UNIVERSIDAD NACIONAL AUTÓNOMA
DE MÉXICO

FACULTAD DE CIENCIAS

PLATAFORMA GENÉRICA PARA AUTOMATIZAR LAS TAREAS DE UNA INSTITUCIÓN BANCARIA

REPORTE DE TRABAJO PROFESIONAL

QUE PARA OBTENER EL TÍTULO DE:

LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A:

EDUARDO GONZÁLEZ MORENO

TUTORA:

M. EN C. MARÍA GUADALUPE ELENA
IBARGÜENGOITIA GONZÁLEZ

2015

Hoja de Datos del Jurado

1. Datos del alumno
González
Moreno
Eduardo
55 2845 4033
Universidad Nacional Autónoma de México
Facultad de Ciencias
Ciencias de la Computación
302082656

2. Datos del tutor
M en C
María Guadalupe Elena
Ibargüengoitia
González

3. Datos del sinodal 1
Dra
Bibiana
Obregón
Quintana

4. Datos del sinodal 2
M en I
Miguel Ehécatl
Morales
Trujillo

5. Datos del sinodal 3
Dra
Hanna Jadwiga
Oktaba

6. Datos del sinodal 4
L A
Miguel Ángel
González
López

7. Datos del trabajo escrito
Plataforma genérica para automatizar las tareas de una institución bancaria
78 p
2015

AGRADECIMIENTOS

A la **Facultad de Ciencias** de la Universidad Nacional Autónoma de México (**UNAM**), no solo como institución educativa sino también a sus excelentes profesores que de una u otra manera colaboraron para la obtención de mi grado académico.

A la **Institución Bancaria** donde actualmente ejerzo mi profesión, por el tiempo, conocimiento y sobretodo la confianza que me brindó para la realización de este trabajo.

A los magníficos sinodales que conformaron mi **Jurado de Grado**:

M. en C. María Guadalupe Elena Ibargüengoitia González

Dra. Bibiana Obregón Quintana

M. en I. Miguel Ehécatl Morales Trujillo

Dra. Hanna Jadwiga Oktaba

L. A. Miguel Ángel González López

AGRADECIMIENTOS A TÍTULO PERSONAL

A mi padres **María Magdalena Moreno González** y **Fermín Eduardo González Rocha**. A mis hermanas **Verónica Alejandra** y **Karla Gabriela**†.

Gracias mamá por siempre darme tu amor desinteresado, aunque haya cometido errores y me haya equivocado muchas veces tu siempre me diste tu apoyo y ayuda, siempre confiando en mí en momentos que ni siquiera yo creía en mí mismo. Gracias por todas las cosas que me has enseñado y por todos los sacrificios que haces por mí, eres una mujer maravillosa.

Gracias papá por haberme dado de tu inteligencia y compartido tus experiencias, por los valores que has inculcado en mi vida que han hecho de mí una mejor persona. Siempre he sabido que has luchado para darme lo mejor que has podido, por eso de esta manera hoy te lo agradezco desde el fondo de mi corazón, te amo mucho.

A los integrantes de mi **Jurado de Grado**, por la conducción de este trabajo, sus permanentes y valiosas sugerencias que han permitido enriquecer el escrito a través de la discusión de los avances, pero sobre todo por el honor de aceptar ser parte de mi jurado, creer en mí y ayudarme a concluir este logro personal.

Agradezco la luz, el amor y la paciencia de **Norma Ivonne Gutierrez Vega**, de quien he recibido un apoyo gigantesco y ha sido el motivo para que iniciara, y finalmente pudiera terminar este trabajo. Con el paso del tiempo te has convertido en una persona importante y muy especial en mi vida.

A mis compañeros de oficina **Horacio Cruz Ávila**, **Carlos Ignacio Irigoyen Urdapilleta**, **Paula Venegas Solís** y **Bernardo Rubio Ávila**, cuya paciencia y enseñanzas, han aportado en gran medida el desarrollo de éste trabajo. Gracias por compartir los buenos momentos tanto en lo profesional como en lo personal que han enriquecido y han hecho placenteras las largas jornadas de trabajo.

A mis amigos y compañeros de la Facultad de Ciencias, **Cesar Eduardo Nieto González** y **Darío Emmanuel Vázquez Ceballos**, que siempre fueron un fuerte aliado durante mi estadía en la universidad y de quienes obtuve todo su apoyo siempre de manera incondicional.

Por último agradezco a los profesores, amigos, compañeros y a todos aquellos que han estimulado y alentado mi desarrollo intelectual.

A LA MEMORIA DE MI HERMANA

Karla Gabriela González Moreno

(1987-2010)

Contenido

Introducción

Dada las crecientes necesidades que en diferentes ámbitos una institución bancaria debe cumplir por motivos de seguridad, disposiciones, legalidad, entre otras, se vuelve imprescindible el contar con sistemas de información gerencial oportuna para la toma de decisiones de forma rápida y precisa para poder proveer a las instituciones reguladoras de reportes normativos precisos.

El uso de sistemas computacionales se vuelve la solución más deseada ya que la elaboración manual por parte del recurso humano podría implicar una mayor cantidad de tiempo para generarlos, además de introducir errores en la gestión y generación de los mismos dados los niveles de operaciones que maneja la institución.

Por supuesto, no todas las instituciones bancarias cuentan con el suficiente presupuesto para costear los diferentes sistemas que pudieran verse obligados a adquirir a fin de lograr dichos objetivos. Además de que, en su gran mayoría, ningún sistema del mercado se adapta o entiende completamente las necesidades de la empresa o institución, debido a que los diseños no son ex profeso a éstas y, en su lugar, la implementación y adecuación de los sistemas adquiridos involucran un gasto excesivo de capital, haciendo que su uso quede relegado y resulten una inversión perdida.

Es por ello que el presente trabajo, muestra cómo se desarrolló una plataforma tecnológica genérica que sirve como base y marco de trabajo en la construcción de los diferentes sistemas computacionales que requiere la institución bancaria, utilizando para ello solo al personal del área de sistemas interna y así evitar la compra o soporte de terceras partes que pudiesen derivar en altos costos de mantenimiento o actualización de los sistemas. Por motivos de confidencialidad y seguridad, se ha omitido el nombre de la entidad donde se realizó el desarrollo e implementación de la plataforma y sistema, y a quien solo puedo agradecer la confianza que esta institución bancaria me brindó para la realización de este trabajo, el cual aún sigue en uso y constante desarrollo dentro de la institución.

Para el sistema bancario de México el tema del riesgo operacional va ligado con los escándalos financieros de Enron y WorldCom, aunque su aplicación no se limita a empresas de la magnitud de las antes mencionadas.

Enron era la séptima empresa energética más importante de los Estados Unidos, quebró a inicios del año 2002, entre sus muchas causas se encuentra el fraude interno dado que se ocultó información, además de realizar manipulación contable en la que deberían mostrarse variaciones de la situación patrimonial, ocultando los riesgos y pérdidas de la empresa, lo que se conoce como "creatividad contable" y que se basa en alterar estados de resultados para reflejar ganancias inexistentes y ocultar pérdidas existentes.

WorldCom por otra parte fue una empresa importante de telecomunicaciones en Estados Unidos. A partir de 1999, comenzó a declarar costos menores a los reales y a inflar de más las ganancias en un intento por estabilizar el valor de sus acciones. En 2001, el director general de WorldCom, Bernard Ebbers, tomó prestados más de USD$400 millones para cubrir gastos excesivos en sus otros negocios. Los auditores internos finalmente descubrieron el fraude contable lo que posteriormente llevaría a la quiebra de la empresa en julio de 2002.

Los desastres financieros antes expuestos originaron el desarrollo de normativas internacionales que atenuaron los riesgos a que se encuentra expuesta toda empresa.

Así pues, toda actividad realizada en una empresa, sea chica, mediana o grande, tiene ciertos riesgos de magnitudes diferentes como pudieran ser de tipo operacional, financiero, de mercado, entre otros, que pueden ser mitigados o incluso eliminados si se implementan las medidas de control adecuadas.

Objetivos

Debido a las necesidades de la institución bancaria al no contar con un sistema gestor de riesgo operacional, se emprendieron algunas actividades de las cuales fue necesario determinar los siguientes objetivos:

- Poseer una plataforma tecnológica genérica que para fines prácticos en lo sucesivo nos referiremos a ésta como plataforma genérica, para la construcción de diversos sistemas que la institución requiere para la automatización de sus tareas.

- Proveer al personal de un sistema para capturar eventos de pérdida por riesgo operacional y generación de reportes.

- El sistema debe tener como características básicas las siguientes:

    - Módulo flexible para la creación y modificación de la estructura bancaria a nivel geográfico e institucional.
    - Módulo de agregación, edición y eliminación de usuarios así como de controles de acceso definidos mediante perfiles y facultades.
    - Módulo para editar catálogos referentes al riesgo operacional.

- El sistema debe ser escalable y de fácil mantenimiento.

- Como punto medular, debe ser un sistema libre de licencias, para evitar dependencias de tipo comercial.

Este trabajo contará con 4 capítulos. El primero, presenta una descripción de las necesidades de la institución bancaria, un breve antecedente de los Acuerdos de Basilea, el marco de riesgo operacional, así como el planteamiento del problema de la entidad bancaria para satisfacer dichas necesidades.

El segundo capítulo aborda los conceptos a utilizar y el método aplicado para dar solución al problema de la institución bancaria mediante la construcción de una plataforma genérica.

El capítulo 3 presenta en detalle el diseño y organización de la plataforma genérica desarrollada mediante una descripción técnica de la misma.

Finalmente, el capítulo 4 muestra como configurar un sistema gestor de riesgo operacional construido sobre la plataforma genérica. Además se incluyen los resultados obtenidos en base a los objetivos establecidos y las conclusiones.

Capítulo 1

1. Planteamiento del problema

El Comité de Basilea es la denominación usual con la que se conoce al Comité de Basilea de Supervisión Bancaria (BCBS, sigla de *Basel Committee on Banking Supervision* en inglés), la organización mundial recibe su nombre a partir de la ciudad de Basilea, Suiza, donde el BCBS mantiene su secretariado en la sede del Banco de Pagos Internacionales. El Comité fue establecido en 1975 por los presidentes de los bancos centrales de los once países miembros del Grupo de los Diez (G-10) en aquel momento; actualmente todos los demás países significativos del G-20 están representados, así como algunas de las mayores plazas bancarias como Hong Kong y Singapur.

Desde su surgimiento, el Comité de Basilea se constituyó en un foro de discusión para fomentar la mejora y la convergencia de las prácticas y normativas de supervisión bancaria, buscando perfeccionar las herramientas de fiscalización internacional a fin de fortalecer la solidez de los sistemas financieros, a través de acercamientos y de estándares comunes.

El Comité de Basilea no es una organización multilateral clásica, por lo tanto no tiene autoridad para imponer recomendaciones, si bien la mayor parte de los países así como algunos otros que no forman parte del mismo tienden a implementar las políticas del Comité. Esto significa que las recomendaciones son aplicadas a través de leyes y regulaciones nacionales (o a nivel comunitario en la Unión Europea), antes que como resultado de una recomendación internacional del Comité, de modo que es preciso un cierto período de tiempo desde que se aprueba una recomendación hasta que esta es aplicable a nivel nacional. Entre las normas de importancia que el Comité ha emitido, se encuentran las recomendaciones sobre blanqueo de capitales.

Los Acuerdos de Basilea son aquellos que reúnen las recomendaciones emitidas por éste Comité. Están formados por los acuerdos Basilea I, Basilea II y Basilea III.

En 1994, el Comité de Basilea publicó una guía para la gestión de riesgos de productos derivados[1], documento que contiene la primera definición formal y explícita para riesgo operacional, donde "riesgo operativo" y "riesgo operacional" son considerados como lo mismo. Aquella deficiencia puede explicar la confusión entre ambos términos, a pesar de que no sólo tienen una diferencia semántica sino que también una distinción conceptual, puesto que riesgo operativo contempla principalmente fallos en operaciones internas de una entidad, mientras que el riesgo operacional tiene un ámbito bastante más amplio, se define no solo como la pérdida potencial por fallas o deficiencias en los controles internos sino también por errores en el procesamiento y almacenamiento de las operaciones o en la transmisión de información, así como por resoluciones administrativas y judiciales adversas, fraudes o robos, y comprende, entre otros, al riesgo tecnológico y al riesgo legal.

En septiembre de 1998, el Comité de Basilea publicó el artículo "*Operational Risk Management*"[2], donde se presentan los principales resultados y conclusiones de un estudio sobre el estado del arte de la administración de riesgo operacional en las instituciones financieras. El estudio deja en evidencia cinco problemas que debían abordarse:

1) Una baja o nula frecuencia en la medición y reporte de este riesgo;
2) Pobre o vaga determinación de los factores que lo determinan;
3) Vaga definición de este riesgo;
4) Ausencia de actividades de auditoría interna como uno de los precursores de la gestión del riesgo operacional;

[1] Los derivados son uno de los principales instrumentos financieros que, entre otras cosas, permiten a las personas y empresas anticiparse y cubrirse de los riesgos o cambios que pueden ocurrir en el futuro, a tal manera de evitar ser afectados por situaciones adversas.

[2] El documento puede leerse directamente desde el sitio web del Banco de Pagos Internacionales (BIS; sigla de *Bank for International Settlements* en inglés) es el banco central de bancos centrales con sede en Basilea, Suiza. Mantiene dos oficinas de representación, una en Hong-Kong y otra en Ciudad de México. El enlace directo se encuentra en http://www.bis.org/publ/bcbs42.pdf

5) La necesidad de una definición y marco de buenas prácticas por parte de los supervisores.

Basilea II (el segundo de los Acuerdos) fue publicado inicialmente en junio de 2004 y tiene como propósito la creación de un estándar internacional que sirva de referencia a los reguladores bancarios, con objeto de establecer los requerimientos de capital necesarios para asegurar la protección de las entidades frente a los riesgos financieros y operacionales.

En México, la autoridad financiera que se encarga de supervisar y regular a las instituciones integrantes del sistema financiero mexicano en el rubro de riesgo operacional, es la Comisión Nacional Bancaria y de Valores (CNBV), a fin de procurar su estabilidad y correcto funcionamiento, en protección de los intereses del público.

La CNBV se rige bajo un estricto código de ética el cual tiene como los siguientes valores institucionales; Legalidad, Equidad, Respeto y Lealtad, mismos que la han posicionado como una autoridad eficiente, moderna y respetada que procura la estabilidad del Sistema Financiero Mexicano, acorde con mejores prácticas internacionales.

Es la CNBV quien mediante la Circular Única de Bancos (CUB[3]) define las disposiciones de carácter general aplicables a las instituciones de crédito. En particular la CUB contempla la Serie R28 relativa a la Información de riesgo operacional. Esta serie se comprende de tres[4] subreportes definidos como:

- R28 A-11: Eventos de pérdida por riesgo operacional.
- R28 A-12: Estimación de niveles de riesgo operacional.
- R28 A-13: Actualización de eventos de pérdida por riesgo operacional.

[3] Circular Única de Bancos es el nombre abreviado para Disposiciones de Carácter General Aplicables a las Instituciones de Crédito (DCGAIC).

[4] La Circular Única de Bancos ya contempla un cuarto subreporte (R28 A-14), sin embargo, a la fecha de éste trabajo, dicho subreporte es aún discutido y sigue pendiente en establecer un *layout* oficial que adopten las instituciones financieras.

Debido a las necesidades para atender el riesgo operacional y cubrir las fallas antes mencionadas, la Dirección General de Administración de Riesgos de la entidad bancaria, requiere de un sistema que sea capaz de registrar los diversos eventos de pérdida por riesgo operacional reportados en las diferentes áreas, para así poder dar cumplimiento ante la CNBV en el marco relativo al riesgo operacional.

Por políticas de ahorro (particulares a la institución bancaria), dicho sistema debe ser *"in-house"*, y con proyección de crecimiento, razón por la cual debe estar implementado bajo algún tipo de diseño o estructura que sea capaz de mantenerse de manera ordenada y sin necesidad de recurrir a soportes de proveedores o terceros ajenos a la propia institución. Por tal motivo, consideramos es imprescindible contar con una plataforma genérica sobre la cual puedan crearse y robustecerse los diversos sistemas creados para las diferentes tareas de evaluación del riesgo en la institución y así crecer conforme las necesidades lo requieran.

Las bondades que ofrece una plataforma genérica, se ejemplificarán mediante un sistema gestor de riesgo operacional para la elaboración de reportes que permita al usuario tener acceso a registro y consulta a fin de atender las diferentes necesidades de la institución, así como satisfacer y prevenir con información oportuna los riesgos.

Capítulo 2

2. El patrón de diseño MVC

Vamos a diseñar una plataforma genérica bajo una arquitectura o patrón que por sus siglas en inglés significa: Modelo (*Model*), Vista (*View*), Controlador (*Controller*), en adelante MVC, usando el lenguaje de programación PHP. La creación de ésta plataforma genérica nos permitirá construir sistemas que cubran de necesidades básicas de construcción, tales como la implementación de seguridad, clases e interfaces de controles de acceso a usuarios, entre otras; pero que a su vez tenga la posibilidad de robustecerse conforme a las necesidades lo requieran. Una de las ventajas de hacer esto es el aprendizaje y entendimiento de cómo funcionan este tipo de herramientas diseñadas para hacer sistemas web robustos, seguros, estructurados y escalables.

2.1 ¿Cómo funciona el patrón MVC?

Con la evolución de las capacidades de internet y la demanda de sistemas y aplicaciones, surgió una nueva forma de desarrollar software en PHP basada en el patrón de diseño MVC.

Este modelo de desarrollo propone separar el funcionamiento de un programa en tres componentes principales:

- El Modelo representa la información con la que trabaja el sistema.
- La Vista representa la información del Modelo en páginas web que permiten al usuario interactuar con ella.
- El Controlador se encarga de la lógica del negocio procesando las interacciones del usuario y realizando los cambios apropiados en el Modelo o en la Vista.

La Figura 2.1 ilustra el funcionamiento del patrón MVC.

El patrón MVC separa la lógica de negocio y la presentación, por lo que se consigue un mantenimiento más sencillo sobre los sistemas. Si por ejemplo un mismo sistema debe ejecutarse tanto en un navegador estándar como en un navegador de un dispositivo móvil, solamente es necesario crear una Vista nueva para cada dispositivo; manteniendo el Controlador y el Modelo original.

El Controlador se encarga de aislar al Modelo y a la Vista de los detalles del protocolo utilizado para las peticiones (HTTP, consola de comandos, email, entre otros).

