Universidad de las Ciencias Informáticas

Facultad 3

"Módulo de seguridad para el marco de trabajo Symfony2."

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Autores: *Jeandy Miranda Díaz*
Jesús Cobacho Aguilera.

Tutores: *Dr. Oiner Gómez Baryolo*

Ing. Arnolis Salgueiro Arzuaga

Ing. Tania Pérez Ramírez.

La Habana, 2014. Año 56 de la Revolución

Declaración de autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____

Jeandy Miranda Díaz

Autor

_____

Jesús Cobacho Aguilera

Autor

_____

Ing. Arnolis Salgueiro Arzuaga

Tutor

_____

Ing. Tania Pérez Ramírez.

Tutor

_____

Dr. Oiner Gómez Baryolo

Tutor

Jeandy:

Dedico esta tesis a mis padres por siempre mi ejemplo y guía, a mi hermanita quien me hace querer ser una mejor persona y servirle de ejemplo.

Jesús:

Le dedico esta tesis a mis padres por ser un ejemplo de humildad y sacrificio, gracias a ellos soy la persona que soy.

Agradecimientos

Jeandy:

A mi madre por ser la mejor madre y amiga del mundo, por todo el sacrificio que hace día a día para que yo tenga un buen futuro, por su amor incondicional que nunca me ha faltado a pesar de la distancia, por ser Barbarita ,por querer siempre lo mejor para mí, por cada consejo dado. A mi padre ser el mejor ejemplo de cómo debe ser un padre, por ser amigo, por todas las veces que me dio pastillas en vez de inyectarme, por enseñarme a separar lo que está bien de lo que está mal. A mi hermanita por siempre tenerme presente, por quererme tanto a pesar de que casi nunca la veo y por ser la hermana más linda del mundo. A mi familia por la preocupación mostrada durante todos estos años. A Mijail, al Puro, a Carlito, a mi primo Ismel. A mi compañero de tesis. A mis amigos de casi toda una vida: Frank David, Ricardo, Reynier, Emil y Raudel. A las amistades de la UCI: Juan Ignacio, Miguelon, Anel, Arturo, Dargel, El Pulga, Gendry, Javico, Julio, El Migue. A todos aquellos que de una forma u otra contribuyeron a la formación de mi personalidad. A los profesores que me impartieron clases durante estos 5 cursos.

Jesús:

A toda mi familia especialmente a mis padres por su apoyo incondicional, su preocupación, y siempre confiar en mí. A mi hermano por ayudarme y aconsejarme siempre, a mis tías "la negra y mima" que han tomado el papel de madre y padre durante estos años. A mis amistades del barrio: El luisi, Javier, martiato, yamil, los narras. A mi compañero de tesis. A las amistades de la UCI: Arturo, Dargel, El pulga, Rachel, Gendry, El javico, EL migue y de manera general mi grupo y a todas esas personas que me ayudaron durante estos cinco años.

Índice

Índice de tablas

Índice de figuras

Resumen

Las aplicaciones web constituyen una herramienta para la gestión de la información, donde entidades hacen uso de estas para la colaboración entre ellas y sus necesidades específicas. En dichas entidades la información toma un papel fundamental, dando paso a la necesidad de garantizar su seguridad de una manera centralizada.

El Centro de Informatización de Entidades perteneciente a la Universidad de las Ciencias Informáticas es uno de los centros encargados de desarrollar soluciones que faciliten la gestión empresarial. Actualmente en dicho centro no se cuenta con un componente que garantice la gestión centralizada de la seguridad, en los sistemas que se implementen sobre el marco de trabajo Symfony2. Con el propósito de eliminar dicho problema, el presente trabajo propone el desarrollo de un módulo de seguridad para el marco de trabajo Symfony2, donde se expone como resultado un bundle que soporta los entornos multi-dominio y multi-sistema. Dicho bundle contiene cuatro subsistemas: Sistemas, Usuarios, Servidores y Nomencladores, permitiendo una correcta gestión de la seguridad. Además se gestionan los procesos de autenticación y autorización de usuarios en el sistema, ayudando así al aumento de la seguridad de las aplicaciones desarrolladas por el marco de trabajo Symfony2.

Palabras claves: seguridad, marco de trabajo, multi-sistema, multi-dominio, autenticación, autorización

Introducción

Las Tecnologías de la Información y las Comunicaciones (TIC) en la actualidad, están expuestas a sufrir constantes cambios, siempre en pos del mejoramiento y facilidad de uso en el trabajo diario de las personas. Esto se evidencia en el uso extendido de las aplicaciones web, las cuales desempeñan un papel fundamental en el mercado del software siendo cada vez mayor el número de empresas que prestan sus servicios a través de la web. Cuba, a pesar del nocivo impacto del bloqueo económico, dirige sus esfuerzos a informatizar los negocios de entidades u organizaciones en aras de alcanzar un desarrollo económico-social a la altura de los países desarrollados. Con este objetivo fue concebida la Universidad de las Ciencias Informáticas (UCI), centro de altos estudios donde se vinculan el estudio, la investigación y la producción.

El Centro de Informatización de Entidades (CEIGE), perteneciente a la UCI, cuenta con un departamento encargado de gestionar las tecnologías utilizadas para el desarrollo de software. En dicho departamento se desea implementar un sistema sobre el marco de trabajo Symfony2, que facilite el desarrollo de software de gestión. A raíz de esto, ocupa el centro de atención: “la información”, la cual puede poner en peligro a entidades y a sus respectivos clientes, de ser difundida sin tener en cuenta la confidencialidad, disponibilidad e integridad de los datos. Es por ello que para el desarrollo de aplicaciones web es necesario mantener un sistema de seguridad que garantice los elementos anteriormente expuestos.

El módulo de seguridad del marco de trabajo Symfony2 cuenta con un archivo denominado security.yml, donde se hace uso de los componentes que brinda Symfony2 para la gestión de la seguridad. En este archivo son utilizadas una serie de configuraciones para garantizar temas como la autenticación y la autorización, entre otros. En aplicaciones complejas realizar estas configuraciones resultaría engorroso, y significaría pérdida de tiempo destinada al desarrollo de sistemas. Además el módulo de seguridad de Symfony2 no soporta entornos multi-dominio, ni multi-sistema, lo que limita el alcance de las aplicaciones realizadas. “Un entorno multi-dominio está asociado a una estructura jerárquica compleja, que agrupa a varias organizaciones atendiendo a un criterio común, para facilitar el acceso e interoperabilidad entre sus sistemas informáticos distribuidos de una forma segura”(1). “El aspecto multi-sistema se centra en que los usuarios puedan cumplir un conjunto de roles comunes en un conjunto de sistemas y dominio de entidades”(2).

Partiéndose de la situación problemática descrita anteriormente se plantea como **problema a resolver:** ¿Cómo gestionar los procesos de identificación, autenticación y autorización en entornos multi-sistemas y multi-dominios de los sistemas desarrollados sobre el marco de trabajo Symfony2?

