Universidad de las Ciencias Informáticas

Facultad 15

Libro de Ayuda del Marco de Trabajo Sauxe, En su versión 2.0.

Trabajo de Diploma para optar por el título de
Ingeniero en Ciencias Informáticas

Autor

Yanisleydi Cañete Pupo

Tutor

Ing. Marianela Tenrero Cabrera

Co-Tutor

Ing. Noel Jesús Rivero Pino

Ciudad de la Habana, Junio 2010

AGRADECIMIENTOS

El hecho de que este trabajo exista se debe primeramente al apoyo incondicional de mi mamá y mi hermana Yaimara, que sin ellas no hubiese podido llegar hasta donde he llegado. Mil gracias a mi tutora Marianela, a mi co-tutor Noel por haber puesto todos sus conocimientos para ayudarme en el desarrollo de este trabajo y a mi novio Oiner que además de mi novio y amigo, ha sido más que un tutor para mí. Gracias también a todas mis amistades, profesores y compañeros del proyecto, que han hecho que mi estancia en esta escuela sea más agradable y divertida, y a mi familia tanto a los de Camagüey, cómo a los de aquí de La Habana, a los de allá por apoyarme desde lejos y a los de aquí porque a pesar de que nunca los había visto, me acogieron y me hicieron parte de ellos con todo el amor del mundo.

DEDICATORIA

Este trabajo va dedicado primeramente a mis dos abuelitos (Reynerio y Rolando), a Reynerio por darme todo su amor y hacer el papel de padre cuando he necesitado de ese sentimiento y a Rolando porque a pasar que ya no se encuentra con nosotros siempre confió en mí y dijo que yo sería la primera ingeniera de la familia Cañete. A mi familia en general por darme la educación que tengo y por guiarme por los buenos caminos. A mi novio por estar siempre ahí, por aguantar todas mis malcriadeces, por aparecer sin que yo lo esperara pero tal como lo necesitaba.

"Un buen libro es aquel que se abre con interés y se cierra con provecho, que te enseña lo que debes hacer, te instruye sobre lo que debes evitar y te muestra el fin a que debes aspirar."

Alcott

RESUMEN

En el mundo existen numerosos Marcos de Trabajos que brindan un conjunto de componentes genéricos reutilizables que agilizan el proceso de desarrollo de software de gestión. A la universidad de las ciencias informáticas (UCI) en conjunto con la Unidad de Compatibilización, Integración y Desarrollo de Software para la Defensa (UCID) se le dio la tarea de desarrollar el sistema ERP más grande construido en Cuba hasta el momento. Para el desarrollo del mismo se necesitaba contar con una arquitectura tecnológica robusta que soportara los complejos procesos de gestión generadas por el sistema. Esta misión se le asignó al departamento de tecnología del Centro para la Informatización de la Gestión de Entidades (CEIGE). El mismo realizó un estudio de las principales plataformas de desarrollo para la construcción de sistemas de este tipo, como resultado del análisis estableció utilizar como Marco de Trabajo base Zend Framework por todas las ventajas que brinda. En el transcurso del desarrollo se identificaron un conjunto de escenarios que no cumplía Zend, desde este momento surge ZendExt Framework en el cual se agruparon todas las extensiones o desarrollo realizados para solucionar dichos escenarios. Como se trataban de nuevos componentes no contaban con la documentación necesaria para su entendimiento a la hora de extenderlos o reutilizarlos. Coherencia

Para darle solución a la problemática existente se decidió crear un libro de ayuda, el mismo está estructurado en cuatro capítulos, donde se describe la instalación y configuración de las herramientas necesarias para dar paso a la instalación del Marco de Trabajo que creará las bases para desarrollos posteriores, manual de instalación, configuración y uso de la arquitectura tecnológica, descripción de las taxonomías estructurales, vistas estilísticas, estándares de codificación, especificación de la capas y componentes que conforman la arquitectura.

Índice

Introducción

Los Sistemas de Planificación de Recursos de la Empresa (ERP), son sistemas de gestión de información que integran y automatizan muchas de las prácticas de negocio asociadas con los aspectos operativos o productivos de una empresa. Los sistemas ERP son sistemas integrales de gestión para la empresa. Se caracterizan por estar compuestos por diferentes partes integradas en una única aplicación. Estas partes son subsistemas o módulos que gestionan por ejemplo: producción, ventas, compras, logística, contabilidad (de varios tipos), gestión de proyectos, Sistema de Información Geográfica (GIS), inventarios y control de almacenes, pedidos, nóminas, etc. Solo se puede definir un ERP como la integración de todas las partes. Este tiene gran importancia ya que es una vía o alternativa para la mejora e integración de los procesos de negocio de la empresa trayendo consigo la obtención de ganancias. Las razones principales por las cuales las empresas implantan sistemas de este tipo, son: aumentar su competitividad, controlar mejor sus operaciones e integrar su información. (Iberia, 2010)

Para estimar el impacto que puede significar el hecho de contar con un verdadero sistema ERP para la economía cubana es necesario crear un equipo multidisciplinario que pueda disponer de los datos necesarios a nivel nacional, a pesar de esto se pueden identificar algunos elementos que pueden ser ilustrativos:

1. Recursos que se pierden o extravían por no tener un sistema de control efectivo.
2. El tiempo que se emplea en consolidar y relacionar datos manualmente en nuestras Organizaciones.
3. Gasto por concepto de toma de decisiones basadas en datos incorrectos o inexactos.
4. Tiempo y recursos que se requieren para hacer un levantamiento nacional de información en un sector determinado de la economía o los servicios.
5. Inversión en pagos de licencias de uso a empresas extranjeras por sistemas ERP que prácticamente nunca funcionan como tal y requieren luego de un largo proceso de “Soporte basado en Parches”.
6. Tiempo y recursos que se invierten en soporte y capacitación de los usuarios.

Este sistema tiene mucha importancia para la economía del país porque permite ahorrar cuantiosos recursos monetarios por concepto de importación de software, ya que de esta manera la nación no tendría que invertir fondos en la compra de licencias, plugins y otros softwares. De esta forma se obtiene la soberanía tecnológica pues no se tendría que depender de otros países en la elaboración de programas o actualizaciones para poder usarlos. Todo esto trae consigo la uniformidad en el procesamiento de la información porque todas las empresas cubanas tendrían el mismo sistema contribuyendo así al ordenamiento institucional.

Como los sistemas ERP son muy costosos, se le dio a la Universidad de las Ciencias Informáticas (UCI) en conjunto con la Unidad de Compatibilización, Integración y Desarrollo de Software para la Defensa (UCID) la tarea de desarrollar un ERP que cuente con todos los módulos necesarios para la gestión empresarial del país. La magnitud, complejidad y seguridad de un sistema de este tipo requiere hacer un diseño arquitectónico potente que soporte la gestión y transferencia de información. Para ello se creó un grupo central de Arquitectura que fue definiendo por cada capa, los componentes que formarían parte de la línea base. De estos componentes, algunos fueron reutilizados y otros extendidos de Zend Framework para dar lugar al Marco de Trabajo de Cedrux que adoptó el nombre de Sauxe. Éste contiene un conjunto de componentes reutilizables que provee la estructura genérica y el comportamiento para una familia de abstracciones, logrando una mayor estandarización, flexibilidad, integración y agilidad en el proceso de desarrollo.

A medida que el sistema Cedrux avanza, mayores son los requerimientos que debe cumplir la Arquitectura para soportar y brindar las prestaciones requeridas. Actualmente, Sauxe se encuentra en la versión 2.0 y continúa desarrollando tecnologías robustas y cada vez son más los interesados en aprenderlas y trabajar sobre esta Arquitectura. Por el nivel de integración, asimilación y robustez de este Marco de Trabajo se encuentra generalizado en 15 proyectos de la UCI (Aduana, Akademos2.0, Sistema para la gestión de las transportaciones militares en las FAR(DITRANS), Fuerza de Trabajo Calificada (FTC), Generador de reportes, Sistema de supervisión y control de los PSI (Hoyo), Informatización UCID, Plataforma Soberana para el Desarrollo de Sistemas de Información Geográfica (GENESIG), Minería de datos, (SEUNE), Sistema Informático para la Gestión de Auditoría y Control SIGAC, Sistema de Mando y Estado Mayor (SIMEM), Sistema de gestión estadística, Sistema de Registro, Control y Análisis de los Medios y Equipos de la Reserva Militar(SRCAMERM), el MINFAR y otras entidades del país como el Polo de Desarrollo de Holguín alcanzando resultados satisfactorios. La dirección de producción de la Universidad basada en los resultados obtenidos decidió que todos los proyectos de gestión en PHP que se comenzaran a desarrollar debían utilizar este Marco de Trabajo. (Baryo & García, 2009)

Por tanto se hace indispensable el poder contar con una documentación precisa, clara, formalizada y entendible para que tanto los desarrolladores de Cedrux así como los clientes o usuarios finales puedan interactuar sin dificultades con dicha Arquitectura.

En este momento no se cuenta con una documentación de este tipo lo que trae como consecuencia dudas a la hora de reutilizar sus componentes, incremento del tiempo de estudio, mayor necesidad por parte de los desarrolladores de realizar consultas constantes al equipo de Arquitectura, lo que puede desencadenar un atraso considerable en los cronogramas de trabajo además del descontento por parte de los usuarios finales.

Dada la situación planteada anteriormente el **problema a resolver** queda formulado de la siguiente manera: La documentación existente de la Arquitectura Tecnológica de Sauxe no permite entender con claridad a los desarrolladores y usuarios finales cómo usar sus componentes y los escenarios a los que da solución.

El **objeto de estudio** de este trabajo lo constituye la arquitectura de software.

El **campo de acción** queda enmarcado en la Arquitectura tecnológica para aplicaciones web de gestión en entornos multi-entidad.

El **objetivo general** es desarrollar el libro de ayuda de la Arquitectura Tecnológica de Sauxe versión 2.0.

Para dar cumplimiento a este objetivo general se han trazado los siguientes **objetivos específicos**:

- Recopilar la documentación existente de la Arquitectura Tecnológica de Sauxe.
- Actualizar manual de instalación y configuración de las herramientas.
- Realizar manual de instalación y configuración del Marco de Trabajo Sauxe.
- Realizar estándar de codificación. Realizar estilo de código para PHP y JavaScript.
- Definir taxonomía arquitectónica.
- Especificar rasgos arquitectónicos del Marco de Trabajo por cada componente.

Se tiene como **idea a defender** de la presente investigación:

Si se formaliza y centraliza en un libro de ayuda la documentación de la Arquitectura Tecnológica Sauxe versión 2.0, se logrará entender con claridad cómo usar sus componentes y los escenarios a los que da solución.

Resultado Esperado:
La documentación necesaria que cumpla con la organización y actualización de la Arquitectura Tecnológica de Sauxe.

Estructura de los capítulos de la tesis
La tesis estará compuesta por cuatro capítulos:
Capítulo 1: Instalación y configuración de las herramientas.
Capítulo 2: Instalación y configuración de Sauxe.
Capítulo 3: Estilo de codificación para PHP y JavaScript, taxonomía arquitectónica y conectores de estilo.
Capítulo 4: Descripción de cada uno de los componentes de Sauxe 2.0.

1 Capítulo 1: Instalación y configuración de herramientas.

Fundamentación teórica.

Introducción

En este capítulo se mencionan los diferentes Marcos de Trabajo analizados y estudiados. Se exponen las principales características de Zend así como la extensión que surgió de él. Esta extensión junto con ExtJS que es una potente librería y Doctrine que es el Mapeador de Objeto Relacional, ayudaron a formar lo que es actualmente Sauxe. Se hace un detallado estudio de la documentación del Marco de Trabajo Zend Framework. Se exponen además, algunas características y actividades del modelo que se utiliza en el Departamento de Tecnología basado en las diferentes metodologías ágiles. Además se dará a conocer el impacto que tiene el Marco de Trabajo y por último se ofrece una detallada descripción sobre la instalación y configuración de las herramientas que se necesitan para luego dar paso a la instalación del mismo.

1.1 Estado del arte.

1.1.1 Marcos de Trabajos.

Los Marcos de Trabajos son desarrollados con el objetivo de brindarles a los programadores y diseñadores una mejor organización y estructura a sus proyectos. En sus inicios para llevar a cabo el desarrollo de Sauxe, se hizo necesario realizar un estudio de los Marcos de Trabajos más reconocidos mundialmente por las facilidades que brindan a la hora de optimizar código, y estandarizar la programación de una manera eficiente. Como principales y más usados se destacan: Symfony, CakePHP, CodeIgniter y KumbiaPHP y Zend Framework. De ellos, se seleccionó este último debido a que brinda facilidades para desarrollar aplicaciones y servicios web, además por su facilidad a la hora de extender sus componentes.

1.1.2 Zend Framework.

Se trata de un Marco de Trabajo para el desarrollo de aplicaciones y servicios Web con PHP, brinda soluciones para construir sitios web modernos, robustos y seguros. Este Marco de Trabajo está formado por una serie de métodos estáticos y componentes que usarán estos métodos. Los componentes son varios y variados y aunque alguno es posible que no se use nunca, hay otras que puede que se usen hasta la saciedad, por ejemplo el componente para la base de dato. Como características Zend Framework fundamentales tiene:

- Trabaja con Modelo Vista Controlador (MVC).

- El Marco de Zend también incluye objetos de las diferentes bases de datos, por lo que es extremadamente simple para consultar su base de datos, sin tener que escribir ninguna consulta SQL.
- Una solución para el acceso a base de datos que balancea el Mapeador de Objeto Relacional con eficiencia y simplicidad.
- Completa documentación y test de alta calidad.
- Soporte avanzado.
- Robustas clases para autenticación y filtrado de entrada.
- Muchas otras clases útiles para hacerlo tan productivo como sea posible. (Leopoldo, 2007)

Los 51 componentes de Zend Framework conforman un potente y extensible Marco de Trabajo de aplicaciones web al combinarse entre sí. De todos estos Sauxe utiliza solamente los siguientes:

1. Zend_Cache
2. Zend_Config
3. Zend_Controller
4. Zend_Loader,
5. Zend_Log
6. Zend_Registry
7. Zend_Session,
8. Zend_View

A partir de la extensión de algunos componentes de Zend Framework surge ZendExt, desarrollado por el Departamento de Tecnología y la UCID, con el objetivo de crear un Marco de Trabajo extensible y configurable centrando el desarrollo de las aplicaciones, en la lógica del negocio, en las interfaces de usuario, alejando a los programadores de los detalles arquitectónicos, con soporte para entornos multi-entidad y para una arquitectura de sistema orientada a componentes. (Baryolo, Tenrero, & Silega, 2008)

La unión de ZendExt, Doctrine y ExtJS dio lugar al Marco de Trabajo Sauxe.

1.1.3 Marco de trabajo Sauxe.

Sauxe es un Marco de Trabajo que contiene un conjunto de componentes reutilizables que provee la estructura genérica y el comportamiento para una familia de abstracciones, logrando una mayor estandarización, flexibilidad, integración y agilidad en el proceso de desarrollo. (Baryolo, Tenrero, & Silega, 2008)

Sauxe en su versión 2.0 está formado por los componentes: ZendExt_App, ZendExt_Aspect, ZendExt_ADT, ZendExt_Cache, ZendExt_ExpImp, ZendExt_MVC, ZendExt_Exception,

ZendExt_FastResponse, ZendExt_GlobalConcept, ZendExt_IoC, ZendExt_Nomencladores, ZendExt_Trace, ZendExt_Validation, ZendExt_Portal, ZendExt_TransactionManager.