El Modelo se encarga de la abstracción de la lógica relacionada con los datos, haciendo que la Vista y las acciones sean independientes de, por ejemplo, el tipo de gestor de bases de datos utilizado por el sistema.

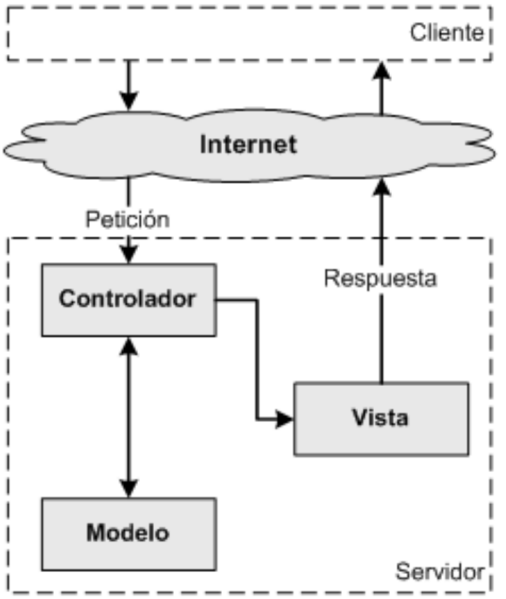

Figura 2.1 El patrón MVC

Para entender mejor las ventajas de utilizar el patrón MVC, transformaremos un sistema realizado con PHP en un sistema que sigue la arquitectura MVC. Abordaremos como ejemplo una lista con las últimas entradas o artículos de un blog.

2.2 Programación

Utilizando solamente PHP normal y corriente, el código necesario para mostrar los artículos almacenados en una base de datos se muestra a continuación:

Código 2.1 - Codificación simple

```
<?php

// Conectar con la base de datos y seleccionarla
$conexion = mysql_connect('localhost', 'miusuario', 'micontrasena');
mysql_select_db('blog_db', $conexion);

// Ejecutar la consulta SQL
$resultado = mysql_query('SELECT fecha, titulo FROM articulos', $conexion);
?>

<html>
  <head>
    <title>Listado de Artículos</title>
  </head>
  <body>
    <h1>Listado de Artículos</h1>
    <table>
      <tr><th>Fecha</th><th>Titulo</th></tr>

<?php

     // Mostrar los resultados con HTML
     while ($fila = mysql_fetch_array($resultado, MYSQL_ASSOC))
     {
       echo "\t<tr>\n";
       printf("\t\t<td> %s </td>\n", $fila['fecha']);
       printf("\t\t<td> %s </td>\n", $fila['titulo']);
       echo "\t</tr>\n";
     }
?>

    </table>
  </body>
</html>

<?php

// Cerrar la conexion
mysql_close($conexion);
?>
```

El código anterior es fácil de escribir y rápido de ejecutar, pero muy difícil de mantener y actualizar. Los principales problemas del código anterior son:

- No existe protección frente a errores (¿Qué ocurre si falla la conexión con la base de datos?).
- El código HTML y el código PHP están mezclados en el mismo archivo e incluso en algunas partes están entrelazados.
- El código solo funciona si la base de datos es MySQL.

2.3 Separando la presentación

Las llamadas a las funciones de PHP *echo* y *printf* del Código 2.1 dificultan la lectura del mismo. De hecho, modificar el código HTML del código anterior para mejorar la presentación es un problema debido a cómo está programado. Para tratar de solventar estos inconvenientes dividiremos el código en dos partes. En primer lugar, el código PHP puro, donde toda la lógica de negocio se incluirá en el código del Controlador, como se muestra en el Código 2.2.

Código 2.2 - La parte del Controlador, en *index.php*

```
<?php

// Conectar con la base de datos y seleccionarla
$conexion = mysql_connect('localhost', 'miusuario', 'micontrasena');
mysql_select_db('blog_db', $conexion);

// Ejecutar la consulta SQL
$resultado = mysql_query('SELECT fecha, titulo FROM articulo', $conexion);

// Crear el array de elementos para la capa de la Vista
$articulos = array();
while ($fila = mysql_fetch_array($resultado, MYSQL_ASSOC))
{
  $articulos[] = $fila;
}

// Cerrar la conexión
mysql_close($conexion);

// Incluir la lógica de la Vista
require('vista.php');
```

El código HTML, que contiene cierto código PHP a modo de plantilla, se almacenará en el código de la Vista, como se muestra en el Código 2.3.

Código 2.3 - La parte de la Vista, en *vista.php*

```
<html>
  <head>
    <title>Listado de Artículos</title>
  </head>
  <body>
    <h1>Listado de Artículos</h1>
    <table>
      <tr><th>Fecha</th><th>Título</th></tr>
      <?php foreach ($articulos as $articulo): ?>
        <tr>
          <td><?php echo $articulo['fecha'] ?></td>
          <td><?php echo $articulo['titulo'] ?></td>
        </tr>
      <?php endforeach; ?>
    </table>
  </body>
</html>
```

Una buena regla general para determinar si la parte de la Vista está suficientemente *limpia* de código PHP, es que debería contener una cantidad mínima de código PHP, la suficiente como para que un diseñador HTML sin conocimientos de PHP pueda entenderla. Las instrucciones más comunes en la parte de la Vista suelen ser *echo*, *if*/*endif*, *foreach*/*endforeach* y poco más.

Además, no se deben incluir instrucciones PHP que generen etiquetas HTML.

Toda la lógica se ha centralizado en el código del Controlador, que solamente contiene código PHP y ningún tipo de HTML. De hecho, el mismo Controlador se puede reutilizar para otros tipos de presentaciones completamente diferentes, como por ejemplo un archivo PDF o una estructura de tipo XML.

2.4 Separando la manipulación de los datos

La mayor parte del código del Controlador se encarga de la manipulación de los datos. Pero, ¿qué ocurre si se necesita la lista de entradas del blog para otro Controlador, por ejemplo uno que se dedica a generar el canal RSS de las entradas del blog? ¿Y si se quieren centralizar todas las consultas a la base de datos en un único sitio para evitar duplicidades? ¿Qué ocurre si cambia el Modelo de datos y la tabla cambia al llamarse de otra forma? ¿Y si se quiere cambiar a PostgreSQL en vez de MySQL? Para poder hacer todo esto, es

imprescindible eliminar del Controlador todo el código que se encarga de la manipulación de los datos y ponerlo en otro código, llamado *modelo.php*, tal y como se muestra en el Código 2.4.

Código 2.4 - La parte del Modelo, en *modelo.php*

```
<?php

function getTodosLosArticulos()
{
  // Conectar con la base de datos y seleccionarla
  $conexion = mysql_connect('localhost', 'miusuario', 'micontrasena');
  mysql_select_db('blog_db', $conexion);

  // Ejecutar la consulta SQL
  $resultado = mysql_query('SELECT fecha, titulo FROM articulo', $conexion);

  // Crear el array de elementos para la capa de la Vista
  $articulos = array();
  while ($fila = mysql_fetch_array($resultado, MYSQL_ASSOC))
  {
    $articulos[] = $fila;
  }

  // Cerrar la conexión
  mysql_close($conexion);
  return $articulos;
}
```

El Controlador modificado se puede ver en el Código 2.5.

Código 2.5 - La parte del Controlador modificado, en *index.php*

```
<?php

// Incluir la lógica del Modelo
require_once('modelo.php');

// Obtener la lista de artículos
$articulos = getTodosLosArticulos();

// Incluir la lógica de la Vista
require('vista.php');
```

Ahora el Controlador es mucho más fácil de leer. Su única tarea es la de obtener los datos del Modelo y pasárselos a la Vista. En sistemas más complejos, el Controlador se encarga además de procesar las peticiones, las sesiones de los usuarios, la autenticación, entre otros. El uso de nombres apropiados para las

funciones del Modelo hace que sea innecesario añadir comentarios al código del Controlador.

El código del Modelo solamente se encarga del acceso a los datos y puede ser reorganizado a tal efecto. Todos los parámetros que no dependen de la capa de datos (como por ejemplo los parámetros de la petición del usuario), se deben obtener a través del Controlador y por tanto, no se puede acceder a ellos directamente desde el Modelo. Las funciones del Modelo se pueden reutilizar fácilmente en otros Controladores.

2.5 Separación en capas más allá del MVC

El principio más importante del patrón MVC es la separación del código del programa en tres capas, dependiendo de su naturaleza. La lógica relacionada con los datos se incluye en el Modelo, el código de la presentación en la Vista y la lógica del sistema en el Controlador.

La programación se puede simplificar si se utilizan otros patrones de diseño. De esta forma, las capas del Modelo, la Vista y el Controlador se pueden subdividir en más capas.

2.6 Abstracción de la base de datos

La capa del Modelo se puede dividir en la capa de acceso a los datos y en la capa de abstracción de la base de datos. De esta forma, las funciones que acceden a los datos no utilizan sentencias ni consultas que dependen de una base de datos, sino que utilizan otras funciones para realizar las consultas. Así, si se cambia de sistema gestor de bases de datos, solamente es necesario actualizar la capa de abstracción de la base de datos.

El Código 2.6 muestra un ejemplo de capa de abstracción de la base de datos y el Código 2.7 muestra una capa de acceso a datos específica para MySQL.

Código 2.6 - La parte del Modelo correspondiente a la abstracción de la base de datos

```
<?php

function crear_conexion($servidor, $usuario, $contrasena)
{
  return mysql_connect($servidor, $usuario, $contrasena);
}

function cerrar_conexion($conexion)
{
  mysql_close($conexion);
}

function consulta_base_de_datos($consulta, $base_datos, $conexion)
{
  mysql_select_db($base_datos, $conexion);
  return mysql_query($consulta, $conexion);
}

function obtener_resultados($resultado)
{
  return mysql_fetch_array($resultado, MYSQL_ASSOC);
}
```

Código 2.7 - La parte del Modelo correspondiente al acceso a los datos

```
function getTodosLosArticulos()
{
  // Conectar con la base de datos
  $conexion = crear_conexion('localhost', 'miusuario', 'micontrasena');

  // Ejecutar la consulta SQL
  $resultado = consulta_base_de_datos('SELECT fecha, titulo FROM articulo',
'blog_db',$conexion);

  // Crear el array de elementos para la capa de la Vista
  $articulos = array();
  while ($fila = obtener_resultados($resultado))
  {
    $articulos[] = $fila;
  }

  // Cerrar la conexión
  cerrar_conexion($conexion);

  return $articulos;
}
```

Como se puede comprobar, la capa de acceso a datos no contiene funciones dependientes de ningún sistema gestor de bases de datos, por lo que es independiente de la base de datos utilizada. Además, las funciones creadas en la

capa de abstracción de la base de datos se pueden reutilizar en otras funciones del Modelo que necesiten acceder a la base de datos.

Nota: Los ejemplos del Código 2.6 y Código 2.7 no están completos, y todavía hace falta añadir código para tener una abstracción completa de la base de datos (abstraer el código SQL mediante un constructor de consultas independiente de la base de datos, añadir todas las funciones a una clase, entre otros.) El propósito de esta sección no es mostrar cómo se puede escribir todo ese código, sino proveer una idea general de cómo se hace.

2.7 Los elementos de la Vista

La capa de la Vista también puede aprovechar la separación de código. Los sistemas web suelen contener elementos que se muestran de forma idéntica a lo largo de todo el sistema: cabeceras de la página, el *layout* genérico, el pie de página y la navegación global. Normalmente sólo cambia el interior de la página, por este motivo, la Vista se separa en un *layout* y en una plantilla.

Por lo general, el *layout* es global en todo el sistema o al menos en un grupo de páginas. La plantilla sólo se encarga de visualizar las variables definidas en el Controlador. Para que estos componentes interaccionen entre sí correctamente, es necesario añadir cierto código. Siguiendo estos principios, la parte de la Vista del Código 2.3 se puede separar en tres partes, como se muestra a continuación:

Código 2.8 - La parte de la plantilla de la Vista, en *plantilla.php*

```
<h1>Listado de Artículos</h1>
<table>
  <tr><th>Fecha</th><th>Título</th></tr>
  <?php foreach ($articulos as $articulo): ?>
    <tr>
      <td><?php echo $articulo['fecha'] ?></td>
      <td><?php echo $articulo['titulo'] ?></td>
    </tr>
  <?php endforeach; ?>
</table>
```

Código 2.9 - La parte de la lógica de la Vista

```
<?php

$titulo = 'Listado de Artículos';
$contenido = include('plantilla.php');
```

Código 2.10 - La parte del layout de la Vista

```
<html>
  <head>
    <title><?php echo $titulo ?></title>
  </head>
  <body>
    <?php echo $contenido ?>
  </body>
</html>
```

2.8 Acciones y Controlador frontal

En el ejemplo anterior, el Controlador no se encarga de realizar muchas tareas, pero en sistemas web reales, el Controlador suele tener mucho trabajo. Una parte importante de su labor es realizar tareas comunes a todos los Controladores del sistema. Entre las tareas comunes se encuentran el manejo de las peticiones del usuario, el manejo de la seguridad, cargar la configuración del sistema y otras tareas similares. Por este motivo, el Controlador normalmente se divide en un Controlador frontal, que es único para cada sistema, y las acciones que incluyen el código específico del Controlador de cada página.

Una de las principales ventajas de utilizar un Controlador frontal es que ofrece un punto de entrada único para todo el sistema. Así, en caso de que sea necesario impedir el acceso al sistema, solamente es necesario editar el código correspondiente al Controlador frontal. Si el sistema no dispone de Controlador frontal, se debería modificar cada uno de los Controladores.

2.9 Orientación a Objetos

Los ejemplos anteriores utilizan la Programación Procedimental. Las posibilidades que ofrecen los lenguajes de programación modernos para trabajar con objetos permiten simplificar la programación, ya que los objetos pueden encapsular la lógica, pueden heredar métodos y atributos entre diferentes objetos, y proporcionan una serie de convenciones claras sobre la forma de nombrar a los objetos.

La implementación del patrón MVC en un lenguaje de programación cuyo paradigma no es la Programación Orientada a Objetos (POO), puede encontrarse con problemas de *namespace* y código duplicado, dificultando la lectura del código fuente del programa.

La orientación a objetos permite a los desarrolladores trabajar con objetos de la Vista, objetos del Controlador y clases del Modelo, transformando las funciones de los ejemplos anteriores en métodos. Se trata de un requisito obligatorio para las arquitecturas de tipo MVC.

Capítulo 3

3. Detalle técnico de la plataforma genérica

La estructura de directorios propuesta se formará de manera que resulte simple e intuitiva, tomando en consideración que se implementará el patrón de diseño MVC.

Vamos a crear la siguiente estructura de directorios y archivos base para dicha plataforma genérica:

- **application**
  - **controllers**
    - *AuthController.php*
    - *ErrorController.php*
    - *IndexController.php*
  - **models**
  - **views**
    - **_layouts**
      - *layout.phtml*
    - **auth**
    - **index**
      - *index.phtml*
    - **error**
      - *403.phtml*
      - *404.phtml*
      - *500.phtml*
      - *error.phtml*
- **config**
  - *Bootstrap.php*
  - *config.php*

        - *Router.php*
- **db**
- **library**
    - **core**
        - *ACL.php*
        - *ACLDbTable.php*
        - *Auth.php*
        - *AuthDbTable.php*
        - *Browser.php*
        - *Controller.php*
        - *DateTimeFormats.php*
        - *Db.php*
        - *Dispatcher.php*
        - *FileUpload.php*
        - *FlashMessages.php*
        - *Helpers.php*
        - *Log.php*
        - *Model.php*
        - *Paginator.php*
        - *RARFile.php*
        - *RenderException.php*
        - *RouterBase.php*
        - *View.php*
- **private**
- **public**
    - **css**
    - **images**
    - **js**
    - **uploads**
    - *.htaccess*
    - *index.php*
- **scripts**
- **tmp**
    - **cache**
    - **files**

    - **logs**
    - **sessions**
- *.htaccess*

A continuación vamos a describir de manera breve cada una de las clases que la integran, así como sus áreas de oportunidad, lo cual puede ser útil para hacer que la propia plataforma genérica pueda crecer conforme nuestras necesidades lo requieran y con ello poder desarrollar sistemas más completos.

Antes de entrar en detalle con las clases vamos a describir propiamente la estructura de directorios:

- **application**: este directorio contendrá todo lo necesario para que el sistema responda mediante el patrón MVC. Las carpetas que contienen son muy intuitivas ya que se llaman ***models***, ***views*** y ***controllers*** (Ver Figura 3.1).

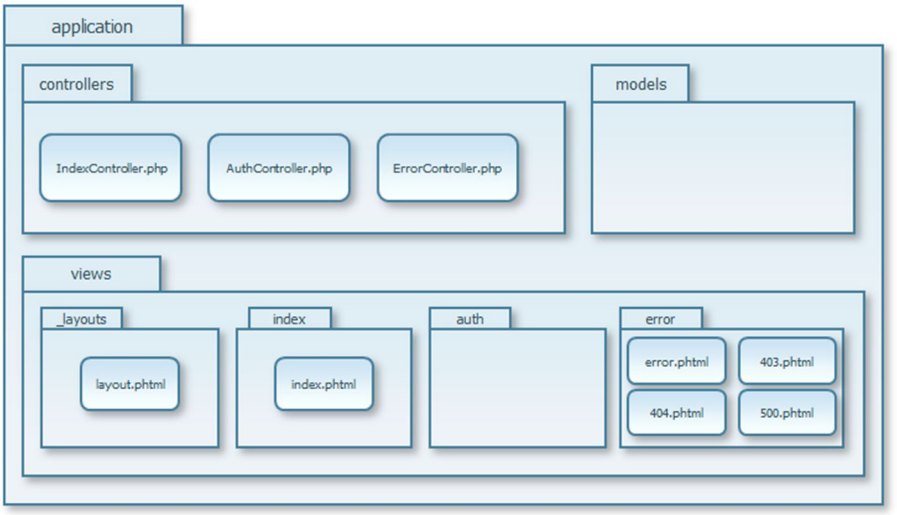

Figura 3.1 Directorio "application"

- **config**: la función de este directorio es la configuración del sistema, desde el registro de acceso a la base de datos, rutas (URLs) y cargar

todas las clases y archivos necesarios para la correcta ejecución de arranque (Ver Figura 3.2).