Se traza como **Objetivo General**: Desarrollar un módulo de seguridad para el marco de trabajo Symfony2 que permita gestionar los procesos de identificación, autenticación y autorización en entornos multi-sistemas y multi-dominios para fortalecer la seguridad de las aplicaciones implementadas.

Desglosándolo en los **Objetivos Específicos**:

- Construir el marco teórico-referencial de la investigación a partir del estudio de las particularidades del Módulo de Seguridad de Symfony2 ya existente.
- Realizar el análisis y diseño del Módulo de Seguridad de Symfony2.
- Implementar las funcionalidades necesarias siguiendo buenas prácticas del desarrollo de software.
- Realizar las pruebas pertinentes para validar el Módulo de Seguridad de Symfony2.

Para enmarcar los límites de la presente investigación se define como **Objeto de Investigación**: Sistemas de seguridad en aplicaciones web. Delimitando el **Campo de Acción** en: Módulo de Seguridad del Marco de Trabajo Symfony2.

Se plantea como **Idea a Defender:** Con el desarrollo del Módulo de seguridad para el marco de trabajo Symfony2 se fortalecerá la seguridad de las aplicaciones web a partir de una mayor gestión del módulo.

Durante el desarrollo de la investigación se emplearon los siguientes **métodos científicos:**

Teóricos:

Histórico–lógico: A través de este método se estudia la trayectoria real de los elementos que se utilizan en la implementación de un sistema de procesamiento de opiniones. Este método es de gran importancia, su uso durante la realización del estudio del estado del arte permitió evaluar el desarrollo de los sistemas de procesamiento de opiniones o sistemas similares, la evolución de sistemas y herramientas de desarrollo que pueden ser usadas para la elaboración de la solución propuesta.

Analítico-Sintético: Mediante este método se identifican todos los conceptos así como las definiciones más importantes relacionadas con el tema, permitiendo generar una propuesta adecuada a la situación planteada. Este método fue de vital importancia y fue utilizado en todo el desarrollo del trabajo investigativo cediendo al reconocimiento de los principales conceptos y fundamentos a usar que permitieran el desarrollo satisfactorio de las acciones.

Empíricos:

La observación: Mediante el cual se puede observar los avances realizados en el estudio de las diferentes herramientas y tecnologías a usar. Se usa durante todo el desarrollo del trabajo permitiendo a los investigadores evaluar el proceso y los avances realizados en la elaboración de la solución propuesta.

Luego de cumplir el objetivo general, los **posibles resultados** del presente trabajo serán:

Modelo del negocio

a) Mapa de procesos del negocio
b) Descripción de procesos del negocio

Validación de procesos del negocio

a) Modelo conceptual
b) Descripción de requisitos (incluyendo prototipos de interfaz de usuario)
c) Validación de requisitos
d) Descripción de Documentos de caso de prueba (DCP)

Modelo de diseño

a) Modelo de diseño
b) Validación del diseño

Implementación

a) Implementación del módulo de seguridad de Symfony2.
b) Pruebas de caja blanca y caja negra.

Estructura del documento

El presente trabajo de diploma está compuesto por 3 capítulos desglosados a continuación.

Capítulo 1 Fundamentación Teórica: En este capítulo se abordarán los principales conceptos relacionados con el desarrollo de aplicaciones web y se hará una breve descripción de las metodologías, herramientas y tecnologías que se utilizarán en el desarrollo de la solución.

Capítulo 2 Análisis y diseño del módulo de seguridad: En este capítulo se muestran todas las características que tendrá el módulo de seguridad, partiendo de la descripción general de la propuesta del mismo. Se generan así los artefactos correspondientes a las fases de modelación y descripción de requisitos especificados en el modelo de desarrollo aplicado. Se obtendrá además el diseño de la solución propuesta.

Capítulo 3 Implementación y prueba: En este capítulo se abarcan los elementos referentes al Desarrollo del módulo de seguridad para el marco de trabajo Symfony2. Se valida el diseño y la implementación de la solución mediante métricas y pruebas de caja blanca y caja negra.

Capítulo 1: Fundamentación Teórica

1.1 Introducción

En el presente capítulo se abordan los principales conceptos necesarios para entender el desarrollo de esta investigación. Se analiza la situación actual en los procesos de seguridad de las aplicaciones web. Se establece una comparación entre las opciones existentes y las tendencias actuales, en cuanto a seguridad a nivel mundial y se detallan las herramientas y tecnologías utilizadas para dar solución al problema planteado.

1.2 Principales conceptos

Seguridad informática

La seguridad informática es una disciplina que se relaciona a diversas técnicas, aplicaciones y dispositivos encargados de asegurar la integridad y privacidad de la información de un sistema informático y sus usuarios (3).

FrameWork (marco de trabajo)

En general, con el término marco de trabajo, se refiere a una estructura software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación, cuyo objetivo es: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como es el uso de patrones (4).

Bundle

Un bundle simplemente es un conjunto estructurado de archivos en un directorio que implementa una sola característica. Cada directorio contiene todo lo relacionado con esa característica, incluyendo archivos PHP, plantillas, hojas de estilo, archivos Javascript, pruebas y cualquier otra cosa necesaria. Cada aspecto de una característica existe en un bundle y cada característica vive en un bundle (5).

RBAC (Control de Acceso Basado en Roles)

Es un sistema para controlar cuáles usuarios tienen acceso a recursos basados en el rol del usuario. Los derechos de acceso se agrupan por nombres de roles, y el acceso a los recursos se restringe a los usuarios que han sido autorizados para asumir el rol asociado (6).

Gestor de base de datos

Un sistema gestor de base de datos se define como el conjunto de programas que administran y gestionan la información contenida en una base de datos (7).

Proporciona una interfaz entre los datos, los programas que los manejan y los usuarios finales. Cualquier operación que el usuario hace contra la base de datos está controlada por el gestor (7).

IDE (Integrated development enviorament)

Existen diferentes formas de introducir un programa Java en una computadora. Se puede utilizar un Ambiente de Desarrollo Integrado (IDE, por sus siglas en inglés) o un editor de textos plano. Un IDE es más que una larga pieza de software, que permite introducir, compilar y ejecutar programas. La introducción, compilación y ejecución son parte del desarrollo de un programa y están integradas juntas en un ambiente, de ahí el nombre ambiente de desarrollo integrado (8).

Lenguaje de programación

Es un lenguaje diseñado para describir el conjunto de acciones consecutivas que un equipo debe ejecutar. Por lo tanto, un lenguaje de programación es un modo práctico para que los seres humanos puedan dar instrucciones a un equipo (9).

Interfaz de usuario

En los sistemas informáticos, la relación humano-computadora se realiza por medio de la interfaz, que se podría definir como mediador (10).