1.1.4 Doctrine.

Doctrine es un potente y completo sistema Mapeador de Objeto Relacional (ORM) para PHP 5.2.3 + que incorpora una Capa de Abstracción a Base de Datos (DBL). (Server Gove, 2010)

Sauxe utiliza en la capa de acceso a datos el Lenguaje de Consulta de Datos (DQL) que implementa Doctrine. Este es un potente y completo sistema ORM para PHP 5.2.3+ con un DBAL (Capa de Abstracción de Base de Datos) incorporado. Se está empezando a ver su potencial, pero de la documentación se puede decir que tiene todas las características necesarias para ser funcional en casi cualquier proyecto. Entre otros elementos se tiene la posibilidad de exportar una base de datos existente a sus clases correspondientes y también a la inversa, es decir convertir clases (convenientemente creadas siguiendo las pautas del ORM) a tablas de una base de datos. Por otro lado, como la librería es bastante grande ésta tiene un método para ser 'compilada' al pasar a producción. (Baryolo, Tenrero, & Silega, 2008)

Ventajas que facilitan enormemente tareas comunes y de mantenimiento:

Reutilización: La principal ventaja que aporta un ORM es la reutilización permitiendo llamar a los métodos de un objeto de datos desde distintas partes de la aplicación e incluso desde diferentes aplicaciones.

Encapsulación: La capa ORM encapsula la lógica de los datos pudiendo hacer cambios que afectan a toda la aplicación únicamente modificando una función.

Portabilidad: Utilizar una capa de abstracción que permite cambiar en mitad de un proyecto de una base de datos MySQL (Sistema de gestión de base de datos relacional) a una Oracle sin ningún tipo de complicación.

Esto es debido a que no se utiliza una sintaxis (MySQL, Oracle o SQLite) para acceder a nuestro modelo, sino una sintaxis propia del ORM utilizado que es capaz de traducir a diferentes tipos de bases de datos.

Seguridad: Los ORM suelen implementar mecanismos de seguridad que protegen nuestra aplicación de los ataques más comunes como una Inyección SQL.

Mantenimiento del código: Gracias al correcto ordenamiento de la capa de datos, modificar y mantener nuestro código es una tarea sencilla (Mata, 2009).

1.1.5 ExtJS.

Como se expuso anteriormente Sauxe también contiene ExtJS.

ExtJS es una librería JavaScript ligera y de alto rendimiento, compatible con la mayoría de los navegadores que permite crear páginas e interfaces web dinámicas. (Comunidad de desarrollo, 2006-2010)

Sauxe utiliza ExtJS en la capa de presentación, por la gran gama de componentes que se pueden reutilizar para agilizar el proceso de desarrollo y mostrarle al usuario una interfaz más amigable y funcional. (Baryolo, Tenrero, & Silega, 2008)

Esta librería incluye:

- Componentes Interfaz de Usuario (UI) del alto performance y personalizables.
- Modelo de componentes extensibles.
- Un API fácil de usar.
- Licencias de códigos abiertos y comerciales. (Corso, 2008)

Una de las grandes ventajas de utilizar ExtJS es que permite crear aplicaciones complejas utilizando componentes predefinidos así como un manejador de layouts similar al que provee Java Swing, gracias a esto provee una experiencia consistente sobre cualquier navegador, evitando el tedioso problema de validar que el código escrito funcione bien en cada uno (Firefox, Internet Explorer, Safari, etc.). Además, la ventana flotante que provee ExtJS es excelente por la forma en la que funciona. Al moverla o redimensionarla solo se dibujan los bordes haciendo que el movimiento sea fluido lo cual le da una ventaja tremenda frente a otros. (Corso, 2008)

Como se ha visto hasta el momento Sauxe es el resultado de una importante mezcla entre: la extensión de algunos componentes de un potente Marco de Trabajo, un sistema ORM y una muy popular librería, dando lugar a un novedoso Marco de Trabajo, con funcionalidades y prestaciones específicas pero también fácilmente extensibles a cualquier aplicación web de gestión.

1.1.6 Documentación de Zend Framework.

A la hora de desarrollar la documentación de Sauxe, se partió del análisis de la documentación Zend Framework, porque fue el Marco de Trabajo del cual se partió para extender sus componentes.

Zend Framework contiene una documentación abarcadora, explícita y actualizada, cuenta con una "Guía de referencia para programadores" en la que se detallan todos y cada uno de sus componentes, también contiene una lista extensa de ejemplos en el que se especifican diferentes casos que pueden surgir durante el desarrollo. También contiene una "Guía de estudio", en la cual se explican elementos básicos sobre Zend Framework, así como una guía de actividades que permite ejercitar conocimientos. Ambas guías se encuentran en idioma Inglés. Además en la web tiene publicado un sitio en el que aparece la descripción de todos los componentes, algunos de estos han sido traducidos al español. Este sitio, da la posibilidad a todos aquellos usuarios interesados en colaborar con la comunidad de Zend, de realizar la traducción al español de los componentes que

deseen y que ésta sea publicada en la web, este sitio se encuentra en la dirección: http://zfdes.com/index.php/Documentación .

Independientemente de la extensa y abarcadora documentación con la que cuenta Zend, no llega a ser suficiente para Sauxe, debido a que, aunque éste lo utiliza. Sauxe tiene extensiones de algunos de sus componentes dando lugar a un nuevo Marco de Trabajo, que cuenta además con Doctrine para la gestión de la capa de Acceso a Datos y ExtJS en la capa de presentación. Por tanto, surge la necesidad de desarrollar una documentación a la medida que se ajuste a los elementos, particularidades y características propias de Sauxe.

1.1.7 Propuesta de modelo de desarrollo.

El desarrollo de la documentación se hará utilizando una propuesta de modelo de desarrollo elaborada en el Departamento de Tecnología, la misma tiene su basamento en los principios y buenas prácticas de las metodologías ágiles estudiadas para garantizar rapidez y un buen funcionamiento en el desarrollo de los procesos. (Pérez & Hernández, 2009)

El modelo provee los procesos, actividades, roles involucrados y artefactos generados durante el proceso de desarrollo tecnológico en la Subdirección Tecnológica del CEIGE. Este está formado por 8 procesos y una propuesta de estructura organizativa. Cada proceso es especificado teniendo en cuenta su flujo de actividades, los roles involucrados y artefactos generados en cada actividad. (Pérez & Hernández, 2009)

La documentación se guiará de acuerdo a lo que establece el “Proceso de desarrollo de componentes tecnológicos”.

Este proceso inicia cuando se solicita a la Subdirección Técnica del centro iniciar una iteración de desarrollo tecnológico, a partir de una lista de requisitos. (Pérez & Hernández, 2009)

Este proceso contiene un flujo de actividades a seguir para darle cumplimiento al mismo, para el desarrollo de la documentación las actividades a tener en cuenta son:

Actividad 4: Descripción y especificación del estilo arquitectónico, Marcos de Trabajos, librerías y componentes tecnológicos.

Descripción: A partir de una orden emitida por los arquitectos tecnológicos comienza a desarrollarse esta actividad que consiste en la descripción y especificación del estilo arquitectónico, los Marcos de Trabajos y librerías. En esta actividad se describen y especifican además los componentes tecnológicos con sus conectores registrando los requisitos y la responsabilidad arquitectónica de los mismos en el documento de especificación técnica de componentes**,** así como, las relaciones y restricciones de acuerdo a los atributos de calidad y las categorías o aspectos arquitectónicamente significativos de la aplicación.

Artefactos:

- Documento de especificación técnica de componentes.

Objetivo: Descripción de los requisitos funcionales y no funcionales y especificación de los prototipos de interfaz de usuario por cada componente tecnológico definido.

Actividad 6.3: Desarrollo de la documentación del Marco de trabajo.

Descripción: Desarrollar toda la documentación necesaria para utilizar e implantar el Marco de Trabajo.

Artefactos:

- Manual de usuario.

Objetivo: Dar a conocer a los usuarios finales las funcionalidades y el modo de usar el software.

- Video tutorial.

Objetivo: Dar a conocer a los usuarios finales las funcionalidades y el modo de usar el software.

- Caso de estudio.

Objetivo: Guiar a los desarrolladores en el uso del marco de trabajo.

- Manual de instalación.

Objetivo: Explicar cada uno de los pasos a seguir para la instalación del software.

1.1.8 Sauxe. Impacto.

1.1.8.1 Impacto Económico.

Con el desarrollo y reutilización de Sauxe se ahorran cuantiosos recursos humanos y materiales puesto que adquirir una tecnología de este tipo para desarrollar aplicaciones de gestión resultaría muy costoso; el valor más alto de esta Arquitectura está en la reutilización por parte de los proyectos que se decidan desarrollar en lenguaje PHP, estos proyectos no tendrían que invertir esfuerzos ni recursos en construir todos los componentes externos puesto que este Marco de Trabajo se ocupa de ellos, solo tienen que centrarse en implementar el sistema específicamente. Algo muy importante es que se han creado nuestros propios componentes libres de huecos de seguridad que puedan ser utilizados para realizar un ataque y causar daños económicos irreparables. Esta primera versión puede ser comercializada y el país obtener por ella cuantiosas sumas ya que lo más importante para un proyecto es dar una solución rápida, vistosa, eficiente y Sauxe garantiza estos aspectos.

1.1.8.2 Impacto Social.

Este Marco de Trabajo ha tenido un gran impacto social en la formación de desarrolladores puesto que estos han aprendido todo lo referente al desarrollo de aplicaciones web y el trabajo en equipo de forma organizada, lo cual facilita ahorro de tiempo y mejor calidad de desarrollo. Ha posibilitado a los usuarios finales obtener soluciones con entornos de trabajos amigables, amplio nivel de configuración y altos niveles de integración, el producto se encuentra debidamente documentado permitiendo que usuarios y desarrolladores dominen a fondo todas las funcionalidades y puedan adaptarlo a sus necesidades.

1.1.8.3 Impacto Tecnológico Novedoso.

Es un producto novedoso desarrollado sobre tecnologías libres como el lenguaje PHP, gestor de base de datos Postgres, servidor web Apache, etc., aún cuando en el mundo los principales productos de este tipo están implementados sobre otras tecnologías. Su administración centralizada de todos los aspectos necesarios a tener en cuenta en el desarrollo de aplicaciones de gestión lo convierte en un producto de punta en esta rama. Permite realizar un número de funcionalidades que hace esta Arquitectura muy aplicable para cualquier entorno web en PHP. Es la primera vez que un producto de este tipo permite la gestión de multi-entidad y con esta la compartimentación de la información de cada una de ellas. Además hasta hoy no se realizaba la administración dinámica de perfiles de usuarios que permitiera a las empresas controlar todos los datos deseados de los usuarios. Tampoco se permitía la administración de conexiones a las bases de datos y los permisos sobre estas de forma centralizada. Algo novedoso también resulta la incorporación de la auditoría, que permita consultar todas las acciones realizadas en los sistemas, con toda la información asociada, dígase tiempo, valores, interacción. El tener incorporado un componente que se encarga de la gestión dinámica de estructuras de un país lo convierte en una tecnología aún más potente y reutilizable.

1.1.9 Impacto de la documentación.

El desarrollo de la documentación del Marco de Trabajo traerá consigo un impacto considerable tanto en la productividad del programador como en la integridad del sistema así como en la usabilidad del software.

1.1.9.1 Impacto en la productividad del programador.

La existencia de una documentación detallada, constituirá para los desarrolladores una guía referencial que les permitirá realizar las tareas acordes a su rol, tomando de ésta los conocimientos que le son necesarios para el desarrollo e implantación de los componentes. Esto permite reducir el número de consultas a los arquitectos, ahorro de tiempo y mayor uniformidad en el proceso de programación.

1.1.9.2 Impacto en la integridad del sistema.

La documentación debe estar centralizada y completa, de esta forma se evita tener que realizar encuentros innecesarios, talleres, etc. Esto contribuye a que el sistema sea cada vez más robusto, tenga mayor rendimiento y cumpla con todos los requerimientos especificados por el cliente.

1.1.9.3 Impacto en la usabilidad del software.

Contar con una documentación fiable, y sólida en su contenido, es esencial para llevar a cabo cualquier actividad de desarrollo. Es fundamental para el logro en tiempo y forma de cualquier tarea productiva; constituye una fuente de motivación para los responsables de llevar a cabo el desarrollo

de determinado producto. Esto asegurará mayores resultados en la utilización del proyecto, mayor auge en la venta del producto así como la satisfacción del equipo de desarrollo y usuarios finales.

Desarrollo.

1.2 Instalación y Configuración de herramientas.

Este manual es una guía de todos los pasos y procedimientos que se deben seguir para alcanzar resultados óptimos en cuanto al desarrollo de aplicaciones con el Marco de Trabajo Sauxe en su versión 2.0, y tiene como objetivo fundamental lograr que el usuario adquiera conocimientos en cuanto a cómo instalar y configurar cada una de las herramientas requeridas para el trabajo con este Marco de Trabajo.

Windows

Para los usuarios que trabajan con sistema operativo Windows se utilizaran las herramientas siguientes:

1. Cliente SVN TorstoiseSVN para manejar el control de versiones.
2. Wamp para instalar los servicios de Apache como servidor web y PHP como lenguaje de programación.
3. Configuración de un Virtualhost para la creación de máquinas virtuales.
4. PostgreSQL 8.3 como Gestor de Base de Datos.
5. Mozilla Firefox como explorador web u otro que implemente el DOM 2.0 y que soporte JavaScript.
6. IDE de desarrollo para PHP y/o JavaScript.
    - ZendStudio Neon para PHP y para Zend Framework.
    - Eclipse 3.3, agregar plugin para PHP y plugin para JavaScript (Spket o Aptana).
    - Aptana para JavaScript.
    - Spket para JavaScript.

Linux

Para los usuarios que trabajan con sistema operativo Linux se utilizaran las herramientas siguientes:

1. Cliente SVN RapidSVN para manejar el control de versiones.
2. Apache 2.0 como servidor web.
3. PHP 5.2.4 como lenguaje de programación.
4. PostgreSQL 8.3 como Gestor de Base de Datos
5. Configuración de un Virtualhost para la creación de máquinas virtuales.
6. Mozilla Firefox como explorador web u otro que implemente el DOM 2.0 y que soporte JavaScript.
7. IDE de desarrollo para PHP y/o JavaScript.
    - ZendStudio Neon para PHP y para Zend Framework.

En los anexos se mostrará con detalles la instalación y configuración de las herramientas.

1.3 Requerimientos de Software y Hardware.

Además de las herramientas explicadas anteriormente, se deben tener en cuenta algunos requerimientos específicos, para lograr óptimos resultados en el desarrollo de aplicaciones sobre el Marco de Trabajo.

	Cliente	Servidor (1)	Servidor (2)
Software	• Mozilla Firefox 2.2	• Ubuntu Server. • Apache 2.0 • PHP 5	• Ubuntu Server. • PostgreSQL 8.3
Hardware	• Procesador: 1.40 GHZ • RAM: 256 MB (recomendado 512 Mb) • Tarjeta de Red: 1	• Procesador: 3.00 GHZ • RAM: 1GB • Disco duro: 160 GB • UPS: 1 • Lector de CD: 1 • Tarjeta de Red: 1	• Procesador: 3.00 GHZ • RAM: 1GB • Disco duro: 160 GB • UPS: 1 • Lector de CD: 1 • Tarjeta de Red: 1

Conclusiones Parciales.