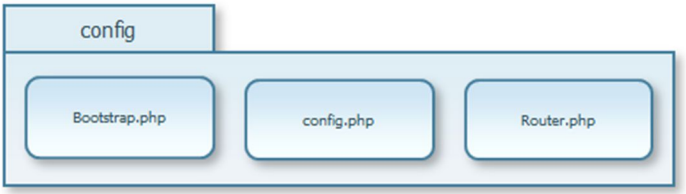

Figura 3.2 Directorio "config"

- **db**: aquí opcionalmente puede almacenarse toda clase de archivos relacionados con la base de datos, desde el punto de vista de recuperación o creación del esquema utilizado por el sistema (Ver Figura 3.3).

Figura 3.3 Directorio "db"

- **library**: este directorio almacena archivos auxiliares para el sistema, como *plugins* y bibliotecas de externos. También se ubica el directorio *core* que es donde se encuentran archivos vitales para que la plataforma genérica exista (Ver Figura 3.4).

- **private:** el contenido en este directorio puede utilizarse para guardar archivos que suelen ser privados en ciertos sistemas como imágenes, audio, video, entre otros (Ver Figura 3.5).

- **public:** este directorio debe ser el único punto de entrada a todo el sistema, en particular mediante el archivo *index.php* (para garantizar esto, es necesario tener correctamente configurado los accesos al

directorio del sistema en el servidor utilizado), por ello aquí se localiza todo aquello que el sistema dispondrá como de carácter público. Además se incluye una estructura de directorios común para sitios web 2.0 como los directorios ***css***, ***images***, ***js*** y ***uploads*** (Ver Figura 3.6).

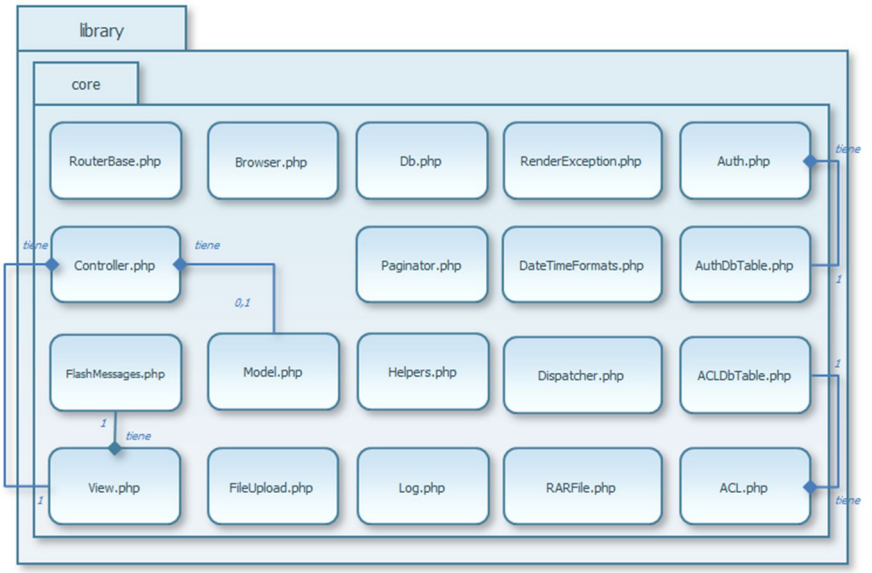

Figura 3.4 Directorio "library"

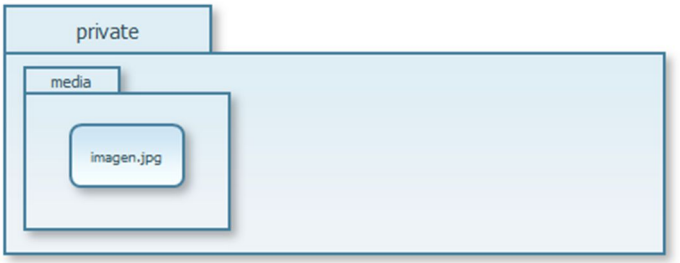

Figura 3.5 Directorio "private"

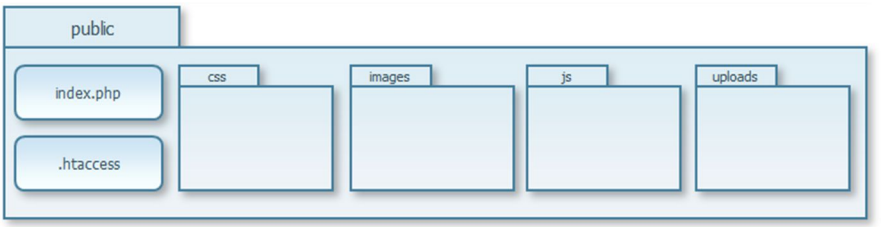

Figura 3.6 Directorio "public"

- **scripts:** este directorio puede utilizarse para ubicar archivos de ejecución automática como los generados para *Crontab* (Ver Figura 3.7).

Figura 3.7 Directorio "scripts"

- **tmp:** aquí va todo aquel cuyo contenido no sea de vital importancia y cuyo almacenamiento en el servidor sea temporal, se incluyen los directorios ***cache***, ***files***, ***logs*** y ***sessions*** (Ver Figura 3.8).

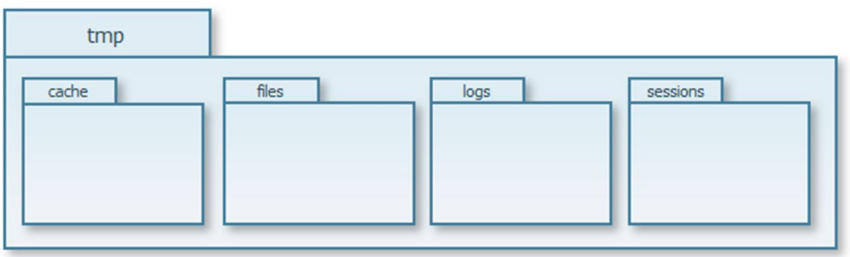

Figura 3.8 Directorio "tmp"

Una vez explicada de manera general la función de cada uno de los directorios que comprende nuestra plataforma genérica, vamos a describir de manera breve los distintos archivos y clases del mismo.

Como medida de seguridad para nuestro sistema, este debe tener solo un punto de acceso público definido, y la carpeta *public* está diseñada para dicha cuestión. A continuación se sugiere una configuración en un Virtual Host sobre un servidor *Apache 2*:

```
<VirtualHost *:80>
  DocumentRoot "/usr/local/apache/htdocs/proyecto/public"
  ServerName mi-aplicacion
  <Directory "/usr/local/apache/htdocs/proyecto/public">
    DirectoryIndexindex.php
    AllowOverrideAll
    orderallow,deny
    Allowfromall
    OptionsFollowSymLinks
      RewriteEngineOn
      RewriteCond %{REQUEST_FILENAME} -s [OR]
      RewriteCond %{REQUEST_FILENAME} -l [OR]
      RewriteCond %{REQUEST_FILENAME} -d
      RewriteRule ^.*$ - [NC,L]
      RewriteRule ^.*$ /index.php [NC,L]
  </Directory>
</VirtualHost>
```

Otra sugerencia de seguridad para la estructura, es mantener permisos de acceso **644** para los directorios y **755** para los archivos. Esto es una medida de seguridad común aplicada en los servidores.

Con esta configuración el único punto de entrada al sistema será mediante el archivo *index.php* ubicado dentro de la carpeta *public*, por lo que el servidor una vez recibida cualquier petición, lo primero que hará será leer el archivo *.htaccess* en este directorio.

El archivo *.htaccess* es un archivo de configuración muy popular en servidores web basados en Apache, que permite a los administradores aplicar distintas políticas de acceso a directorios o archivos con la idea de mejorar la seguridad de su página web y, por tanto, evitar acceso a terceros. Cuando visitamos una página web cualquiera y pulsamos sobre un enlace o queremos descargar un archivo, en el proceso de trámite de la petición, el servidor web consulta el archivo *.htaccess* con la idea de aplicar las directivas y restricciones definidas antes de cursar la petición y, lógicamente, cancelar peticiones que se encuentren prohibidas dentro de este archivo (cuyo ámbito de actuación es el directorio en el que se encuentra y todos los subdirectorios que se encuentran por debajo de éste).

El segundo archivo *.htaccess* ubicado en la raíz de nuestra estructura de directorios indica que cualquier petición será atendida a partir del directorio *public*, enviando todas las peticiones al archivo *index.php.* Esto logra mayor seguridad en el acceso sobre los archivos en nuestro servidor.

3.1 Archivos de arranque

Al inicio de cualquier petición sobre el sistema, existen archivos específicos en la plataforma genérica que siempre deben ser ejecutados de manera primordial, cuyo código programado se encarga de dar un orden a la ejecución subsecuente sobre cada uno de estos. A continuación se enlista y detalla brevemente cada uno de estos archivos.

3.1.1 Archivo index.php

Lo que hace el código *index.php* es definir constantes globales que son útiles para el ambiente del sistema. En este código pueden definirse todas aquellas variables globales relacionadas al funcionamiento propio de la plataforma genérica.

Y dado que *index.php* es el primer código en ejecutarse con cada petición al sistema, actúa como nuestro Controlador frontal.

Además es el encargado de llamar a los archivos de arranque, como por ejemplo al archivo *Bootstrap.php*.

3.1.2 Archivo Bootstrap.php

En este archivo se va a poner todo lo que el sistema debe conocer o cargar primero para el correcto funcionamiento. También hace inclusiones y usa tres archivos más: *config.php*, *Router.php* y *Log.php*. Pero la parte más importante de este código radica en el uso del método mágico *__autoload()* de PHP. Dicho método carga todas las clases y archivos del directorio *core, library, controllers y models* de nuestra plataforma genérica. Esto nos evita las constantes llamadas de inclusión a los diversos archivos o clases que podamos generar para nuestro sistema.

Si necesitamos establecer acciones o demás tareas que ocurran antes de cada petición, se declararán justamente sobre este archivo.

3.1.3 Archivo config.php

En este archivo vamos a definir las constantes globales acorde al sistema, por ejemplo, el nombre de la sesión de PHP, nombre de las rutas base del dominio, parámetros de la conexión a la base de datos, entre otros.

3.1.4 Archivo Router.php

En el archivo de rutas se pueden establecer los nombres de cada ruta a nuestro sistema; de esta manera contamos con grandes ventajas en los buscadores más populares de internet, además de proveer acceso sumamente descriptivo para el usuario. Esto es conocido como URLs amigables (*frindly-urls*) y para hacer uso de este enfoque es necesario que nuestro servidor cuente con el módulo habilitado para ello. En Apache, el módulo es *mod_rewrite*.

En el archivo *Router.php* también se pueden definir nombres de rutas que utilicen expresiones regulares, esto es gracias a la clase *RouterBase* de la cual hereda; dicha clase la explicaremos más adelante.

3.2 Entendiendo el *Core*

A continuación se describen las clases que integran el directorio más importante de la plataforma genérica para el correcto funcionamiento de nuestros sistemas, el directorio *core*.

3.2.1 Archivo Dispatcher.php

En el archivo *Bootstrap.php*, se hace la llamada al método *init()* de una clase llamada *Dispatcher,* la cual sigue el patrón *Singleton* para garantizar que esta clase sólo tenga una instancia y proporcione un punto de acceso global a ella.

Dispatcher se encarga de desmenuzar las peticiones del usuario y limpiar los parámetros de manera que resulten seguros, restringiendo la presencia de caracteres reservados o inválidos en una URL. Aunque la principal tarea de ésta clase es asociar la petición a una acción de un Controlador mediante las rutas establecidas para el sistema.

3.2.2 Archivo Browser.php

La clase *Singleton Browser* obtiene los datos relevantes del navegador del usuario. Dado que con cada petición esta clase estaría ejecutándose para obtener siempre la misma información y, debido a que no es una necesidad de vital importancia, resultaría útil almacenar en una sesión la información obtenida para un acceso más rápido a la misma y un mejor rendimiento sobre el sistema. Por esta razón esta clase incluye el método *storeInfo()*. A medida que vayan actualizando o naciendo nuevos navegadores, esta clase resulta útil para distinguirlos y darle una experiencia agradable al usuario sin importar el navegador con el que acceda al sistema web.

3.2.3 Archivo RouterBase.php

Esta clase es de gran importancia para la plataforma genérica, ya que su principal tarea es dar seguimiento a las peticiones del usuario a través de las URL, cualquier aspecto relacionado con las rutas debe hacerse en el archivo *Router.php*, visto anteriormente, donde se define la URL, el Controlador, parámetros y la acción a la cual debe responder.

La clase *RouterBase* se puede ampliar conforme las necesidades de los diversos sistemas y así también servirá para robustecer a la plataforma genérica. Cabe señalar que la clase *RouterBase* está implementada de tal forma que es capaz de trabajar con URL en expresiones regulares. Así, por ejemplo, una ruta con la forma *entradas/:int* serviría para tener múltiples rutas que terminen en un número: *entradas/1*, *entradas/2*, *entradas/3*, y así sucesivamente. Esto sería útil para una posible paginación de entradas de artículos en un blog.

3.2.4 Archivo Controller.php

La clase *Controller* es de las más importantes, ya que como su nombre lo indica, se encarga de controlar las peticiones del usuario. Su método constructor se encarga de asociar a la Vista y, en su caso, al Modelo acorde a la acción solicitada, también puede realizar tareas relacionadas a seguridad de accesos como comprobaciones de identidad (sesión de usuario) o asignar credenciales (permisos) con las que cuenta el usuario.

Otro método importante en esta clase es *response*, quien se encarga de responder en caso de errores o inaccesibilidad (códigos de estado propios del protocolo HTTP), por ejemplo cuando una página no existe, es inaccesible o el usuario no tiene los permisos necesarios para visualizarla.

Cualquier clase controladora de nuestro sistema, debe heredar de esta clase.

3.2.5 Archivo View.php

La clase *View* trata todo lo relacionado a la petición de la Vista, es capaz de incrustar código HTML sobre la Vista actual, además de incluir código de Javascript o reglas de cascada de estilos (CSS). Sin embargo su principal tarea se lleva a cabo en el método *render(),* el cual básicamente lee el archivo de *layout* o plantillas y prepara la salida del mismo con la correcta inclusión de datos obtenidos de la acción del Controlador.

3.2.6 Archivo Model.php

La clase *Model* es sin embargo más sencilla, debido a que no todos los Controladores necesitan consumir de una base de datos para funcionar. De hecho en esta clase se podrían escribir funciones generales a consultas aunque básicas, ya que como se verá más adelante las consultas y extracción de datos se harán desde las propias clases de Modelos definidas para el sistema.

3.2.7 Archivo Log.php

Este archivo saca ventaja de la variable de entorno definida en nuestro archivo *.htaccess*, la cual determina el ambiente de nuestro sistema y así podremos mostrar los errores en pantalla como es lo habitual en un ambiente de desarrollo o bien solo registrarlos en un archivo como ocurre en un ambiente productivo.

3.2.8 Archivo Auth.php

Es muy frecuente que los sistemas requieran de un manejo simple de autenticación de usuarios mediante un nombre de usuario y una contraseña, por lo que ésta clase tiene el propósito de crear o destruir sesiones de usuario mediante métodos básicos para llevar a cabo la autenticación. Cabe mencionar que sigue el patrón *Singleton* pues solo es necesaria una instancia de la misma. La interacción con la base de datos destinada al almacenamiento de usuarios se hace a través de la clase *AuthDbTable*.

3.2.9 Archivo AuthDbTable.php

La clase *AuthDbTable* se encarga de interactuar directamente con la base de datos para obtener la información necesaria del acceso de usuario. En el

constructor de la clase es posible indicar el nombre de las tablas y columnas de la base de datos necesarias para la autenticación.

3.2.10 Archivo ACL.php

En ciertos sistemas no basta con un acceso basado simplemente en poseer un usuario y una contraseña para disponer de toda las funciones disponibles, a veces suele ser necesario contar con una Lista de Control de Accesos a fin de que los usuarios puedan acceder a ciertas partes del sistema mediante roles (perfiles) y permisos (facultades). La clase *ACL* proporciona métodos para la recuperación y otorgamiento de éstos al usuario. *ACL* también sigue el patrón *Singleton* ya que no es necesario crear más de una instancia por sesión. Gran parte del trabajo que realiza esta clase se apoya en la clase *ACLDbTable*.

3.2.11 Archivo ACLDbTable.php

La clase *ACLDbTable* se encarga de interactuar directamente con la base de datos para obtener la información necesaria del perfil del usuario. En el constructor de la clase es posible indicar el nombre de las tablas y columnas de la base de datos que almacenan la lógica de permisos.

3.2.12 Archivo DateTimeFormats.php

Mediante la clase *DataTimeFormats* podemos obtener elementos básicos de una fecha para representarla de diferentes formas, como por ejemplo formatos específicos en español, abreviaturas, entre otros.

3.2.13 Archivo Db.php

Una clase simple que crea la cadena conectora a la base de datos y le establece algunos atributos. El tipo de conector depende del motor de base de datos que estemos utilizando, por lo que es justo este archivo donde podremos modificar si estamos usando MySQL, PostgreSQL, Oracle, u otro sistema de gestión.

3.2.14 Archivo FileUpload.php

Una clase que está hecha para ayudar con el envío de archivos al servidor de manera fácil y segura.

3.2.15 Archivo FlashMessages.php

Es muy frecuente que cuando los usuarios interactúan con los sistemas automatizados, requieran de retroalimentación del mismo cuando efectúan acciones, como puede ser el envío de un formulario exitoso o fallido, edición de datos, entre otras. Para ello la clase *FlashMessages* se encarga de crear los mensajes necesarios para indicarle al usuario sobre el resultado de las acciones que ejecute. Dichos mensajes se crean mediante una sesión del navegador y su duración es de solo una petición al servidor.

3.2.16 Archivo Helpers.php

Esta clase se encarga de proporcionar un método rápido de recuperación de valores enviados por el método *GET*.

3.2.17 Archivo Paginator.php

A menudo es necesario limitar la cantidad de registros devueltos por una base de datos por cuestiones de rendimiento y claridad. Esta clase genera las etiquetas HTML para la creación de enlaces de paginación. Su función únicamente se limita al cálculo de páginas necesarias, por lo que no debe confundirse con la separación de registros limitados a una cierta cantidad en la base de datos.

3.2.18 Archivo RARFile.php

Esta clase resulta útil para el tratamiento de archivos compresos tipo rar. *RARFile* puede abrir y/o extraer los archivos de éste contenedor y almacenarlos en el servidor.

3.2.19 Archivo RenderException.php

Es una clase que se solicita cuando ocurre algún tipo de error con las peticiones al servidor. Se complementa junto con el Controlador de errores definidos en el sistema.