URL

Son las siglas de Localizador de Recurso Uniforme (en inglés Uniform Resource Locator), la dirección global de documentos y de otros recursos en la World Wide Web (11).

Servicios web

Conjunto de aplicaciones o de tecnologías con capacidad para interoperar en la Web. Estas intercambian datos entre sí con el objetivo de ofrecer servicios. Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios los solicitan llamando a estos procedimientos a través de la Web (12).

APIs

(Application Programming Interface - Interfaz de Programación de Aplicaciones). Grupo de rutinas (conformando una interfaz) que provee un sistema operativo, una aplicación o una biblioteca, que definen cómo invocar desde un programa un servicio que éstos prestan. En otras palabras, una API representa un interfaz de comunicación entre componentes software. También Es el método que utiliza un programa para interactuar con funciones del sistema operativo (13).

Plataforma

En informática, una plataforma es un sistema que sirve como base para hacer funcionar determinados módulos de hardware o de software con los que es compatible. Dicho sistema está definido por un estándar alrededor del cual se determina una arquitectura de hardware y una plataforma de software (incluyendo entornos de aplicaciones). Al definir plataformas se establecen los tipos de arquitectura, sistema operativo, lenguaje de programación o interfaz de usuario compatibles (14).

Servidor web

Es la tecnología que tiene implícito programas informáticos que procesan aplicaciones realizando conexiones bidireccionales y/o unidireccionales y síncronas o asíncronas con el cliente, generando o cediendo una respuesta en cualquier lenguaje o aplicación del lado del cliente (15).

Lamp

(Linux-Apache-MySQL- PHP/Python/PERL). El término hace referencia al sistema creado por la conjunción de esas aplicaciones libres (de código abierto). Este grupo de aplicaciones generalmente son usados para crear servidores web.

LAMP provee a los desarrolladores con los cuatro elementos necesarios para un servidor web: un sistema operativo (Linux), un manejador de base de datos (MySQL), un software para servidor web (Apache) y un software de programación script web (PHP, Python o PERL). Se ha convertido un estándar de facto para los servidores web pues es completamente libre y no se necesitan pagar licencias (lo que encarecería el servicio) (16).

WampServer

WampServer es un entorno de desarrollo web para Windows en el cual se podrán crear aplicaciones web con Apache, PHP y base de datos en MySQL (motor de base de datos). Esta herramienta incluye además con un administrador de base de datos PHPMyAdmin con el cual podremos crear una nueva base de datos e ingresar la data de las tablas creadas en ella, realizar consultas y generar scripts SQL, como exportar e importar scripts de base de datos. WampServer ofrece a los desarrolladores herramientas necesarias para realizar aplicaciones web de manera local, con un sistema operativo (Windows), un manejador de base de datos (MySQL), un software de programación script web PHP. WampSever se caracteriza por que puede ser usado de forma libre es decir no debemos de contar con alguna licencia el cual nos permita el uso de la misma, ya que pertenece a la corriente de "open source" (17).

Virtual host(host virtuales)

El concepto de “virtual host” se refiere a que en un mismo servidor web se pueden hospedar múltiples proyectos cada uno con su propio dominio aunque todos pertenezcan a la misma dirección IP pública. Apache es un servidor web que permite el uso de este método (18).

Multiplataforma

En informática se dice que un elemento (programas, lenguajes de programación, elementos de hardware)es multiplataforma cuando tiene la capacidad de funcionar en más de un sistema operativo con similares características y sin que su funcionalidad varíe en exceso (19).

1.3 Marco Conceptual

A continuación se definen los principales aspectos teóricos sobre la seguridad en las aplicaciones web, haciendo énfasis en su tratamiento.

1.3.1 Seguridad en aplicaciones web

Todos los sistemas informáticos, no importa cuán sofisticados en cuanto a seguridad sean, siempre estarán expuestos a sufrir un error o una alteración maliciosa intencional. Cualquier organización que expone sus servicios informáticos a redes de acceso tendrá que realizar un esfuerzo significativo para asegurar que la información y los recursos estén protegidos.

La seguridad en una aplicación web, se puede dividir en los siguientes aspectos básicos : (20)

- **Integridad:** Propiedad o característica consistente en que el activo de información no ha sido alterado de manera no autorizada.
- **Autenticidad:** Propiedad o característica consistente en que una entidad es quien dice ser o bien que garantiza la fuente de la que proceden los datos.
- **Disponibilidad:** Propiedad o característica de los activos, consistente en que las entidades o procesos autorizados tienen acceso a los mismos cuando lo requieren.
- **Confidencialidad:** Propiedad o característica consistente en que la información ni se pone a disposición, ni se revela a individuos, entidades o procesos no autorizados.
- **Trazabilidad:** Propiedad o característica consistente en que las actuaciones de una entidad pueden ser imputadas exclusivamente a dicha entidad.

1.3.2 Principales vulnerabilidades en aspectos de seguridad en las aplicaciones web

Muchas veces los desarrolladores descuidan los aspectos básicos de seguridad en la implementación de la aplicación condicionados por la premura de tiempo para el desarrollo de los sistemas y enfrascados en resolver las exigencias del cliente. Con el objetivo de crear conciencia sobre la seguridad en aplicaciones web, el Proyecto Abierto de Seguridad en Aplicaciones Web (OWASP por sus siglas en inglés) genera una lista de vulnerabilidades que requieren inmediata solución en un sistema web, estas vulnerabilidades son priorizadas de acuerdo con el nivel de explotación, detección e impacto estimado. A continuación se muestra la lista de las primeras cinco vulnerabilidades identificadas en el 2013 : (21)

- **Fallos de Inyección**

Los fallos de inyección, tales como SQL y LDAP (Protocolo Ligero de Acceso a Directorios), ocurren cuando datos no confiables son enviados a un intérprete como parte de un comando o consulta. Los datos hostiles del atacante pueden engañar al intérprete y ejecutar comandos no intencionados o acceder a datos sin autorización.

- **Pérdida de Autenticación y Gestión de Sesiones**

Las credenciales de cuentas y los testigos de sesión (sesión token) con frecuencia no son protegidos adecuadamente. Los atacantes obtienen contraseñas, claves, o testigos de sesión para obtener identidades de otros usuarios. Cross-Site Scripting (XSS): Ocurre cuando existe validación pobre de la información ingresada por el atacante.

- **Referencia Insegura y Directa a Objetos**

Una referencia directa a objetos ("direct object reference") ocurre cuando un programador expone una referencia hacia un objeto interno de la aplicación, tales como un fichero, directorio, registro de base de datos, o una clave tal como una URL o un parámetro de formulario Web. Un atacante podría manipular este tipo de referencias en la aplicación para acceder a otros objetos sin autorización.