Con el estudio del Estado del Arte se pudieron identificar los aspectos a tener en cuenta en el desarrollo del Libro de Ayuda del Marco de Trabajo Sauxe, para lograr disminuir la curva de aprendizaje de los equipos de desarrollo y en el proceso de generalización. Se planteó la vía de solución teniendo en cuenta la metodología señalada. Se conformó la propuesta de solución, desarrollando el manual de instalación y configuración de las herramientas lo mismo para Windows que para Linux y se especificaron lo requisitos de software y hardware necesarios para el correcto funcionamiento de las aplicaciones que se desarrollen sobre estas tecnologías. Se valoró el impacto social, tecnológico novedoso y económico que puede aportar un Marco de Trabajo de este tipo en el desarrollo de aplicaciones web de gestión.

2 Capítulo 2: Instalación y configuración de Sauxe.

Introducción.

La Arquitectura de software es una práctica joven de apenas unos pocos años de trabajo constante, sin embargo ya se ha convertido en un factor de vital importancia para lograr que los sistemas de software tengan un alto nivel de calidad. Poseer una buena Arquitectura de software es de suma importancia, ya que ésta es el corazón de todo sistema de software y determina cuáles serán los niveles de calidad asociados al sistema.

La Subdirección Tecnológica del Centro de Soluciones de Gestión de Entidades (CESGE) de conjunto con la Unidad de Compatibilización, Integración y Desarrollo (UCID) llevaron a cabo el desarrollo del Marco de Trabajo Sauxe para desarrollar grandes aplicaciones web de gestión. Al Sauxe no tener una documentación actualizada que describa detalladamente las herramientas que se deben utilizar, así como su configuración, se hace necesario la realizar de la actualización del presente manual.

Desarrollo.

El presente documento tiene como objetivo fundamental lograr que el usuario adquiera conocimientos en cuanto a cómo instalar y configurar cada una de las herramientas requeridas para el trabajo con el Marco de Trabajo Sauxe en su versión 2.0.

Este manual es una guía de todos los pasos y procedimientos que se deben seguir para alcanzar resultados óptimos en cuanto al desarrollo de aplicaciones con el Marco de Trabajo Sauxe en su versión 2.0.

2.1 Manual de instalación de Sauxe.

Una vez que han sido instaladas y configuradas las herramientas para el Marco de Trabajo Sauxe, tal y como se explicó en el “Manual de instalación de las herramientas para Sauxe 2.0 Windows y Linux”, ya se está en condiciones de crear subsistemas, módulos, funcionalidades, otorgar permisos entre otras acciones, a través del Marco de Trabajo.

Una vez que se haya accedido al ambiente del sistema Cedrux, como se muestra en el Manual de instalación de las herramientas (figura 1), lo primero que se debe hacer es crear un nuevo sistema.

Fig. 1

En la figura 2 se muestra como a través del menú Inicio/Configurar sistemas/Sistemas, se crea el sistema que se desee.

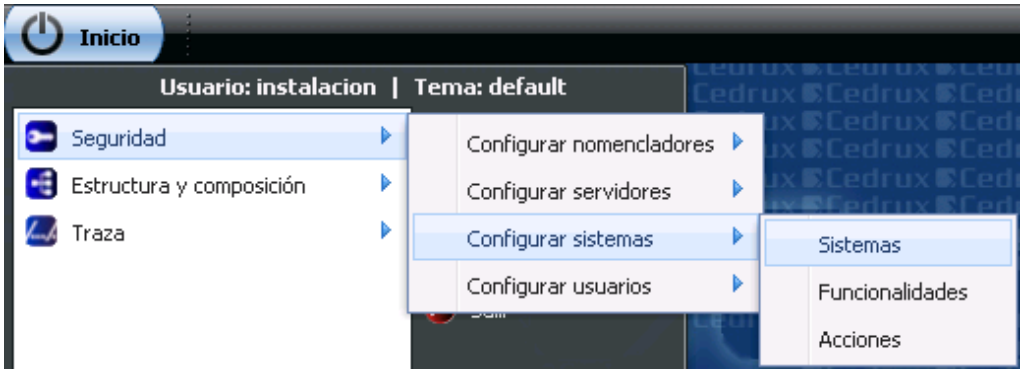
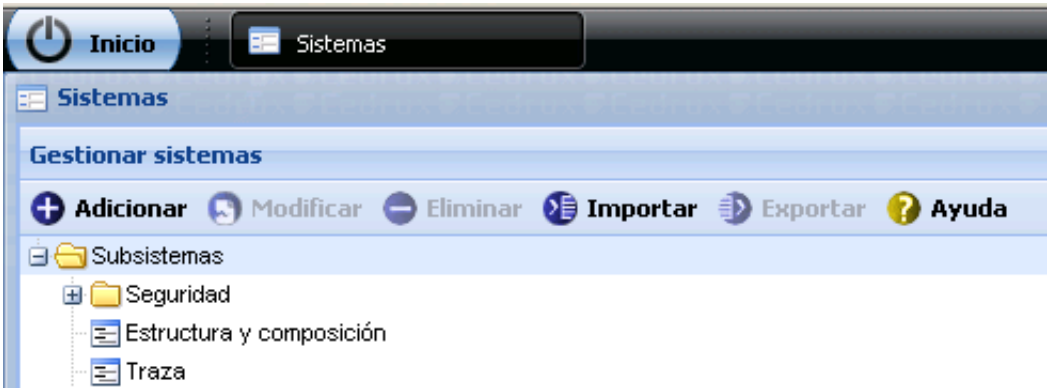


Fig. 2

Inmediatamente aparece la ventana que muestra la figura 3. Al hacer clic sobre “Subsistemas” se pueden ver todos los que hay creados hasta el momento. Haciendo clic sobre el botón “Adicionar” se crea un nuevo subsistema.



Aparece la ventana que se muestra, en el que se deben llenar los datos, para el ejemplo que se muestra la denominación del subsistema será “talleres”, luego presione el botón “Aceptar” y así queda creado.

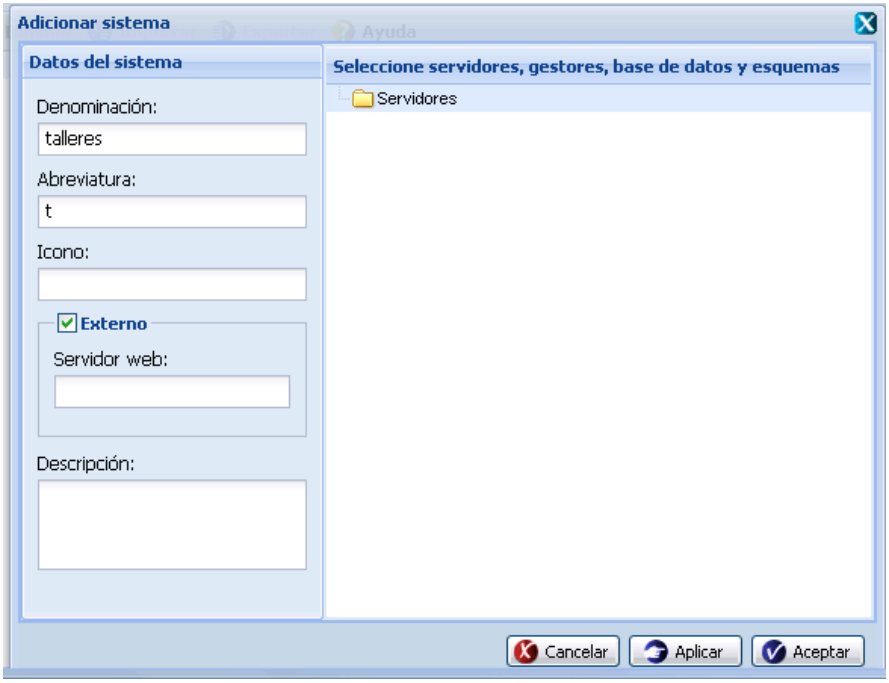


Fig. 4

La figura 5 muestra como el subsistema “talleres” fue creado al mismo nivel de los demás subsistemas que habían sido creados anteriormente.

Una vez que ha sido creado el subsistema, el siguiente paso será crear un módulo dentro de este. Esto se realiza dentro de la misma ventana de “Gestionar sistemas” que se muestra en la figura 5. Haga clic sobre el subsistema “talleres” y luego clic en el botón Adicionar, repita el mismo procedimiento que realizo para crear “talleres”. Al módulo que se crea dentro de “talleres” se le denominará “almacen”.

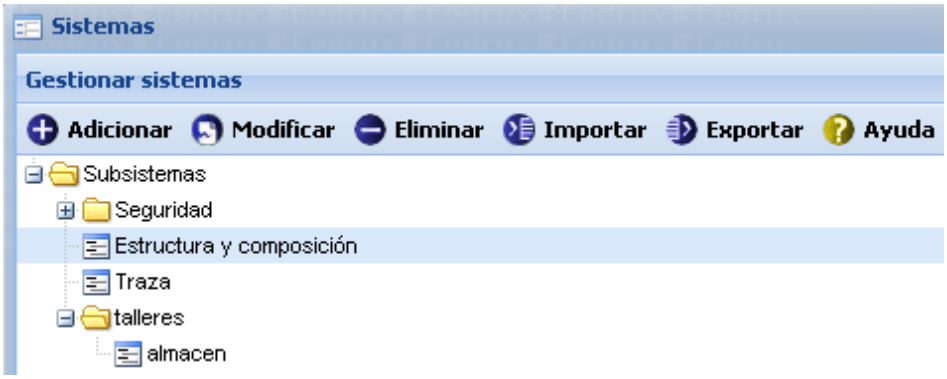


Fig. 6

La figura 5 muestra como el módulo “almacen” fue creado dentro del subsistema “talleres”.

Una vez que ha sido creado el subsistema, el siguiente paso será crear una funcionalidad, esto se hace a través del menú Inicio/Configurar sistemas/Funcionalidades, ver la figura 7.

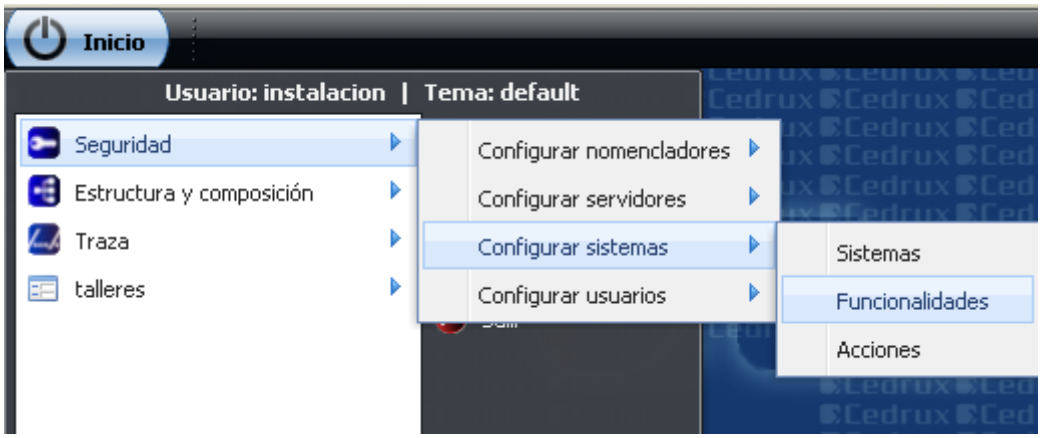


Fig. 7

Inmediatamente aparece la ventana que muestra la figura 8. Al hacer clic sobre “almacen” y luego en el botón “Adicionar” aparece la ventana que muestra la figura 9.

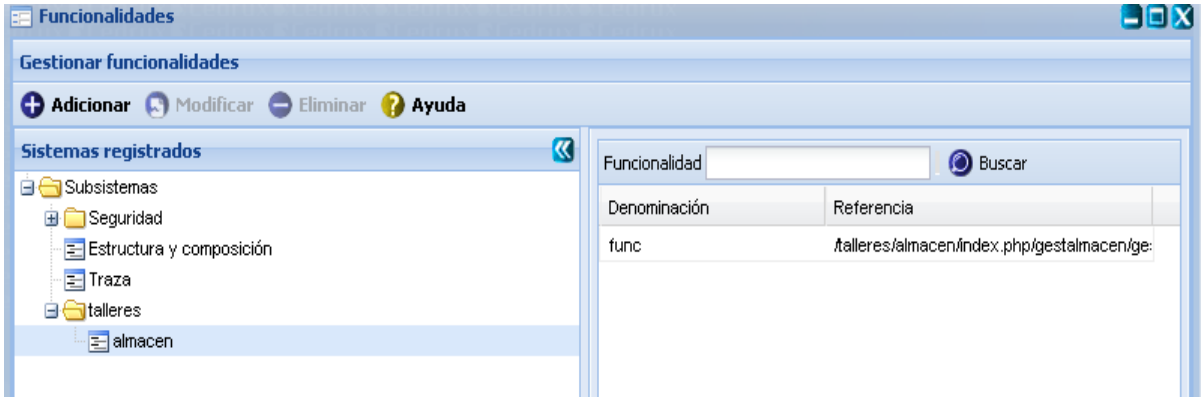


Fig. 8

Se debe insertar el nombre de la funcionalidad así como la referencia, que es la dirección en la que se encuentra programada esta funcionalidad. Para poder tener esta dirección correctamente se debe crear la estructura de carpetas y programar esta funcionalidad.

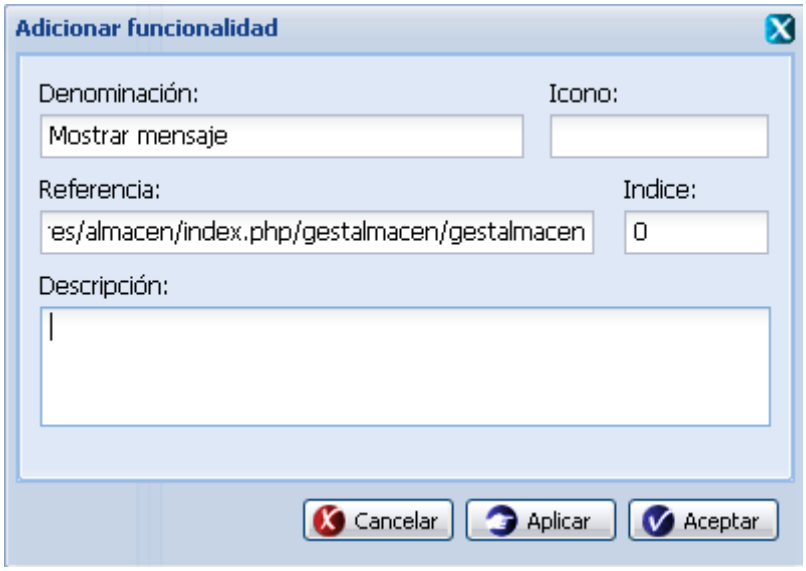


Fig. 9

Primeramente se debe ir a la carpeta donde se tiene copiado la estructura del Marco de Trabajo.

Fig. 10

Dentro de la carpeta “apps” se debe crear una carpeta con el mismo nombre del subsistema “talleres” y dentro de esta una con el mismo nombre del módulo “almacen”. Ver la figura 11.