3.3 El diseño con MVC para la plataforma genérica

La plataforma genérica contiene un directorio llamado *application* que a su vez, contiene tres subdirectorios más: *models*, *views* y *controllers*. Como es de esperarse cada uno de ellos contiene los archivos relacionados al patrón MVC.

Para poner en contexto esto, tenemos tres Controladores necesarios en nuestra plataforma genérica:

- ***IndexController*:** Es el Controlador por defecto para el sistema, puede utilizarse comúnmente para las páginas de bienvenida, contacto y demás secciones comunes en los sitios web.
- ***ErrorController*:** Se trata de un Controlador necesario para el envío de páginas con código de tipo error sobre la petición, como por ejemplo *404* cuando una página no existe, *403* cuando no tiene permisos suficientes para visualizar la página, entre otros.
- ***AuthController*:** Mediante este Controlador es posible indicar si el sistema requiere de un manejo de autenticación para el acceso o inclusive si es necesario también una lista de control de accesos.

Por cada Controlador que tengamos, será necesario tener en el directorio *views* un subdirectorio con el nombre de dicho Controlador escrito en minúsculas y sin la palabra *Controller*, además de estos subdirectorios existe uno fundamental para almacenar las Vistas de layouts. Nuestra plataforma genérica entonces cuenta con los siguientes directorios dentro de *views*:

- ***_layouts*:** Es el directorio para almacenar todos los posibles *layouts* de nuestro sistema. Por lo general en los sistemas suelen existir un par de éstos, uno para el diseño global y otro para el diseño de la pantalla de autenticación. Nuestra plataforma genérica por defecto utiliza uno llamado *layout.phtml*
- ***index*:** Dado que se tiene el Controlador por default *IndexController*, su directorio de Vistas es *index*. Por ejemplo, un sistema simple que cuente con tres secciones como la página de home, servicios y contacto debería tener por lo menos las plantillas *home.phtml*, *servicios.phtml* y *contacto.phtml*

- ***error*:** Cuando se necesite mostrar plantillas de error, este es el directorio donde pueden ubicarse dichas plantillas, por ejemplo *404.phtml*, *403.phtml*, entre otras.
- ***auth:*** un directorio que almacenaría las Vistas relacionadas a la autenticación del usuario, por ejemplo *login.phtml*

La lógica de la plataforma genérica sugiere que sea intuitiva para que por consecuente sea fácil asociar y recordar la manera de programar, incluso para quienes vean la estructura de archivos sin tener el conocimiento previo de tal estructuración. Por ejemplo si tenemos un sistema con un módulo de libros, es de pensarse que podríamos tener un **Controlador** encargado de manejar las acciones de los libros como podría ser listarlos, agregarlos, editarlos, eliminarlos, ponerlos en una lista RSS, entre otras. Por supuesto que podríamos contar con tantos libros que es necesario tener una base de datos que almacene dichos registros de libros, esto viene siendo nuestro **Modelo**; además una vez que el usuario solicita alguna acción sobre los libros entonces debemos mostrarle en pantalla su petición gráficamente, estas serían nuestras **Vistas**.

Supongamos entonces que tenemos un Controlador de libros, dicha clase y nombre de archivo serían *LibrosController* y *LibrosController.php* respectivamente; *LibrosController* debe heredar de la clase *Controller*, que como ya vimos es parte de nuestro Core y gracias también a nuestro archivo *Bootloader.php* no es necesario hacer ninguna inclusión al respecto. El código de esta clase sería el siguiente:

```
<?php

class LibrosController extends Controller
{
  public function __construct($action, $model)
  {
    parent::__construct($action, $model);
    $this->_setModel("Libro");
  }

  public function indexAction()
  {
    // Obtenemos el listado de libros
    $registros = $this->_model->getLibros();

    // Enviamos la lista de libros a la Vista
    $this->_view->set('libros', $registros);
  }
```

```
  // Mas acciones…
}
```

Podemos ver que la clase se llama *LibrosController*, con este nombre estamos obligados a crear un subdirectorio dentro del directorio *views* con las palabras o secuencia de caracteres que anteceden a la palabra Controller y todo en minúsculas, en este caso sería un directorio llamado *libros*.

En el código podemos percatarnos que existe una función llamada *indexAction()* cuyo propósito particular será enlistar todos los libros de nuestro inventario. La plataforma genérica entenderá que cualquier función de acceso público cuyo nombre termine con la palabra *Action*, será una acción disponible como petición del usuario al sistema, así pues cualquier palabra o juego de caracteres que antecedan a la palabra *Action* deberán corresponder con un nombre de archivo en minúsculas almacenado en el respectivo directorio de Vistas del Controlador, en este caso sería un archivo llamado *index* con extensión *phtml*. Dicho archivo podría contener lo siguiente:

```
<ul>
  <?php foreach ($libros as $libro): ?>
    <li><?php echo $libro['titulo']; ?>
  <?php endforeach; ?>
</ul>
```

Donde la ruta y nombre del archivo sería ubicado en *application/views/ libros/index.phtml*

La variable `$libros` es enviada a la Vista de manera automática gracias a la instrucción *$this->_view->set('libros', $registros)* que proviene de la clase *View* de nuestro Core y cuya labor es mapear la variable recibida como segundo argumento a una nueva variable definida bajo el nombre del primer argumento.
Por último, también podemos darnos cuenta de que el método constructor contiene una instrucción *$this->_setModel("Libro"),* lo cual indica que en nuestro directorio *models* debemos de crear un archivo *LibroModel.php* que extienda de la clase *Model*:

```
<?php

class LibroModel extends Model {

  private $_isbn;
  private $_titulo;

  public function __construct()
  {
    parent::__construct();
    $this->_isbn = null;
    $this->_titulo = null;
  }

  public function getIsbn()
  {
    return $this->_isbn;
  }

  public function setIsbn($value)
  {
    $this->_isbn = $value;
    return $this;
  }

  public function getTitulo()
  {
    return $this->_titulo;
  }

  public function setTitulo($value)
  {
    $this->_titulo = $value;
    return $this;
  }

  public function getLibros()
  {
    $sql = "SELECT isbn, titulo FROM libros";

    $this->_setSql($sql);

    $rows = $this->getAllRows();

    if (!empty($rows))
    {
      return $rows;
    }

    return false;
  }
}
```

El nombre del Modelo no está asociado a un Controlador de manera explícita, ya que no todos los Controladores necesitan consumir datos desde una base. Es por eso que se necesita incluir la llamada al método *_setModel()* del Controlador.

Gráficamente y de manera más abstracta, la Figura 3.9 muestra cómo se relacionan las clases en la estructura de la plataforma genérica:

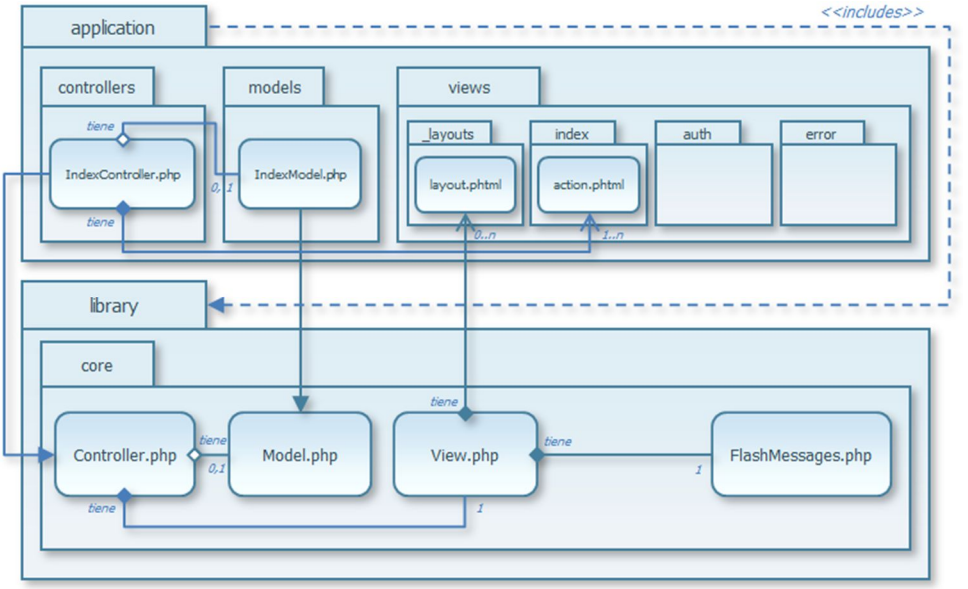

Figura 3.9 Diagrama relacional de la plataforma genérica

Este ejemplo se enfoca en el funcionamiento general de la plataforma genérica, por lo cual sólo se muestra el paquete del sistema "*application*" y únicamente 4 clases del Core de nuestra biblioteca constituida en el paquete "*library/core*".

La clase *Controller* es la clase esencial en la arquitectura, ya que como puede observarse, se trata de una súper clase de la cual heredan todos los Controladores del sistema, en este caso *IndexController*.

Además *Controller* tiene una relación de composición con una y solo una clase *View* asociada. En cambio mantiene una relación de agregación con la clase *Model*, ya que no todos los Controladores implementan un Modelo de datos. En caso contrario el Controlador tendría asociado un solo Modelo directo.

La clase *View* puede tener a su disposición múltiples *layouts* o plantillas, además ésta clase también mantiene una asociación de composición con la clase

FlashMessages, que es la encargada de enviar mensajes de sesión al usuario para mostrar los resultados a determinadas acciones.

El Controlador *IndexController* perteneciente al sistema puede tener relacionados una o múltiples archivos de Vista (archivos .*phtml*), como fuesen necesarios para mostrar las peticiones de las acciones por parte del usuario.

Por último existe una dependencia "*includes*", que nos dice que el paquete *application* depende del paquete "*library/core*" y que cualquier cambio en éste último debe tomarse en consideración por el otro paquete.

Capítulo 4

4. Ejemplificación del uso de la plataforma genérica mediante un sistema

El sistema construido sobre la plataforma genérica es un gestor de riesgo operacional, el cual servirá para la creación de los reportes regulatorios de riesgo operacional, debe configurarse de acuerdo a las recomendaciones emitidas por Basilea II, esto se logra con la implementación de sus diversos catálogos de clasificación.

Para ello el sistema se ha construido de manera que resulte seguro tanto para su administración, como para mantener a salvo la información sensible de posibles usos malintencionados.

A continuación vamos a describir la manera técnica de configurar el sistema.

4.1 Configuración estructural

En esta sección cubriremos tres áreas:

- Configuración de las *Ubicaciones* en el Mapa organizacional
- Configuración de las *Unidades* en la organización
- Configuración de *Usuarios*

El área de la estructura organizacional, es un repositorio con toda la información relativa a la ubicación jerárquica y datos de las unidades organizacionales dentro del sistema. Los usuarios también se pueden configurar aquí.

4.2 Configurando Niveles de Mapa

La sección de Mapa contiene información sobre la ubicación geográfica dentro de la organización y se visualiza mediante un formato de árbol jerárquico.

El mantenimiento de las ubicaciones dentro del mapa así como sus niveles se reservan al administrador. Los niveles de más profundidad (más bajos), se utilizan como identificadores sobre las estructura de unidades y usuarios de la organización. Todos los niveles menos profundos (más altos), se utilizan para unir a los miembros del nivel más bajo en grupos lógicos para la generación de informes.

Antes de que las ubicaciones se puedan añadir, debemos definir y configurar sus propios niveles (Ver Figura 4.1). Los niveles no deberían redefinirse con mucha frecuencia, sin embargo, el sistema proporciona la facilidad para hacer los cambios cuando sea necesario.

Figura 4.1 Lista de niveles de mapa

4.2.1 Agregar niveles al mapa

1. En el menú de Mapa seleccione Niveles, a continuación se mostrará la página con la *Jerarquía de niveles*.
2. Seleccione Agregar un nivel. Se mostrará la página de Nuevo Nivel de Mapa, donde podrá elegir si se debe añadir un nuevo nivel de mapa por debajo de otro nivel o bien añadir en la parte más baja del árbol jerárquico (Ver Figura 4.2).

3. Introduzca el <nombre del nivel>.

4. Presione **Agregar Nivel** para confirmar la inserción de éste nuevo registro o presione **Cancelar** para salir de la página sin guardar.

4.2.2 Editar un nivel de mapa

1. Navegue hasta la página de *Jerarquía de niveles*.

2. Presione el botón de **Editar** para acceder a la página de edición sobre el elemento elegido – únicamente el *Nombre* puede ser editado.

3. Presione **Guardar cambios** para confirmar la modificación de éste registro o presione **Cancelar** para salir de la página sin guardar.

4.2.3 Eliminar un nivel del mapa

Nota: Antes de eliminar un nivel del mapa, el usuario debe asumir los siguientes dos procesos que de manera implícita se ejecutarán en el sistema:

- Al elemento hijo del nivel a eliminar, se le asignará como nuevo nivel padre a aquel que se encuentre inmediatamente arriba del elemento a eliminar.

- A cada ubicación cuyo nivel padre pertenezca al elemento a eliminar, se les asignarán como elemento padre, a la misma ubicación padre del nivel a eliminar.

1. Navegue hasta la página de *Jerarquía de niveles*.

2. Presione el botón de **Eliminar** para acceder a la página de confirmación de eliminación. Elija Si y presione **Eliminar Nivel** o presione **Cancelar** para salir de la página sin guardar.

Figura 4.2 - Formulario de "Nuevo nivel" de mapa

4.3 Trabajando con Ubicaciones de Mapa

Figura 4.3 - Vista con el menú de opciones emergente sobre los elementos de ubicaciones de Mapa

Presione el ícono '+' para expandir el árbol jerárquico y visualizar las ubicaciones subordinadas.

4.3.1 Agregar ubicaciones al mapa

1. En el menú de Mapa seleccione Ubicaciones, a continuación se mostrará la página con la *Jerarquía de ubicaciones*. Presione el botón de **Opciones** perteneciente al elemento padre elegido, por ejemplo, para agregar una nueva región a *México*, seleccione la opción **Añadir descendiente** (Ver Figura 4.3).

2. Se mostrará la página de Nueva Ubicación de Mapa.

3. Introduzca el <nombre de la ubicación>, <código> e indique si la ubicación estará habilitada.
4. Presione **Agregar Ubicación** para confirmar la inserción de éste nuevo registro o presione **Cancelar** para salir de la página sin guardar.

4.3.2 Editar una ubicación del mapa

1. Navegue hasta la página de *Jerarquía de ubicaciones*.
2. Presione el botón de **Editar** para acceder a la página de edición sobre el elemento elegido – únicamente el *Nombre*, *Nivel, Ubicación* y *habilitación* pueden ser editados (Ver Figura 4.4).
3. Presione **Guardar cambios** para confirmar la modificación de éste registro o presione **Cancelar** para salir de la página sin guardar.

4.3.4 Eliminar una ubicación del mapa

Nota: Antes de eliminar una ubicación del mapa, el usuario debe asumir el siguiente proceso que de manera implícita se ejecutará en el sistema:

- A cada ubicación cuyo nivel padre pertenezca al elemento a eliminar, se les asignarán como elemento padre, a la misma ubicación padre de la ubicación a eliminar.

1. Navegue hasta la página de *Jerarquía de ubicaciones*.
2. Presione el botón de **Eliminar** para acceder a la página de confirmación de eliminación. Elija Si y presione **Eliminar Ubicación** o presione **Cancelar** para salir de la página sin guardar.

4.3.5 Cambiar elemento padre de una ubicación del mapa

Nota: Es posible reasignar dicho elemento a una ubicación padre distinta, inclusive si está en otro nivel, el usuario debe asumir que todos los elementos hijos de la ubicación seguirán siendo hijos del mismo, por lo que el elemento padre quedaría sin ubicaciones descendientes. En caso de deshabilitar la ubicación, todos los elementos hijos serán deshabilitados también.

1. Navegue hasta la página de *Jerarquía de ubicaciones*. Presione el botón de **Opciones** perteneciente al elemento padre elegido, por ejemplo, para cambiar la ubicación padre de *Centro Sur*, seleccione la opción **Editar ubicación**.
2. Se mostrará la página de Editar Ubicación de Mapa. Seleccione el nivel y nombre apropiado desde las listas respectivamente.
3. Presione **Guardar cambios** para confirmar la modificación de éste registro o presione **Cancelar** para salir de la página sin guardar.

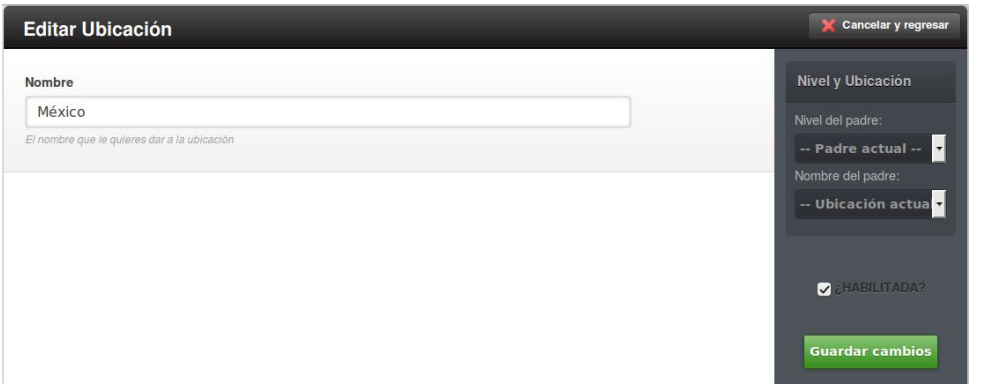

Figura 4.4 - Vista del formulario "Editar Ubicación"

4.4 Configurando Niveles de Organización

La sección de Organización contiene información sobre la estructura de las unidades organizacionales de la institución y se visualiza mediante un formato de árbol jerárquico.

El mantenimiento de las unidades dentro de la organización así como sus niveles se reservan al administrador. Todos los niveles menos profundos (más altos), se utilizan para unir a los miembros del nivel más bajo en grupos lógicos para la generación de informes.

Antes de que las unidades se puedan añadir, debemos definir y configurar sus propios niveles (Ver Figura 4.5). Los niveles no deberían redefinirse con mucha frecuencia, sin embargo, el sistema proporciona la facilidad para hacer los cambios cuando sea necesario.

