CAPÍTULO 3

DISEÑO Y DESARROLLO

METAS DEL SISTEMA LMP

Después de las juntas entre el cliente y el equipo de desarrollo del PROVEEDOR, obtuvimos un panorama general acerca de los requerimientos del sistema, de los módulos y de los casos de uso que describen el funcionamiento del sistema.

El objetivo es que la aplicación cumpliera con los requerimientos del cliente y además con los elementos básicos que una webapp debe incluir para ser una aplicación utilizable, navegable y con un diseño apropiado.

Las metas a las que se pretendían llegar mismas que se alcanzaron al término del proyecto y que fueron factores clave para el éxito del mismo son las siguientes:

Simplicidad: La aplicación debe contener la información necesaria ni más ni menos, para permitir al usuario final intuir que paso sigue dentro de un proceso.

Congruencia: El sitio debe ser construido de modo congruente, es decir que la imagen, tipografía, tamaños de textos, imágenes y todos los elementos gráficos deben mantener un cierto estilo para dotar a la webapp de identidad y permita al usuario recordar nuestra aplicación.

Robustez: Dotar a la aplicación de funciones útiles para el usuario final que sean relevantes para sus necesidades.

Compatibilidad: Hacer que nuestra webapp pueda convivir con distintos hardware, software, tipos de conexión, sistemas operativos, navegadores etc.

Escalabilidad: Desarrollar una aplicación que nos permita ir mejorándola, actualizándola y agregando nuevos módulos de acuerdo a las necesidades que surjan en determinado momento.

Comunicabilidad: Permitir a la aplicación convivir con otros sistemas e interactuar intercambiando información a través de diversos mecanismos que no permitan a la webapp ser una aplicación cerrada, ya que eso limitaría su usabilidad y tiempo de vida.

Seguridad: La información contenida dentro del sistema LMP contiene información delicada acerca del negocio del cliente por lo que la seguridad dentro de la webapp es de vital importancia. Para evitar accesos y modificaciones no deseados al sistema.

Disponibilidad. Garantizar que la webapp esté accesible las 24 horas los 365 días del año, además de la compatibilidad con diferentes navegadores en distintos sistemas operativos hacen que nuestra aplicación se accesible para más usuarios. En el caso de la aplicación LMP, garantizar que en todos las Salas en donde utilizan, diferentes sistemas operativos y diferentes navegadores la aplicación se pueda utilizar sin problemas.

Estabilidad. Que la aplicación LMP pueda dar servicio a un número significativo de usuarios y volumen transacciones.

Con estos objetivos en mente, trabajé en el diseño de la aplicación utilizando las mejores prácticas en Desarrollo web de sistemas, por lo que me fue necesario realizar una investigación previa antes de decidir con qué arquitectura de diseño debíamos abordar el desarrollo de este sistema. Durante este proceso pude hacer uso de los conocimientos obtenidos en materias como Programación de Sistemas y Estadística.

Actualmente la evolución tecnológica que el sector ha sufrido durante los últimos años ha permitido otra evolución paralela, la de la arquitectura de las aplicaciones web. A medida que aparecían nuevos recursos técnicos, los patrones de diseño se amoldaban para aprovechar las nuevas característica que estás novedades ofrecían. De esta forma, el modelo de las aplicaciones de Internet ha sufrido dos grandes saltos desde la aparición de los primeros portales. Los distintos modelos son los mostrados en la Tabla 3.1.

Tabla 3.1 Modelos arquitectónicos de aplicaciones Web

Modelo	**Descripción**
Modelo 1	Son las más primitivas. Se identifican con este modelo las clásicas aplicaciones web CGI, basadas en la ejecución de procesos externos al servidor web, cuya salida por pantalla era el HTML que el navegador recibía en respuesta a su petición. Presentación, negocio y acceso a datos se confundían en un mismo script Perl.
Modelo 2	Como evolución del modelo 1 con la incorporación del patrón MVC a este tipo de aplicaciones, se conocen como Modelo 2 de la arquitectura web. Se incorpora un elemento controlador de la navegación de la aplicación. El modelo de negocio queda encapsulado y la vista del usuario queda

	visible recibiendo el resultado de las operaciones del controlador.
Modelo 2X	El modelo 2X aparece como evolución del modelo 2, y con objeto de dar respuesta a la necesidad, cada vez más habitual, de desarrollar aplicaciones multicanal, es decir, aplicaciones web pueden ser atacadas desde distintos tipos de clientes remotos. Así, una aplicación web multicanal podrá ejecutarse desde una PDA, desde un terminal de telefonía móvil, o desde cualquier navegador HTML estándar. El medio para lograr publicar la misma aplicación para distintos dispositivos es emplear plantillas XSL para transformar los datos XML y determinar la plantilla a emplear en base al parámetro user-agent recibido en el request.

Las arquitecturas utilizadas en desarrollo web son las descritas en la Tabla 3.2

Tabla 3.2 Arquitecturas de Desarrollo

Arquitectura	Descripción	Ventajas	Desventajas
Monocapa	Tanto los datos de aplicación como la interfaz como la lógica de modelo residen en una misma identidad	∝ Toda la aplicación reside en un mismo elemento de hardware y de software ∞ El tiempo de desarrollo es elevado ya que no permite el trabajo en equipo separado por responsabilidades.	∝No permite interacción con más de un usuario. ∞La actualización depende de personal autorizado. ∞Difícil mantenimiento y actualización.
Dos Capas	La arquitectura tradicional de cliente/servidor también es conocida como arquitectura de dos capas. Requiere una interfaz de usuario que se instala y corre en una PC o estación de trabajo y envía solicitudes a un servidor para ejecutar operaciones complejas. Por ejemplo, una estación de trabajo utilizada	∝ El tiempo de desarrollo de aplicaciones es más rápido que en un sistema Monocapa. ∞ Permite el trabajo en equipo al tener una separación de responsabilidades.	∝El mantenimiento puede resultar un problema ya que hay que realizar una actualización en cada cliente del sistema. ∞La dependencia a largo plazo de cualquier herramienta, puede complicar el escalamiento futuro o las implementaciones.

	como cliente puede correr una aplicación de interfaz de usuario que interroga a un servidor central de bases de datos.		∝Los ambientes de dos capas requieren control excesivo de las versiones y demandan esfuerzo de distribución de la aplicación cuando se les hacen cambios. Esto se debe al hecho de que la mayoría de la aplicación lógica existe en la estación de trabajo del cliente
Tres Capas	La arquitectura de tres capas introduce una capa intermedia en el proceso. Cada capa es un proceso separado y bien definido corriendo en plataformas separadas. En la arquitectura tradicional de tres capas se instala una interfaz de usuario en la computadora del usuario final (el cliente). La arquitectura basada en Web transforma la interfaz de búsqueda existente (el explorador de Web), en la interfaz del usuario final.	∝ Las llamadas de la interfaz del usuario en la estación de trabajo, al servidor de capa intermedia, son más flexibles que en el diseño de dos capas, ya que la estación solo necesita transferir parámetros a la capa intermedia. ∝Con la arquitectura de tres capas, la interfaz del cliente no es requerida para comprender o comunicarse con el receptor de los datos. Por lo tanto, esa estructura de los datos puede ser modificada sin cambiar la interfaz del usuario en la PC. ∝El código de la capa intermedia puede ser reutilizado por múltiples aplicaciones si está diseñado en formato modular. ∝La separación de roles en tres capas, hace más fácil reemplazar o modificar una capa sin afectar a los módulos restantes.	∝Los ambientes de tres capas pueden incrementar el tráfico en la red y requiere más balance de carga u tolerancia a las fallas. ∝Los exploradores actuales no son todos iguales. La estandarización entre diferentes proveedores ha sido lenta en desarrollarse.

:

De acuerdo a la información de la tabla 3.2 y a las metas del sistema LMP, la arquitectura seleccionada fue la de tres capas, para la elección de ésta se tomaron los siguientes criterios:

- Escalabilidad.
- Separación las responsabilidades de desarrollo.
- Mantenibilidad.
- Desarrollo ágil dentro de un ambiente en un equipo de trabajo.

Debido a estas características el equipo de desarrollo del PROVEEDOR tomó la decisión de implementar un modelo MVC (Modelo – Vista - Controlador) que se ajusta a este tipo de arquitectura en capas y que es ideal para el sistema LMP.

PATRÓN DE DISEÑO MVC

MVC (Modelo-Vista-Controlador) es un patrón de diseño que considera dividir una aplicación en tres módulos o capas claramente identificables y con funcionalidad bien definida: El Modelo, las Vistas y el Controlador.

Modelo: El modelo es un conjunto de clases que representan la información del mundo real que el sistema debe procesar.

Vista: Presenta el modelo con el que va a interactuar el usuario, más conocida como interfaz.

Controlador: El controlador responde más bien a eventos, normalmente son acciones que el usuario invoca, implica cambios en el modelo y también en la vista (interfaz).

El controlador es un objeto que se encarga de dirigir el flujo del control de la aplicación debido a mensajes externos, como datos introducidos por el usuario u opciones del menú seleccionadas por él. De acuerdo a estos mensajes, el controlador se encarga de modificar el modelo o de abrir y cerrar vistas. El controlador tiene acceso al modelo y a las vistas, pero las vistas y el modelo no conocen de la existencia del controlador.