- **Configuración defectuosa de seguridad**

Una buena seguridad requiere tener definida e implementada una configuración segura para la aplicación, marcos de trabajo, servidor de aplicación, servidor web, base de datos, y plataforma. Todas estas configuraciones deben ser definidas, implementadas, y mantenidas ya que por lo

general no son seguras por defecto. Esto incluye mantener todo el software actualizado, incluidas las librerías de código utilizadas por la aplicación.

- **Exposición de datos sensibles**

Las aplicaciones Web casi nunca utilizan adecuadamente funciones criptográficas para proteger datos y credenciales. Se refiere a la protección incorrecta de datos críticos tales como, números de tarjetas de crédito, contraseñas, entre otros.

1.3.3 Seguridad en Symfony2

El sistema de seguridad de Symfony2 se basa en identificar primero al usuario (autenticación) y comprobar después si ese usuario tiene acceso al recurso solicitado (autorización).

Firewalls (autenticación)

El sistema de seguridad de Symfony2 se activa cuando un usuario hace una petición a una URL que está protegida por un firewall o cortafuegos. El trabajo del firewall consiste en determinar si el usuario necesita estar autenticado, y si lo necesita, enviar una respuesta al usuario para iniciar el proceso de autenticación (22).

Un firewall se activa cuando la URL de una petición entrante concuerda con el valor de su opción de configuración pattern. En este ejemplo el valor de pattern (^/) concuerda con cualquier petición entrante. No obstante, el hecho de que el firewall esté activado no significa que el navegador muestra la caja de login+contraseña para todas las URL. Los usuarios pueden acceder por ejemplo a /foo sin que la aplicación les pida que se autentiquen.

Este funcionamiento es posible en primer lugar porque el firewall permite el acceso a los usuarios anónimos debido a la opción de configuración anonymous. En otras palabras, el firewall no exige que todos los usuarios se autentiquen nada más acceder a la aplicación. Y como en la configuración de la sección access_control no se indica que los usuarios deban tener ningún rol especial para acceder a /foo la petición se procesa sin requerir al usuario que se autentique.

Si se elimina la opción *anonymous*, el efecto es que ahora el firewall pide autenticación a cualquier usuario que solicite cualquier recurso (22).

Control de acceso (autorización)

Siguiendo con el mismo ejemplo, si un usuario solicita /admin/foo, la aplicación se comporta de manera diferente. Esto es debido a la configuración de la sección access_control, que indica que cualquier URL que coincida con la expresión regular ^/admin (es decir, la URL /admin o cualquier otra URL que coincida con /admin/*) requiere el rol ROLE_ADMIN. Los roles son la clave del

sistema de autorización: el usuario puede acceder a /admin/foo sólo si cuenta con el rol ROLE_ADMIN (22).

1.4 Sistemas de gestión de seguridad existentes en el mundo y en la UCI

A continuación se describen una serie de sistemas cuya solución se relaciona con la seguridad en las aplicaciones web. Se valora cada uno de estos sistemas, teniendo en cuenta su posible utilización en el diseño de la solución propuesta.

1.4.1 Aduana

El Departamento de Soluciones para la Aduana desarrolló un Mecanismo de Control de Acceso y Autenticación para Aplicaciones en Symfony, el cual es un componente con estructura de plugin. En este sistema el control de acceso se maneja a nivel de usuarios y roles.

El sistema interactúa con los sistemas que estén instalados sobre el mismo proyecto, en el mismo servidor de aplicaciones de las funcionalidades a las que controla la seguridad. Está almacenado en forma de plugin en el proyecto, permitiendo el acceso de sus funcionalidades a las aplicaciones que estén bajo el dominio del mismo proyecto. Para el registro de los subsistemas y sus funcionalidades, el sistema carga la estructura de cada aplicación con sus respectivos módulos y acciones, que se encuentran en un archivo de configuración YAML (23).

El módulo de seguridad para la Aduana brinda un mecanismo de seguridad para garantizar el control de acceso en sus sistemas. En dicho módulo no se incluye la gestión del perfil de los usuarios, funcionalidad requerida por el departamento de Tecnologías para la manipulación de usuarios con distintas características. Además fue implementado haciendo uso de una versión inferior al marco de trabajo a utilizar, que incluye cambios significativos en los paradigmas de desarrollo y en la estructura de las aplicaciones creadas, dificultando acciones de migración y/o reutilización.

1.4.2 Acaxia

La seguridad de las aplicaciones del Sistema de Gestión de Entidades Cedrux, perteneciente al Centro de Informatización de la Gestión de Entidades, se gestiona mediante el sistema Acaxia, desarrollado sobre el marco de trabajo Sauxe. Acaxia gestiona la seguridad en un entorno de varias aplicaciones y garantiza temas tan importantes como:

- Autenticación.
- Autorización.
- Auditoría.

- Administración de perfiles.
- Administración de conexiones.
- Entornos multi-entidad.
- Entornos multi-lenguaje.
- Entornos multi-temas.
- Compartimentación de la información.

Brinda tres tipos de integración para reutilizar todas las ventajas que ofrece:

- Integración a nivel de interfaz. Esta forma permite mostrar diferentes aplicaciones desarrolladas en distintas plataformas web en el portal principal.
- Integración a nivel de servicios internos. Para la integración mediante esta forma las aplicaciones deben estar desarrolladas sobre el mismo marco de trabajo, pudiendo así consumir sus servicios.
- Integración a nivel de servicios web. Este tipo de integración permite a sistemas de arquitecturas diferentes, comunicarse con Acaxia mediante servicios web.

  Brinda sus servicios a todos los sistemas que se suscriban a él. Para ello se gestionan las conexiones a la base de datos, las funcionalidades asociadas y las acciones que realizan las mismas.

El módulo de seguridad Acaxia es flexible, adaptable y genérico, además cumple con las características deseadas, pero de coexistir con Symfony2 implicaría problemas con el rendimiento y atentaría además contra el volumen de datos de las aplicaciones creadas. A pesar de las limitantes antes mencionadas se utilizará como guía para el desarrollo de la solución.

1.4.3 FOSUserBundle

Facilita la gestión de los usuarios en las aplicaciones desarrolladas en Symfony2. Se encarga de realizar tareas muy comunes en las aplicaciones que manejan usuarios y que Symfony2 por el momento no soporta. Entre otras, simplifica el registro de usuarios (incluyendo el envío opcional de un email de confirmación) y la opción "olvidé mi contraseña". Compatible con Doctrine, Propel, y MongoDB/CouchDB (24).

1.4.4 SonataAdminBundle

Genera automáticamente la parte de administración de las aplicaciones. Se integra perfectamente con Doctrine2. Su objetivo es convertirse en el "admingenerator" oficial de Symfony2.

El funcionamiento del bundle no se basa en archivos de configuración YAML sino en definir una clase de tipo Admin para cada entidad que quieras administrar. Dentro de esa clase, se añaden métodos configureXXXFields() para definir qué propiedades de la entidad se ven en cada página de administración (24).