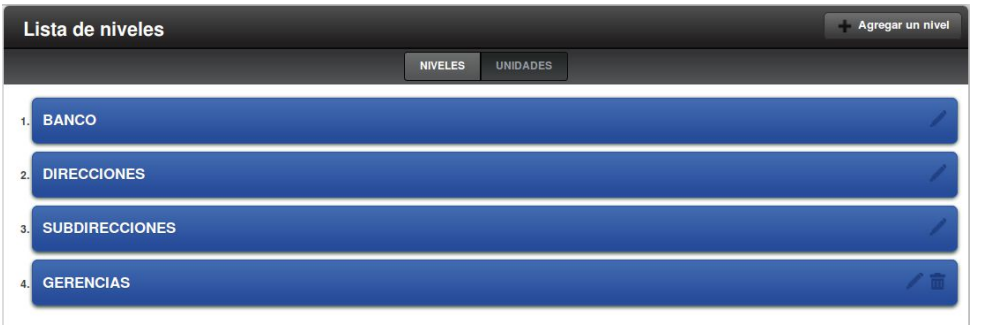

Figura 4.5 - Lista de niveles de organización

4.4.1 Agregar niveles a la organización

1. En el menú de Organización seleccione Niveles, a continuación se mostrará la página con la *Jerarquía de niveles*.
2. Seleccione Agregar un nivel. Se mostrará la página de Nuevo Nivel de Organización, donde podrá elegir si se debe añadir un nuevo nivel de organización por debajo de otro nivel o bien añadir en la parte más baja del árbol jerárquico (Ver Figura 4.6).
3. Introduzca el <nombre del nivel>.

4. Presione **Agregar Nivel** para confirmar la inserción de éste nuevo registro o presione **Cancelar** para salir de la página sin guardar.

4.4.2 Editar un nivel de organización

1. Navegue hasta la página de *Jerarquía de niveles*.
2. Presione el botón de **Editar** para acceder a la página de edición sobre el elemento elegido – únicamente el *Nombre* puede ser editado.
3. Presione **Guardar cambios** para confirmar la modificación de éste registro o presione **Cancelar** para salir de la página sin guardar.

4.4.3 Eliminar un nivel de la organización

Nota: Antes de eliminar un nivel de la organización, el usuario debe asumir los siguientes dos procesos que de manera implícita se ejecutarán en el sistema:

- Al elemento hijo del nivel a eliminar, se le asignará como nuevo nivel padre a aquel que se encuentre inmediatamente arriba del elemento a eliminar.
- A cada unidad cuyo nivel padre pertenezca al elemento a eliminar, se les asignarán como elemento padre, a la misma unidad padre del nivel a eliminar.

1. Navegue hasta la página de *Jerarquía de niveles*.
2. Presione el botón de **Eliminar** para acceder a la página de confirmación de eliminación. Elija Si y presione **Eliminar Nivel** o presione **Cancelar** para salir de la página sin guardar.

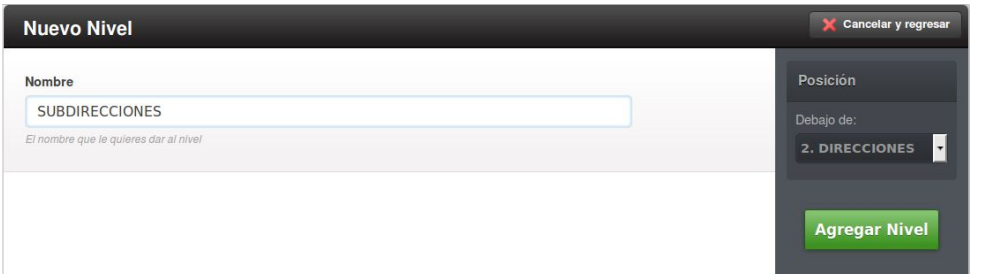

Figura 4.6 - Formulario de "Nuevo nivel" de organización

4.5 Trabajando con Unidades Organizacionales

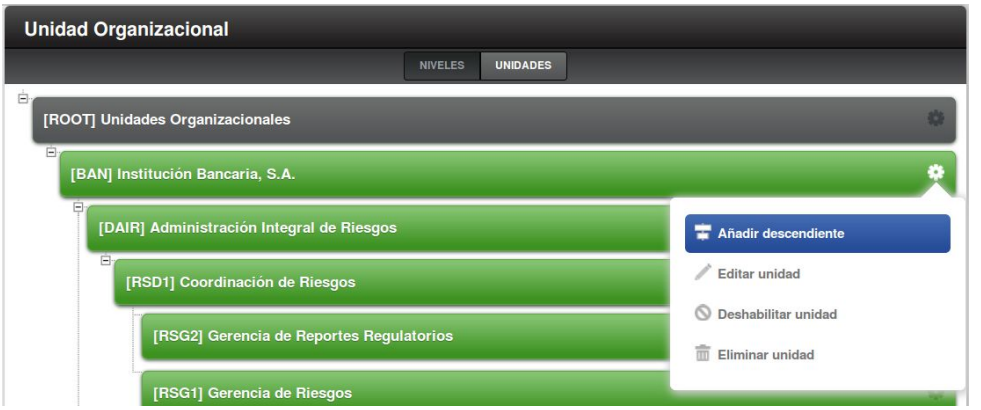

Figura 4.7 - Vista con el menú de opciones emergente sobre los elementos de unidades organizacionales

Presione el ícono '+' para expandir el árbol jerárquico y visualizar las unidades subordinadas.

4.5.1 Agregar unidades organizacionales

1. En el menú de Organización seleccione Unidades, a continuación se mostrará la página con la *Jerarquía de unidades*. Presione el botón de **Opciones** perteneciente al elemento padre elegido, por ejemplo, para agregar una nueva división a *Institución Bancaria, S.A.*, seleccione la opción **Añadir descendiente** (Ver Figura 4.7).

2. Se mostrará la página de Nueva Unidad Organizacional.
3. Introduzca los diversos campos que constituyen la unidad (Ver Figura 4.24).
4. Presione **Agregar Unidad** para confirmar la inserción de éste nuevo registro o presione **Cancelar** para salir de la página sin guardar.

4.5.2 Editar una unidad organizacional

1. Navegue hasta la página de *Jerarquía de unidades*.
2. Presione el botón de **Editar** para acceder a la página de edición sobre el elemento elegido.
3. Presione **Guardar cambios** para confirmar la modificación de éste registro o presione **Cancelar** para salir de la página sin guardar.

4.5.4 Eliminar una unidad organizacional

Nota: Antes de eliminar una unidad de la organización, el usuario debe asumir el siguiente proceso que de manera implícita se ejecutará en el sistema:

- A cada unidad cuyo nivel padre pertenezca al elemento a eliminar, se les asignarán como elemento padre, a la misma unidad padre de la unidad a eliminar.

1. Navegue hasta la página de *Jerarquía de unidades*.
2. Presione el botón de **Eliminar** para acceder a la página de confirmación de eliminación. Elija Si y presione **Eliminar Unidad** o presione **Cancelar** para salir de la página sin guardar.

4.5.5 Cambiar elemento padre de una unidad organizacional

Nota: Es posible reasignar dicho elemento a una unidad padre distinta, inclusive si está en otro nivel, el usuario debe asumir que todos los elementos hijos de la unidad seguirán siendo hijos del mismo, por lo que el elemento padre quedaría sin unidades descendientes. En caso de deshabilitar la unidad, todos los elementos hijos serán deshabilitados también.

1. Navegue hasta la página de *Jerarquía de unidades*. Presione el botón de **Opciones** perteneciente al elemento padre elegido, por ejemplo, para cambiar la ubicación padre de *Gerencia de Riesgos*, seleccione la opción **Editar unidad**.
2. Se mostrará la página de Editar Unidad Organizacional. Modifique los campos apropiados respectivamente.
3. Presione **Guardar cambios** para confirmar la modificación de éste registro o presione **Cancelar** para salir de la página sin guardar.

4.6 Configuración de Usuarios

La sección de *Personas* es un área que contiene detalles de todos los usuarios dentro del sistema. Es un requisito que tanto usuarios activos como inactivos sean almacenados para un futuro uso, por ejemplo, asignarlos como responsables de un control.

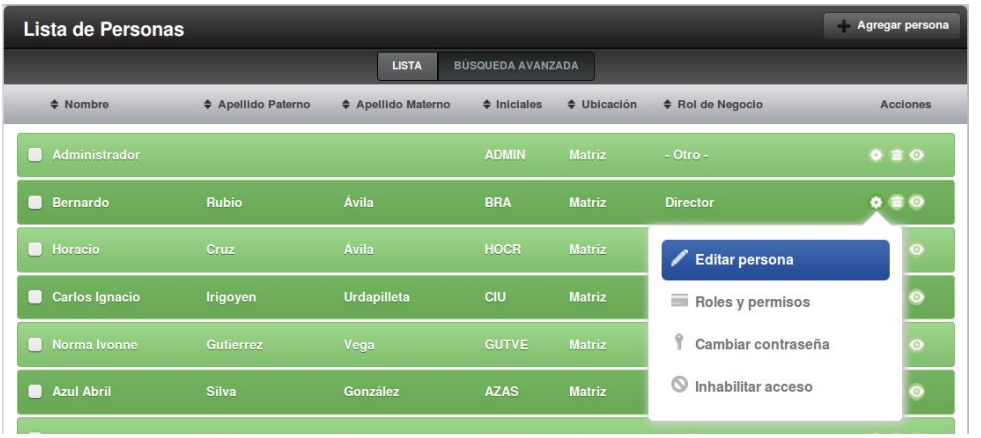

Figura 4.8 - Vista con el menú de opciones emergente sobre los elementos de la lista de personas

Nota: Las opciones de esta sección están reservadas exclusivamente al administrador o a usuarios con permisos específicos sobre el módulo *Personas* (Ver Figura 4.8).

4.6.1 Listado de Personas

1. En el menú de Personas seleccione Lista, a continuación se mostrará la página con el *Listado de personas* mostrando todos los usuarios registrados en el sistema. Un breve resumen de sus detalles será visualizado junto con una serie de elementos que nos proporcionarán las siguientes características:

    - La lista puede ser ordenada al hacer clic sobre el encabezado de la columna en la parte alta de la página y tenga un símbolo de ordenación.
    - La columna ubicada en la parte derecha de la lista nos proporcionará opciones dependiendo de los permisos de acceso del usuario. También se tiene el botón de **Agregar persona** en la parte superior derecha.

4.6.2 Agregar personas al sistema

1. Navegue hasta la página de *Listado de personas* y presione el botón **Agregar persona**.
2. Introduzca la información apropiada:
    - *Login:* Este debe ser un nombre de usuario único que servirá para identificar al usuario dentro del sistema.
    - *Nombre:* El nombre o los nombres de la persona.
    - *Apellido paterno:* El apellido paterno de la persona.
    - *Apellido materno:* El apellido materno de la persona.
    - *Iniciales:* Estas <iniciales> serán utilizadas para propósitos de reportes. Iniciales duplicadas en el sistema son permitidas.
    - *Correo electrónico:* La dirección de correo electrónico es requerida para permitir a los usuarios recibir información desde el propio sistema vía email. Tal información puede incluir, por ejemplo, reportes programados o alertas. Este campo debe tener un formato de email, i. e. en la forma:

      <nombre de buzón>@<dominio>.<extensión>

    - *Puesto:* El cargo de la persona dentro de la organización, solo se emplea para propósitos de reportes.
    - *Zona de trabajo:* La ubicación física donde labora la persona.

      Nota: la lista desplegable muestra las ubicaciones definidas en el Mapa del sistema.

    - *Área:* La unidad organizativa a la que pertenece la persona.

      Nota: la lista desplegable muestra las unidades definidas en la Organización del sistema.

- *Roles de la persona:* El perfil o rol que tendrá configurado el usuario; pueden seleccionarse múltiples roles si así se requiere.

  Nota: el rol *Usuario* debe elegirse siempre para un usuario activo. En cambio, el rol *Otros* debe reservarse para un usuario inactivo en el sistema.

- *Importancia en incidentes:* Corresponde al valor de relevancia por defecto cuando ésta persona reporta un incidente de riesgo operacional.

- *Importancia en pérdidas:* Corresponde al valor de relevancia por defecto cuando ésta persona reporta un evento de pérdida por riesgo operacional.

3. Presione **Agregar persona** para confirmar la inserción de éste nuevo registro o presione **Cancelar** para salir de la página sin guardar.

Nota: Una vez que se agrega una persona, se le deberá asignar una contraseña inicial y habilitar su acceso al sistema.

4.6.3 Editar el perfil de una persona

Esta sección describe como un usuario con perfil de *administrador* o con permisos específicos sobre el módulo *Personas* puede editar el perfil de una persona.

Los usuarios que no tengan un perfil *administrador* pueden realizar ciertas ediciones a su propio perfil. Esto es tratado más adelante.

1. Navegue hasta la página de *Listado de personas.* Presione el botón de **Opciones** perteneciente al usuario elegido en la columna de *Acciones* para desplegar las ediciones disponibles.

2. Para editar campos relativos al perfil de la persona elija la opción *Editar persona*, proceda con las modificaciones y guarde los cambios.

3. Puede cambiar la contraseña de un usuario, elija la opción *Cambiar contraseña*. La contraseña puede ser introducida de manera textual o bien generar una sugerencia de formato seguro para la misma.

4. Para editar roles y permisos específicos, elija la opción *Roles y permisos.* Ésta edición le permitirá reasignar roles o permitir ciertos permisos de manera específica para el usuario.

5. Para otorgar el acceso al sistema, se debe elegir la opción *Habilitar/Deshabilitar acceso*. Ésta edición es sumamente importante para permitir al usuario iniciar una sesión dentro del sistema. Si el usuario está deshabilitado, no podrá accesar al mismo, pero podrá seguir recibiendo reportes a su dirección de correo electrónico.

4.7 Lógica del negocio. Configuración de catálogos

La sección *Lógica de Negocio* es un área donde es posible configurar los catálogos relativos al riesgo operacional. Dichos catálogos son necesarios para el correcto funcionamiento del módulo *Captura de Riesgo*.

4.8 Gestión de catálogos

En general todos los catálogos se definen mediante niveles de profundidad, de este modo cada elemento del catálogo puede contener un ID único, esto sirve para que el sistema pueda interactuar con el árbol jerárquico y sea posible cambiar niveles, eliminarlos, reestructurarlos o asignarles nuevas propiedades (Ver Figura 4.9).

Figura 4.9 - Vista de niveles genéricos a los catálogos

4.8.1 Agregar niveles al catálogo

1. En el menú relativo a un catálogo seleccione Niveles, a continuación se mostrará la página con la *Jerarquía de niveles*.

2. Seleccione Agregar un nivel. Se mostrará la página de Nuevo Nivel de catálogo donde podrá elegir si se debe añadir un nuevo nivel por debajo de otro ya existente o bien añadirlo en la parte más baja del árbol jerárquico.

3. Introduzca el <nombre del nivel>.

4. Presione **Agregar Nivel** para confirmar la inserción o presione **Cancelar** para salir de la página sin guardar.

4.8.2 Editar un nivel del catálogo

1. Navegue hasta la página de *Jerarquía de niveles*.

2. Presione el botón de **Editar** para acceder a la página de edición sobre el elemento elegido – únicamente el *Nombre* puede ser editado.

3. Presione **Guardar cambios** para confirmar la modificación de éste registro o presione **Cancelar** para salir de la página sin guardar.

4.8.3 Eliminar un nivel del catálogo

Nota: Antes de eliminar un nivel del catálogo, el usuario debe asumir los siguientes dos procesos que de manera implícita se ejecutarán en el sistema:

- Al elemento hijo del nivel a eliminar, se le asignará como nuevo nivel padre, a aquel que se encuentre inmediatamente arriba del elemento a eliminar.
- A cada elemento del catálogo cuyo nivel padre pertenezca al elemento a eliminar, se les asignarán como elemento padre, al mismo elemento padre del nivel a eliminar.

1. Navegue hasta la página de *Jerarquía de niveles*.

2. Presione el botón de **Eliminar** para acceder a la página de confirmación de eliminación. Elija Si y presione **Eliminar Nivel** o presione **Cancelar** para salir de la página sin guardar.

4.9 Trabajando con las definiciones del catálogo

Figura 4.10 - Vista con el menú de opciones emergente sobre los elementos del catálogo

Presione el ícono '+' para expandir el árbol jerárquico y visualizar las definiciones subordinadas.

4.9.1 Agregar elementos al catálogo

1. En el menú relativo a un catálogo seleccione Catálogo, a continuación se mostrará la página con la *Jerarquía de definiciones*. Presione el botón de **Opciones** perteneciente al elemento padre elegido y a continuación elija **Añadir descendiente** (Ver Figura 4.10).
2. Se mostrará la página de *Nuevo Tipo de Catálogo*.
3. Introduzca el <nombre del elemento>, <código>, <definición> e indique si la definición estará habilitada.
4. Presione **Agregar** para confirmar la inserción de éste nuevo registro o presione **Cancelar** para salir de la página sin guardar.

4.9.2 Editar una definición del catálogo

1. Navegue hasta la página de *Jerarquía de definiciones*.

2. Presione el botón de **Editar** para acceder a la página de edición sobre el elemento elegido – únicamente el *Nombre*, *Nivel, Posición, Definición* y *habilitación* pueden ser editados (Ver Figura 4.11).

3. Presione **Guardar cambios** para confirmar la modificación de éste registro o presione **Cancelar** para salir de la página sin guardar.

4.9.3 Eliminar una definición del catálogo

Nota: Antes de eliminar un elemento del catálogo, el usuario debe asumir el siguiente proceso que de manera implícita se ejecutará en el sistema:

- A cada definición cuyo nivel padre pertenezca al elemento a eliminar, se les asignarán como definición padre, al mismo elemento padre de la definición a eliminar.

1. Navegue hasta la página de *Jerarquía de definiciones*.

2. Presione el botón de **Eliminar** para acceder a la página de confirmación de eliminación. Elija Si y presione **Eliminar** o presione **Cancelar** para salir de la página sin guardar.

4.9.4 Cambiar elemento padre de una definición del catálogo

Nota: Es posible reasignar dicho elemento a un elemento padre distinto, inclusive si está en otro nivel, el usuario debe asumir que todos los elementos hijos de la definición seguirán siendo hijos del mismo, por lo que el elemento padre quedaría sin elementos descendientes. En caso de deshabilitar la definición, todos los elementos hijos serán deshabilitados también.

1. Navegue hasta la página de *Jerarquía de definiciones*. Presione el botón de **Opciones** perteneciente al elemento padre elegido y a continuación elija **Editar**.

2. Se mostrará la página de *Editar elemento*. Para cambiar el elemento padre seleccione su nivel y nombre correspondiente desde las listas respectivamente.

3. Presione **Guardar cambios** para confirmar la modificación de éste registro o presione **Cancelar** para salir de la página sin guardar.