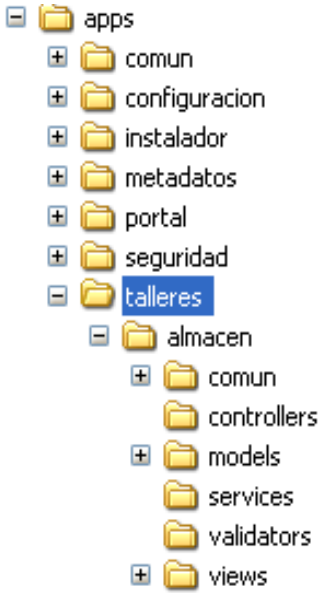


Fig. 11

La carpeta del módulo “almacen” tendrá dentro la siguiente estructura de carpetas:

comun: contiene los XML de las excepciones del módulo.

controllers: en esta carpeta se almacenan todas las clases controladoras del módulo. La estructura del nombre de la clase será: el nombre propio de la misma en mayúsculas seguido por la palabra Controller, para el caso del ejemplo que se explica un nombre sería: GestalmacenController.php. Las funciones que sean programadas dentro de esta clase tendrá al final la palabra Action, para el caso del ejemplo que se explica un nombre de función sería: gestalmacenAction.

models: contiene las carpetas “domain” y “business”. En los ficheros del “domain” se programan las consultas y en el “business” las modificaciones como son: los métodos invocados desde la funcionalidad de la clase controladora.

services: contiene los servicios que utiliza por ejemplo, servicios web, servicios de seguridad, etc.

validators: esta carpeta contiene las clases de tipo php que van a realizar acciones de validación en el componente, como por ejemplo las precondiciones que se deben cumplir antes de que un determinado método sea ejecutado.

views: en esta carpeta se recopilan los ficheros que van a gestionar la capa de presentación, estos son idioma y scripts. Dentro de la carpeta idioma existirán dos subcarpetas, una “es” donde se almacenan los archivos que gestionan una presentación en español como por ejemplo ficheros de tipo json que recopilan etiquetas para mostrar mensajes en el idioma correspondiente, y otra carpeta llamada “en” donde se gestiona la presentación en inglés. Dentro de la carpeta scripts se incluyen todas las vistas, para ello se crea una carpeta para cada clase controladora y dentro se incluye la vista o script. Estos no son más que archivos de extensión phtml donde se especifica el título de la página que se gestiona y se carga el archivo js que mostrará la presentación como tal.

La estructura de carpetas explicada se muestra en la figura 12.

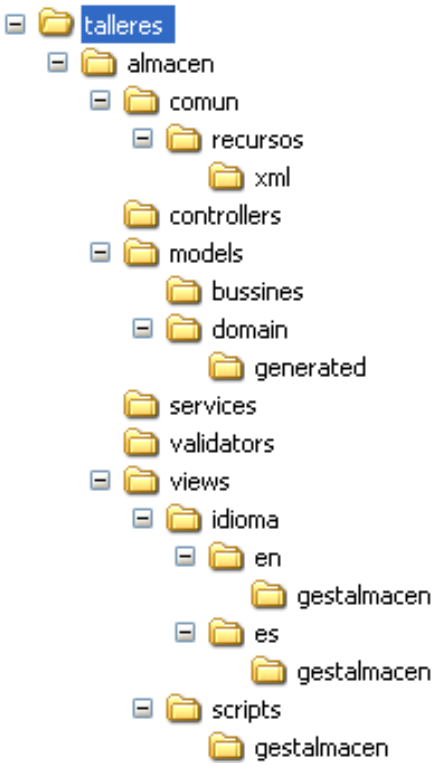


Fig. 12

Una vez creada correctamente la carpeta “talleres” con la estructura explicada dentro de la carpeta “apps”, lo próximo será crear también una carpeta “talleres” y dentro de esta una con el mismo nombre del módulo “almacen”, dentro de la carpeta “web”. Para entender con claridad a qué nivel se encuentra localizada “web”, ver la figura 10.

La carpeta del módulo “almacen” tendrá dentro la siguiente estructura de carpetas:

views: en esta carpeta se recopilan los ficheros que van a gestionar la capa de presentación, en este caso dentro de esta estarán css y js. Dentro de la carpeta css se encuentran las plantillas para el diseño del módulo, por lo general no se utiliza ninguna plantilla ya que con el Marco de Trabajo ExtJS se está resolviendo todo el tema de la presentación. Un fichero js no es más que un documento con la extensión .js donde se escribirá el código correspondiente a la capa de presentación, aquí es donde se carga el diseño de la aplicación, para cada clase controladora se crea una carpeta que tendrá incluido el fichero js correspondiente a dicha clase control.

webservices: contiene los servicios que utiliza por ejemplo, servicios web, servicios de seguridad, etc.

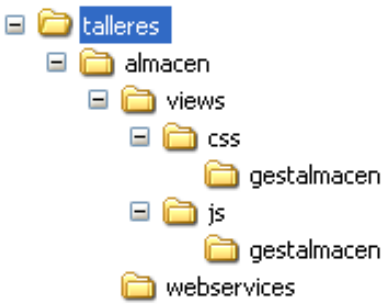


Fig. 13

Una vez creada correctamente la estructura de carpetas lo siguiente será programar una funcionalidad para el módulo almacen. Para el caso del ejemplo la funcionalidad será mostrar un mensaje.

¿Cómo se hace?

En la clase controladora que se encuentra en: \apps\talleres\almacen\controllers se escribe el siguiente código:

```
function gestalmacenAction()
{
      $this->render();
}
```

Luego en el fichero js que se encuentra en: web\talleres\almacen\views\js\gestalmacen se escribe el siguiente código:

```
function cargarInterfaz()
{
      alert ("Hola mundo");
}
cargarInterfaz();
```

Una vez programada la funcionalidad se regresa a la figura 9, interfaz "Adicionar funcionalidad". Como ya se explicaba, se debe insertar el nombre que desee a la funcionalidad así como la referencia, esta ya se tiene, y sería: \talleres\almacen\index.php\nombre de la clase\nombre de la acción. Por tanto, para el caso del ejemplo la dirección quedaría: /talleres/almacen/index.php/gestalmacen/gestalmacen.

Es importante tener en cuenta que se deben cambiar los slash.

Una vez que la funcionalidad haya sido creada aparecerá el mensaje:

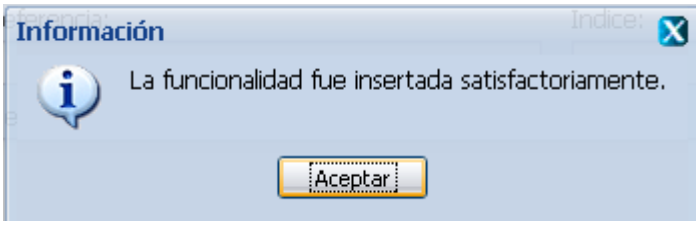


Fig. 14

Al ir al menú inicio, ya aparece la funcionalidad. Y al hacer clic sobre ella aparece el mensaje. Ver figura 16.

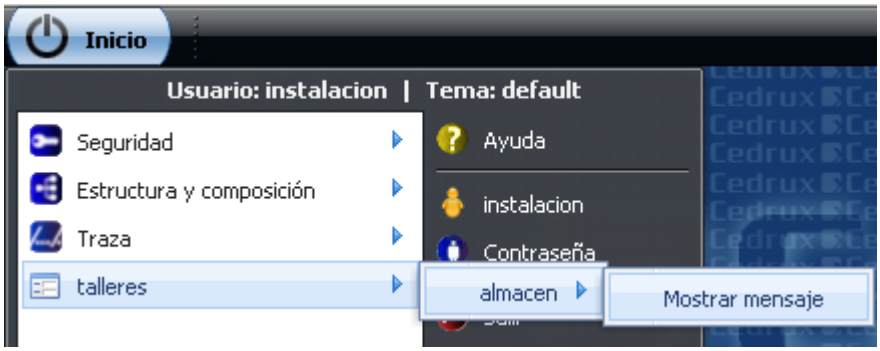


Fig. 15

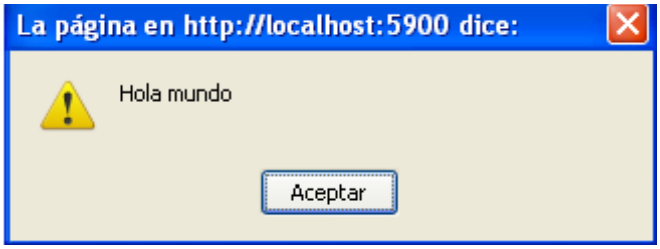


Fig. 16

Conclusiones Parciales.

En este capítulo se realizó la descripción del proceso de instalación y configuración de Sauxe. En la misma se especifican las vistas estructurales de la arquitectura, la estructura que debe cumplir un subsistema y los pasos a seguir para lograr la integración y visualización de cada una de las funcionalidades. Se abordó a nivel macro las principales funcionalidades del Sistema de Gestión Integral de Seguridad Acaxia para lograr la seguridad desde el mismo inicio del desarrollo. Además se mostraron las pautas a seguir para lograr la coherencia, comunicación e integración entre los componentes de cada una de las capas.

3 Capítulo 3. Estilos y taxonomías.

Introducción.

Los estilos de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código.

Las taxonomías, en su sentido más general, es la ciencia de la clasificación, la ciencia de ordenar.
La taxonomía estructural va a regir la organización de los elementos arquitectónicos así como los conectores de estilo presentes en la arquitectura.

Desarrollo.

3.1 Normas y estilos de codificación.

Los estándares de codificación en el Marco de Trabajo del ERP van a permitir una mejor integración entre las líneas de producción y se establecerán las pautas que conlleven a lograr un código más legible y reutilizable, de tal forma que se pueda aumenta su mantenibilidad a lo largo del tiempo.

Este epígrafe constituye una guía para el desarrollo en las líneas de producción, desde el punto de vista arquitectónico, con el propósito de lograr una estandarización del código.

3.1.1 Políticas de trabajo.

3.1.1.1 Revise estas políticas.

Las mismas se irán actualizando para acomodarlas a los problemas o necesidades que vayan surgiendo en el proyecto.

3.1.1.2 Utilizar normas y estándares de codificación definidos para el desarrollo en el proyecto.

Se deben tener siempre presente las reglas definidas en el documento: “Normas y estándares de codificación” para el desarrollo en el proyecto.
Ver documento en el repositorio.
http://10.12.179.3:3389/svn/erp/ERP/Ingenieria/Arquitectura/Desarrollo/Documentos%20y%20Presnetaciones/Documentos/Revisiones%20tecnicas%20al%20desarrollo/Listas%20de%20chequeo/

3.1.2 Estándares de Nomenclatura.

3.1.2.1 Nomenclatura de las clases.

Los nombres de las clases comienzan con la primera letra en mayúscula y el resto en minúscula, en caso de que sea un nombre compuesto se empleará notación *PascalCasing**. Con sólo leerlo se reconoce el propósito de la misma.

Ejemplo: GestionarUsuario

3.1.2.2 Nomenclatura según el tipo de clases.

➢ **Clases controladoras**

Las clases controladoras después del nombre llevan la palabra:"Controller".

Ejemplo: GestionarUsuarioController

➢ **Clases de los modelos**

- Business (Negocio)

Las clases que se encuentran dentro de Business después del nombre llevan la palabra: "Model".
Ejemplo: MonedaContModel

- Domain (Dominio)

Las clases que se encuentran dentro de Domain el nombre que reciben es el de la tabla en la Base de Datos.
Ejemplo: User

- Generated (Dominio bases)

Las clases que se encuentran dentro de Generated el nombre comienza con la palabra: "Base" y seguido el nombre de la tabla en la Base de Datos.

Ejemplo: BaseUser

➢ **Clases del framework**

Como parte del marco de desarrollo de Zend existe el Zend_Loader (Cargador) que, además del cumplimiento de ciertas normas para la nomenclatura de las clases, garantiza que a partir de una ruta de inclusión, este sea el responsable de la inclusión de los recursos requeridos en el proceso.
Ejemplo: Los nombres de las clases contienen la dirección donde se encuentran, de la siguiente forma, si tenemos una clase llamada Condado en la siguiente estructura Cuba/VC/SantaClara/Condado.php entonces un identificador de la misma debe ser Cuba_VC_SantaClara_Condado lo que garantiza que a través de una casuística particular el cargador localice estos recursos.

3.1.2.3 Nomenclatura de las funciones.

El nombre a emplear para las funciones se escribe con la primera palabra en minúscula, en caso de que sea un nombre compuesto se empleará notación *CamelCasing**, y con sólo leerlo se reconoce el propósito de la misma.

Ejemplo: insertarMoneda

En caso de ser una acción de la clase controladora se le pone el nombre y seguida la palabra:"Action"

Ejemplo: insertarMonedaAction

3.1.2.4 Nomenclatura de las variables

El nombre a emplear para las variables se escribe con la primera palabra en minúscula, en caso de que sea un nombre compuesto se empleará notación *CamelCasing*,* y comenzando con un prefijo según el tipo de datos.

Ejemplo: arrMoneda

3.1.2.5 Prefijos para los tipos de datos.

Los prefijos a utilizar en la creación de variables serán los siguientes:

Tipos de Datos	Prefijos
Arreglos	arr
Objetos	obj
Enteros	int
Cadena	str
float	flt
Boolean	Boo

3.1.2.6 Nomenclatura de las constantes.

El nombre a emplear para las constantes se escribe con todas las letras en mayúscula.

Ejemplo: MONEDA.

3.1.2.7 Nomenclatura de los atributos.

El nombre a emplear para los atributos se escribe con la primera palabra en minúscula, en caso de que sea un nombre compuesto se empleará notación *CamelCasing*.* Además en caso de ser un objeto se comienza con:"_" y después se escribe el nombre.

Ejemplo: intMoneda=dinero

objMoneda=_dinero

3.1.3 Normas de comentariado.

Es una necesidad comentar todo lo que se haga dentro del desarrollo, es decir, establecer las pautas que conlleven a lograr un código más legible y reutilizable, de manera que se pueda aumentar su mantenibilidad a lo largo del tiempo.

3.1.3.1Nomenclatura de los comentarios.

Los comentarios deben ser lo bastante claro y preciso de forma tal que se entienda el propósito de lo que se está desarrollando.

- **En las clases**

Antes de la declaración de una clase se escribe una breve descripción donde se explique el propósito de la misma. Que se escribe de la siguiente forma:

/**
* Nombre de la clase *
* Descripcion *
* @author *
* @package *(módulo)
* @subpackage *(sub módulo)
* @copyright *
* @version (versión - parche) */

Ejemplo:

```
/**
 * UsuarioModel
 * Modelo de Negocio para gestionar usuarios
 *
 * @author Yoandry Morejon Borbon
 * @package ERP
 * @subpackage Usuario
 * @copyright UCID-ERP Cuba
 * @version 1.0-0
 */
```

➢ **En las funciones**

Antes de la declaración de la función se escribe una breve descripción donde se explique el propósito de la misma. Que se escribe de la siguiente forma:

/**
* Nombre de la función *
* Descripcion *
* @author * (en caso de que no sea el autor de la clase)
* @param *(los parámetros que se le pasan a la función con su descripción)
* @throws *(en caso de que dispare una excepción)
* @return *(se pone lo que devuelve la función y un comentario)
*/

Ejemplo:

```
/**
 * insertarUsuario
 * Insertar un usuario
 *
 * @param DatPersona $persona - Entidad de dominio de persona
 * @param DatUsuario $usuario - Entidad de dominio de usuario
 * @param DatClavesacceso $clave, Entidad de dominio de claveacceso
 * @throws Doctrine_Exception - excepcion disparada por doctrine
 * @return boolean, si se inserto correctamente retorna true sino retorna false
 */
```

➢ **Función complicada**

En caso de ser una función complicada se comentaría dentro de la misma para lograr una mejor comprensión del código.