1.4.5 Valoración de los sistemas estudiados

Symfony2 cuenta con un mecanismo de seguridad sencillo y eficiente, que nutre como base a otros sistemas desarrollados en Symfony2, como el FOSUserBundle y el SonataAdminBundle. Estos sistemas incluyen una serie de funcionalidades y configuraciones que contribuyen a fortalecer la seguridad de las aplicaciones desarrolladas en Symfony2. De esta manera disminuye el tiempo destinado a la implementación de la seguridad en las aplicaciones desarrolladas, pero tienen como limitante de que su integración con aplicaciones complejas se torna bastante engorroso. Además no soportan los entornos multi-dominio ni multi-sistema, imposibilitando la gestión de la seguridad de una manera centralizada como ACAXIA o el Módulo de seguridad para la Aduana.

1.4.6 Modelo de Desarrollo

Se emplea el Modelo de desarrollo definido por el centro CEIGE el cual permite identificar las actividades de cada una de las fases por las que se debe transitar el ciclo de vida del proyecto y el conjunto de artefactos a generar en cada una de ellas. Este modelo define varias disciplinas por cada una de las fases por las que transita.

Dichas fases se describen a continuación:

1. Inicio o Estudio preliminar

   En esta fase se realiza un estudio de factibilidad del proyecto que permite determinar si este tiene alcance, además de hacer un análisis para planificar y asignar los recursos necesarios para realizar el proyecto.
2. Desarrollo: Esta fase es la encargada de ejecutar las actividades requeridas para desarrollar el software, con el objetivo de obtener un sistema que satisfaga las necesidades por las que surgió el proyecto.

Disciplinas que componen los proyectos de desarrollo de software:

En la fase de desarrollo se definen los requisitos, se lleva a cabo el análisis y el diseño, se implementa y se libera la solución informática. Estos aspectos constituyen las disciplinas que se deben ejecutar en los proyectos de desarrollo de software (25).

A continuación se identifican las disciplinas que serán utilizadas para obtención de la solución esperada:

1. Modelado del Negocio: En esta fase se lleva a cabo un estudio que permite identificar y analizar los procesos que tendrán que ser llevados a cabo para la realización de la solución que se desea implementar.
2. Descripción de Requisitos: Esta disciplina permite obtener un modelo del software que se quiere construir. Genera un grupo de artefactos que permite determinar, administrar y desarrollar los requisitos de la solución.
3. Análisis y diseño: Es necesario contar con esta fase, pues en la misma se llevará a cabo el modelado del sistema, convirtiéndolo en un plano para la siguiente fase.
4. Implementación: Esta disciplina constituye la parte resultante del análisis y el diseño pues en ella se llevará a cabo la implementación del sistema a partir de la generación de componentes que se unirán para lograr la solución deseada.
5. Pruebas Internas: Se hace necesario realizarle pruebas al sistema resultante de la implementación para corregir posibles errores cometidos tanto en la documentación como en el software, por ello la importancia de esta fase.
6. Pruebas de Liberación: Para obtener la aceptación final del cliente se deben realizar pruebas al entregable del proyecto por una entidad externa capacitada para realizar dicha labor.

La realización de este trabajo se enfoca en darle cumplimiento a la fase de *Desarrollo* del modelo presentado, comprendiendo las disciplinas de Modelado, Descripción de requisitos, Análisis y diseño, Implementación y Pruebas internas.

1.5 Tecnologías utilizadas en el desarrollo de la propuesta de solución

1.5.1 PHP v5.4.12

PHP (Hypertext Preprocessor) es un lenguaje script (no se compila para conseguir códigos máquina si no que existe un intérprete que lee el código y se encarga de ejecutar las instrucciones que contiene éste código), para el desarrollo de páginas web dinámicas del lado del servidor, cuyos fragmentos de código se intercalan fácilmente en páginas HTML, debido a esto, y a que es de Open Source (código abierto), es el más popular y extendido en la web (26).

1.5.2 Symfony v2.3.5

Symfony es un marco de trabajo PHP que facilita el uso de la arquitectura MVC (Modelo-Vista-Controlador). Fue escrito desde un origen para ser utilizado sobre la versión 5 de PHP ya que

hace ampliamente uso de la orientación a objetos que caracteriza a esta versión y desde la versión 2 de Symfony se necesita mínimamente PHP 5.3.3 (27).

1.5.3 Twig v1.13

Es un motor de plantillas para el lenguaje de programación PHP. Su sintaxis se origina a partir de Jinja y plantillas de Django. El producto de código abierto se distribuye bajo licencia BSD y desarrollado por Fabien Potencier. El framework Symfony2 viene con un soporte incluido para Twig como su motor de plantillas por defecto.

Características:

- **Rápido**: Twig compila las plantillas a código PHP sencillo y optimizado. La sobrecarga en comparación con el ordinario de código PHP se ha reducido a la mínima expresión.
- **Seguro**: Twig tiene un modo de recinto para evaluar código de la plantilla que no es de confianza. Esto permite que Twig sea utilizado como un lenguaje de plantillas para aplicaciones donde los usuarios pueden modificar el diseño de la plantilla.
- **Flexible**: Twig es impulsado por un léxico flexible y analizador. Esto permite al desarrollador definir sus propias etiquetas y filtros personalizados, y crear su propio DSL (28).

1.5.4 Doctrine v2

Doctrine 2 es un ORM (Object-Relational Mapper) para PHP que incorpora por defecto Symfony2. Un ORM provee un sistema de persistencia para objetos PHP. Doctrine 2 se sitúa encima de una capa de abstracción de bases de datos (DBAL). La principal tarea del mapeador es traducir de forma trasparente los objetos PHP en filas de una base de datos relacional y viceversa. Una de las características claves es escribir consultas en un dialecto propio similar al SQL, llamado Doctrine Query Language (DQL), inspirado por HQL de Hibernates (29).

1.5.5 ExtJS v4.2

ExtJS es una biblioteca de JavaScript para el desarrollo de aplicaciones web interactivas que además de flexibilizar el manejo de componentes de la página como el DOM, Peticiones AJAX, DHTML, tiene la funcionalidad de crear interfaces de usuario bastante funcionales. Creado por Jack Slocum, Brian Moeskau, AaronConran, RichWaters (30).

Ventajas

- Una de las grandes ventajas de utilizar ExtJS es que nos permite crear aplicaciones complejas utilizando componentes predefinidos.

- Evita el problema de tener que validar el código para que funcione bien en cada uno de los navegadores (Firefox, IE, Safari, Opera etc.).
- El funcionamiento de las ventanas flotantes lo pone por encima de cualquier otro.
- Relación entre Cliente-Servidor balanceado: Se distribuye la carga de procesamiento, permitiendo que el servidor pueda atender más clientes al mismo tiempo.
- Eficiencia de la red: Disminuye el tráfico en la red, pues las aplicaciones cuentan con la posibilidad de elegir qué datos desea transmitir al servidor y viceversa (Criterio este que puede variar con el uso de aplicaciones de pre-carga).
- Comunicación asíncrona. En este tipo de aplicación el motor de render puede comunicarse con el servidor sin necesidad de estar sujeta a un clic o una acción del usuario, dándole la libertad de cargar información sin que el cliente se dé cuenta (30).