Figura 4.11 – Vista del formulario genérico de edición de elementos de catálogo

4.10 Uso del Sistema

El sistema gestor de riesgo operacional comprende de varios módulos, los cuáles están estructurados de manera que, dependiendo el rol o perfil del usuario, haga uso de ellos. La prioridad en el sistema es que el usuario reporte eventos o incidencias causadas por riesgo operacional.

Cuando el usuario reporta dichos sucesos, el mismo puede ver y darle seguimiento, de éste modo la interfaz del sistema propone un enfoque de bandejas (similares a los sistemas de correo electrónico), donde cada entrada es un registro que ha sido reportado.

Para que un usuario pueda reportar dichos eventos o incidencias, se realizarán mediante el envío de formularios que incluyen todos los campos solicitados por la entidad regulatoria, en este caso la CNBV.

Dado que el sistema debe configurarse en base a los catálogos de riesgo operacional emitidos en la CUB, también es importante que el usuario sepa cómo adecuar dichos catálogos en caso de posibles modificaciones a los mismos.

Una característica importante del sistema, es que permite la configuración del mismo de acuerdo a la propia estructura de la entidad bancaria, siendo posible la creación de nuevas áreas o estructura global de la misma, como por ejemplo, la creación de nuevas sucursales bancarias podría involucrar nuevo personal y, como es de esperarse, aquella nueva sucursal deberá de contar con sus propios resúmenes y generación de reportes.

A continuación explicaremos la manera en la que el usuario debe usar el sistema para sacar el máximo provecho del mismo.

Para dar acceso al sistema se requiere de un nombre de usuario y una contraseña, los cuales son proporcionados por el administrador del mismo.

La pantalla de *Inicio de sesión* se muestra en la Figura 4.12:

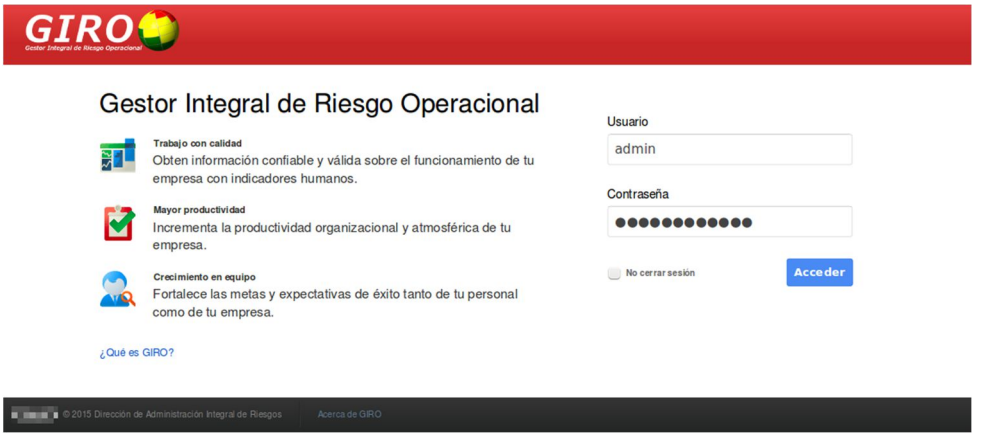

Figura 4.12 - Vista de "Inicio de sesión"

Una vez introducido los campos con datos válidos de **Usuario** y **Contraseña**, se podrá hacer uso de las opciones disponibles dentro del sistema.

4.11 Bandeja de formularios

En la sección *Captura de Riesgo* se encuentran ubicadas las Bandejas de Entradas para cada formulario (“R28 A-2811”, “R28 A-2812” y “R28 A-2813”), dependiendo del perfil asignado al usuario, se podrá visualizar todos los eventos de pérdida que ha reportado.

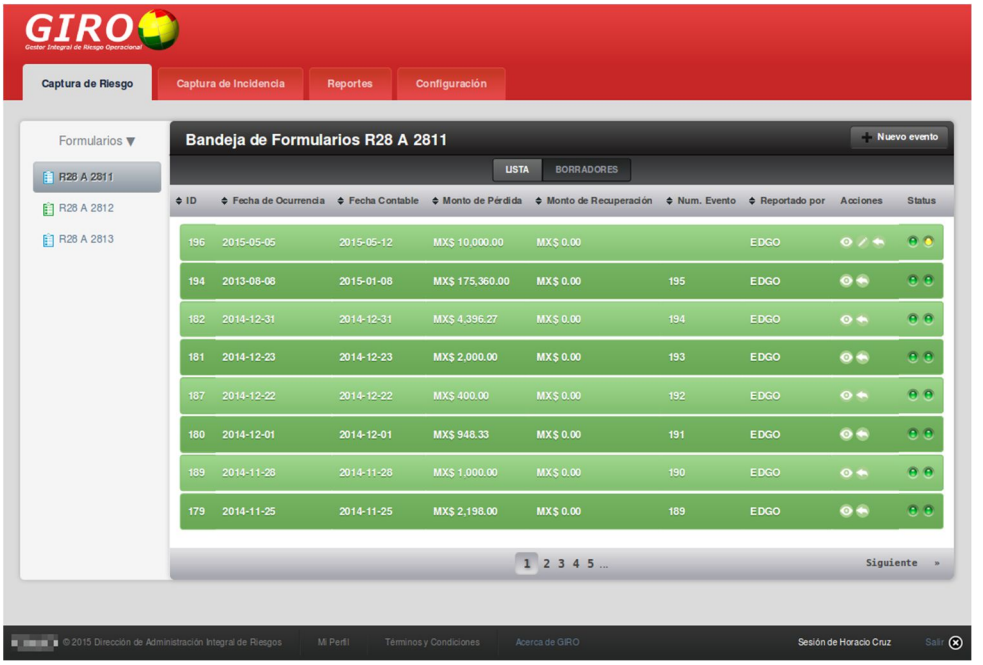

Figura 4.13 - Vista del listado de entradas liberadas

Existen dos Bandejas de Entradas para cada tipo de reporte, la primera corresponde a *Entradas liberadas* (Ver Figura 4.13) y la segunda a *Entradas borrador* (Ver Figura 4.15).

4.12 Registro de eventos

Cuando el usuario desee agregar un nuevo evento de pérdida por riesgo operacional, deberá hacer clic en el botón *Nuevo evento* y así accederá a la visualización del formulario de acuerdo a la Bandeja donde nos encontremos,

por ejemplo, el formulario correspondiente a eventos de pérdida “R28 A-2811” se muestra en la Figura 4.14.

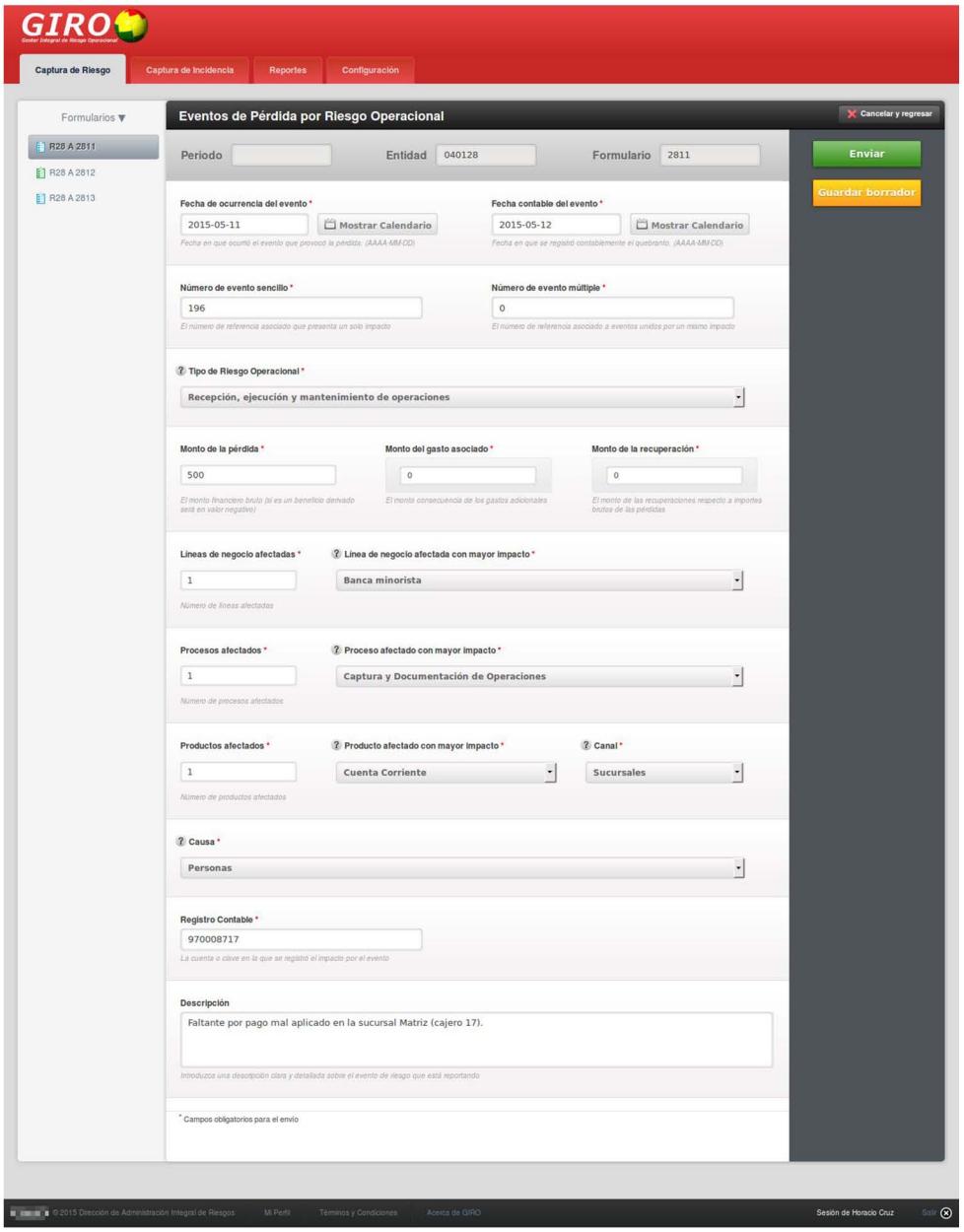

Figura 4.14 - Vista del formulario para registro de evento R28 A-2811

El formulario puede guardarse como un borrador, en este caso los campos no serán validados o bien enviarse para su revisión y, de este modo, el sistema revisará el correcto tipo de dato suministrado para cada campo del formulario.

La bandeja de *Entradas borrador* se muestra a continuación:

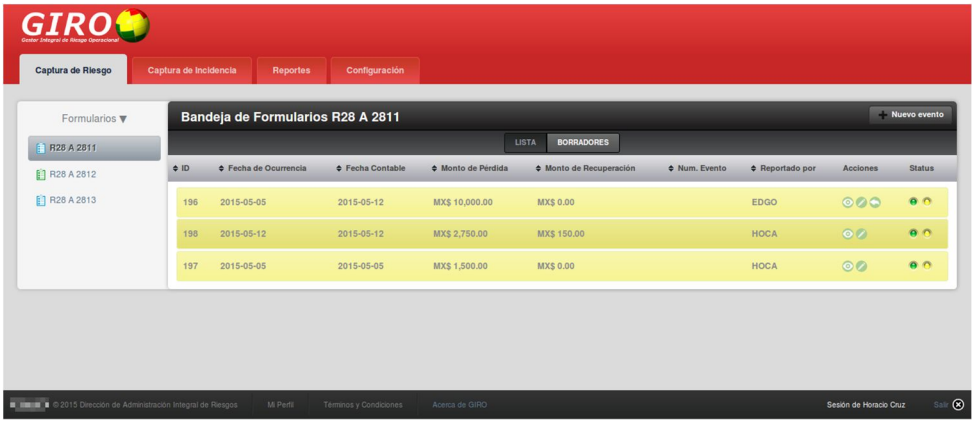

Figura 4.15 - Vista del listado de entradas en borradores

Un evento puede ser guardado para su próxima re-edición y posteriormente ser enviado para su revisión y liberación.

Cuando una entrada es liberada, puede verse el detalle de la misma mediante un reporte (como se detalla más adelante) o bien desde la opción Ver sobre el registro en la bandeja de entradas (Ver Figura 4.16).

4.13 Registro de incidencias

Para registrar una incidencia de riesgo operacional, el usuario deberá de ingresar a la sección *Captura de Incidencia*, en donde encontrará una bandeja de entrada similar a las que se encuentran en la sección *Captura de Riesgo*, de este modo puede ver todas aquellas incidencias que los usuarios han reportado.

Una vez en esta bandeja, el usuario hará clic sobre el botón *Nueva incidencia* y podrá reportarla mediante un formulario en el cual puede incluso adjuntar archivos que sirvan como evidencia. Ver **¡Error! No se encuentra el origen de la referencia.** y **¡Error! No se encuentra el origen de la referencia.**.

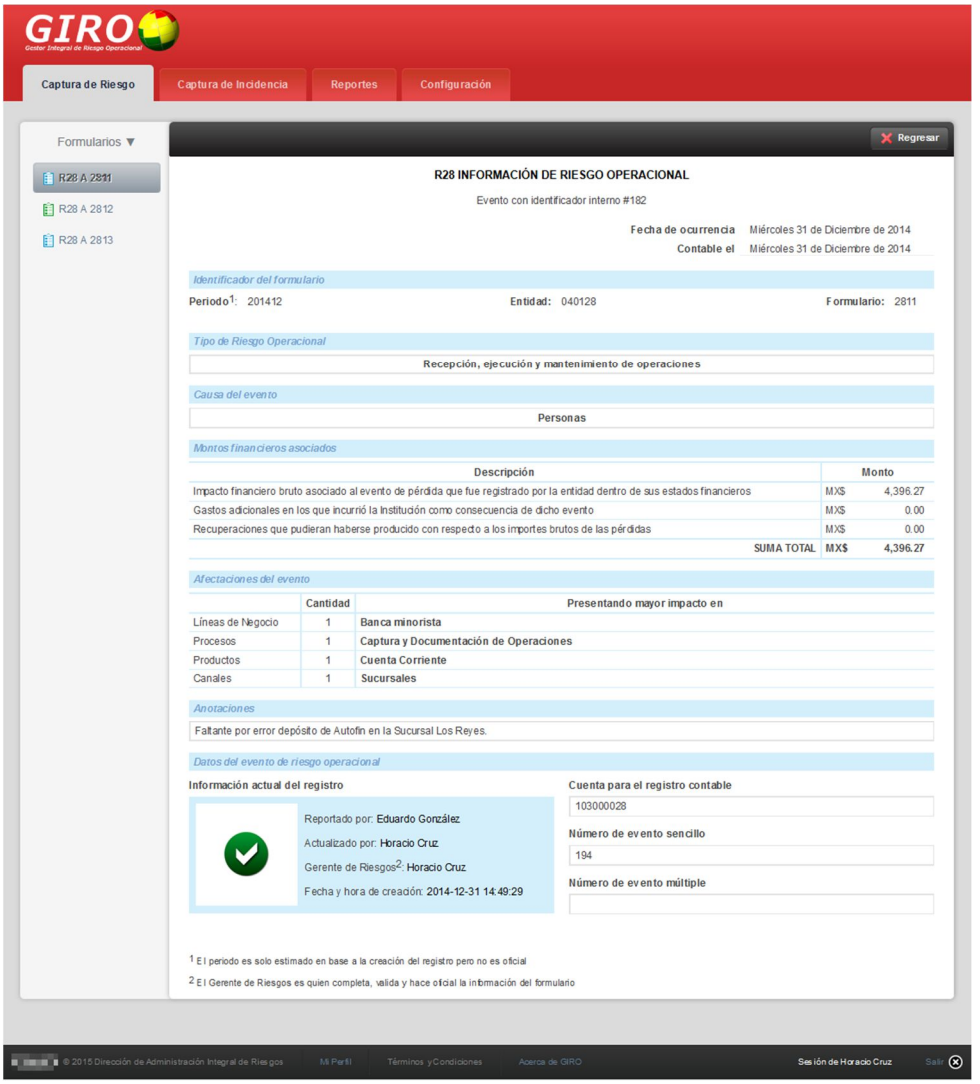

Figura 4.16 - Vista del detalle de un evento de pérdida

Nota: La diferencia entre un evento de perdida y una incidencia por riesgo operacional, es que el primero implica una pérdida económica consumada; mientras que la segunda aún no se consuma una posible pérdida relacionada.