Ejemplo:

```
public function insertarUsuario(DatPersona $persona, DatUsuario $usuario, DatClavesacceso $clave)
{
    //Obteniendo el nuevo identificador de persona
    $idpersona = $persona->genIdPersona();

    //Insertando la persona
    $persona->idpersona = $idpersona;
    $persona->save();

    //Insertando el usuario
    $usuario->idpersona = $idpersona;
    $usuario->save();

    //Generando el nuevo identificador de clave de acceso
    $idclaveacceso = $clave->genIdClaveAcceso();

    //Registrando la clave de acceso
    $clave->idclaveacceso = $idclaveacceso;
    $clave->idpersona = $idpersona;
    $clave->claveacceso = md5($clave->claveacceso);
    $clave->fechainicio = date('d/m/Y');
    $clave->fechafin = '01/01/0001';
    $clave->save();
}
```

➢ **En las llaves**

Cada vez que se vaya a cerrar una llave se escribe en forma de comentario lo siguiente: Fin y el nombre de lo que se esté cerrando.

Ejemplo:

```
class Example {
    var $theInt = 1;
    function foo($a, $b) {
        switch ( $a) {
            case 0 :
                $Other->doFoo ();
            break;
            default :
                $Other->doBaz ();
        }// fin del switch
    } //fin de la función foo
    function bar($v) {
        for($i = 0; $i < 10; $i ++) {
            $v->add ( $i );
        }// fin del for
    }// fin de la función bar
}// fin de la clase example
?>
```

3.1.4 Estilo del Código.

En la implementación cuando se vaya a escribir una sentencia en php la forma de utilizar los tab del mismo es la siguiente:

```php
<?php
//código
?>
```

3.1.4.1Sangría o Indexado.

La política de sangría a utilizar en la implementación es por **tab**

- **Declaraciones dentro del cuerpo de la clase**

Las clases se comienzan a declarar pegado al margen izquierdo, después de poner el nombre de la clase se pone un espacio y se abre llave en la misma línea.

Ejemplo:

```php
<?php
/**
 * Indentation
 */
class Example {
    var $theInt = 1;
    function foo($a, $b) {
    switch ( $a) {
    case 0 :
    $Other->doFoo ();
    break;
    default :
    $Other->doBaz ();
    }
    }
    function bar($v) {
    for($i = 0; $i < 10; $i ++) {
    $v->add ( $i );
    }
    }
}
?>
```

- **Declaraciones dentro del método / el cuerpo de la función**

Las declaraciones dentro del método / el cuerpo de la función se realizan como se muestra en el ejemplo.

Ejemplo:

```php
<?php
/**
 * Indentation
 */
class Example {
var $theInt = 1;
function foo($a, $b) {
    switch ( $a) {
    case 0 :
    $Other->doFoo ();
    break;
    default :
    $Other->doBaz ();
    }
}
function bar($v) {
    for($i = 0; $i < 10; $i ++) {
    $v->add ( $i );
    }
}
}
?>
```

- **Declaraciones dentro de los bloques**

Las declaraciones dentro de los bloques se realizan como se muestra en el ejemplo.

Ejemplo:

```php
<?php
/**
 * Indentation
 */
class Example {
var $theInt = 1;
function foo($a, $b) {
switch ( $a) {
case 0 :
$Other->doFoo ();
break;
default :
$Other->doBaz ();
}
}
function bar($v) {
for($i = 0; $i < 10; $i ++) {
    $v->add ( $i );
}
}
}
?>
```

- **Declaraciones dentro del cuerpo del interruptor o switch**

Las declaraciones dentro del cuerpo del switch se realizan como se muestra en el ejemplo.

Ejemplo:

```php
<?php
/**
 * Indentation
 */
class Example {
var $theInt = 1;
function foo($a, $b) {
switch ( $a) {
    case 0 :
    $Other->doFoo ();
    break;
    default :
    $Other->doBaz ();
}
}
function bar($v) {
for($i = 0; $i < 10; $i ++) {
$v->add ( $i );
}
}
}
?>
```

- **Declaraciones dentro del cuerpo del caso o case**

Las declaraciones dentro del cuerpo del case se realizan como se muestra en el ejemplo.

Ejemplo:

```php
<?php
/**
 * Indentation
 */
class Example {
var $theInt = 1;
function foo($a, $b) {
switch ( $a) {
case 0 :
    $Other->doFoo ();
break;
default :
    $Other->doBaz ();
}
}
function bar($v) {
for($i = 0; $i < 10; $i ++) {
$v->add ( $i );
}
}
}
?>
```

Ejemplo General: En este ejemplo se muestra como debe quedar el código después de aplicarle las declaraciones de sangría anteriores.

```php
<?php
/**
 * Indentation
 */
class Example {
    var $theInt = 1;
    function foo($a, $b) {
        switch ( $a) {
            case 0 :
                $Other->doFoo ();
            break;
            default :
                $Other->doBaz ();
        }
    }
    function bar($v) {
        for($i = 0; $i < 10; $i ++) {
            $v->add ( $i );
        }
    }
}
?>
```

3.1.4.2 Brazas o Llaves.

En la declaración de clases o interfaces, métodos, bloques y switch, la apertura de llaves se hace en la misma línea.

Ejemplo:

```php
<?php
/**
 * Braces
 */
interface EmptyInterface {
}

class Example {
    function bar($p) {
        for($i = 0; $i < 10; $i ++) {
        }
        switch ( $p) {
            case 0 :
                $fField->set ( 0 );
            break;
            case 1 :
                {
                    break;
                }
            default :
                $fField->reset ();
        }
    }
}
?>
```

3.1.4.3 Espacio en blanco.

➢ **Declaraciones**

La declaración de los espacios en blanco en las clases, campos y métodos es como se muestra en el ejemplo.

Ejemplo:

```
<?php
class MyClass implements IO, I1, I2 {
}
class MyClass {
    public $a = 0, $b = 1, $c = 2, $d = 3;
    const MY_TRUE = 1, MY_FALSE = 2;
}
function foo() {
}
function bar(int $x, $y, $z = 1) {
}
?>
```

➢ **Controles**

La declaración de los espacios en blanco en los controles es de la siguiente forma:

- **Blocks**

Ejemplo:

```
<?php
if (true) {
    return 1;
} else if (true) {
    return 3;
} else {
    return 2;
}
try {
    echo $b;
} catch ( Exception $e ) {
}
?>
```

- ‘**if’ ‘else’**

Ejemplo:

```
<?php
if ($condition) {
    return $foo;
} else {
    return $bar;
}
?>
```

- **‘for’**

Ejemplo:

```php
<?php
for($i = 0, $j = 0; $i < 8; $i ++, $j --) {
}
?>
```

- **'foreach'**

Ejemplo:

```php
<?php
foreach ( $s as $key => $value ) {
}
?>
```

- **'switch'**

Ejemplo:

```php
<?php
switch ( $number) {
    case RED :
        return GREEN;
    case GREEN :
        return BLUE;
    case BLUE :
        return RED;
    default :
        return BLACK;
}
?>
```

- **'while' & 'do while'**

Ejemplo:

```php
<?php
while ( $condition ) {
}
do {
} while ( $condition );
?>
```

- **'catch'**

Ejemplo:

```php
<?php
try {
    echo $b;
} catch ( Exception $e ) {
}
?>
```

- **'static'**

Ejemplo:

```php
<?php
static $a, $b;
?>
```

- **'global'**

Ejemplo:

```php
<?php
global $a, $b;
?>
```

- **'echo'**

Ejemplo:

```php
<?php
echo 'hello ', $a;
?>
```

➢ **Expresiones**

La declaración de los espacios en blanco en las expresiones es como se muestra en el ejemplo.

Ejemplo:

```php
<?php
$myClass->$attr;
MyClass::$attr;
MyClass::MY_CONST;
foo ();
oar ( $x, $y );
MyClass::foo ();
$myClass->foo ();
$a = - 4 + - 9;
$b = $a ++ / -- $number;
$c += 4;
$value = true && false;
$s = ( string ) $object;
$a = $condition ? TRUE : FALSE;
(($a));
?>
```

➢ **Arreglos**

La declaración de los espacios en blanco en los arreglos es como se muestra en el ejemplo.

Ejemplo:

```php
<?php
list ( $a, $b ) = array (1, 2, 3 );
$array = array (1 => 2, 2 => 3 );
$array [$i]->foo ();
$array [] = 'first cell';
?>
```

3.1.4.4 Líneas en blanco.

Las líneas en blanco antes de la declaración de las clases, constantes, campos, funciones, métodos, al empezar una función o cuerpo de un método son nulas.

Ejemplo:

```php
<?php
/**
 * Blank Lines
 */
class Example {
    const CONST2 = 3;
    var $theInt = 1;
    public function foo($a, $b) {
        switch ( $a) {
            case 0 :
                $Other->doFoo ();
            break;
            default :
                $Other->doBaz ();
        }
    }
    function bar($v) {
        for($i = 0; $i < 10; $i ++) {
            $v->add ( $i );
        }
    }
}
interface MyInterface {
}
?>
```

3.1.4.5 Nuevas líneas.

Las nuevas líneas se utilizan en el cuerpo de la clase, en el cuerpo del método y en el bloque cuando son vacíos.

Ejemplo:

```php
<?php
/**
 * New Lines
 */
class EmptyBody {
}
class Example {
    function emptyFoo() {
    }
    function foo() {
        do {
        } while ( false );
        for(; ; ) {
        }
    }
}
$array = array (1, 2, 3, 4, 5 );
while ( $a )
    ;
?>
```

3.2 Taxonomía estructural.

La arquitectura es basada en capas. Está compuesto básicamente por cinco niveles o capas:

1. Capa de Presentación: En esta capa se emplea las facilidades que brinda el Marco de Trabajo ExtJS para la construcción de interfaces amigables a la vista de los usuarios. Ext centra su desarrollo en tres componentes fundamentales JS-File, CSS-File y Client-page y Server-Page.

2. Capa de Control o Negocio: En esta capa se emplea el patrón de arquitectura Modelo Vista Controlador (MVC). De forma vertical al modelo descrito hasta este momento, estarán los aspectos fundamentales del sistema así como el encargado del tratamiento de la seguridad a nivel de aplicación Web.

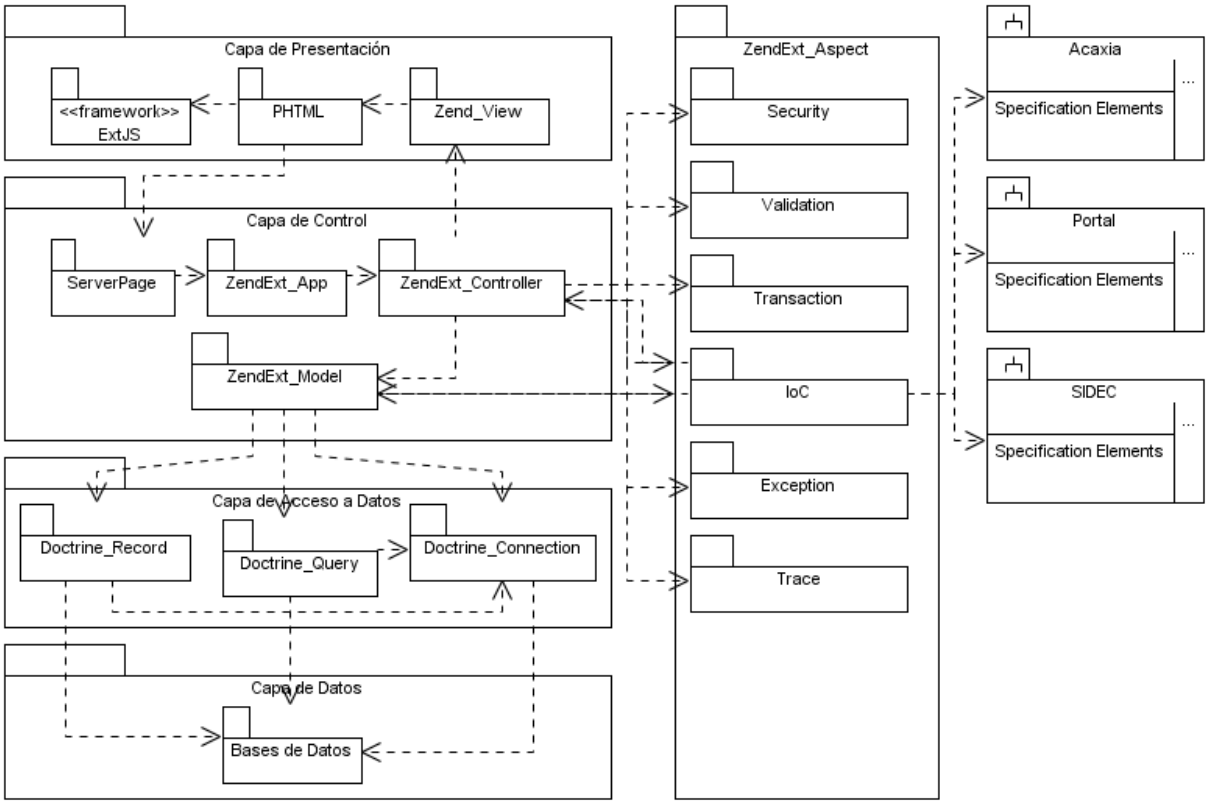
3. Capa de Acceso a Dato: En esta capa estará presente el ORM (Object Relational Mapping) Doctrine, como Marco de Trabajo de persistencia para la comunicación con el servidor de datos mediante el protocolo PDO, también estará un Persistidor de Configuración que es el encargado de comunicarse vía XML con los Ficheros de Configuración del sistema denominado FastResponse.

4. Capa de Dato: En esta capa estará ubicado como servidor de base de datos PostgreSQL y un conjunto de Ficheros de Configuración de la arquitectura tecnológica.

5. Capa de Servicio: En esta última capa se encuentran todos los subsistemas que prestan y consumen servicios entre sí.

Esta gran estructura descrita, se comunica con unas series de servicios Web mediante protocolos SOAP, e IoC, que interactúan con el sistema proporcionándole un conjunto de funcionalidades tanto de seguridad como de negocio.

3.2.1 Gráfico estructural



3.2.2 Especificación estructural.

Elementos de la estructura	
Estructura 1 – Capa de Presentación Es la capa más cercana al usuario, le comunica y captura la información del usuario en un mínimo de proceso, también es conocida como interfaz gráfica y debe tener la característica de ser amigable, es decir entendible y fácil de usar.	
Elemento 1: Framework ExtJS Es una librería construida con JavaScript que proporciona una interfaz muy potente. Su potencia radica en la rica colección de componentes para el diseño de interfaces de usuarios del lado del cliente haciendo uso extensivo de Ajax. Entre los componentes que esta librería ofrece encontramos cuadros de diálogo, menús, tablas editables, layouts, paneles, pestañas y todo lo necesario para construir atractivos desarrollos al estilo de Web 2.0.	
Responsabilidad Arquitectónica	Desarrollar las vistas para mostrar al usuario una interfaz más amigable y funcional se utilizará en las vistas.