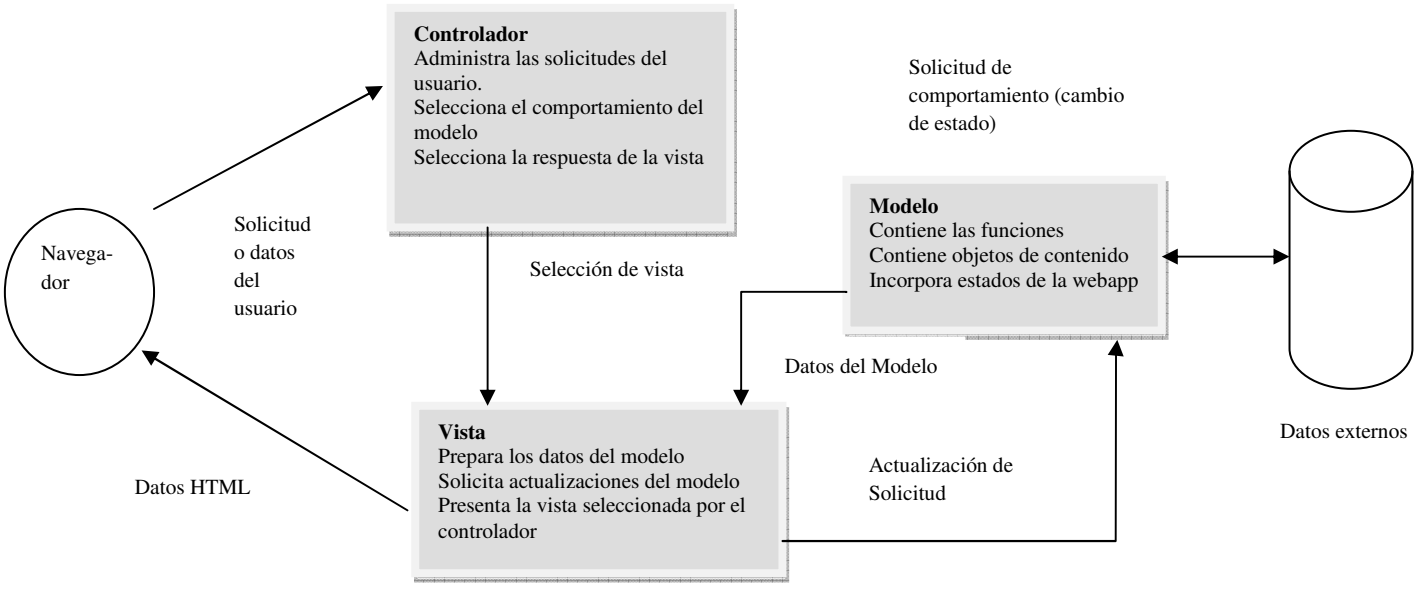


Figura 3.1 Patrón MVC

Como se muestra en la Figura 3.1, el controlador maneja las solicitudes o datos del usuario. El controlador también selecciona el objeto de vista que sea aplicable con base en la solicitud del usuario. Una vez determinado el tipo de solicitud, se transmite al modelo un pedido de comportamiento, que implementa la funcionalidad o recupera el contenido requerido para dar acomodo a la solicitud, se transmite al modelo un pedido de comportamiento que implementa la funcionalidad o recupera el contenido requerido para dar acomodo a la solicitud. El objeto de modelo accede a los datos almacenados en una base de datos corporativa, como parte de un almacén de datos locales o como una colección de archivos independientes. El objeto de vista apropiado debe dar formato y organizar los datos desarrollados por el modelo para luego transmitirlos desde el servidor de la aplicación hacia el navegador del cliente para que se desplieguen en su navegador.

Con base en la investigación realizada y en mi experiencia en el desarrollo de sistemas WEB, se tomó la decisión de implementar un patrón MVC para la aplicación LMP y Tarjeta BEC, debido a que las características de este patrón cumplen con las metas de escalabilidad y modularidad, las cuales permitirán a la aplicación crecer de acuerdo a los nuevos requerimientos que vayan surgiendo.

TECNOLOGÍA APLICADA AL SISTEMA LMP

El tiempo de desarrollo fue un factor importante en la creación del sistema LMP, si bien utilizar un patrón MVC garantiza utilizar las Best Practices para el desarrollo de sistemas de webapps, también es cierto que el desarrollar e implementar este patrón desde cero podría ser una tarea que implicara mucho más tiempo del deseado. Afortunadamente ya existen herramientas que implementan este patrón por default para que el desarrollador sólo se preocupe por diseñar el modelo de datos, de las reglas de negocio y de la vista de la aplicación.

Estas herramientas son llamadas Frameworks, "Marcos de Trabajo" o WAF (Web Application Framework), son una serie de librerías (toolkits) que se han unido bajo un único esquema de colaboración para que de manera rápida (RAD: Rapid Application Development) se desarrollen las aplicaciones.

Entre las ventajas de utilizar un framework para el desarrollo de webapps se encuentran:

- ∞ No hay que definir "marco de desarrollo", solo "rellenar" los huecos que los frameworks nos indican.
- ∞ Trabajar sobre un Estándar que miles de personas ya conocen.
- ∞ Facilidad para encontrar herramientas, librerías o documentación.
- ∞ Relación Coste de aprendizaje vs. Aplicabilidad mínima.

En la actualidad existen frameworks que utilizan el patrón MVC en cada lenguaje de programación en nuestro caso, debido a que se seleccionó el lenguaje PHP5 para desarrollar el sistema LMP, revisamos cual sería el más afín con las necesidades de programación.

Algunos Framework que utilizan el model MVC de mayor presencia en el desarrollo de sistemas web en la actualidad son:

- ∞ Zend Framework
- ∞ CakePHP
- ∞ CodeIgniter
- ∞ Symfony

Cada uno tiene características propias que no lo hacen ni mejor o peor que otro si no que con sus características propias le permiten ajustarse a las necesidades del negocio, en este caso a LMP. Durante esta etapa de selección uno de los miembros del equipo de

trabajo del PROVEEDOR recomendó utilizar el Framework CodeIgniter mientras que por mi parte la recomendación fue utilizar Zend Framework. Algunas de las características y más sobresalientes, son las listadas en la Tabla 3.3.

Tabla 3.3 Comparativa de Frameworks de PHP

Característica	Zend Framework	CodeIgniter
1. Configuración	Se requiere la creación de un archivo de arranque con los valores de inicialización. El framework es relativamente grande - cerca de 12.4Mb y el proceso de configuración tomó unos 30 minutos.	El tamaño del framework es de apróx 2MB y la instalación consiste en copiar todos los archivos.
2. Documentación	Zend Framework tiene muy detallada documentación con una gran cantidad de ejemplos.	Documentación estructurada, aunque menos detallada que ZF.
3. Plantillas (Layouts)	Zend Framework incluye una clase de diseño para proporcionar un diseño común (o varios diseños) para todo el sitio web o aplicación.	Cuenta con una clase dedicada a usar Templates.
4. Componentes	ZF cuenta con un número masivo de las clases y componentes. Están bien documentados, aunque el uso es generalmente un poco más difícil que en CI.	Tienen gran cantidad de librerías disponibles
5. Acceso a BD	Zend_Db y sus clases, ofrecen una sencilla interfaz de base de datos SQL.	Contiene una clase de conexión a BD sencilla de utilizar.
6. Flexibilidad	ZF es simplemente una colección de clases y como tal cualquier archivo o carpeta puede colocarse en cualquier lugar, siempre y cuando la ubicación se agrega al archivo de arranque.	Es muy flexible que permite casi todos los valores predeterminados para ser modificado.
7. Validación	El componente Zend_Validate proporciona un conjunto de validadores con frecuencia necesarios. También proporciona un mecanismo de encadenamiento por el cual varios validadores se puede aplicar a un solo de forma ordenada.	La validación de datos en CodeIgniter se maneja a través de una clase de validación. Un conjunto de reglas se definen y se asigna al objeto de validación de validación object.
8. Formularios	Zend_Form simplifica la creación de formularios y manejo. Usando, ZF puede representar un form completamente en el código PHP, incluyendo etiquetas, validación y mensajes de error.	No elimina completamente la necesidad de escribir código HTML.
9. Rendimiento	El Zend Framework es aproximadamente la mitad de rápido que CodeIgniter.	Tiene casi el doble del rendimiento del Zend Framework.
10. Internacionalización	Si	No
11. Licencia	Nueva BSD	BSD

De acuerdo a estas características ambos frameworks poseen cualidades interesantes, pero para el desarrollo de LMP el equipo de desarrollo decidió utilizar Zend Framework principalmente por los puntos 2, 6 y 7 que corresponde a Documentación, Flexibilidad y Validación de la tabla 3.3.

Estas características tuvieron impacto en el desarrollo de la aplicación entre los cuales están:

Documentación: Al tener mayor nivel de detalle, permitió a los desarrolladores reducir la curva de aprendizaje, debido a que se encontró mayor información y ejemplos en Internet.