1.5.6 PostgreSQL v9.2

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD y con su código fuente disponible libremente. Es el sistema de gestión de bases de datos de código abierto más potente del mercado.

PostgreSQL utiliza un modelo cliente/servidor y usa multiprocesos en vez de multi-hilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando (31).

A continuación se muestran las características más importantes y soportadas por PostgreSQL:

- Es una base de datos 100% ACID
- Copias de seguridad en caliente (Online/hotbackups)
- Unicode
- Juegos de caracteres internacionales
- Regionalización por columna
- Acceso encriptado via SSL
- Completa documentación
- Llaves primarias y foráneas.
- Check, Unique y Notnullconstraints.
- Restricciones de unicidad postergables.
- Columnas auto-incrementales.
- Índices compuestos, únicos, parciales y funcionales en cualquiera de los métodos de almacenamiento disponibles, B-tree, R-tree, hash o GiST.
- Joins.

- Vistas (views).
- Disparadores (triggers) comunes, por columna, condicionales.
- Herencia de tablas (31).

1.5.7 Apache v2.2

Apache es un servidor de páginas web de código abierto, multiplataforma y modular, se desarrolla dentro del proyecto Servidor HTTP de la Fundación de Software Apache. Presenta entre otras características mensajes de error configurables, bases de datos de autenticación y negociado de contenido, pero fue criticado por la falta de una interfaz gráfica que ayude en su configuración. Se ha adaptado siempre a una gran variedad de entornos a través de su diseño modular, que permite a los administradores de sitios web elegir qué características van a ser incluidas en el servidor, y seleccionar qué módulos se van a cargar, ya sea al compilar o al ejecutar el servidor (32).

Características principales:

- Trabaja sobre múltiples plataformas (Unix, Linux, MacOSX, Win32, OS2).
- Incluye módulos que se cargan de forma dinámica.
- Soporta CGI, Perl, PHP.
- Posee soporte para bases de datos.
- Soporta SSL para transacciones seguras.
- Incluye soporte para host virtuales.
- Soporta HTTP 1.1.
- Es rápido y eficiente.
- Ventajas:
- Ayuda en la mejora del posicionamiento.
- Este servidor junto con el módulo mod_rewrite puede convertirse en una herramienta muy útil para adicionar páginas con enlaces amigables para los buscadores.
- Es un software libre.
- Open Source.
- Modular.
- Extensible.
- Presenta mensajes de error altamente configurables.

1.6 Herramientas utilizadas en el desarrollo de la propuesta de solución

1.6.1 NetBeans v7.3

El IDE NetBeans es una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java, pero puede servir para cualquier otro lenguaje de programación. Es un producto libre y gratuito sin restricciones de uso. Netbeans permite crear aplicaciones Web con PHP 5, posee un potente debugger integrado y además viene con soporte para Symfony2. Al tener también soporte para AJAX, es una muy buena opción para desarrollar aplicaciones LAMP o WAMP (33).

1.6.2 PgAdmin v3

Es una aplicación de diseño y manejo de bases de datos para su uso con PostgreSQL. La aplicación se puede utilizar para manejar postgreSQL 7.3 y superiores y funciona sobre casi todas las plataformas. Este software fue diseñado para responder a las necesidades de todos los usuarios, desde la escritura de simples consultas SQL a la elaboración de bases de datos complejas. La interfaz gráfica es compatible con todas las características de PostgreSQL y facilita la administración (34).

1.6.3 Lenguaje Unificado de Modelado

(UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados (35).

Es importante remarcar que UML es un "lenguaje de modelado" para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo.

Herramienta para el modelado, Visual Paradigm para UML v8.0

Visual Paradigm para UML es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software. Permite modelar todo tipo de diagramas UML, generar código desde diagramas, generar documentación, realizar ingeniería tanto directa como inversa, entre otras funciones. Este software permite crear modelos de gran calidad en poco tiempo y a un menor costo, y presenta una licencia gratis para uso no comercial.

Es multiplataforma y posee una interfaz amigable que permite un mejor desenvolvimiento por parte del equipo de desarrollo, que además ya se ha familiarizado con ella, pues tiene un papel protagónico en el estudio de la Ingeniería de Software como asignatura en la Universidad de las Ciencias Informáticas.

1.7 Conclusiones del capítulo

A lo largo del capítulo se conceptualizaron una serie de aspectos asociados a la seguridad en las aplicaciones web, así como diferentes tecnologías y herramientas que se utilizan para proveer dicha seguridad, lo cual permitió adquirir un mayor conocimiento sobre el objeto de estudio de la investigación.

Con el estudio y análisis de diferentes sistemas existentes en el mundo y en la UCI que llevan a cabo la gestión de la seguridad se evidenció la necesidad de desarrollar un módulo que permita gestionar la seguridad de las aplicaciones web implementadas sobre el marco de trabajo Symfony2.

El estudio de los lenguajes, tecnologías y herramientas, así como del modelo a utilizar en desarrollo de la solución, aportó un mayor conocimiento sobre los mismos, lo cual facilitó su utilización.

Capítulo 2: Análisis y diseño del componente

2.1 Introducción

En el presente capítulo se describen las características de la solución propuesta de manera que facilite un mayor entendimiento de la misma. De esta forma se describe la propuesta de solución para el Desarrollo del módulo de seguridad para el marco de trabajo Symfony2. Se definen los principales conceptos asociados al negocio, se describen y especifican los requisitos funcionales y no funcionales. Se realiza el diseño de las clases y se definen los patrones arquitectónicos y de diseño a utilizar.

2.2 Propuesta de solución

Una vez analizadas y valoradas las diferentes soluciones que de alguna manera llevan a cabo la gestión de la seguridad en aplicaciones web, se plantea el desarrollo de un módulo de seguridad para Symfony2 que soporte entornos multi-sistema y multi-dominios, para ello se deben gestionar las estructuras de los diferentes sistemas. Un sistema puede estar compuesto por subsistemas, funcionalidades y acciones. Los permisos sobre los diferentes sistemas, funcionalidades y acciones están agrupados por roles y estos a su vez deberán ser asignados a los usuarios en un conjunto de entidades pertenecientes a un dominio, de esta manera se establece un modelo RBAC para el control de acceso. Esta solución permitirá fortalecer la seguridad en aplicaciones web, brindando una mejor configuración de esta a los usuarios.

La siguiente **figura** representa el modelo RBAC a utilizar para el desarrollo de la solución.

