="canchaDAO" parent="baseDAO"
class="com.gerelca.fvtenis.alquiler.dal.dao.CanchaDAO" />
            <bean id="alquilerDAO" parent="baseDAO"
class="com.gerelca.fvtenis.alquiler.dal.dao.AlquilerDAO" />
            <bean id="feriadoDAO" parent="baseDAO"
class="com.gerelca.fvtenis.alquiler.dal.dao.FeriadoDAO" />
            <bean id="adminDAO" parent="baseDAO"
class="com.gerelca.fvtenis.alquiler.dal.dao.AdminDAO" />

       </beans>
```

CONCLUSIONES

El desarrollo de software es una actividad que implica análisis y preparación en el que se debe tomar en cuenta aspectos como la tecnología a utilizar, su costo y la metodología.

El enfoque de desarrollo AJAX proporciona una nueva forma de interactuar con las aplicaciones Web, la cual hace que el usuario se comunique de una forma mas fluida con el sistema.

Después de realizar esta investigación queda evidenciado que un software basado en el enfoque de desarrollo AJAX puede ser implementado sin ningún inconveniente en centros como la Federación de Tenis con el fin de optimizar sus procesos y comunicaciones, tanto internas como externas ya que se lograron cumplir todos los requerimientos solicitados por la comunidad deportiva.

Para la culminación del TEG se lograron alcanzar exitosamente los siguientes objetivos específicos:

- Diseño de una arquitectura basada en el modelo MVC ya que se logró centrar la lógica de negocios dentro de objetos de dominio separados y se pudo abstraer la lógica de presentación en vistas, que muestran los datos provenientes de los objetos del dominio. El controlador maneja la navegación entre vistas, procesa la entrada de datos del usuario y asegura el correcto flujo entre la vista y el modelo, y éste a su vez encapsula el almacenamiento de datos y la implementación de las reglas de negocio.

- Implementación exitosa de la lógica del lado del cliente utilizando, los *frameworks* RichFaces para el desarrollo basado en el enfoque AJAX.

- Desarrollo exitoso de la capa de persistencia, utilizando el *framework* Hibernate, aprovechando así las bondades y facilidades para el desarrollo y configuración de una capa de persistencia más liviana y flexible.

- Desarrollo de la capa de lógica de negocio, utilizando el *framework* Spring DAO y JSP para lograr la integración exitosa de todos los componentes.

- La metodología de trabajo "Proceso Unificado Ágil" sirvió para el desarrollo y construcción de todo el TEG, llevando a cabo cada uno de los requerimientos y objetivos específicos.

- Se utilizó AUP porque se ajusta a los valores y principios de la Alianza Ágil en donde solo se toman en cuenta las actividades importantes por el usuario final, simplificando cada uno de los detalles.

Experiencia durante el desarrollo del TEG

El problema de toda nueva tecnología es no solo comprender los nuevos *frameworks*, sino llegar a dominarlos y aprovechar toda la potencia y funcionalidades que ofrecen.

A pesar de ello, los *frameworks* de código abierto (JSF, Hibernate, Spring DAO, etc) favorecen en gran medida la resolución de dudas y problemas a través de foros muy participativos y existe amplia documentación tanto en forma impresa como en Internet.

En la experiencia al desarrollar este TEG pude conseguir mucha información para llevar a cabo todo el modelado del sistema. Para el uso de la vista como fue el caso de JSF y RichFaces (que sirvio de gran utilidad para la integración de Ajax) fue sencilla de utilizar y de aprender.

Contribuciones

Los principales aportes que se generaron al culminar el Sistema de Alquiler de Canchas de Tenis fueron los siguientes:

- Se desarrolló un sistema que permita automatizar actividades como:
  - Mantener registro de las canchas alquiladas.
  - Visualizar por fecha las canchas disponibles
  - Buscar a las personas afiliadas, mostrar precios y calcular el descuento si se da el caso.