Flexibilidad: Esta característica nos permitió implementar la aplicación en diferentes ambientes de desarrollo, dotándonos de la capacidad de realizar una configuración propia a cada sistema sin realizar cambios importantes a la aplicación.

Validación: Permite que la validación de los formularios de la aplicación se realizaran de forma ordenada y estructurada. Reduciendo el tiempo de desarrollo y por ende mejorando los tiempos de entrega.

PLANIFICACIÓN DEL PROYECTO LMP

Una vez definidos los requerimientos del cliente, de haber realizado un análisis modular, de definir la tecnología que se aplicaría al sistema LMP, el siguiente paso fue realizar la planificación del proyecto en cuanto a recursos y tiempo de desarrollo.

Las tareas que se siguieron para realizar la planificación fueron las siguientes.

1. Establecimiento de Recursos.
2. Estimación de costos y esfuerzo.
3. Calendario del proyecto.

1. ESTABLECIMIENTO DE RECURSOS

Los recursos son los elementos con los que contamos o requerimos para lograr el objetivo del desarrollo de software. La figura 3.2 muestra las principales categorías de recursos utilizados en desarrollo de software:

a) Personal
b) Componentes de software reutilizables
c) Entorno de desarrollo (herramientas de software y de hardware).

Cada recurso se especifica con cuatro características: descripción del recurso, disponibilidad, momento en que se requiere y tiempo de aplicación de recurso.

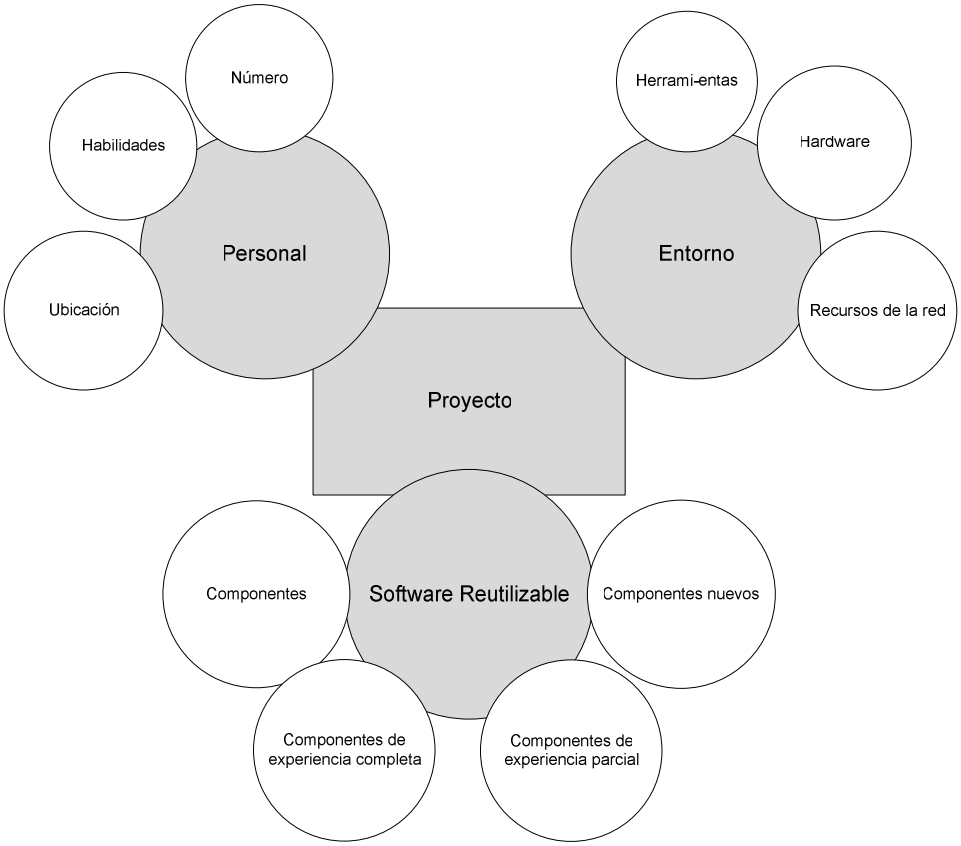


Figura 3.2 Recursos del Proyecto

a) **Personal:** Se especifica tanto la posición organizacional como la especialidad, en algunos proyectos un solo individuo puede realizar varias tareas, ese fue el caso del sistema LMP, ya que el PROVEEDOR asignó directamente a los recursos humanos de acuerdo a la tecnología a utilizar. Para este proyecto se contó con los recursos humanos listados en la Tabla 3.4.

Tabla 3.4 Recursos humanos asignados a desarrollo LMP

Nombre	Posición	Especialidad
Javier Villegas Avilés (Quien presenta este informe de actividades)	Project Manager, Diseño y desarrollo	∝ Gestión de proyecto ∞ Diseño y desarrollo de sistemas web ∞ Base de datos ∞ Programación ∞ Testing
Erick Martínez Loran	Diseño y desarrollo	∝ Diseño y desarrollo de sistemas web. ∞ Base de datos ∞ Programación ∞ Servidores
Debora Martínez	Testing	∝ Pruebas d Software ∞ Usabilidad ∞ Validación de procesos
Jorge Acosta Rojero	Ventas	∝ Gestión con el cliente ∞ Test de aplicación ∞ Levantamiento de requerimientos

b) **Componentes de software reutilizables**: La ingeniería de software hace énfasis en la reusabilidad es decir, en la creación y reutilización de bloques constructores de software. A estos bloques con frecuencia se les llama componentes los cuales deben catalogarse para facilitar su integración y validar su aplicación. En el caso especifico para el desarrollo del sistema LMP el recurso principal fueron las librerías que el framework de zend nos proporciono.

c) **Entorno de desarrollo**: El entorno que soporta a un proyecto de software, con frecuencia llamado entorno de ingeniería, incorpora elementos de hardware y de software. El hardware proporciona una plataforma que soporta las herramientas de software requeridas para producir un software de acuerdo a lo requerido. De acuerdo al análisis previo del sistema LMP, los recursos ambientales para desarrollo en hardware y software son los listados en la Tabla 3.5

Tabla 3.5 Recursos ambientales para el sistema LMP

Tipo Recurso	**Recurso**	**Descripción**
Hardware	Servidor Desarrollo	Servidor Dedicado para el desarrollo del sistema LMP Características: ∞ Dell Optiplex 755 ∞ Core 2 Due ∞ 2GB RAM ∞ 120 GB HD
Hardware	PC Desarrollo	PC de escritorio dedicada a desarrollo de la aplicación ∞ Pentium IV ∞ 1GB RAM ∞ 80GB HD
Hardware	PC Desarrollo	PC de escritorio dedicada a desarrollo de la aplicación ∞ Pentium IV ∞ 1GB RAM ∞ 80GB HD
Software	Sistema Operativo Linux	El SO base del servidor del desarrollo. Red Hat Enterprise 5
Software	Sistema Operativo Windows	Sistema Operativo base de las PC's de desarrollo.
Software	IDE Desarrollo Zend Studio	Ambiente de desarrollo optimizado para uso con el framework de Zend
Software	Tools BD MySQL	Herramientas de administración de base de datos MYSQL

2. ESTIMACIÓN DE COSTO Y ESFUERZO

La estimación de costo y esfuerzo del software nunca será una ciencia exacta, ya que existen muchas variables que pueden afectar el costo final del software y el esfuerzo aplicado para su desarrollo, aun así se vuelve necesario tener una estimación con un riesgo aceptable.

Los componentes de riesgo se definen en la forma siguiente:

- ∞ Riesgo de rendimiento: grado de incertidumbre de que el producto satisfará sus requisitos y se ajustara al uso pretendido.
- ∞ Riesgo de costo: grado de incertidumbre de que el presupuesto del proyecto se mantendrá.
- ∞ Riesgo de apoyo: grado de incertidumbre de que el software resultante será fácil de corregir, adaptar y mejorar.
- ∞ Riesgo de calendario: grado de incertidumbre de que el calendario del proyecto se mantendrá y de que el producto se entregará en tiempo.

El área de finanzas estimó el riesgo aceptable para este proyecto y lo hizo de acuerdo a la experiencia en otros desarrollos web diseñados por el PROVEEDOR y la negociación que se tuvo con el CLIENTE.
Por lo que el riesgo aceptable es del 20% para el riesgo de costo y el riesgo de calendario lo que permitiría al PROVEEDOR cubrir las expectativas del sistema en tiempo y forma.

En cuanto a la estimación de tiempo se refiere existen algunas recomendaciones entre las que se encuentran:

1. Estimaciones en base a proyectos anteriores
2. Usar técnicas de descomposición simples
3. Usar uno o más modelos empíricos

Para realizar una estimación aproximada utilicé técnicas de descomposición, debido a que resulta más fácil descomponer el problema en varios más pequeños y por consecuencia más manejable lo que permite una estimación más cercana a la realidad.