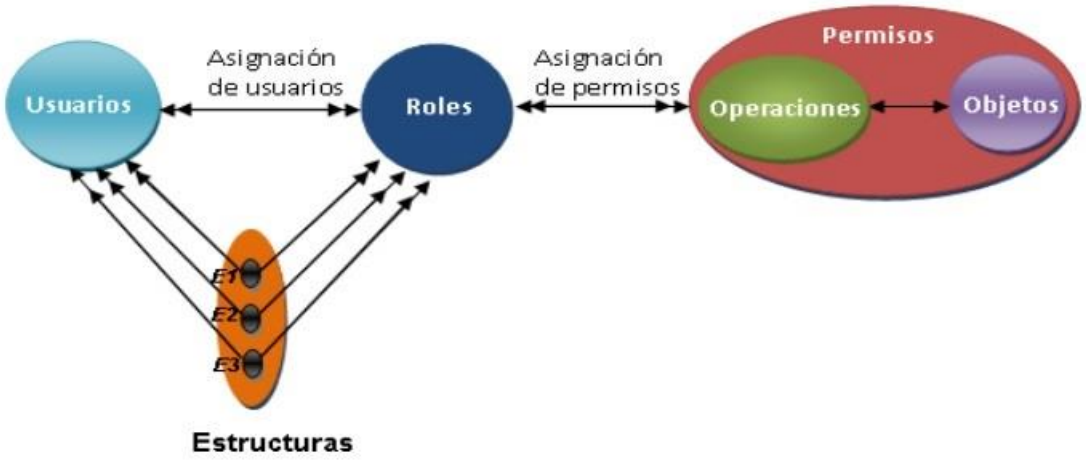


Figura 1 .Modelo RBAC extendido para entornos de multi-entidad.(2)

2.3 Modelación del Negocio

El objetivo del modelado del negocio es identificar y analizar los procesos que se llevan a cabo en el negocio que se desea obtener.

2.3.1 Modelo Conceptual

Un modelo conceptual comprende y describe las clases más importantes dentro del contexto del sistema.

2.3.1.1 Diccionario de datos

- **Acciones:** Actividad o acción concreta que realiza el usuario en el sistema.
- **Campos de perfil de usuario:** Define los atributos que va a contener el perfil de un usuario. Este concepto es configurable, es decir puede contener más atributos asociados.
- **Claves:** Representa los elementos que debe contener una contraseña para ser válida.
- **Dominio:** Se encarga de agrupar un conjunto de estructuras, estas estructuras pueden ser ministerios, agrupaciones, entidades, áreas, estructuras internas o cargos.
- **Entidades:** Forma parte de la estructura de un país y puede comportarse como un ministerio, entidad, área o cargo.
- **Expresiones:** Expresiones regulares para validar los campos del perfil de usuario.
- **Escritorios:** Representa los escritorios que va a tener el sistema asociado a un determinado usuario.
- **Funcionalidades:** Agrupa un conjunto de acciones del sistema.
- **Gestor de Base de Datos:** Es un conjunto de programas que permiten crear y mantener una base de datos, asegurando su integridad, confidencialidad y seguridad.
- **Idiomas:** Representa los idiomas que va a soportar el sistema.
- **Permiso:** Tipo de autorización que puede tener un rol o usuario sobre determinado recurso del sistema.
- **Perfil de usuario:** Contiene los datos personales asociados a un usuario. Los campos del perfil varían ya que este concepto es configurable.

- **Servidor:** Computadores que proporcionan servicios a las estaciones de trabajo de la red y donde se alojan los datos y la información asociada a un sistema o directorio activo.
- **Servidor de autenticación:** Define los parámetros a configurar para un servidor de autenticación.
- **Servidor de base de datos:** Define los parámetros a configurar para un servidor de base de datos.
- **Sistema:** Es el producto al que se le desea brindar seguridad.
- **Temas:** Representa representaciones visuales que va a tener el sistema asociado a un determinado usuario.
- **Usuario:** Cualquier persona que interactúa con el sistema y juega un rol determinado en el mismo.
- **Rol:** Agrupa una serie de permisos sobre sistemas, funcionalidades y acciones que se le asignarán a un conjunto de usuarios.

La siguiente **figura** representa el modelo conceptual del Módulo de Seguridad para el marco de trabajo Symfony2.

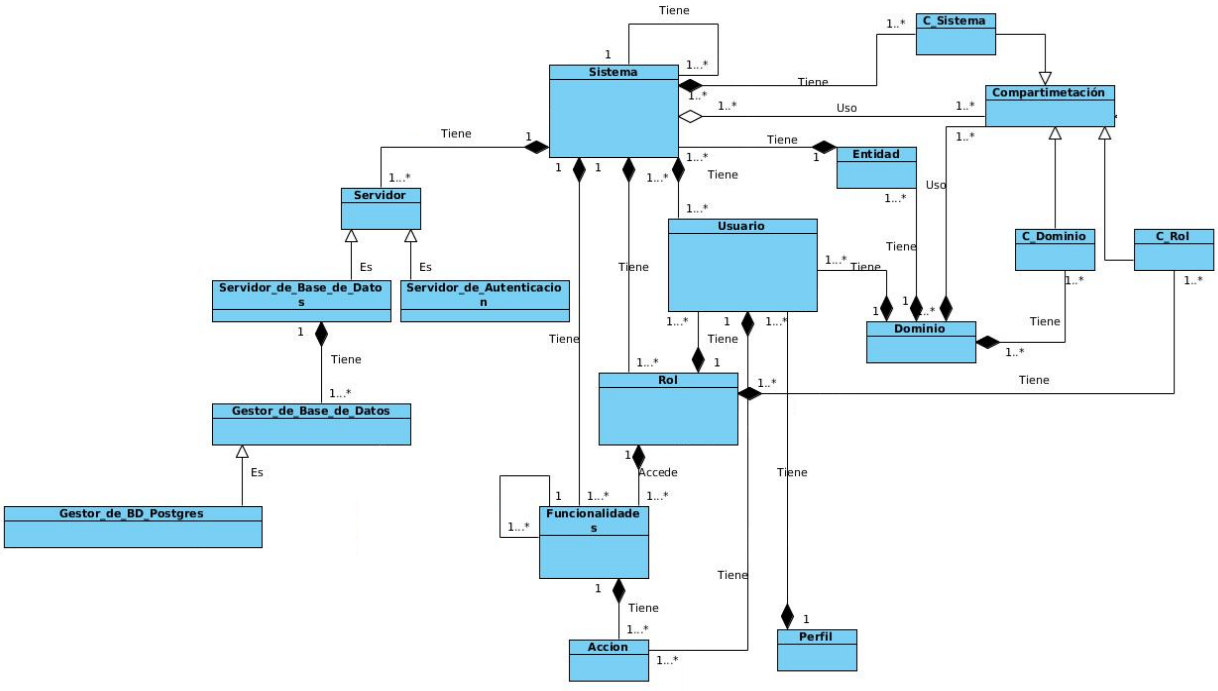


Figura 2. Modelo Conceptual del Módulo de Seguridad de Symfony2

2.4 Requisitos

Un requisito de software es una condición o capacidad que necesita un usuario para resolver un problema o lograr un objetivo. También puede verse como una condición o capacidad que tiene que ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato, estándar, u otro documento impuesto formalmente**. (IEEE: Standard Glossary of Software Engineering Terminology)**

2.4.1 Requisitos funcionales

Según Ian Sommerville: Los requisitos funcionales son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que éste debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares. En algunos casos estos pueden también declarar lo que el sistema no debe hacer (36).