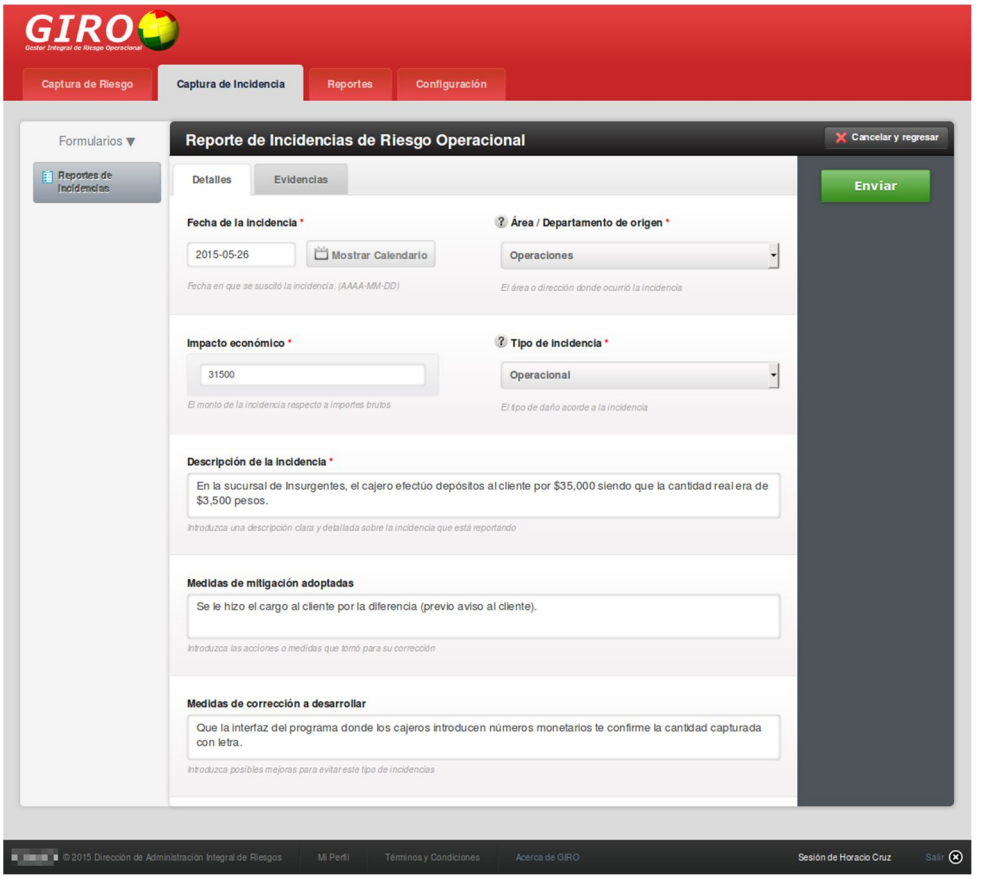

Figura 4.17 - Formulario de Incidencia de Riesgo Operacional, pestaña Detalles

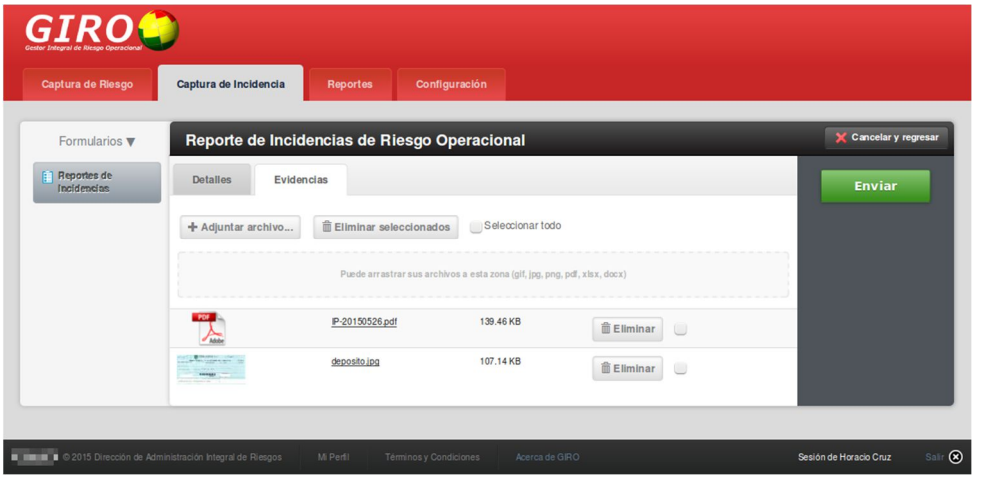

Figura 4.18 - Formulario de Incidencia de Riesgo Operacional, pestaña Evidencias

4.14 Generación de reportes

Para la generación de reportes nos ubicamos en la sección *Reportes* del menú principal, y elegimos el tipo de reporte en el submenú. A continuación aparecerá un listado con los registros finales y validados por un usuario con perfil de Gerente de Riesgos (GRIS); los filtros de registros pueden ser tan simples como un intervalo de fechas o avanzados que nos permitirían un filtrado por diferentes campos del registro. Una vez establecidos los filtros procederemos a hacer clic sobre el botón *Descargar como…* (Ver Figura 4.20). El sistema permite elegir el formato de descarga de reporte, pudiendo ser de tipo csv, xlsx o pdf (Ver **¡Error! No se encuentra el origen de la referencia.**).

4.15 Edición del perfil de Usuario

El usuario también puede editar datos de su cuenta en el sistema, pudiendo ser datos no sensibles como su dirección de correo electrónico, puesto, contraseña, entre otros (Ver Figura 4.19). Para ello debe ingresar a través del menú inferior del sistema en la opción *Mi Perfil,* el cual enviará al usuario al formulario con datos de su cuenta actual.

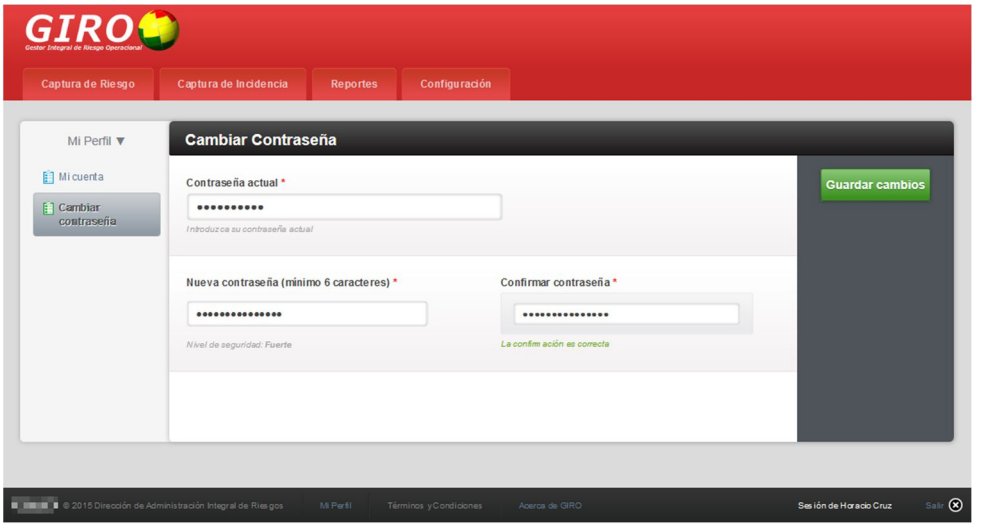

Figura 4.19 - Formulario "Mi Perfil"

4.16 Configuración

Es posible dependiendo del perfil de usuario, establecer configuraciones sobre el sistema, algunas de ellas son la gestión de usuarios, registro de ubicaciones físicas de la institución bancaria, catálogos, entre otras.

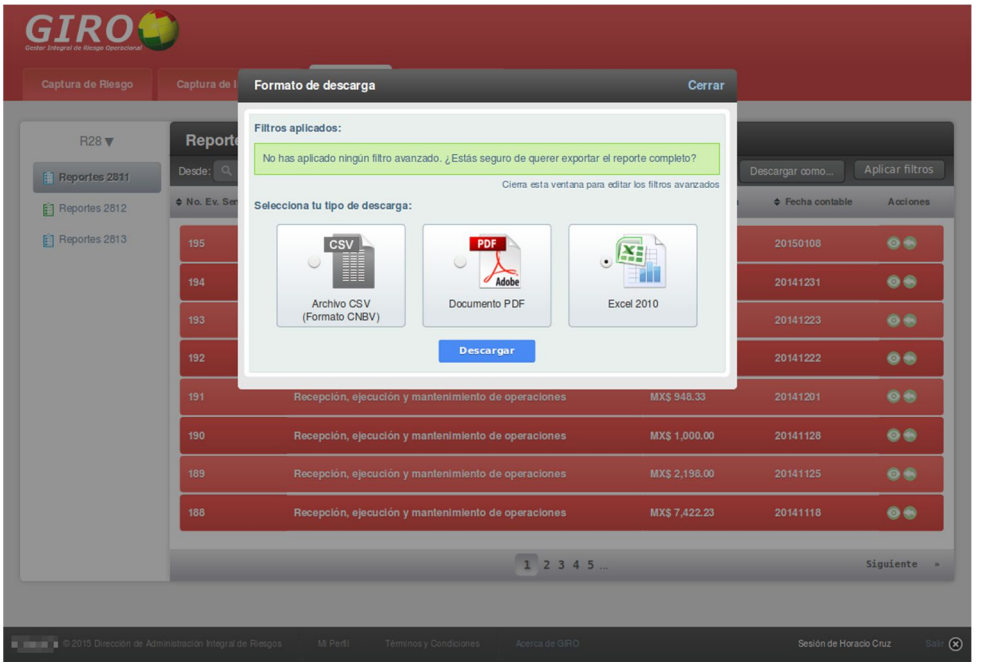

Figura 4.20 - Vista con la opción de descarga activa sobre reportes

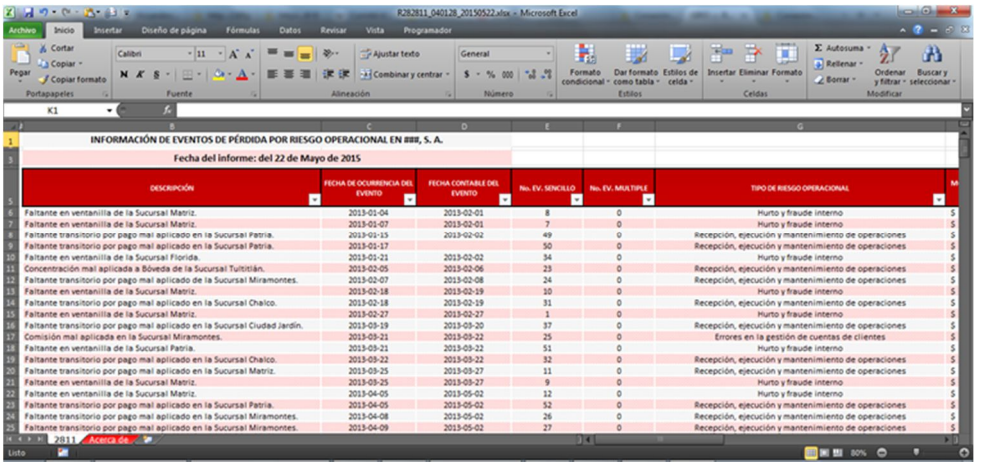

Figura 4.21 - Ejemplo de un archivo de tipo xlsx generado por el sistema

4.17 Registro de Ubicaciones físicas de la entidad bancaria

Para acceder al registro de Ubicaciones debe ir a *Configuración -> Mapa* dentro del cual se establece primeramente la cantidad de **Niveles** de profundidad (Véase 4.2 Configurando Niveles de Mapa).

Una vez establecido los niveles de profundidad, es posible agregar las **Ubicaciones** (Ver Figura 4.22), para ello seleccionamos del menú desplegable sobre cada nivel al cual se agregará cierto elemento (Véase 4.3 Trabajando con Ubicaciones de Mapa).

El formulario de agregación de ubicaciones es similar al de la Figura 4.4.

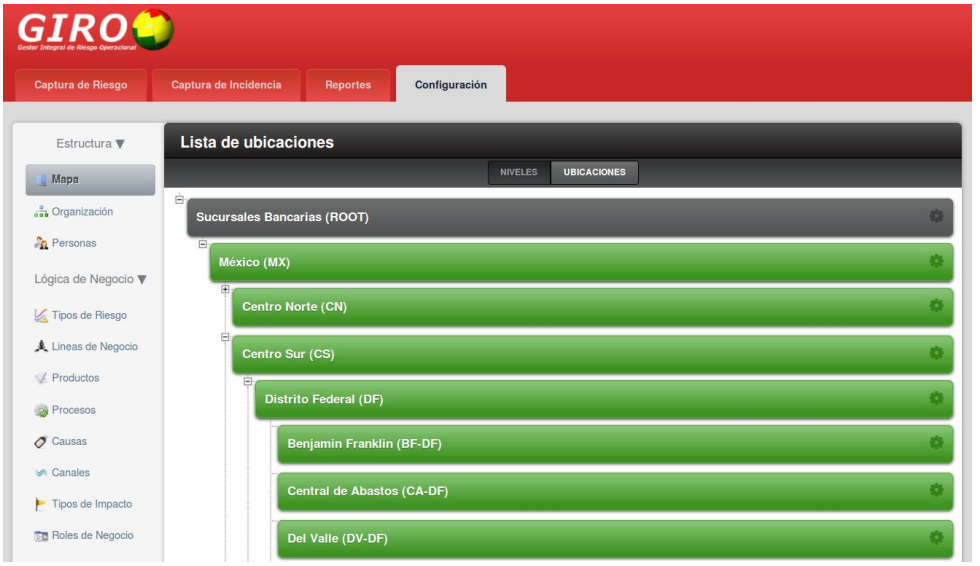

Figura 4.22 - Lista de ubicaciones de mapa

4.18 Registro de Organizaciones en la entidad bancaria

Para acceder al registro de Organizaciones debe ir a *Configuración -> Organizaciones* dentro del cual se establece primeramente la cantidad de **Niveles** de profundidad (Véase 4.4 Configurando Niveles de Organización).

Una vez establecido los niveles de profundidad, es posible agregar las **Unidades** (Ver Figura 4.23), para ello seleccionamos del menú desplegable sobre cada nivel al cual se agregará cierto elemento (Véase 4.5 Trabajando con Unidades Organizacionales).

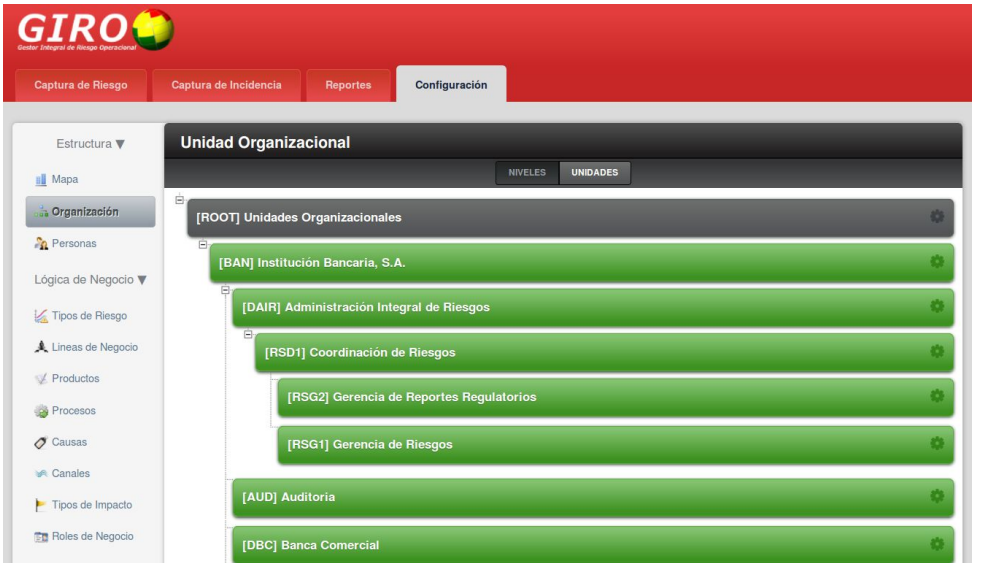

Figura 4.23 - Vista de la lista de unidades organizacionales

El formulario para la inserción de Organizaciones se muestra en la Figura 4.24.

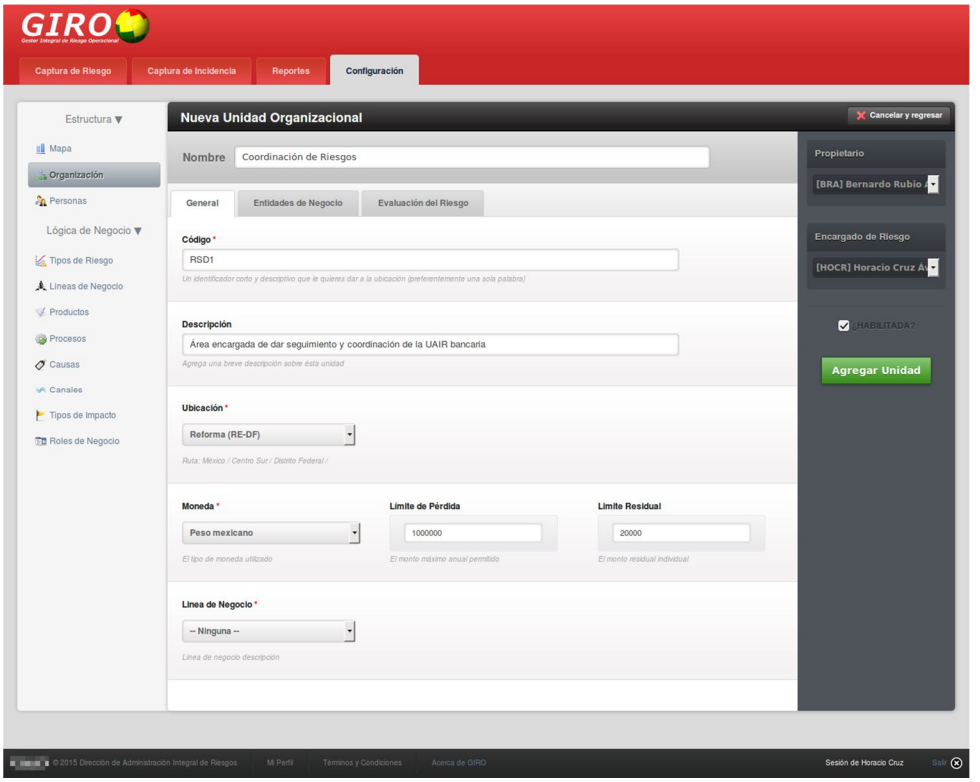

Figura 4.24 - Vista del formulario "Nueva Unidad Organizacional"

4.19 Creación del catálogo de *Tipos de riesgo*

Para acceder al catálogo de *Tipos de Riesgo* debe ir a *Configuración -> Tipos de Riesgo* dentro del cual se establece primeramente la cantidad de **Niveles** de profundidad (Véase 4.8 Gestión de catálogos).

Una vez establecido los niveles de profundidad, es posible agregar los elementos del **Catálogo**, para ello seleccionamos del menú desplegable sobre cada nivel al cual se agregará cierto elemento.

El formulario de agregación de elementos para el catálogo *Tipos de Riesgo* es similar al de la Figura 4.11.

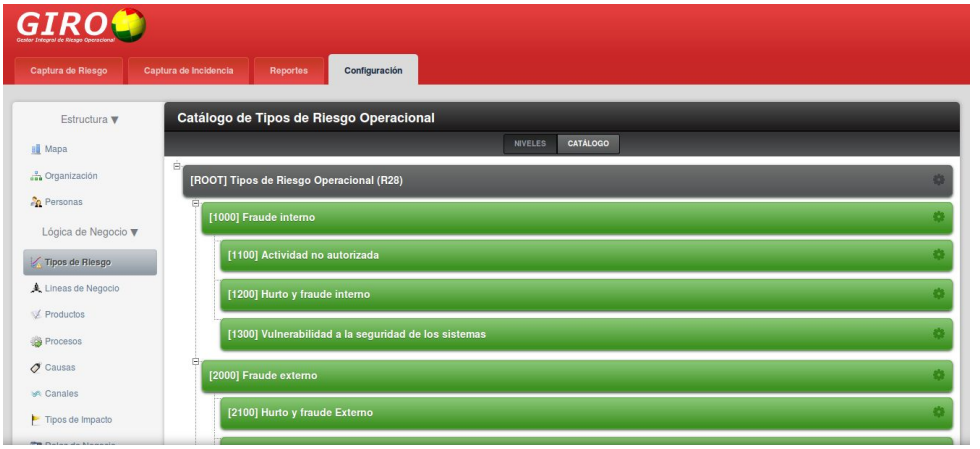

Figura 4.25 – Vista de la lista del catálogo de "Tipos de Riesgo Operacional"

4.20 Creación del catálogo de *Líneas de negocio*

Para acceder al catálogo de *Líneas de negocio* debe ir a *Configuración -> Líneas de Negocio* dentro del cual se establece primeramente la cantidad de **Niveles** de profundidad (Véase 4.8 Gestión de catálogos).