Específicamente, utilizando la técnica de estimación por procesos orientada a web, y de acuerdo a los casos de uso anteriores obtuve la estimación en tiempo en días por proceso los cuales se muestran de las tablas 3.6 a 3.14

Tabla 3.6 Estimación Módulo Campañas

Módulo Campañas		**TIEMPO EN DIAS**				
Módulo	**Cantidad**	**Análisis**	**Diseño**	**Código**	**Prueba**	**Totales**
Entradas	6	2	1	4	1	8
Salidas	8	2	1	4	1	8
Tablas	6	4	2	3	1	10
Consultas	10	6	3	3	1	13
Totales		14	7	14	4	39

Tabla 3.7 Estimación Módulo Redenciones

Módulo Redenciones		**TIEMPO EN DIAS**				
Módulo	**Cantidad**	**Análisis**	**Diseño**	**Código**	**Prueba**	**Totales**
Entradas	4	2	1	4	1	8
Salidas	8	2	1	4	1	8
Tablas	6	3	2	3	1	9
Consultas	8	4	2	3	1	10
Totales		11	6	14	4	35

Tabla 3.8 Estimación Módulo Clientes

Módulo Clientes		**TIEMPO EN DIAS**				
Módulo	**Cantidad**	**Análisis**	**Diseño**	**Código**	**Prueba**	**Totales**
Entradas	7	2	1	4	1	8
Salidas	6	2	1	4	1	8
Tablas	6	3	2	3	1	9
Consultas	5	3	2	3	1	9
Totales		10	6	14	4	34

Tabla 3.9 Estimación Módulo Campañas

Módulo Manejos		**TIEMPO EN DIAS**				
Módulo	**Cantidad**	**Análisis**	**Diseño**	**Código**	**Prueba**	**Totales**
Entradas	1	1	1	3	1	6
Salidas	2	1	1	3	1	6
Tablas	1	1	1	3	1	6
Consultas	5	1	1	3	1	6
Totales		4	4	12	4	24

Tabla 3.10 Estimación Módulo Puntos

Módulo Puntos		**TIEMPO EN DIAS**				
Módulo	**Cantidad**	**Análisis**	**Diseño**	**Código**	**Prueba**	**Totales**
Entradas	1	1	1	3	1	6
Salidas	2	1	1	3	1	6
Tablas	1	1	1	3	1	6
Consultas	5	1	1	3	1	6
Totales		4	4	12	4	24

Tabla 3.10 Estimación Módulo Configuración

Módulo Configuración		**TIEMPO EN DIAS**				
Módulo	**Cantidad**	**Análisis**	**Diseño**	**Código**	**Prueba**	**Totales**
Entradas	40	10	4	15	5	34
Salidas	35	10	4	14	5	33
Tablas	25	5	3	10	5	23
Consultas	45	11	3	15	5	34
Totales		36	14	54	20	124

Tabla 3.11 Estimación Módulo Mailing

Módulo Mailing		**TIEMPO EN DIAS**				
Módulo	**Cantidad**	**Análisis**	**Diseño**	**Código**	**Prueba**	**Totales**
Entradas	10	2	2	5	1	10
Salidas	11	2	2	5	1	10
Tablas	5	2	2	5	1	10
Consultas	10	2	2	5	1	10
Totales		8	8	20	4	40

Tabla 3.12 Estimación Módulo Reporting

Módulo Reporting		**TIEMPO EN DIAS**				
Módulo	**Cantidad**	**Análisis**	**Diseño**	**Código**	**Prueba**	**Totales**
Entradas	5	3	2	4	1	10
Salidas	9	3	2	4	1	10
Tablas	50	3	2	3	1	9
Consultas	20	3	2	3	1	9
Totales		12	8	14	4	38

Tabla 3.13 Estimación Módulo Valores LMP

Módulo Valores LMP		**TIEMPO EN DIAS**				
Módulo	**Cantidad**	**Análisis**	**Diseño**	**Código**	**Prueba**	**Totales**
Entradas	1	1	1	2	1	5
Salidas	1	1	1	2	1	5
Tablas	1	1	1	2	1	5
Consultas	1	1	1	2	1	5
Totales		4	4	8	4	20

Tabla 3.14 Estimación Módulo Finanzas BEC

Módulo Finanzas BEC		**TIEMPO EN DIAS**				
Módulo	**Cantidad**	**Análisis**	**Diseño**	**Código**	**Prueba**	**Totales**
Entradas	1	2	1	4	1	8
Salidas	2	2	1	4	1	8
Tablas	10	2	1	3	1	7
Consultas	2	2	1	3	1	7
Totales		8	4	14	4	30

Los parámetros que se contemplaron para realizar esta estimación en días fueron:

- ∞ Una persona realizando cada operación.
- ∞ La experiencia adquirida al programar otros sistemas web.
- ∞ El uso de un framework de desarrollo.

3. CALENDARIO DE PROYECTO

Cuando se crea un calendario de proyecto de software, se comienza con un conjunto de tareas (la estructura de la distribución de trabajo). Si se usan herramientas automatizadas, la distribución del trabajo se ingresa como una red o esbozo de tareas. Luego se ingresa duración y fecha de inicio de cada tarea. Además, las tareas pueden asignarse a individuos específicos.

Como consecuencia de esta entrada se genera un cronograma, también llamado Gráfica de Gantt. Es posible desarrollar un cronograma para todo el proyecto. Alternativamente, pueden generarse gráficos separados para cada módulo del proyecto.

Para el sistema LMP se generó un calendario general de toda la aplicación. La Figura 3.3, muestra el cronograma general de la aplicación

Para darle seguimiento al calendario se utilizaron diferentes estrategias de monitoreo y control. Entre esas formas se encuentran:

- ∞ Se realizaron reuniones periódicas del estado del proyecto, en las que se comentaron reportes, avances y problemas.
- ∞ Se evaluaron resultados de todas las revisiones realizadas.
- ∞ Revisamos si los tiempos se cumplieron en la fecha prevista.
- ∞ Al ser un equipo de pocas personas, hubo mucha relación en cuanto a los estatus de los procesos.

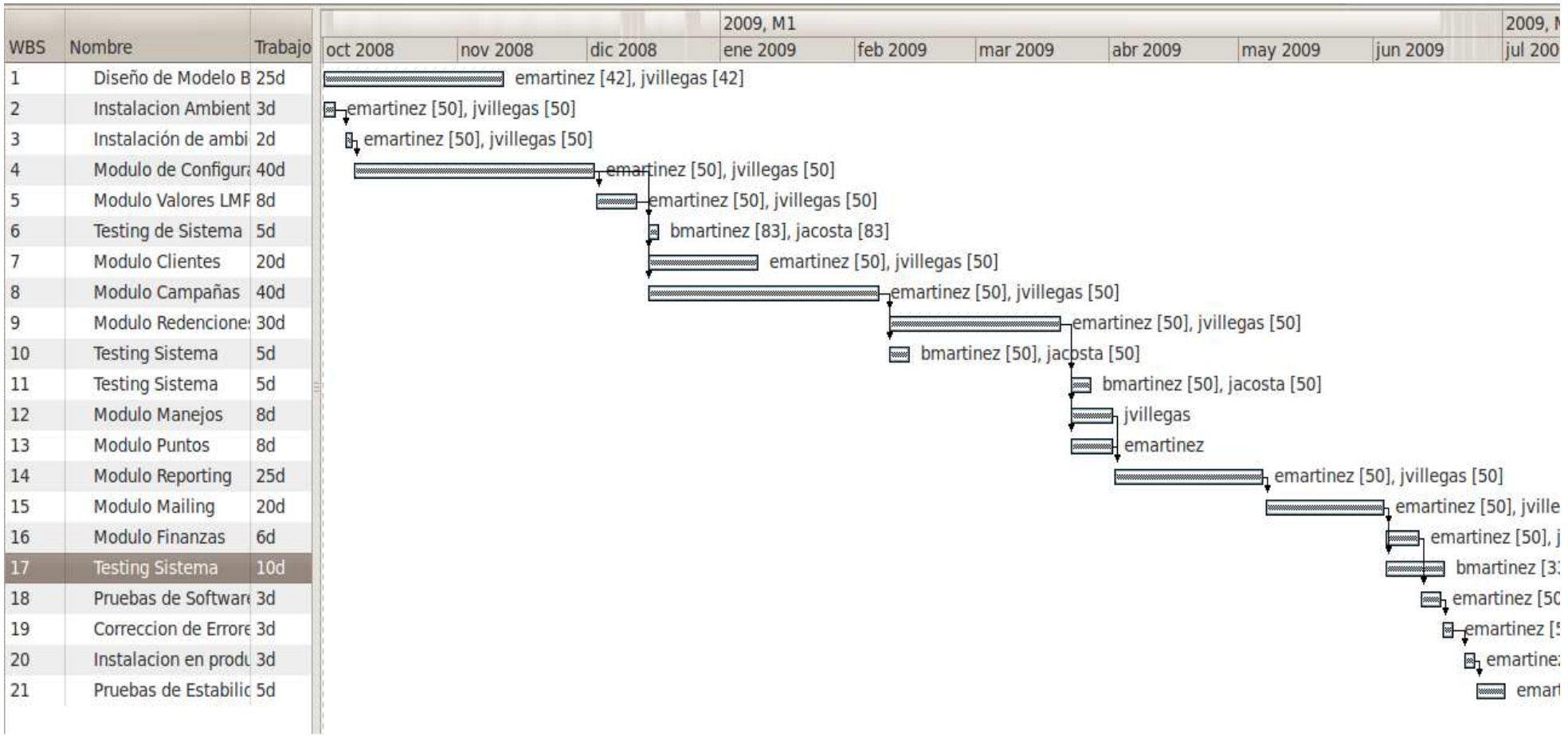


Figura 3.3 Cronograma de Tareas LMP

DISEÑO DE LA BASE DE DATOS

Después de realizar el cronograma de actividades, tal y como se proyectó, se realizó el diseño de la base de datos que alojaría el modelo del sistema LMP.