  - Imprimir la hoja con la reservación de la persona
  - Se pueda Buscar por fecha las reservaciones, buscar por persona las reservaciones y buscar por persona las reservaciones.
  - Hacer un cierre de caja.
  - Agregar una Cancha nueva al sistema, modificar los días feriados, Editar los precios de las canchas y Editar el descuento para los afiliados.
- Usando herramientas de desarrollo de Aplicaciones de Internet Enriquecidas se pudo obtener un sistema con una interacción mucho más fluida y rápida que con una aplicación Web tradicional.

Recomendaciones y trabajos futuros

El presente trabajo de grado se enfocó en la implementación del módulo principal de la aplicación: el control de reservaciones de las canchas. Es así como la investigación que se realizó en este trabajo pretende servir de base para trabajos futuros que puedan implementar y/o enriquecer los siguientes módulos sobre la arquitectura planteada.

Tomando en cuenta lo anterior, se proponen los siguientes trabajos futuros y recomendaciones:

- Integrar este modulo con el sistema ya existente en la federación para emitir las facturas a los clientes.
- Implementación de un módulo de reportes sobre los datos manejados por el sistema.
- Implementación para que el sistema pueda ser accedido por cualquier usuario en Internet y los pagos sean electrónicos.

Referencias Bibliográficas

[1] TEG: Reingeniería de una aplicación para el seguimiento de hojas de tiempo utilizando el enfoque AJAX y la tecnología JPA. A.Leal, J.F. Ravelo, C. Moreno, H. Gonzalez.

[2] Asleson, R. & Schutta, R. T. (2005). *Foundations of Ajax*. Apress.

[3] Gadge, V. V. (2006). Technology options for Rich Internet Applications. http://www-128.ibm.com/developerworks/web/library/wa-richiapp.

[4] Crane, D., Pascarello, E. & James, D. (2005). AJAX in action. Manning Publications Co.

[5] El Proceso Unificado de Desarrollo de Software. http://yaqui.mxl.uabc.mx/~molguin/as/RUP.htm

[6] John Hunt, (2003) Guide to the Unified Process featuring UML, Java and Design Patterns

[7] Philippe Kruchten, (2000) The Rational Unified Process: An Introduction (2nd Edition)

[8] Hamilton, K. & Miles, R. (2006). Learning UML 2.0. O'Reilly

[9] Direct Web Remoting (2008). Documentation.

http://directwebremoting.org/dwr/documentation.

[10] Prototype JavaScript Framework (2008). Prototype.

http://www.prototypejs.org.

[11] Scriptaculous (2008). Scriptaculous.

http://script.aculo.us.

[12] Wikipedia (2008). Ritech Internet Application.

http://en.wikipedia.org/wiki/Rich_Internet_application.

[13] Wikipedia (2008). AJAX.

http://es.wikipedia.org/wiki/AJAX.

[14] Modelos Clientes/Servidor, [en línea]. Disponible en:

http://es.wikipedia.org/wiki/Modelo_Cliente_Servidor/

[15] JavaServer Faces (JSF), [en línea]. Disponible en:

http://java.sun.com/javaee/javaserverfaces/

[16] RichFaces, [en línea]. Disponible en:

http://livedemo.exadel.com/RichFaces-demo/RichFaces/actionparam.jsf

[17] Java EE 5, [en línea]. Disponible en:

http://es.wikipedia.org/wiki/J2EE/

[18] JSpring, [en línea]. Disponible en:

http://www.springframework.org/

[19] Hibernate, [en línea]. Disponible en:

http://www.hibernate.org/

[20] MySQL, [en línea]. Disponible en:

http://dev.mysql.com

[21] Proceso Unificado, [en línea]. Disponible en:

http://es.wikipedia.org/wiki/Proceso_Unificado

[22] Spring, [en línea]. Disponible en:

http://es.wikipedia.org/wiki/Spring_Framework