Una vez establecido los niveles de profundidad, es posible agregar los elementos del **Catálogo**, para ello seleccionamos del menú desplegable sobre cada nivel al cual se agregará cierto elemento.

Figura 4.26 - Vista de la lista del catálogo de "Líneas de Negocio"

El formulario de agregación de elementos para el catálogo *Líneas de Negocio* es similar al de la Figura 4.11.

4.21 Creación del catálogo de *Productos*

Para acceder al catálogo de *Productos* debe ir a *Configuración -> Productos* dentro del cual se establece primeramente la cantidad de **Niveles** de profundidad (Véase 4.8 Gestión de catálogos).

Una vez establecido los niveles de profundidad, es posible agregar los elementos del **Catálogo**, para ello seleccionamos del menú desplegable sobre cada nivel al cual se agregará cierto elemento.

El formulario de agregación de elementos para el catálogo *Productos* es similar al de la Figura 4.11.

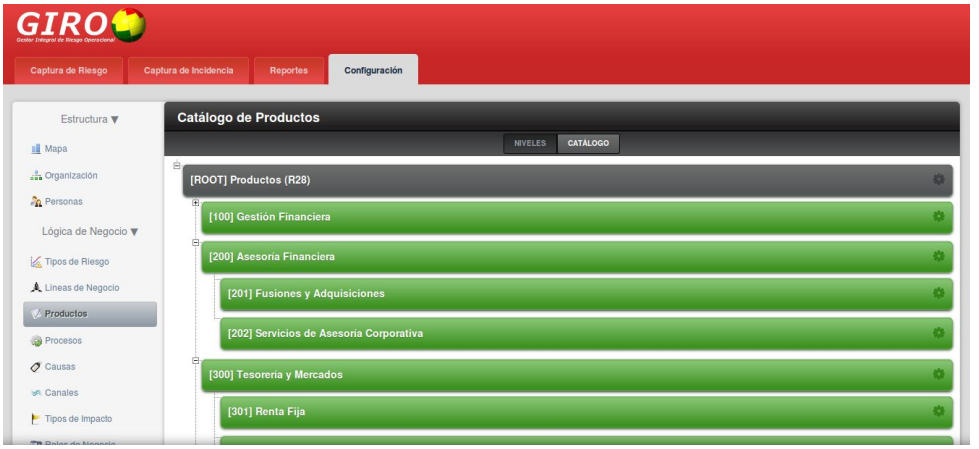

Figura 4.27 - Vista de la lista del catálogo de “Productos”

4.22 Creación del catálogo de *Procesos*

Para acceder al catálogo de *Procesos* debe ir a *Configuración -> Procesos* dentro del cual se establece primeramente la cantidad de **Niveles** de profundidad (Véase 4.8 Gestión de catálogos).

Una vez establecido los niveles de profundidad, es posible agregar los elementos del **Catálogo**, para ello seleccionamos del menú desplegable sobre cada nivel al cual se agregará cierto elemento.

El formulario de agregación de elementos para el catálogo *Procesos* es similar al de la Figura 4.11.

Figura 4.28 - Vista de la lista del catálogo de “Procesos”

4.23 Creación del catálogo de *Causas*

Para acceder al catálogo de *Causas* debe ir a *Configuración -> Causas* dentro del cual se establece primeramente la cantidad de **Niveles** de profundidad (Véase 4.8 Gestión de catálogos).

Una vez establecido los niveles de profundidad, es posible agregar los elementos del **Catálogo**, para ello seleccionamos del menú desplegable sobre cada nivel al cual se agregará cierto elemento.

El formulario de agregación de elementos para el catálogo *Causas* es similar al de la Figura 4.11.

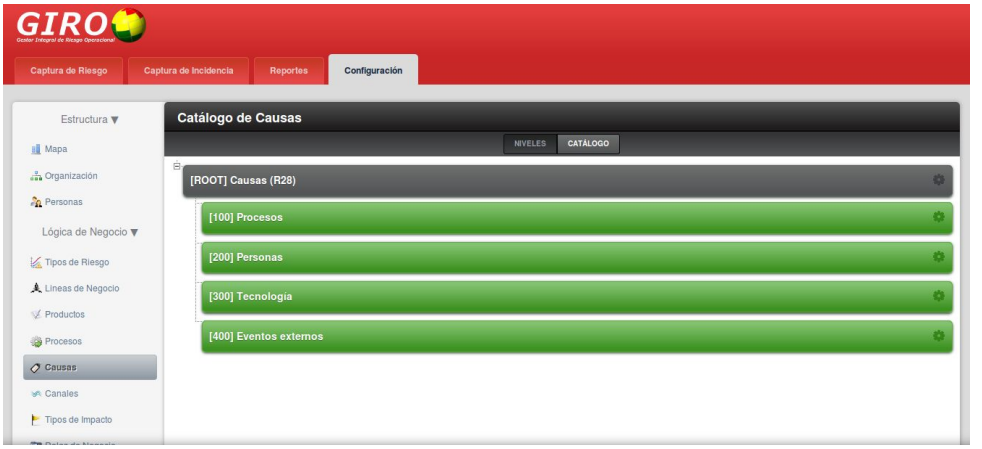

Figura 4.29 - Vista de la lista del catálogo de “Causas”

4.24 Creación del catálogo de *Canales*

Para acceder al catálogo de *Canales* debe ir a *Configuración -> Canales* dentro del cual se establece primeramente la cantidad de **Niveles** de profundidad (Véase 4.8 Gestión de catálogos).

Una vez establecido los niveles de profundidad, es posible agregar los elementos del **Catálogo**, para ello seleccionamos del menú desplegable sobre cada nivel al cual se agregará cierto elemento.

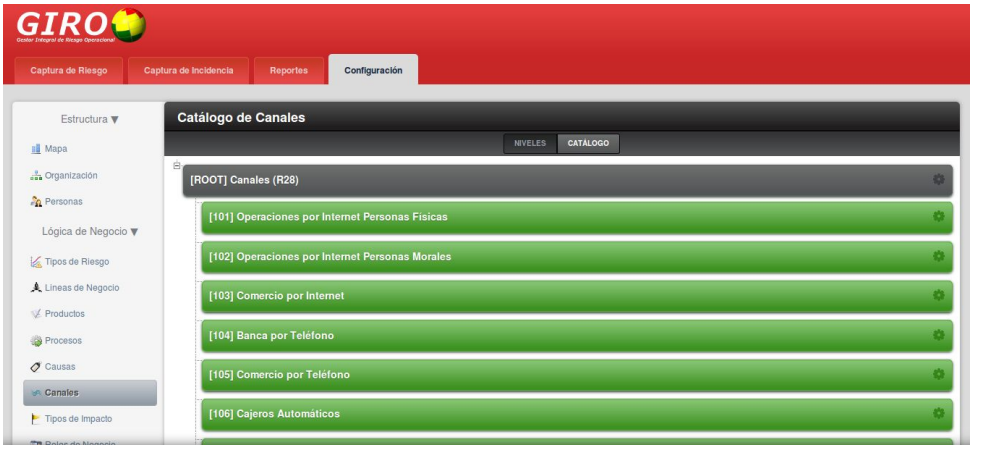

Figura 4.30 - Vista de la lista del catálogo de “Canales”

El formulario de agregación de elementos para el catálogo *Canales* es similar al de la Figura 4.11.

4.25 Registro de usuarios

Para acceder al formulario de inserción de usuarios (personas) debe ir a *Configuración -> Personas* dentro del cual podemos agregar o editar datos y ajustes para cada usuario (solo el perfil de *administrador* o usuarios con permisos específicos pueden efectuar estos cambios).

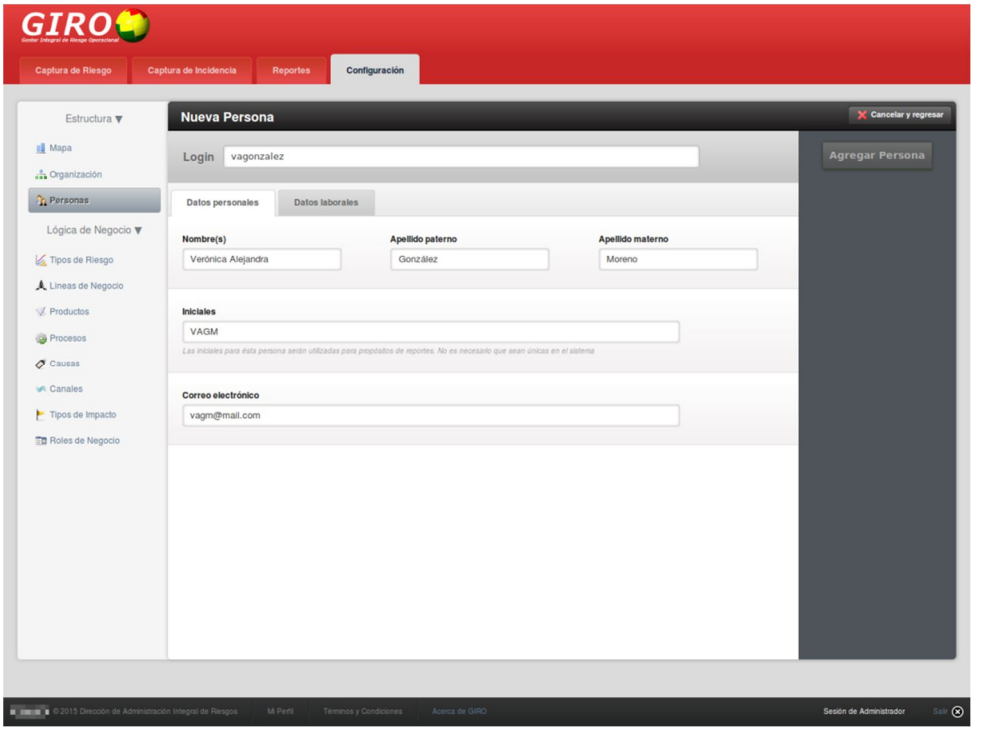

Figura 4.31 – Pestaña Datos personales del formulario "Nueva Persona"

El formulario para la gestión de usuarios permite registrar los datos personales y laborales de éste en el sistema (Ver Figura 4.31 y Figura 4.32).

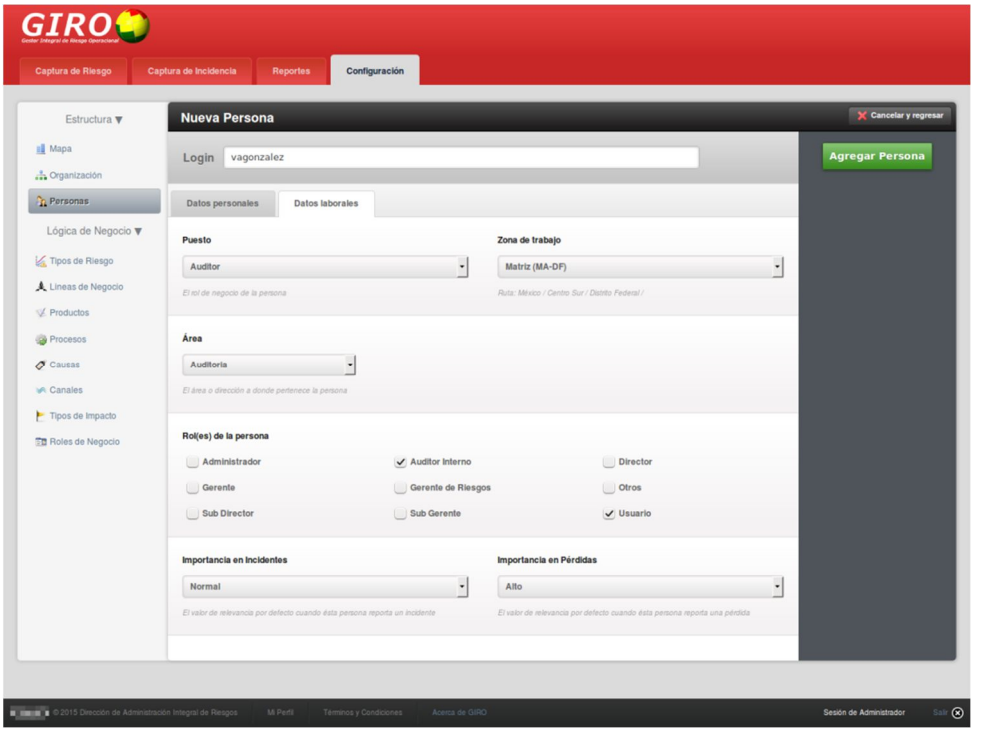

Figura 4.32 - Pestaña Datos laborales del formulario "Nueva Persona"

También es posible permitir, asignar roles, cambiar contraseñas o restablecer accesos de uso al sistema desde este menú, el cual se encuentra disponible solo con un perfil de *administrador*.

Resumen de resultados

El sistema web desarrollado mediante la plataforma genérica satisfizo las necesidades para la correcta administración de los eventos surgidos, en las diferentes áreas de la institución bancaria por riesgo operacional.

Además, la institución puede crear o restructurar las nuevas sucursales o unidades y asignarle los usuarios correspondientes a cada una, otorgándoles sus respectivas credenciales de perfiles o facultades y manteniendo el control de los reportes generados por cada sucursal o unidad.

Uno de los mejores aspectos es que conforme surjan nuevas disposiciones o recomendaciones por parte de las autoridades en lo que a riesgo operacional se refiere, como es el caso del Comité de Basilea, los catálogos pueden modificarse de manera sencilla y la generación de reportes será muy fluida, ya que la automatización permite además la eliminación de errores humanos.

Aún si dichas modificaciones al sistema provenientes por dichas recomendaciones fueran en un grado mayor, sería más sencilla la modificación del mismo, gracias a la estructura de directorios establecidos para la plataforma genérica, donde cada uno cumple con su función gracias a al patrón MVC.

Otra gran ventaja de éste sistema para la institución bancaria, es que al ser un desarrollo "*in-house*", todas estas posibles administraciones y actualizaciones no tendrán costo, ya que serán realizadas por el área de sistemas de la propia institución.

Conclusiones

El desarrollo de software realizado específicamente para una empresa o institución tiene grandes beneficios, ya que de ésta forma el software se ajusta a las necesidades del usuario y no el usuario a las limitantes del software. Y aún mejor es cuando provees una plataforma genérica que sirva como base para desarrollar los diferentes sistemas que la institución requiera, de éste modo el ahorro de capital es significativo, ya que el mantenimiento y expansión corre por cuenta del propio personal de la empresa.

La gran ventaja de desarrollar una plataforma genérica en lugar de utilizar *frameworks* como Zend, Symfony, Yii o algún otro, es el conocimiento total de la plataforma, así pues, el programador conoce las áreas de oportunidad y puede expandirlo de una manera más rápida sin tener que esperar a posibles actualizaciones que pudiesen derivar incluso, en una restructura de programación en el sistema.

Otra ventaja es que la plataforma solo cuenta con aquello que necesita, trayendo como consecuencia un mejor rendimiento que cualquier *framework* pueda ofrecer, ya que éstos son de carácter general y por tal, traen consigo numerosos módulos precargados que muy posiblemente no sean de utilidad para el sistema requerido.

Por último, es muy importante señalar que gracias a la preparación adquirida a lo largo de nuestra formación universitaria en la Facultad de Ciencias, nos ha permitido desarrollar este tipo de programas que involucran buenas prácticas de construcción y que también, implementan la facilidad de robustecerse conforme las necesidades lo requieran.

Bibliografía

"Disposiciones de carácter general aplicables a las instituciones de crédito (Circular Única de Bancos)", versión Noviembre 2014, en http://www.cnbv.gob.mx/SECTORES-SUPERVISADOS/BANCA-MULTIPLE/Paginas/Normatividad.aspx Consultada el 5 de enero de 2015.

Pacheco López, David, "Riesgo Operacional: Conceptos y Mediciones", Superintendencia de Bancos e Instituciones Financieras Chile, versión agosto 2009, en https://www.sbif.cl/sbifweb/internet/archivos/publicacion_8511.pdf Consultada el 16 de enero de 2015.

Rodríguez-Wyler Ortiz, Francisco Alonso, "Riesgo Operacional"; "Antecedentes y Generalidades del Acuerdo de Basilea II", Veritas, Colegio de Contadores Públicos de México, versión diciembre 2010, en http://www.ccpm.org.mx/veritas/diciembre2010/images/Riesgo_Operativo.pdf Consultada el 16 de enero de 2015.

"Comité de Basilea", versión abril 2015, en http://es.wikipedia.org/wiki/Comité_de_Basilea Consultada el 12 de mayo de 2015.

"Acuerdos de Basilea", versión marzo 2015, en http://es.wikipedia.org/wiki/Acuerdos_de_Basilea Consultada el 12 de mayo de 2015.

"Disposiciones de carácter general aplicables a las instituciones de crédito (Circular Única de Bancos)", versión diciembre 2005, en http://www.cnbv.gob.mx/Paginas/NORMATIVIDAD.aspx Consultada el 12 de mayo de 2015.

"Derivado financiero", versión abril 2015, en http://es.wikipedia.org/wiki/Derivado_financiero Consultada el 15 de mayo de 2015.

Buschmann, Frank, et al., "Pattern – Oriented Software Architecture A System of Patterns", Wiley Editorial, Alemania, 2001.

Potencier, Fabien y Zaninotto, François, "Symfony 1.1, la guía definitiva"; "El patrón MVC", versión de noviembre 2008, en http://librosweb.es/symfony_1_1/ Consultada el 20 de enero de 2015.

Rovira, Carlos, “Una Introducción a los conceptos clave en UML aplicados al desarrollo Flash”, versión septiembre 2004, en http://www.carlosrovira.com/en/una-introduccin-a-los-conceptos-clave-en-uml-aplicados-al-desarrollo-flash/ Consultada el 20 de enero de 2015.

Orozco, Sergio, et al., “Uso de UML en Aplicaciones Web”, versión febrero 2010, en http://sg.com.mx/content/view/584 Consultada el 20 de enero de 2015.