Un sistema de bases de datos es una colección de datos interrelacionados y un conjunto de programas que permiten a los usuarios tener acceso a esos datos y modificarlos. Una de las principales finalidades de los sistemas de bases de datos es ofrecer a los usuarios un visión abstracta de los datos. Es decir el sistema oculta ciertos detalles del modo en el que se almacenan y mantienen los datos.

Bajo la estructura de las bases de datos se encuentran el modelo de datos: una colección de herramientas conceptuales para describir los datos, sus relaciones, su semántica y las restricciones de consistencia.

Los modelos de datos pueden clasificarse en cuatro categorías diferentes:

- **Modelo relacional.** El modelo relacional usa una colección de tablas para representar tanto sus datos como sus relaciones. Cada tabla tiene varias columnas, y cada columna tiene un nombre único. El modelo relacional es un ejemplo de un modelo basado en registros. Los modelos basados en registros se denominan así porque la base de datos se estructura en registros de formato fijo de varios tipos. Cada tabla contiene registros de un tipo de dado. Cada tipo de registro define un número fijo de campos, o atributos. Las columnas de la tabla se corresponden con los atributos del tipo registro. El modelo de datos relacional es el modelo más ampliamente usado, y una gran mayoría de sistemas actuales se basan en el modelo relacional.
- **El modelo entidad-relación.** El modelo de datos entidad-relación (E-R) se basa en una percepción del mundo real que consiste en una colección de objetos básicos, denominados entidades, y de las relaciones entre ellos. Una entidad es una "cosa" u "objeto" del mundo real que es distinguible de otros objetos. El modelo entidad-relación se usa mucho en el diseño de base de datos.
- **Modelo de datos orientado a objetos.** El modelo de datos orientado a objetos es otro modelo de datos que está recibiendo atención creciente. El modelo orientado a objetos se puede considerar como una extensión del modelo E-R con los conceptos de la encapsulación, los métodos y la identidad de los objetos.
- **Modelo de datos semiestructurados.** El modelo de datos semiestructurados permite la especificación de datos donde los elementos de datos individuales del mismo tipo pueden tener diferentes conjuntos de atributos. Esto lo diferencia de los modelos de datos mencionados anteriormente, en los que cada elemento de datos de un tipo particular debe tener el mismo conjunto de atributos. El lenguaje de marcas extensible (XML) se emplea mucho para representar datos semiestructurados.

El **modelo de datos de red** y el **modelo de datos jerárquico** precedieron cronológicamente al relacional. En la actualidad son muy poco usados, salvo en bases de datos muy antiguas.

Los sistemas de bases de datos proporcionan un lenguaje de definición de datos para especificar el esquema de la base de datos y un lenguaje de manipulación de datos para expresar las consultas y las modificaciones a la base de datos. En la actualidad el más usado es el lenguaje SQL (Lenguaje Estructurado de Consultas).

En la tabla 3.15 se realiza una comparativa entre los modelos orientados a objetos y el modelo entidad relación.

Modelo de BD	Características	Ventajas	Desventajas
Modelo Orientado a Objetos	Un sistema de BDOO debe satisfacer dos criterios: ∞Debe tener un BDMS(Sistema Manejador de Base de Datos) ∞Debe ser un sistema OO Por ejemplo: para la extensión posible este debe ser consistente en los actuales lenguajes de programación OO ∞ El primer criterio se traduce en 5 características como son: Persistencia, Manejador de almacenamiento secundario, Concurrencia, Recuperación, y Facilidad de Query. ∞ La Segunda se traduce en 8 características: Objetos Complejos, Identidad del objeto, Encapsulación, Tipos ó Clases, Sobre paso con combinación retrasada, Extensibilidad.	∝ La clave que posee la BDOO es el poder que confieren al diseñador para especificar tanto la estructura de objetos complejos como las operaciones que se pueden aplicar a esos objetos. ∞Está su flexibilidad, y soporte para el manejo de tipos de datos complejos. Esto presenta la ventaja adicional que una BDOO puede ajustarse a usar siempre el espacio de los campos que son necesarios, eliminando espacio desperdiciado en registros con campos que nunca usan. ∞La segunda ventaja de una BDOO, es que manipula datos complejos en forma rápida y ágilmente. La estructura de la base de datos está dada por referencias (o apuntadores lógicos) entre objetos	∝ Al considerar la adopción de la tecnología orientada a objetos, la inmadurez del mercado de BDOO constituye una posible fuente de problemas por lo que debe analizarse con detalle la presencia en el mercado del proveedor para adoptar su producto en la línea de producción sustantiva. ∞El segundo problema es la falta de estándares en la industria orientadas a objetos. Sin embargo, el "Grupo Manejador de Objetos" (OMG), es una Organización Internacional de Proveedores de Sistemas de Información y usuarios dedicada a promover estándares para el desarrollo de aplicaciones y sistemas orientados a objetos en ambiente de cómputos de red. La implantación de una nueva tecnología requiere que los usuarios iníciales acepten cierto riesgo.
Modelo Entidad-Relación	∞ El modelo de datos entidad-relación (E-R) se basa en una percepción del mundo real que consiste en una colección de objetos básicos, denominados entidades, y de las relaciones entre ellos. Una entidad es una "cosa" u "objeto" del mundo real que es distinguible de otros objetos. El modelo entidad-relación se usa mucho en el diseño de base de datos.	∝ Provee herramientas que garantizan evitar la duplicidad de registros. ∞Garantiza la integridad referencial, así, al eliminar un registro elimina todos los registros relacionados dependientes. ∞Favorece la normalización por ser más comprensible y aplicable.	∝ Presentan deficiencias con datos gráficos y multimedia. ∞No se manipulan de forma manejable los bloques de texto como tipo de dato.

Si bien las ventajas en una base de datos Orientada a Objetos la hacen candidata para implementación, pero debido a su inmadurez en el mercado no fue una opción viable para el desarrollo de la aplicación LMP, mientras que por otro lado el modelo relación

actualmente usado en muchos sistemas web nos permite contar con un número mayor de recursos para su diseño y desarrollo. Por otro lado dentro del desarrollo del sistema LMP utilizando como Zend Framework, este al estar construido con un paradigma Orientado Objetos nos permitió implementar un desarrollo orientado a objetos a nivel de código de la aplicación, utilizando una base de datos relacional para almacenamiento de la información.

Por lo que para esta aplicación el quipo de desarrollo optó por elegir un modelo relacional para el diseño de la base de datos del sistema LMP.

EL MODELO ENTIDAD-RELACIÓN

El modelo de datos entidad-relación (E-R), está basado en una percepción del mundo real que consiste en un conjunto de objetos básicos, denominados entidades, y de las relaciones entre esos objetos. Una entidad es una "cosa" u "objeto" del mundo real que es distinguible de otros objetos.

Las entidades se describen en las bases de datos mediante un conjunto de **atributos**. Una **relación** es una asociación entre varias entidades. El conjunto de todas las entidades del mismo tipo, y el conjunto de todas las relaciones del mismo tipo se denominan, respectivamente, conjunto de entidades y conjunto de relaciones.

La estructura lógica general (esquema) de la base de datos se puede expresar gráficamente mediante un diagrama E-R, que está constituido por los siguientes componentes.

- **Rectángulos**, que representan conjuntos de entidades.
- **Elipses**, que representan atributos.
- **Rombos**, que representan conjuntos de relaciones entre miembros de varios conjuntos de entidades.
- **Líneas**, que unen los atributos con los conjuntos de entidades entre sí, y también los conjuntos de entidades con las relaciones.