	Elemento 2: PHTML Es una extensión para un tipo de páginas web que llevan código PHP para ser generadas. Cuando una página está escrita en PHP podemos encontrarla con varios tipos de extensiones, como por ejemplo: .php, .php4, .php3, o .phtml.	
	Responsabilidad Arquitectónica	Incluir los ficheros JavaScrip y CSS del Marco de Trabajo Zend_Ext para construir la vista.
	Elemento 3: Zend_View Es la clase que permite trabajar con la vista en el patrón Modelo-Vista-Controlador. El uso de esta clase Zend_View, ocurre en dos grandes pasos: Su controlador de secuencia de comandos crea una instancia de Zend_View y asigna variables a esa instancia. El controlador le dice al Zend_View que emita una vista, y devuelve el control al script de la vista, lo que genera la vista saliente.	
	Responsabilidad Arquitectónica	Existe para ayudar a mantener el script de la vista separado de los scripts del modelo y de los controladores.
Herramienta de la Estructura		
	Mozilla Firefox Es un navegador multiplataforma(es un término usado para referirse a los programas, sistemas operativos, lenguajes de programación, u otra clase de software, que puedan funcionar en diversas plataformas) y está disponible en varias versiones de Microsoft Windows, Mac Os X, GNU/Linux y algunos sistemas basados en Unix que es un sistema operativo portable, multitarea y multiuso.	
	Versión	2.2 o superior
	Fabricante	Fundación Mozilla y voluntarios externos.
	Responsabilidad Arquitectónica	Proporcionar la mejor navegación en cada paso que dé el usuario.
	Licencia	GPL/LGPL/MPL
Elementos de la estructura		
Estructura 2 – Capa de Control o Negocio En esta capa se emplea el patrón de arquitectura Modelo Vista Controlador (MVC) que separa los datos de la aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos, también reúne todos los aspectos del software que automatizan o apoyan los procesos de negocio que llevan a cabo los usuarios.		
	Elemento 1: ServerPage	

	Representa una página Web que tiene scripts ejecutados por el servidor. Estos scripts interactúan con los recursos que se encuentran al alcance del servidor. Sólo puede mantener relaciones con objetos que se encuentren en el servidor.	
	Responsabilidad Arquitectónica	Gestionar las entradas del usuario y es el responsable de controlar todos los eventos generados en la vista y hace función de intermediar entre la vista y el modelo ante las peticiones generadas por el usuario en la interacción con la vista.
	Elemento 2: ZendExt_App Este componente se encarga de realizar las configuraciones iniciales, verificaciones y validaciones. Es el punto de inicio a partir del cual se desencadenan un conjunto de acciones necesarias antes de realizar cualquier operación.	
	Responsabilidad Arquitectónica	Realizar un mecanismo que permita inicializar los elementos arquitectónicamente significativos de la aplicación.
	Elemento 3: ZendExt_Controller Se diseñó para ofrecer sitios Web con direcciones url claras. Para lograr esto, todas las peticiones deberán ser procesadas únicamente por un archivo index.php. Así se ofrece un punto central para todas las páginas de la aplicación y asegura la instalación de un ambiente correcto para ejecutar la aplicación.	
	Responsabilidad Arquitectónica	Extender del Framework Zend para implementar un grupo de funcionalidades propias de ZendExt en la capa de negocio.
	Elemento 4: ZendExt_Model	
	Responsabilidad Arquitectónica	Extiende el modelo, del patrón MVC.
	Elemento 5: Security Consiste en asegurar que los recursos del sistema sean utilizados de la manera que se decidió y que el acceso a la información allí contenida así como su modificación sólo sea posible a las personas que se encuentren acreditadas y dentro de los límites de su autorización.	
	Responsabilidad Arquitectónica	Garantizar la seguridad del sistema.
	Elemento 6: Validation Se diseñó para validar en el servidor todas las acciones que requieran de ellas.	

	Este componente debe validar todos los datos y acciones que se activan ya sean por un usuario o por un sistema externo, que serán ejecutadas en un controlador de una aplicación en específico. La acción será validada antes de llegar al controlador para no hacer peticiones innecesarias a este.	
	Responsabilidad Arquitectónica	Permitir a las aplicaciones que se están desarrollando la validación de todas aquellas acciones y datos que lo requieran, antes de ser procesados por el controlador.
	Elemento 7: Transaction Este componente se diseñó con el propósito de proveer un mecanismo de gestión centralizada de las transacciones de todos los sistemas, en el cual los desarrolladores solo tengan que crear los objetivos y lo deje en manos del Marco de Trabajo para que ejecute las operaciones con la base de datos.	
	Responsabilidad Arquitectónica	Gestionar todas las tracciones que se ejecuten en los sistemas a nivel central.
	Elemento 8: IoC Es un concepto junto a unas técnicas de programación en las que el flujo de ejecución de un programa se invierte respecto a los métodos de programación tradicionales, en los que la interacción se expresa de forma imperativa haciendo llamadas a procedimientos (procedure calls) o funciones.	
	Responsabilidad Arquitectónica	Integrar módulos o subsistemas permitiendo la ejecución de funciones de unos a otros.
	Elemento 9: Exception Las excepciones están destinadas para la detección y corrección de errores. Todas las excepciones lanzadas por las clases de ZendExt deben lanzar una excepción que se deriva de la clase ZendException.	
	Responsabilidad Arquitectónica	Permitirá manejar errores como excepciones.
	Elemento 9: Trace Las trazas son utilizadas, entre otras cosas, para comprobar que se realizan exactamente las funciones esperadas, y no otras, también son usadas con el fin de monitorear las acciones que se realicen y para detectar indicios de hechos relevantes a los efectos de la seguridad que puedan afectar la estabilidad o el funcionamiento del sistema informático.	
	Responsabilidad Arquitectónica	Detectar y registrar los eventos del proceso de negocio del sistema en todos sus

		subsistemas y que visualice este proceso.
Herramienta de la Estructura		
	Apache Es un servidor del protocolo http, comúnmente llamado servidor web de código libre robusto, cuya implementación se realiza de forma colaborativa, con prestaciones y funcionalidades equivalentes a las de los servidores comerciales. El proyecto está dirigido y controlado por un grupo de voluntarios de todo el mundo.	
	Versión	2.0
	Fabricante	Apache Group
	Responsabilidad Arquitectónica	Publicar las aplicaciones para que los usuarios puedan acceder a ellas y realizar configuraciones seguras.
	Licencia	BSD/GNU
Elementos de la estructura		
Estructura 3 – Capa de Acceso a Dato Este nivel lógico es el encargado del acceso a los datos de la aplicación y la persistencia y carga de los mismos. Esta capa aísla definitivamente la capa de funcionamiento de todo lo que tenga que ver con el origen de datos, ya que desde el manejo de la conexión, hasta la ejecución de una consulta, la manejará la capa de Acceso a Datos.		
	Elemento 1: Doctrine_Record Es, conjuntamente con Doctrine_Table, una de las clases fundamentales de Doctrine, se encarga del manejo de los datos de una tabla por cada fila o record. Este componente se utiliza para generar código, tanto de código PHP a base de dato como de base de dato a código PHP. Soporta todos los gestores de base de datos.	
	Responsabilidad Arquitectónica	Servir de puente de comunicación entre la Capa de Funcionamiento y la Capa de Acceso a Datos.
	Elemento 2: Doctrine_Query Es una de las características más fuertes de Doctrine	
	Responsabilidad Arquitectónica	Convertir las consultas del lenguaje de Doctrine QL a lenguaje de gestor SQL
	Elemento 3: Doctrine_Conection Es una clase muy importante de doctrine.	
	Responsabilidad Arquitectónica	Administrar las conexiones de la base de dato.

Herramienta de la Estructura		
	Apache Es un servidor del protocolo http, comúnmente llamado servidor web de código libre robusto, cuya implementación se realiza de forma colaborativa, con prestaciones y funcionalidades equivalentes a las de los servidores comerciales. El proyecto está dirigido y controlado por un grupo de voluntarios de todo el mundo.	
	Versión	2.0
	Fabricante	Apache Group
	Responsabilidad Arquitectónica	Publicar las aplicaciones para que los usuarios puedan acceder a ellas y realizar configuraciones seguras.
	Licencia	BSD/GNU
Elementos de la estructura		
Estructura 4 – Capa de Dato Es donde se almacena de forma persistente toda la información del sistema haciendo uso de servicios de base de datos y los ficheros de configuración. En esta capa no se definió ningún elemento, solo estará la herramienta PostgreSQL		
Herramienta de la Estructura		
	Bases de Datos Son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta. Objetivos que deben cumplir: abstracción de la información, independencia, redundancia mínima, consistencia, seguridad, integridad, respaldo y recuperación, control de la concurrencia, tiempo de respuesta. El Sistema Gestor de Base de Datos seleccionado es PostgreSQL por su seguridad, integridad, potencialidad y libertad de uso. PostgreSQL está considerado como la base de datos de código abierto más avanzada del mundo.	
	Versión	8.3
	Fabricante	PostgreSQL Global Development Group (Grupo de Desarrollo Global de PostgreSQL)
	Responsabilidad Arquitectónica	Soporta una gran parte del SQL estándar y ofrece muchas características modernas: ➢ Consultas complejas ➢ Claves foráneas

		➢ Disparadores ➢ Vistas ➢ Integridad transaccional ➢ Control de Concurrencia (multiversión)
	Licencia	BSD
Elementos de la estructura		
Estructura 5 – Capa de Servicios Es una capa de abstracción que se crea entre la capa de presentación y la capa de negocio, y que agrupa funcionalidad de la capa de negocio para ser expuesta a la capa de presentación.		
	Elemento 1: Acaxia Este componente brinda sus servicios a todos los sistemas que se suscriban a él. Para ello se gestionarán las conexiones a la base de datos, las funcionalidades asociadas y las acciones que realizan las mismas.	
	Responsabilidad Arquitectónica	Garantizar la seguridad de forma centralizada en un entorno de varias aplicaciones.
	Elemento 1: Portal Este componente fue creado con el objetivo de ser capaz de adaptarse a las particularidades de los usuarios con un alto nivel de reutilización cumpliendo con estándares para el diseño de interfaces de usuarios.	
	Responsabilidad Arquitectónica	Gestionar de forma centralizada el menú de los sistemas al cual tiene permiso el usuario, y la visualización de las interfaces de los usuarios.
	Elemento 1:SIDEC Es un componente del framework ZendExt que se desarrolló para gestionar la información.	
	Responsabilidad Arquitectónica	Gestionar la estructura de forma dinámica para lograr cumplir con la multi-entidad y la compartimentación de la información.

3.2.3 Especificación estructural de los conectores del estilo.

Conector 1		
	Nombre	Composición

	Descripción	Es un tipo de relación estática, donde el tiempo de vida del objeto incluido está condicionado por el tiempo de vida del que lo incluye. (Bosch, 2009)	
		Usos	
		Conexion1	PHTML– Framework ExtJS
		Descripción	PHTML muestra todos los objetos creados en el JS correspondiente a esta vista. Para lograr este propósito es necesario que la misma incluya los ficheros del Framework ExtJS.
		Licencia	-
Conector 2			
	Nombre	Include	
	Descripción	Es una simple relación de inclusión, es decir, los escenarios o situaciones posibles detalladas en un caso de uso están incluidas en otro caso de uso (aquel del que, gráficamente, parte la flecha).	
		Usos	
		Conexion1	Zend_View – PHTML.
		Descripción	Zend_View tiene la responsabilidad de identificar cual es la vista o PHTML que se está instanciando, una vez identificada procede a su ejecución.
		Licencia	-
Conector 3			
	Nombre	Show	
	Descripción	Es el conector que permite que se muestren determinada información.	
		Usos	
		Conexion1	PHTM L– ServerPage
		Descripción	Todas las acciones que desencadena un usuario transitan por el controlador frontal del módulo o subsistema, una vez que se obtiene el resultado esperado ya sea a nivel de negocio o a nivel de dato se hace

			necesario mostrar la información al usuario. Esta es una de las funciones del controlador frontal, actualizar la información en las páginas clientes.
		Licencia	-
Conector 4			
	Nombre	Render	
	Descripción	Este conector permite mostrar la vista correspondiente a cada uno de los casos de usos.	
		Usos	
		Conexion1	Zend_Controller –Zend_View
		Descripción	Cada controlador responde al menos a una vista o página cliente, una vez que se solicite la visualización de alguna funcionalidad del sistema, la petición llega hasta el controlador que responde a la misma. El controlador tiene la responsabilidad de ordenar la ejecución de la vista correspondiente, esta acción la ejecuta el método que lleva el mismo nombre que el controlador y que dentro contiene la función render que se encarga de pedirle a Zend_View que muestre la vista solicitada.
		Licencia	-
Conector 5			
	Nombre	Init	
	Descripción	Desencadena un grupo de validaciones y configuraciones de la arquitectura.	
		Usos	
		Conexion1	ServerPage – ZendExt_App
		Descripción	Todas las peticiones que se ejecutan en un subsistema son validadas por el controlador frontal, el mismo contiene un método llamado init que tiene la

			responsabilidad de realizar un conjunto de configuraciones y validaciones como: ➢ Configuración del sistema. ➢ Creación o validación de la sesión. ➢ Configuración de la conexión. ➢ Validación de seguridad. ➢ Gestión del perfil de usuario. ➢ Gestión del controlador solicitado.
		Licencia	-
Conector 6			
	Nombre	Dispatch	
	Descripción	Ejecuta el controlador solicitado.	
		Usos	
		Conexion1	ZendExt_App –ZendExt_Controller
		Descripción	Una vez que es identificado e inicializado el controlador que debe responder a la petición, se procede a la ejecución del mismo por medio del método dispatch, ubicado en Zend_Controller_Front.
		Licencia	-
Conector 7			
	Nombre	**Use** (Uso)	
	Descripción	Representa un tipo de relación muy particular, en la que una clase es instanciada (su instanciación es dependiente de otro objeto/clase).	
		Uso	
		Conexion1	ZendExt_Controller – ZendExt_Model
		Descripcion	La creación del objeto Model está condicionado a la instanciación proveniente desde el objeto Controller.
		Licencia	-
		Conexion2	ZendExt_Model – Doctrine_Query
		Descripcion	La creación del objeto Doctine_Query está condicionado a la instanciación proveniente

			desde el objeto Model.
		Licencia	-
		Conexion3	ZendExt_Model – Doctrine_Record
		Descripcion	La creación del objeto Doctine_Record está condicionado a la instanciación proveniente desde el objeto Model.
		Licencia	-
		Conexion4	Doctrine_Query – Doctrine_Conection
		Descripcion	La creación del objeto Doctrine_Conection está condicionado a la instanciación proveniente desde el objeto Doctrine_Query.
		Licencia	-
		Conexion5	Doctrine_Record – Doctrine_Conection
		Descripcion	La creación del objeto Doctrine_ Conection está condicionado a la instanciación proveniente desde el objeto Doctrine_ Record.
		Licencia	-
Conector 8			
	Nombre	**PDO** (PHP Data Objects)	
	Descripción	Es una librería escrita en C, que provee una capa de abstracción de acceso a datos para PHP 5, con lo cual se consigue hacer uso de las mismas funciones para hacer consultas y obtener datos de distintos manejadores de bases de datos. Es una interfaz de acceso a datos.	
		Usos	
		Conexion1	Doctrine – PostgreSQL
		Descripción	Permite la interacción entre la capa de acceso a dato perteneciente al sistema, y el gestor de base de dato. Es importante destacar que en esta interacción la información fluye en los dos sentidos y para lograr que el proceso sea más eficiente y ágil doctrine utiliza la capa de

			abstracción a datos PDO.
		Licencia	-
Conector 9			
	Nombre	**SOAP** (Simple Object Access Protocol) Protocolo Simple de Acceso a Datos.	
	Descripción	Es uno de los protocolos utilizados en los servicios Web. Define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. SOAP es un marco extensible y descentralizado que permite trabajar sobre múltiples protocolos de redes informáticos. Los procedimientos de llamadas remotas pueden ser modelados en la forma de varios mensajes SOAP interactuando entre sí. SOAP funciona sobre cualquier protocolo de Internet, Proporciona un mecanismo estándar de empaquetar mensajes generalmente HTTP.	
		Usos	
		Conexion1	Arquitectura – Capa de Servicios
		Descripción	Facilita el intercambio de información entre dos o más aplicaciones que se encuentren en distintos espacios de publicación o cuentan con diferentes tecnologías.
		Licencia	-
Conector10			
	Nombre	**IoC** (Inversión de controles)	
	Descripción	Componente creado con el objetivo de integrar sistemas que convivan en una única solución y mantengan comunicación e integridad de la información (datos) entre ellos.	
		Usos	
		Conexion1	Arquitectura – Capa de Servicios
		Descripción	Integra los subsistemas permitiendo la ejecución dinámica de funciones de unos a otros, y el envió y recepción de dato.
		Licencia	-

Conclusiones Parciales.