En la Figura 3.4 se muestra un extracto del diagrama entidad relación del sistema LMP

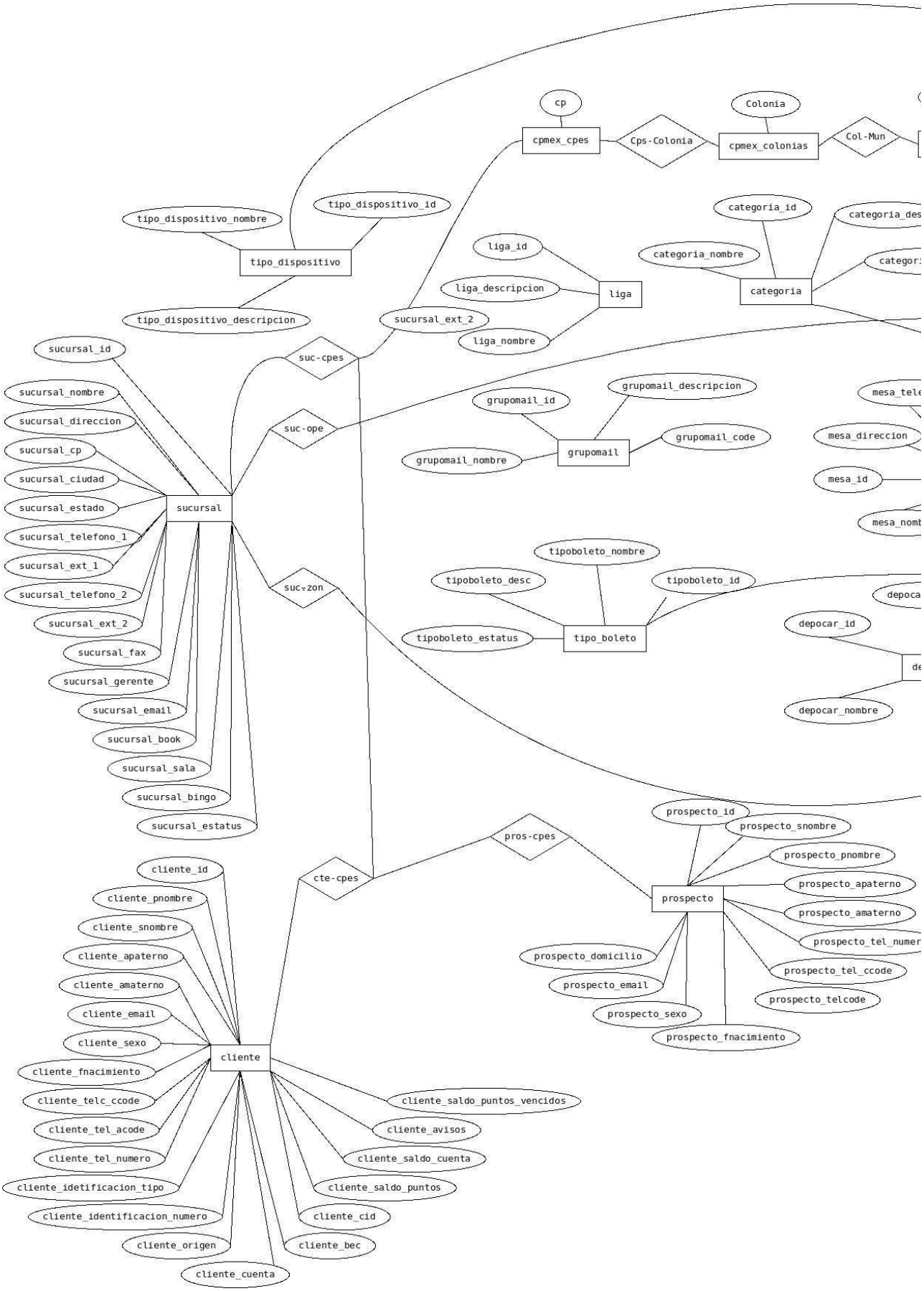


Figura 3.4 Extracto del módelo entidad relación Sistema LMP y Tarjeta BEC

FORMAS NORMALES

Primera Forma Normal o 1FN:

Un dominio es atómico si se considera que los elementos de ese dominio son unidades indivisibles. Se dice que el esquema de la relación R está en la primera forma normal (1FN) si los dominios de todos sus atributos son atómicos.

Abreviada como 1FN, se considera que una relación se encuentra en la primera forma normal cuando cumple lo siguiente:

- ∞ Las celdas de las tablas poseen valores simples y no se permiten grupos ni arreglos repetidos como valores, es decir, contienen un solo valor por cada celda.
- ∞ Todos los ingresos en cualquier columna (atributo) deben ser del mismo tipo.
- ∞ Cada columna debe tener un nombre único, el orden de las columnas en la tabla no es importante.
- ∞ Dos filas o renglones de una misma tabla no deben ser idénticas, aunque el orden de las filas no es importante.
- ∞ Por lo general la mayoría de las relaciones cumplen con estas características, así que podemos decir que la mayoría de las relaciones se encuentran en la primera forma normal.

Segunda Forma Normal o 2FN:

Se dice que un atributo o conjunto de atributos tiene dependencia funcional de otro u otros si a cada uno de los primeros le corresponde sólo uno de los segundos.

Una tabla está en Segunda Forma Normal o 2FN cuando está en 1FN y todo atributo que no pertenece a la clave primaria tiene una dependencia funcional de la clave completa y no de parte de ella. Luego, si la clave principal está formada por un solo atributo y ya está en 1FN, ya estará en 2FN.

Para transformar una tabla con dependencias funcionales, cuya clave está formada por más de un campo, en una tabla en 2FN se necesitan crear tablas nuevas para eliminar las dependencias funcionales, las tablas nuevas tendrán los atributos que dependen funcionalmente de la clave y los que forman la parte de la clave de la que dependen. Una vez creadas las nuevas tablas, se eliminan de la tabla primera los atributos que tenían dependencias funcionales.

Tercera Forma Normal o 3FN:

Se dice que hay dependencia funcional transitiva entre dos atributos cuando un atributo que no pertenece a la clave primaria permite conocer el valor de otro atributo.

Una tabla está en Tercera Forma Normal o 3FN si está en 2FN y no existen atributos que no pertenezcan a la clave primaria que puedan ser conocidos mediante otro atributo que no forma parte de la clave primaria, es decir, no hay dependencias funcionales transitivas.

Forma Normal de Boyce-Codd o FNBC:

Una tabla está en Forma Normal de Boyce-Codd o FNBC si solo existen dependencias funcionales elementales que dependan de la clave primaria o de cualquier clave alternativa. Si la clave primaria está formada por un solo atributo y está en 3FN, ya está en FNBC.

Cuarta Forma Normal o 4FN:

Existe dependencia funcional multivalorada o de múltiples valores si, dados tres atributos de una tabla, si para cada valor del primer atributo existen múltiples valores en el segundo atributo y no hay ninguna relación entre el tercer atributo y el primero, a no ser a través del segundo atributo.

Una tabla está en Cuarta Forma Normal o 4FN si está en FNBC y las únicas dependencias funcionales multivaloradas que existen son las dependencias funcionales de la clave con los atributos que no forman parte de la misma. Estas dependencias multievaluadas de la clave con los atributos que no forman parte de la misma son dependencias triviales, por lo que algunos autores dicen que no existen dependencias multievaluadas en 4FN.

Quinta Forma Normal o 5FN:

Una tabla esta en Quinta Forma Normal (5FN) o Forma Normal de Proyección-Unión si está en 4FN y las únicas dependencias que existen son las dependencias de unión de una tabla con sus proyecciones relacionándose entre las distintas proyecciones mediante la clave primaria o cualquier clave alternativa. La 5FN se emplea cuando en una misma tabla tenemos mucha información redundante, con pocos atributos o cuando una tabla posee una gran cantidad de atributos y se hace por ello inmanejable.

Para conseguir que una tabla 4FN con gran cantidad de atributos esté en 5FN, se parte la tabla original en tantas tablas como se desee, teniendo cada una de ellas en común con las demás los campos que forman la clave primaria en la tabla original.

FORMA NORMAL DEL SISTEMA LMP

El modelo de base de datos para el sistema contiene tablas con FN (Formas Normales) correspondientes a la segunda y tercera forma normal, ya que aunque lo ideal es llegar una tercera forma normal o superior, en algunos casos para evitar que la base creciera con tablas de poca relevancia (P.e. Como podría ser una tabla que contuviera los tipos de sexo) esta información se mantuvo dentro de la misma tabla.

Por ejemplo en algunas tablas del sistema LMP se utilizó un campo de estatus, el cual podría a su vez ser susceptible a pertenecer a una tabla propia pero se decidió no

crearla debido a que solo contendría dos registros, en este caso activo e inactivo, por lo que la información de este campo se definió como 1 ó 0, describiendo su significado en el diccionario de datos.

El fin de este tipo de decisiones de diseño fue tomado en base de que al momento de realizar una consulta para la elaboración de algún reporte, esta se podría tomar una complejidad elevada para quien la diseñe.

En el Anexo 1 se describe el diccionario de datos de la base del sistema LMP para proveer de la documentación completa en caso de que se deseen realizar futuras modificaciones a la estructura de la base de datos actual.

PROGRAMACIÓN DEL SISTEMA LMP

Una vez definidos los módulos del sistema, los casos de uso de los procesos y el modelo de datos del sistema LMP, como siguiente paso el equipo de desarrollo inicio la tarea de codificación de la aplicación.

Para iniciar el desarrollo de la webapp realizamos la instalación de un ambiente base de programación el cual nos permitiría trabajar en conjunto. Este ambiente de desarrollo incluye la instalación de:

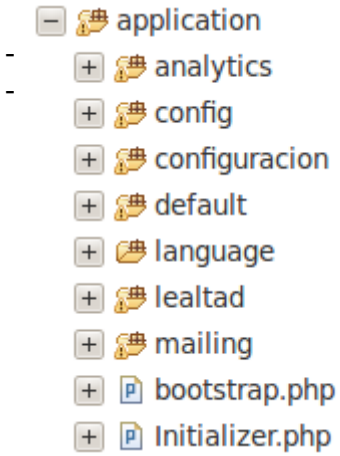
- Servidor web y de base de datos: Se utilizo como servidor web Apache 2 con PHP 5.1 y MySQL 5.0.
- Zend Framework: Framework de código abierto para desarrollar aplicaciones web y servicios web con PHP 5. ZF es una implementación que usa código 100% orientado a objetos
- Manejador de versiones: Se instalo el manejador de versiones Subversion la cual es una aplicación destinada a llevar el control de los cambios y de las diferentes versiones que hacemos a un archivo.

En estos procesos mi participación fue realizar la instalación del framework de desarrollo que incluyo:

- Configuración de Módulos
- Configuración de parámetros base de la aplicación
- Conexión a base de datos.

En el anexo 2 se describen los conceptos básicos para instalar y configurar el framework.

La aplicación LMP se configuro con la estructura mostrada en la figura 3.5:



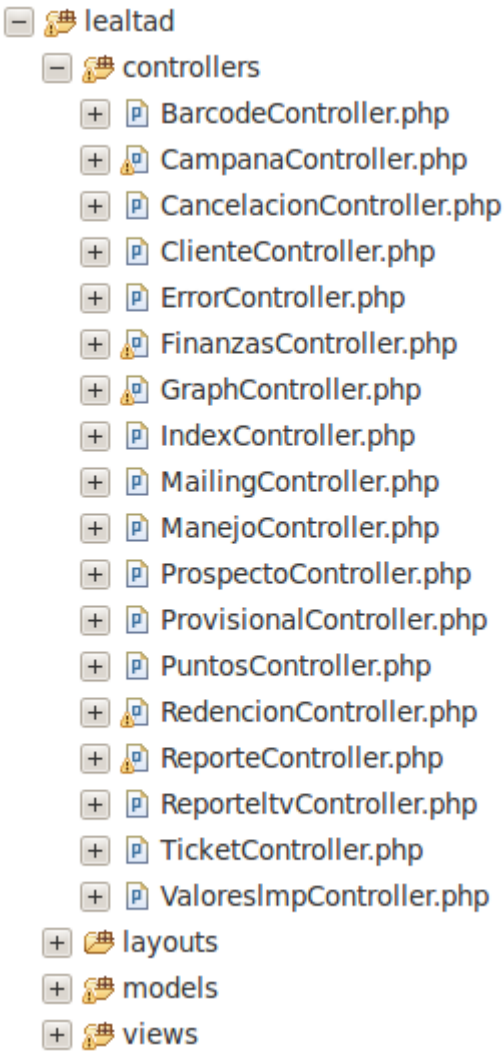
- Application: El directorio base de toda la aplicación web.
- Analitycs: Directorio que contiene a los controladores de reporte de preferencias de los manejos del cliente.
- Config: Configuraciones base de la aplicación como datos de conexión de base de datos, datos de cuentas de correo, preferencias etc.

Figura 3.5 Estructura de la aplicación LMP

- Configuración: Directorio con los controladores de configuración del sistema.
- Default: Directorio accesible desde la web sin públicamente.

Lealtad: Directorio que contiene a los módulos propios del programa de lealtad.
Mailing: Contiene el modulo para envío y recepción de correo electrónico.

Esta es la estructura del sistema de ahí iniciamos con el desarrollo de la aplicación, pondremos como ejemplo el directorio de lealtad mostrado en la Figura 3.6.



3.6 Estructura de Directorio Lealtad

Cada controlador corresponde a un módulo del sistema LMP, a nivel de lenguaje cada uno de ellos es una clase por lo cual tienen características propias de la POO entre las cuales están:

- **Abstracción**: denota las características esenciales de un objeto, donde se capturan sus comportamientos. Cada objeto en el sistema sirve como modelo de un "agente" abstracto que puede realizar trabajo, informar y cambiar su estado, y "comunicarse" con otros objetos en el sistema sin revelar cómo se implementan estas características. Los procesos, las funciones o los métodos pueden también ser abstraídos y cuando lo están, una variedad de técnicas son requeridas para ampliar una abstracción. El proceso de abstracción permite seleccionar las características relevantes dentro de un conjunto e identificar comportamientos comunes para definir nuevos tipos de entidades en el mundo real. La abstracción es clave en el proceso de análisis y diseño orientado a objetos, ya que mediante ella podemos llegar a armar un conjunto de clases que permitan modelar la realidad o el problema que se quiere atacar.

- **Encapsulamiento**: Significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la cohesión de los componentes del sistema. Algunos autores confunden este concepto con el principio de ocultación, principalmente porque se suelen emplear conjuntamente.

- **Modularidad**: Se denomina Modularidad a la propiedad que permite subdividir una aplicación en partes más pequeñas (llamadas módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes. Estos módulos que se puedan compilar por separado, pero que tienen conexiones con otros módulos. Al igual que la encapsulación, los lenguajes soportan la Modularidad de diversas formas.

- **Principio de ocultación**: Cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una interfaz a otros objetos que específica cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas, solamente los propios métodos internos del objeto pueden acceder a su estado. Esto asegura que otros objetos no pueden cambiar el estado interno de un objeto de maneras inesperadas, eliminando efectos secundarios e interacciones inesperadas. Algunos lenguajes relajan esto, permitiendo un acceso directo a los datos internos del objeto de una manera controlada y limitando el grado de abstracción. La aplicación entera se reduce a un agregado o rompecabezas de objetos.

- **Polimorfismo**: comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre, al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando. O dicho de otro modo, las referencias y las colecciones de objetos pueden contener objetos de diferentes tipos, y la invocación de un comportamiento en una referencia producirá el comportamiento correcto para el tipo real del objeto referenciado. Cuando esto ocurre en "tiempo de ejecución", esta última característica se llama asignación tardía o asignación dinámica.

- **Herencia**: las clases no están aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. La herencia organiza y facilita el polimorfismo y el encapsulamiento permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir (y extender) su comportamiento sin tener que volver a implementarlo. Esto suele hacerse habitualmente agrupando los objetos en clases y estas en árboles o enrejados que reflejan un comportamiento común. Cuando un objeto hereda de más de una clase se dice que hay herencia múltiple.

Estas características en conjunto a las ventajas de utilizar el modelo MVC que ofrece Zend Framework nos permiten realizar un desarrollo modular y con un paradigma de programación totalmente orientado a objetos.

Para ejemplificar cómo se combina el uso de las librerías de Zend Framework, con la programación orientada a objetos y el modelo MVC tomaremos como referencia el módulo de cliente. Cada controlador contiene métodos públicos y privados que permitirán la interacción con la aplicación. La Figura 3.7 muestra los métodos de la clase Lealtad_ClienteController.

Figura 3.7 Métodos de la clase Lealtad_ClienteController

Cada controlador tiene tres métodos que se invocan cuando hacemos una petición a algún controller en particular.

Acción indexAction: ZF (Zend Framework) designado como método por default al realizar una petición al controlador.

Acción preDispatch: Método el cual es llamado antes de que una acción sea ejecutada

Acción init: Este es llamado como parte del constructor y nos ayuda a inicializar el controlador.

PLAIN TEXT

PHP:

```
1. /**
2.  * ClienteController
3.  *
4.  * @author proveedor
5.  * @version 1.0
6.  */
7. class Lealtad_ClienteController extends Zend_Controller_Action
8. {
9.     /**
10.      * The default action - show the home page
11.      */
12.     public function preDispatch ()
13.     {
14.         if (isset($_POST['nombre']) || isset($_POST['apaterno']) ||
    isset($_POST['apaterno']) || isset($_POST['snombre'])) {
15.             $this->_request->setParam('pagina', 1);
16.         }
17.     }
18.
19.     public function init ()
20.     {
21.         $ajaxContext = $this->_helper->getHelper('AjaxContext');
22.         $ajaxContext->addActionContext('cpmex', 'json');
23.         $ajaxContext->initContext();
24.     }
25.     public function indexAction ()
26.     {
27.
28.       …….
29.
```

El código cuando se llama a Lealtad_ClienteController la acción predispatch, revisa si existen variables en POST, de ser así asigna a la variable pagina el valor de 1. Posteriormente inicializa la acción cpmex con un contexto json, para hacer llamadas tipo ajax, finalmente se hace la llamada a la acción IndexAction que a su vez tiene salida en un vista con el mismo nombre index.phtml.

Uno de los módulos del cual estuve a cargo de la codificación fue el módulo de LoginController. Este módulo es uno de los más delicados e importantes debido a que la seguridad de la información depende de quién tiene acceso a la aplicación. Los elementos de seguridad que se incluyeron en este módulo son los listados en la Tabla 3.6.