En este capítulo se establecieron un conjunto de políticas para el trabajo de los desarrolladores en cada una de las líneas. Además de las normas y estándares de codificación permitiendo una mejor comprensión del código y mantenimiento del mismo a largo plazo. También se establece un conjunto de normas por las cuales los desarrolladores deben regirse para lograr una mejor organización y entendimiento del trabajo. Se especificó cada una de las capas que componen la arquitectura tecnológica describiendo la vista estilística y la taxonomía estructural.

4 Capítulo 4: Descripción de los componentes.

Introducción.

En este capítulo se hace una detallada descripción de los componentes de Sauxe que se desarrollan para la versión 2.0 del Marco de Trabajo. Hasta el momento son 15 componentes, estos son: ZendExt_Exception, ZendExt_Nomencladores, ZendExt_FastResponse, ZendExt_Validation, ZendExt_IoC, ZendExt_Portal, ZendExt_ExpImp, ZendExt_Cache, ZendExt_GlobalConcept, ZendExt_Trace, ZendExt_ADT, ZendExt_App, ZendExt_MVC, ZendExt_TransactionManager, ZendExt_Aspect. Esta descripción encierra los siguientes aspectos:

- **Requisitos que implementa**

  Los requerimientos o requisitos no son más que condiciones o capacidades que tienen que ser alcanzadas o poseídas por un sistema o componente de un sistema para satisfacer un contrato, estándar, u otro documento impuesto formalmente.

- **Escenarios que resuelve**

  Es una vista a gran escala de las posibles situaciones que pueden darse según el componente en cuestión. Engloba un conjunto de requisitos.

- **Configuración y uso del componente**

  Explica de forma detallada la configuración que necesita el componente para que funcione y la forma correcta en que debe usarse.

- **Caso de estudio**

  Se incluyen todas las posibles formas de uso del componente.

Desarrollo.

4.1 ZendExt_Trace.

4.1.1 Introducción

En todo sistema informático llega un momento en el que surgen problemas. Ninguno es una excepción. Ante comportamientos inesperados, cuando un error inexplicable aparece, siempre se plantea la pregunta: ¿Qué está pasando? Para responder esa interrogante se hace necesario la creación de un mecanismo de registro oficial de eventos (trazas) y datos o información sobre quién, qué, cuándo, dónde y por qué un evento ocurre, ya sea para un módulo del sistema en particular o en toda la aplicación.

Las trazas son utilizadas, entre otras cosas, para comprobar que se realizan exactamente las funciones esperadas, y no otras, así como acreditar las validaciones de datos previstas, sin modificar el sistema en ningún momento. También son usadas con el fin de monitorear las acciones que se realicen (acceso a ficheros, dispositivos, empleo de los servicios, etc.), y para detectar indicios de

hechos relevantes a los efectos de la seguridad que puedan afectar la estabilidad o el funcionamiento del sistema informático. Para ello se recurre a productos de Software muy potentes y modulares que, entre otras funciones, rastrean los caminos que siguen los datos a través del programa.
Al realizar el registro y control de las trazas de un sistema el auditor informático emplea, preferentemente, la información que genera el propio sistema. Se hace una revisión especial en los LOGs (Registro oficial de eventos durante un rango de tiempo en particular) del sistema, donde se recogen las modificaciones de los datos y se detalla la actividad general de dicho sistema.

4.1.1.1 Problema a resolver

En el sistema en innumerables ocasiones ocurren errores que son muy difíciles de detectar o identificar, además no se tiene un registro de los eventos que ocurren en dicho sistema por lo que se hace necesario un componente que notifique la ocurrencia y una aplicación que posibilite la visualización de estos factores.

4.1.1.2 Objetivos

Desarrollar un componente que detecte y registre los eventos del proceso de negocio del sistema en todos sus subsistemas y que visualice este proceso.

4.1.2 Requisitos funcionales

R1–Permitir buscar trazas por:
- R1.1–Fecha de generación
- R1.2–Tipo
- R1.3–Categoría

R2–Generar una traza siempre que:
- R2.1–Se inicia una acción en el sistema
- R2.2–Se produce un acceso a datos
- R2.3–Se visite una URL
- R2.4–Se termine una acción en el sistema
- R2.5–Se produzca un error en una acción
- R2.6–Se produzcan excepciones
- R2.7–Se produzcan errores en el IoC externo
- R2.8–Se produzcan errores en el IoC interno
- R2.9–Se ejecute IoC interno
- R2.10–Se ejecute IoC externo

R3–Registrar traza de:
- R3.1–URL
- R3.2–Autenticación

R3.3–Cierre de sesión

R3.4–Acción

R3.5–Excepción

R3.6–Excepción de integración

R3.7–IoC

R3.8–Rendimiento

R3.9–Integración

R3.10–Dato

R4–Notificar a los sistemas del ERP la ocurrencia de un evento determinado

R5–Exportar las trazas

R6–Requisitos herramienta de monitoreo de trazas

R6.1–Configuración de las trazas

R6.2–Crear un nuevo tipo de traza

R6.3–Modificar las trazas

R6.4–Eliminar las trazas

R6.5–Gestionar las categorías de las trazas

R6.6–Gestionar los tipos de trazas

R7–Hacer reporte de trazas:

R7.1–Según su tipo.

R7.2–Según su categoría.

R7.3–De datos.

R7.4–Exportar reporte creado.

4.1.2.1 Escenarios que resuelve

Permite obtener la secuencia de acciones de un usuario sobre un sistema.

4.1.3 Configuración y uso del componente

Para el seguimiento de las trazas del sistema se Diseñó e implementó una aplicación que permite el registro de los eventos que ocurren en el sistema permitiendo de esta forma poder corregir problemas que se presenten. Estos eventos serían:

- Inicio de una acción: Se dispara un evento al comienzo de una acción.
- Terminación de una acción: Se dispara un evento al finalizar una acción.
- Error en una acción: Se dispara un evento cuando en una acción ocurre un error.
- Autenticar usuario: Se dispara un evento por cada URL visitada.
- IoC Externo: Se dispara un evento cuando ocurre integración entre diferentes subsistemas.

- IoC Interno: Se dispara un evento cuando ocurre integración entre diferentes componentes de un subsistema.
- Excepciones: Se dispara un evento cuando ocurre una excepción en el sistema.
- Rendimiento: Es el tiempo de ejecución de una acción.
- Error IoC Externo: Se dispara un evento cuando ocurre un error en la integración entre diferentes subsistemas.
- Error IoC Interno: Se dispara un evento cuando ocurre un error en la integración entre diferentes componentes de un subsistema.

Salvar las trazas de autenticación que se encuentra en la clase ZendExt_Aspect_Trace.

```
public function traceAutenticacion()
 {
   $ip_pc=$_SERVER['REMOTE_ADDR'];
   $register = Zend_Registry::getInstance();
   $user = $register->session->usuario;
   $global=ZendExt_GlobalConcept::getInstance();
   $idestructura=$global->Estructura->idestructura;
   $categoria=$global->Subsistema->idsubsistema;

   $hisAutentication=new ZendExt_Trace_Container_Autentication($ip_pc,
   $categoria,   $user, $idestructura);

   //$tm=ZendExt_Trace::getInstance();//creo una instancia de esa traza y se
   la paso al manejador
   //$tm=ZendExt_Trace::handle($hisAutentication);

   $db=new ZendExt_Trace_Publisher_Db();//creo el publicador y le paso la
   traza para que la guarde en la bd
   $db->save ($hisAutentication);
 }
```

Salvar las trazas de cierre de sesión que se encuentra en la clase ZendExt_Aspect_Trace

```
public function traceCierreSesion()
 {
   $ip_pc=$_SERVER['REMOTE_ADDR'];
   $register = Zend_Registry::getInstance();
   $user = $register->session->usuario;
   $global=ZendExt_GlobalConcept::getInstance();
   $idestructura=$global->Estructura->idestructura;
   $categoria=$global->Subsistema->idsubsistema;

   $hisCierreSesion=new ZendExt_Trace_Container_CierreSesion( $ip_pc,
   $categoria, $user, $idestructura);

   // $tm=ZendExt_Trace::getInstance();//creo una instancia de esa traza y se
   la paso al manejador
   //$tm=ZendExt_Trace::handle($hisCierreSeccion);

   $db=new ZendExt_Trace_Publisher_Db();//creo el publicador y le paso la
   traza para que la guarde en la bd
   $db->save ($hisCierreSesion);
 }
```

Salvar las trazas de excepciones que se encuentra en la clase ZendExt_Aspect_Trace

```
public function exceptionTraceAction($exception)
 {
   $code = $exception->getIdException();
   $type = $exception->getType();
   $register = Zend_Registry::getInstance();
   $languaje = $register->session->perfil['idioma'];;
   $mensaje = $exception->getMessage();
   $descripcion = (string) $exception->getDescription();
   $global = ZendExt_GlobalConcept::getInstance();
   $idestructura = $global->Estructura->idestructura;
   $moduleReference = $register->get(config)->module_reference;
   $log = $exception->getInnerException ();
   $categoria = $global->Subsistema->idsubsistema;
   $user = $register->session->usuario;
   $exceptionTrace = new ZendExt_Trace_Container_Exception( $code, $type,
   $languaje,  $mensaje, $descripcion, $log, $categoria, $user,
   $idestructura);
   $instance = ZendExt_Trace :: getInstance ();
   $instance->handle ($exceptionTrace);
 }
```

Salvar las trazas de fallo de acción que se encuentra en la clase ZendExt_Aspect_Trace

```
public function
failedTraceIoC($exception,$targetComponent,$class,$method,$iocType)
 {
   $sourceComponent = Zend_Registry::get('config')->module_name;
   $register = Zend_Registry::getInstance();
   $usuario = $register->session->usuario;
   $global = ZendExt_GlobalConcept::getInstance();
   if ($iocType == 'ioc')
      $pIdTraceCategory = '99';
   else
      $pIdTraceCategory = $global->Subsistema->idsubsistema;
      $pIdCommonStructure = $global->Estructura->idestructura;
      $request = Zend_Controller_Front::getInstance () -> getRequest();
   if($request)
     {
      $action = $request->getActionName();
      $controller = $request->getControllerName();
     }
   else
     {
      $action = 'framework';
      $controller = 'framework';
     }
   $code = $exception->getIdException();
   $mensaje = $exception->getMessage();
   $descripcion = (string) $exception->getDescription();
   $log = (string)$exception->getInnerException ();
   $integracion = new ZendExt_Trace_Container_IoCException($code,$mensaje,
   $descripcion, $sourceComponent, $targetComponent, $class, $method, $action,
   $log, $controller, $pIdTraceCategory, $usuario, $pIdCommonStructure);

   $instance = ZendExt_Trace :: getInstance ();
   $instance->handle ($integracion);
 }
```

Salvar las trazas de fin de la acción que se encuentra en la clase ZendExt_Aspect_Trace

```
public function endTraceAction()
 {
   $register = Zend_Registry::getInstance();
   $usuario = $register->session->usuario;
   $categoria = 99;
   $frontController = Zend_Controller_Front::getInstance ();
   $action = $frontController->getRequest ()->getActionName ();
   $controller = $frontController->getRequest ()->getControllerName ();
   $this->script_runtime();
   $exectime=$this->end_script_runtime;
   $global=ZendExt_GlobalConcept::getInstance();
   $idestructura=$global->Estructura->idestructura;
   $moduleReference=$register->config->module_reference;
   $memory = (memory_get_usage() - $this->memory)/1048576;
   $endTrace = new
   ZendExt_Trace_Container_Performance($exectime,$memory,false,
   false,$moduleReference,$controller,
   $action,$categoria,$idestructura,$usuario);
   $instance = ZendExt_Trace :: getInstance ();
   $instance->handle ($endTrace);
 }
```

Salvar las trazas de Inicio de sesión que se encuentra en la clase ZendExt_Aspect_Trace

```
public function beginTraceAction()
 {
   $this->script_runtime();
   $frontController = Zend_Controller_Front::getInstance ();
   $controller = $frontController->getRequest()->getControllerName ();
   $action=$frontController->getRequest ()->getActionName();
   $register = Zend_Registry::getInstance();
   $user = $register->session->usuario;
   $global=ZendExt_GlobalConcept::getInstance();
   $idestructura=$global->Estructura->idestructura;
   $moduleReference=$register->config->module_reference;
   $array=explode('/',$moduleReference);
   $categoria=$global->Subsistema->idsubsistema;
   $beginTrace = new ZendExt_Trace_Container_Action(false, true,
   $moduleReference, $controller, $action, $categoria,  $user, $idestructura);
   $instance = ZendExt_Trace :: getInstance();
   $instance->handle ($beginTrace);
   $this->memory = memory_get_usage();
 }
```

Salvar las trazas de inicio de Integración que se encuentra en la clase ZendExt_Aspect_Trace

```
public function beginTraceIoC($targetComponent, $class, $method, $iocType)
 {
   $sourceComponent = Zend_Registry::get('config')->module_name;
   $register = Zend_Registry::getInstance();
   $usuario = $register->session->usuario;
   $global = ZendExt_GlobalConcept::getInstance();
   if ($iocType == 'ioc')
      $pIdTraceCategory = '99';
   else
      $pIdTraceCategory = $global->Subsistema->idsubsistema;
      $pIdCommonStructure = $global->Estructura->idestructura;
      $request = Zend_Controller_Front::getInstance () -> getRequest();
   if($request)
```

```
        {
          $action = $request->getActionName();
          $controller = $request->getControllerName();
        }
      else
        {
          $action = 'framework';
          $controller = 'framework';
        }
      if($iocType == 'ioc')
        {
          $integracion = new ZendExt_Trace_Container_IoC(false, $sourceComponent,
          $targetComponent, $action, $class, $method, $pIdTraceCategory, $usuario,
          $pIdCommonStructure);
        }
      else
        {
          $integracion = new ZendExt_Trace_Container_IoC(true, $sourceComponent,
          $targetComponent, $action, $class, $method, $pIdTraceCategory, $usuario,
          $pIdCommonStructure);
        }
      $instance = ZendExt_Trace :: getInstance ();
      $instance->handle ($integracion);
    }
```

Salvar las trazas de fallo de Integración que se encuentra en la clase ZendExt_Aspect_Trace

```
    public function
    failedTraceIoC($exception,$targetComponent,$class,$method,$iocType)
     {
      $sourceComponent = Zend_Registry::get('config')->module_name;
      $register = Zend_Registry::getInstance();
      $usuario = $register->session->usuario;
      $global = ZendExt_GlobalConcept::getInstance();
      if ($iocType == 'ioc')
         $pIdTraceCategory = '99';
      else
         $pIdTraceCategory = $global->Subsistema->idsubsistema;
         $pIdCommonStructure = $global->Estructura->idestructura;
         $request = Zend_Controller_Front::getInstance () -> getRequest();
      if($request)
        {
          $action = $request->getActionName();
          $controller = $request->getControllerName();
        }
      else
        {
          $action = 'framework';
          $controller = 'framework';
        }
      $code = $exception->getIdException();
      $mensaje = $exception->getMessage();
      $descripcion = (string) $exception->getDescription();
      $log = (string)$exception->getInnerException ();
      $integracion = new ZendExt_Trace_Container_IoCException($code, $mensaje,
      $descripcion, $sourceComponent, $targetComponent, $class, $method, $action,
      $log, $controller, $pIdTraceCategory, $usuario, $pIdCommonStructure);

      $instance = ZendExt_Trace :: getInstance ();
      $instance->handle ($integracion);
```

```
}
```

4.1.3.1 Ficheros de configuración

Las trazas se activan en el fichero aspect.xml. Se cambia el valor del atributo active a **true**

```
<TraceAutenticacion active="true">
   <class name="ZendExt_Aspect_Trace" singlenton="true">
      <method name="traceAutenticacion" static="false"/>
   </class>
</TraceAutenticacion>

<TraceCierreSesion active="true">
   <class name="ZendExt_Aspect_Trace" singlenton="true">
      <method name="traceCierreSesion" static="false"/>
   </class>
</TraceCierreSesion>

<beginTraceAction active="true">
   <class name="ZendExt_Aspect_Trace" singlenton="true">
      <method name="beginTraceAction" static="false"/>
   </class>
</beginTraceAction>
<endTraceAction active="true">
   <class name="ZendExt_Aspect_Trace" singlenton="true">
      <method name="endTraceAction" static="false"/>
   </class>
</endTraceAction>
<failedTraceAction active="true">
   <class name="ZendExt_Aspect_Trace" singlenton="true">
      <method name="failedTraceAction" static="false"/>
   </class>
</failedTraceAction>
<beginTraceIoC active="true">
   <class name="ZendExt_Aspect_Trace" singlenton="true">
      <method name="beginTraceIoC" static="false"/>
   </class>
</beginTraceIoC>
<failedTraceIoC active="true">
   <class name="ZendExt_Aspect_Trace" singlenton="true">
      <method name="failedTraceIoC" static="false"/>
   </class>
</failedTraceIoC>
```

4.1.4 Caso de estudio

La gestión de las trazas presenta una interfaz gráfica la cual mediante la disposición de elementos para la búsqueda avanzada de las trazas generadas en todo el sistema.