Tabla 3.6 Elementos de Seguridad en LoginController

Elemento de Seguridad	Descripción
Contraseña	La seguridad de la contraseña incluye: - No permitir contraseña que se puedan encontrar fácilmente en un diccionario. - Cifrado MD5 - Agregar una cadena aleatoria adicional a la contraseña llamada salt.
Prevención de ataques CSRF	Los ataques CSRF (del inglés Cross-site request forgery o falsificación de petición en

	sitios cruzados) explotan la confianza que un sitio tiene para un usuario en particular. El sitio es el objetivo del ataque, y el usuario es a la vez víctima y cómplice sin saberlo. Un ataque CSRF fuerza al navegador web validado de una víctima a enviar una petición a una aplicación web vulnerable, la cual entonces realiza la acción elegida a través de la víctima. Hay algunas acciones que se puede tomar para reducir el riesgo de ataques CSRF. Estas medidas menores incluyen el uso de POST en lugar de GET en los formularios HTML que llevan a cabo las acciones, con $ _POST en lugar de $ _REQUEST. Lo más importante que se puede hacer es tratar de forzar el uso de sus propios formularios. Esta validación se realiza mediante un token con un tiempo te vida.
Prevención de ataques de SQL	Inyección SQL es un método de infiltración de código intruso que se vale de una vulnerabilidad informática presente en una aplicación en el nivel de validación de las entradas para realizar consultas a una base de datos. El origen de la vulnerabilidad radica en el incorrecto chequeo y/o filtrado de las variables utilizadas en un programa que contiene, o bien genera, código SQL. Es, de hecho, un error de una clase más general de vulnerabilidades que puede ocurrir en cualquier lenguaje de programación o script que esté embebido dentro de otro. Se conoce como Inyección SQL, indistintamente, al tipo de vulnerabilidad, al método de infiltración, al hecho de incrustar código SQL intruso y a la porción de código incrustado.

Para la codificación de este módulo de utilizaron los siguientes componentes de la librería de ZF:

- **Zend_Form**: Construcción del formulario de Login.
- **Zend_Auth_Adapter_DbTable**: Autenticación en la base de datos.
- **Zend_Session**: Para almacenar los datos de sesión.

∞ **Zend_Db_Table**: mapea la tabla de la base de datos en una clase de PHP.

Tenemos la siguiente estructura dentro del proyecto en la Figura 3.8

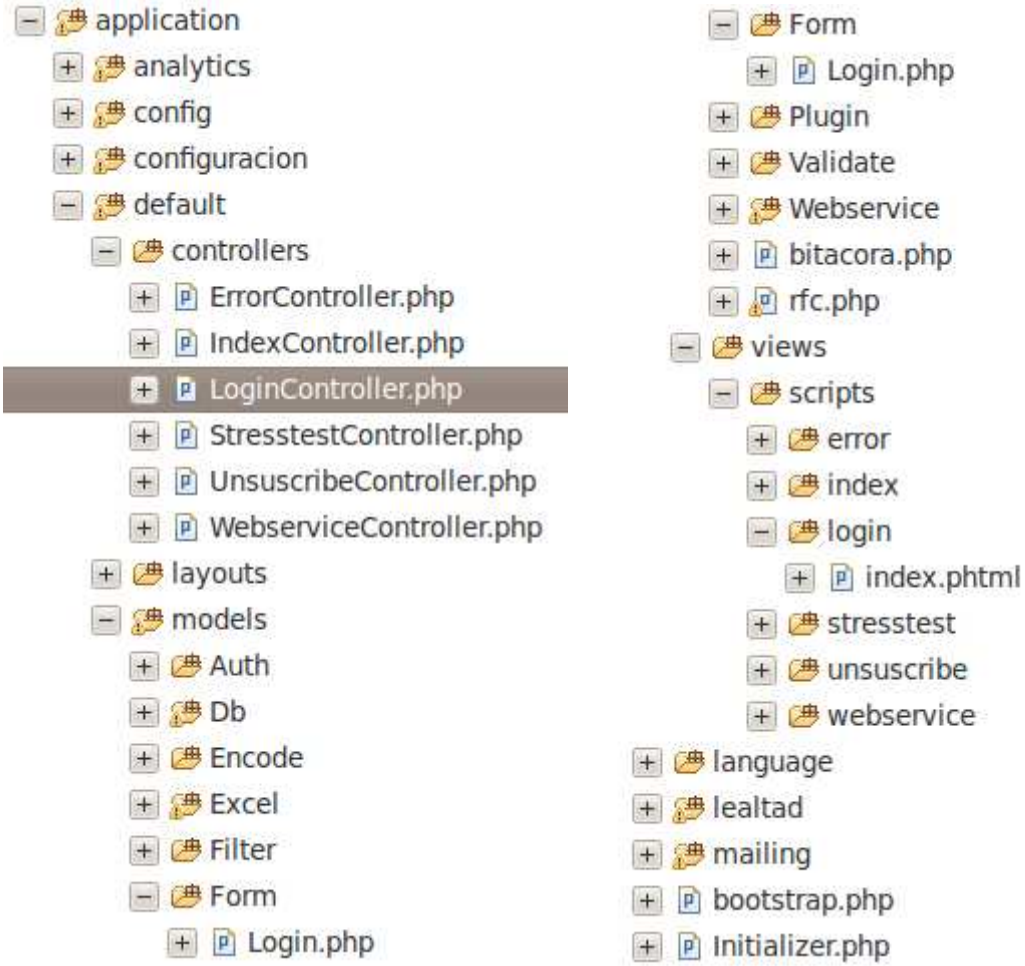


Figura 3.8 Estructura de modulo Login

Iniciamos con la configuración del formulario de login:

```
PLAIN TEXT
```

PHP:

```
1.  <?php
2. class Default_Model_Form_Login extends Zend_Form
3. {
4.  public function init()
5.  {
6.        $username = new Zend_Form_Element_Text(
7.              'username',
8.              array(
9.                  'label'       =>    'loginFormUser',
10.                       'attribs'     =>    array(
11.                           'style'   =>    'width:190px;'
12.                       )
13.                   )
```

```
14.             );
15.
16.             $username->addValidator(new
    Zend_Validate_StringLength(3,15));
17.
18.
19.
20.             $password = new Zend_Form_Element_Password(
21.                   'password',
22.                 array(
23.                     'label'         =>    'loginFormPass',
24.                     'attribs' =>    array(
25.                         'style'     =>    'width:190px;'
26.                     )
27.                 )
28.             );
29.             $password->addValidator(new
    Zend_Validate_StringLength(3,15));
30.
31.             $submit = new Zend_Form_Element_Submit('ingresar',
32.                 array(
33.                     'label'         =>    'Entrar',
34.                     'attribs' =>    array(
35.                         'style'           =>    'background-
    image:url(/resources/imgpg/gradient_tback1.png);width:auto;height:30
    px; background-color:#C00; color:#FFF; font-weight:bold; font-
    family:Arial, Helvetica, sans-serif;padding-top:5pxpadding-
    bottom:5px;padding-left:8px;padding-right:8px;cursor:pointer;'
36.                                 )
37.                 )
38.             );
39.
40.             $token = new Zend_Form_Element_Hash('token');
41.             $token->setSalt(md5(uniqid(rand(), TRUE)));
42.             $token->setTimeout(60);
43.
44.             $this->addElements(array(
45.                   $username,
46.                   $password,
47.                   $submit,
48.                   $token
49.             ));
50.
51.             $translator = Zend_Registry::get('trans');
52.             $this->setTranslator($translator);
53.
54.             foreach ($this->getElements() as $element) {
55.                   $element->addFilter('StringTrim');
56.             }
57.
58.       }
59.
60. }
```

La clase Login extiende a la clase Formulario de ZF, lo que nos permite crear elementos con características pro default por ejemplo en el código anterior de las líneas 6 a la 14, creamos un elemento input de tipo texto, al cual podemos asignarle valores por default, como estilos, id y clases.

Este estilo de crear los formularios permite que sean entendibles y fácilmente modificables cuando haya que dar mantenimiento a la aplicación. Además de asignar validadores predefinidos en ZF como en la línea 16, en la cual asignamos un validador de longitud de cadena a elemento 'username'.

Para asegurar que la petición de login proviene de nuestro formulario utilizamos un token de validación con una duración de 60 seg, lo que nos permite evitar ataques CSRF como se muestra en el código siguiente.

```
PLAIN TEXT

PHP:

1.  $token = new Zend_Form_Element_Hash('token');
2.  $token->setSalt(md5(uniqid(rand(), TRUE)));
3.  $token->setTimeout(60);
```

Por lo que si algún atacante quisiera realizar un request externo en el login del sistema debería contar con un token válido.

De esta forma se fue construyendo la aplicación LMP siempre con base al diseño previo de los módulos y con revisiones técnicas periódicas de tipo formal e informal, además de utilizar codificación estructurada que se especifica en el Anexo 3.

Las revisiones informales incluyen una simple verificación de escritorio de un módulo desarrollado, hecha con parte del equipo de desarrollo de forma casual.

Además de revisiones técnicas formales (RTF) cuyos objetivos son:

1) Descubrir los errores de funcionamiento, lógica o implementación de cualquier representación de software.
2) Verificar que el software que se revisa cumple sus requerimientos.
3) Garantizar que el software está representado de acuerdo a estándares predefinidos por la empresa.
4) Obtener software de desarrollo de manera uniforme.
5) Hacer el proyecto manejable y alcanzable.

Durante esta etapa de diseño y desarrollo de la base de datos pude aplicar los conocimientos obtenidos en materias como:

∞ Programación Orientada a Objetos: Aplique conceptos de la POO, características de clases y objetos además de las relaciones entre ellas.

∞ Base de Datos: Desarrollo del modelo E-R, así como su optimización en las formas normales.

∞ Algebra Lineal: Utilice conocimientos de espacios y relaciones entre arreglos.