Elementos

- Rango de fechas
- Categoría: Engloba todos los subsistemas.
- Tipo: Engloba los tipos de trazas que se generan (Acción, Excepción, Rendimiento, Integración, Excepción de integración).

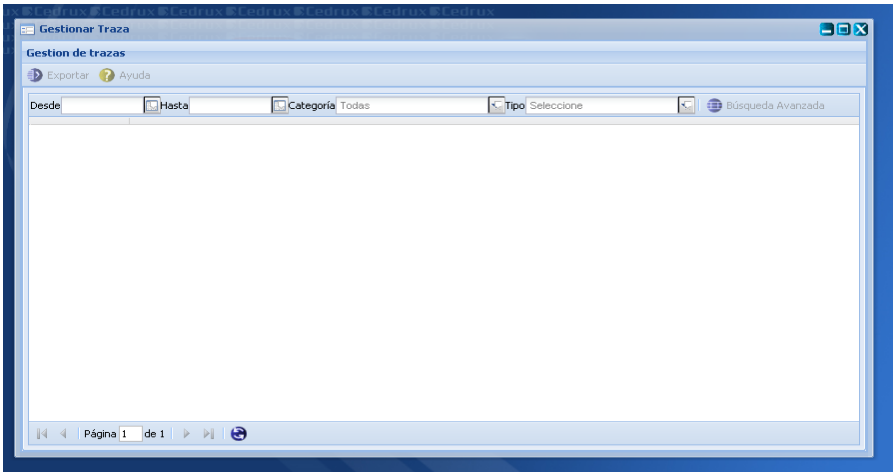


Figure 1. Interfaz de gestión de trazas

Ejemplo: Todas las trazas de **Acción** del subsistema de **Estructura y composición**

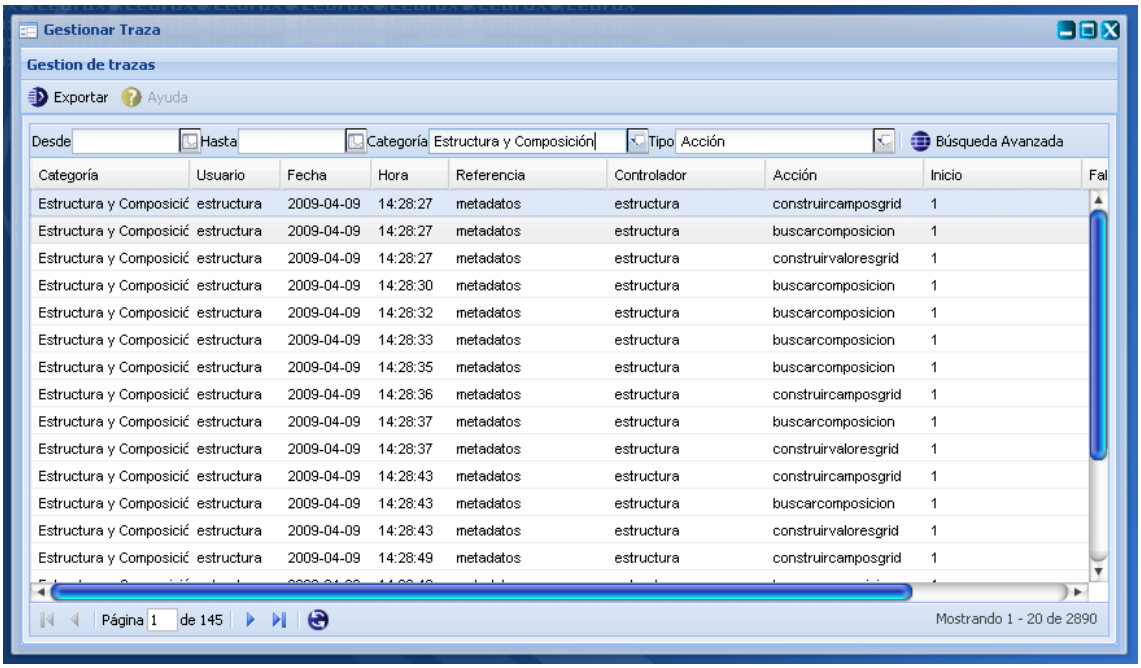


Figure 2. Interfaz de gestionar trazas. Búsqueda

Esta interfaz brinda la posibilidad de exportar las trazas seleccionadas en formato XML

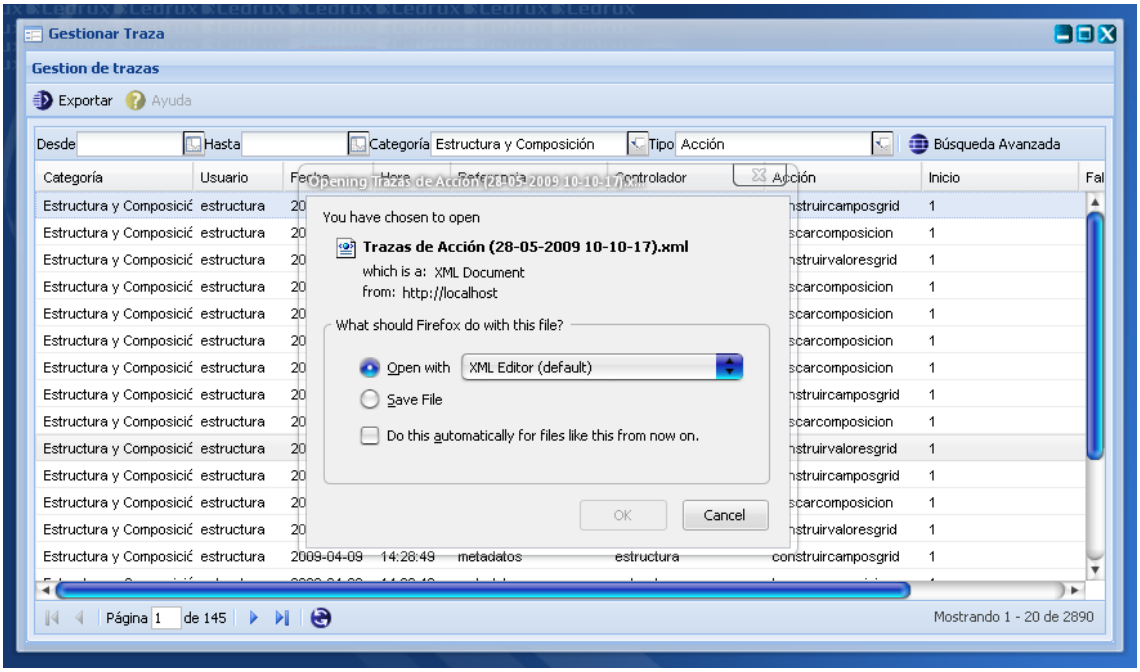


Figure 3. Interfaz gestionar trazas. Exportar trazas.

De la funcionalidad exportar trazas devuelve un XML en el siguiente formato:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xml>
  - <trazas tipo="Acción">
    - <traza_1>
        <idestructuracomun>100000001</idestructuracomun>
        <usuario>estructura</usuario>
        <hora>14:28:27</hora>
        <fecha>2009-04-09</fecha>
        <referencia>metadatos</referencia>
        <controlador>estructura</controlador>
        <accion>construircamposgrid</accion>
        <inicio>1</inicio>
        <falla>No posee</falla>
        <categoria>Estructura y Composición</categoria>
      </traza_1>
    - <traza_2>
        <idestructuracomun>100000001</idestructuracomun>
        <usuario>estructura</usuario>
        <hora>14:28:27</hora>
        <fecha>2009-04-09</fecha>
        <referencia>metadatos</referencia>
        <controlador>estructura</controlador>
        <accion>buscarcomposicion</accion>
        <inicio>1</inicio>
        <falla>No posee</falla>
        <categoria>Estructura y Composición</categoria>
      </traza_2>
```

Los demás componentes se especificarán en los anexos.

Conclusiones Parciales.

Con la descripción de los componentes que conforman el núcleo del Marco de Trabajo se identificaron las responsabilidades arquitectónicas que cumplen cada uno de ellos, para cubrir un conjunto de escenarios indispensables para una tecnología de este tipo. Para lograr una mayor calidad en el proceso de generalización, mantenimiento y extensión de la arquitectura se desarrolló por cada componente un conjunto de temas como la configuración y uso, caso de estudio en el que se ejemplifica de forma real la utilización de las funcionalidades de cada componente y se describieron cada uno de los requisitos que resuelven un conjunto de escenarios.

Validación del Trabajo.

El Libro de Ayuda del Marco de Trabajo es sumamente importante, tanto en el mantenimiento de la aplicación como en el uso de la misma por los desarrolladores y usuarios finales. Esta está validada la documentación de los componentes por los diferentes proyectos productivos que utilizan este Marco de Trabajo tales como La Línea de Sistemas de Información Geográfica (GIS) y la Línea de Contabilidad Material. Además todos los manuales de usuarios fueron aceptados satisfactoriamente por las diferentes entidades en la que se está aplicando este Marco de Trabajo como son Aduana, DESOFT, entre otros proyectos de la UCI.

Conclusiones.

La elaboración del marco teórico permitió identificar los aspectos a tener en cuenta en el desarrollo del Libro de Ayuda del Marco de Trabajo Sauxe, para lograr uniformidad en el proceso de desarrollo de aplicaciones. Se planteó la vía de solución teniendo en cuenta la metodología señalada y a través de esta quedó conformada la propuesta de solución. Primeramente se desarrolló el manual de instalación y configuración de las herramientas tanto para Windows como para Linux. Se valoró el impacto social, tecnológico novedoso y económico que aporta un Marco de Trabajo de este tipo en el desarrollo de aplicaciones web de gestión, así como el impacto de un Libro de Ayuda que contenga toda su información.

Se realizó la descripción del proceso de instalación y configuración del Marco de Trabajo Sauxe. En la misma se especifican las vistas estructurales de la arquitectura, la estructura que debe cumplir un subsistema y los pasos a seguir para lograr la integración y visualización de cada una de las funcionalidades.

Se establecieron normas y estilos de codificación permitiendo una mejor comprensión del código y mantenimiento del mismo a largo plazo. Quedaron explícitamente detalladas las capas por las cuales está conformada la arquitectura de Sauxe, así como las descripciones de la taxonomía estructural y los conectores de estilos.

Con la descripción de los componentes que conforman el núcleo del Marco de Trabajo se identificaron las responsabilidades arquitectónicas que cumplen cada uno de ellos, para cubrir un conjunto de escenarios indispensables para una tecnología de este tipo. Para lograr una mayor calidad en el proceso de generalización, mantenimiento y extensión de la arquitectura se desarrolló por cada componente un conjunto de temas como la configuración y uso, caso de estudio en el que se ejemplifica de forma real la utilización de las funcionalidades de cada componente y se describieron cada uno de los requisitos que resuelven un conjunto de escenarios.

Recomendaciones.

Se recomienda que el resultado propuesto en la investigación sea implantado como uso cotidiano e introducido en el desarrollo de las aplicaciones web de gestión que se desarrollen sobre el Marco de Trabajo Sauxe.

Se recomienda continuar actualizando este trabajo, en la medida que se avance en las nuevas versiones del Marco de Trabajo.

Referencias bibliográficas

Baryo, Gómez, Oiner y García, Tejo, Darien. 2009. *XVI Fórum de Ciencia y Técnica.* 2009.

Baryolo, Gómez, Baryolo, Tenrero, Cabrera, Marianela y Silega, Martínez, Nemuris. 2008. *Plantilla Registro de la Propiedad intelectual(Sauxe).* 2008.

Bosch, Carlos. 2009. Tutorial uml, modelo de clases. [En línea] 2009. http://www.dcc.uchile.cl/~psalinas/uml/modelo.html.

Corso, Giancarlo. 2008. ExtJS lo bueno, lo malo y lo feo. [En línea] 22 de Octubre de 2008. http://blogs.antartec.com/desarrolloweb/2008/10/extjs-lo-bueno-lo-malo-y-lo-feo/.

Iberia, Sescoi. 2010. Las empresas que gestionan proyectos que necesitan soluciones ERP especializadas. [En línea] 2010. http://www.workplan-enterprise.com/fileadmin/pdf/myworkplan/WhitePaper_ERP-JobManag-2_SP.pdf.

Leopoldo, Carlos. 2007. ZendFramework, introducción. [En línea] 2007. http://techtastico.com/post/zend-framework-una-introduccion/.

Pérez, Sarduí, mileydis y Hernández, Cisnero, Sergio. 2009. *Trabajo de Diploma para optar por el título de Ingeniero Informático.* Ciudad de La Habana : s.n., 2009.

Server Gove. 2010. Doctrine. [En línea] 2010. http://www.doctrine-project.org/.

Bibliografía.

http://www.milestone.com.mx/articulos/uso_de_uml_en_aplicaciones_web.htm

http://www.xolucionesinformaticas.com/index.php?option=com_content&view=article&id=77

http://www.softwarelibre.org.bo/esteban/files/86/226/atix08.pdf

http://www.mail-archive.com/programacion@listas.fi.uba.ar/msg01556.html

http://kartones.net/blogs/coco/archive/2009/12/05/la-capa-de-servicios-conceptos-b-225-sicos.aspx

http://www.manycomics.com/ingenieria-del-software/uml/casos-de-uso

http://www.milestone.com.mx/articulos/uso_de_uml_en_aplicaciones_web.htm

http://www.softwarelibre.org.bo/esteban/files/86/226/atix08.pdf

http://www.mail-archive.com/programacion@listas.fi.uba.ar/msg01556.html