A continuación queda reflejado el listado de los requisitos identificados para el desarrollo del Módulo de Seguridad para el marco de trabajo Symfony2.

2.4.2 Especificación de requisitos funcionales

Mediante la especificación de requisitos se obtiene una descripción completa del comportamiento de las funcionalidades que se van a desarrollar. Para su redacción se utiliza un lenguaje informal, de forma que sea fácilmente comprensible para todas las partes involucradas en el desarrollo.

En la tabla #1 se presenta la especificación del requisito funcional ***Adicionar servidor de base de datos***, los restantes requisitos se encuentran especificados en el Expediente de proyecto.

A continuación se muestra el listado de los requisitos funcionales:

- *Activar usuario*
- *Buscar usuario por dominio, denominación y estado.*
- *Desactivar usuario*
- *Asignar roles*
- *Buscar rol de usuario*
- *Cambiar contraseña*
- *Listar roles*
- *Buscar rol*
- *Regular acciones de un rol*
- *Registrar perfil de usuario*
- *Modificar perfil de usuario*
- *Listar perfiles de usuario*
- *Modificar campos de perfil*
- *Eliminar campos de perfil*
- *Listar campos de perfil*
- *Registrar acción*
- *Modificar acción*
- *Eliminar acción*

- *Eliminar usuario*
- *Listar roles*
- *Modificar usuario*
- *Registrar usuario*
- *Registrar sistema*
- *Modificar sistema*
- *Eliminar sistema*
- *Listar sistema*
- *Adicionar rol*
- *Modificar rol*
- *Eliminar rol*
- *Buscar perfil de usuario*
- *Listar clave*
- *Modificar clave*
- *Registrar clave*
- *Registrar funcionalidad*
- *Modificar funcionalidad*
- *Eliminar funcionalidad*
- *Buscar funcionalidad*
- *Listar funcionalidades*
- *Registrar campos de perfil*
- *Listar acción*
- *Adicionar servidor*
- *Modificar servidor*
- *Eliminar servidor*
- *Listar servidor*
- *Adicionar gestor*
- *Eliminar gestor*
- *Listar gestores*
- *Autenticar usuario*

Tabla 1. Especificación del requisito funcional **Adicionar servidor de base de datos**

Precondiciones	El usuario está autenticado y tiene los permisos necesarios para realizar esta acción.
Flujo de eventos	
Flujo básico Adicionar servidor de base de datos	
1	Se introducen los datos del servidor **de base de datos**: Servidor Dirección IP Tipo de servidor(base de datos) Descripción
2	El sistema valida (ver validación 1) los datos introducidos.

3	El sistema confirma el registro de los datos.
4	Concluye el requisito.
Pos-condiciones	
1	Se registró en el sistema un nuevo servidor.
Flujos alternativos	
Flujo alternativo 3.a Información errónea	
1	El sistema señala los datos erróneos y permite corregirlos.
2	El usuario corrige los datos.
3	Volver al paso 2 del flujo básico.
Pos-condiciones	
1	N/A
Flujo alternativo 3.b Información incompleta	
1	El sistema señala los datos vacíos y permite corregirlos.
2	El usuario corrige los datos.
3	Volver al paso 2 del flujo básico.
Pos-condiciones	
1	N/A
Flujo alternativo *.a El usuario cancela la acción	
1	Concluye el requisito.
Pos-condiciones	
1	No se registran los datos.
Validaciones	
1	

Conceptos	**Servidor**	Visibles en la interfaz: Denominación Dirección IP Tipo de servidor Descripción Utilizados internamente: N/A
	Servidor de Bases de Datos	Visibles en la interfaz: Denominación Dirección IP Tipo de servidor Descripción Utilizados internamente: N/A
Requisitos especiales	N/A	
Asuntos pendientes	N/A	

2.4.3 Requisitos no funcionales

Los requisitos no funcionales (RnF) son los requisitos que imponen restricciones al diseño o funcionamiento del sistema. A continuación se especifican los requisitos no funcionales que tienen un impacto directo en la solución:

Usabilidad

- RnF 1: El sistema podrá ser usado por cualquier persona que posea conocimientos básicos en el manejo de la computadora.

Restricciones del diseño

- RnF 2: El lenguaje de programación a utilizar para desarrollar el sistema debe ser php para la lógica del negocio y ExtJS para la capa de presentación.
- RnF 3: Las herramientas y tecnologías a utilizar para desarrollar el sistema deben ser PostgreSql como gestor de base de datos y Pgadmin III como cliente, Doctrine para la capa de acceso a datos y Symfony2 para la lógica del negocio.

- RnF 4: El sistema debe ser multiplataforma, haciendo énfasis en Linux y Windows.

Estándares aplicables

- RnF 5: El sistema debe utilizar el estándar RBAC para la estandarización del proceso de autorización.

Confiabilidad

- RnF 6: El sistema debe utilizar contraseña de acceso para llevar a cabo la autenticación del usuario.
- RnF 7: El sistema debe garantizar protección contra acciones no autorizadas o que puedan afectar la integridad de los datos.
- RnF 8: El sistema debe realizar verificaciones sobre las acciones irreversibles (eliminaciones).

Ambiente

- RnF 9: Un servidor Apache 2.4 o superior con módulo PHP 5.4 disponible, este debe estar configurado con la extensión “pgsql” incluida.

2.4.4 Diagrama de clases del diseño web

El diagrama de clases del diseño describe gráficamente las especificaciones de las clases de software y de las interfaces en una aplicación. En la figura que se muestra a continuación se presenta un diagrama de clases del diseño para la solución propuesta, el cual está basado en estereotipos web.

La *figura 3* muestra el diagrama de clases **gestionar sistema**, los restantes diagramas de diseño se encuentran especificados en el Expediente de proyecto.

2.4.5 Diagrama de secuencia

Los diagramas de secuencia muestran la interacción de un conjunto de objetos en una aplicación a través del tiempo. Se modelan para cada requisito funcional y contienen detalles de implementación del escenario, incluyendo los objetos y clases que se usan para implementar el mismo, además de los mensajes intercambiados entre los objetos. La *figura 4* muestra el diagrama de secuencia del requisito funcional ***Adicionar sistema***. Los demás se encuentran en el Expediente de proyecto.

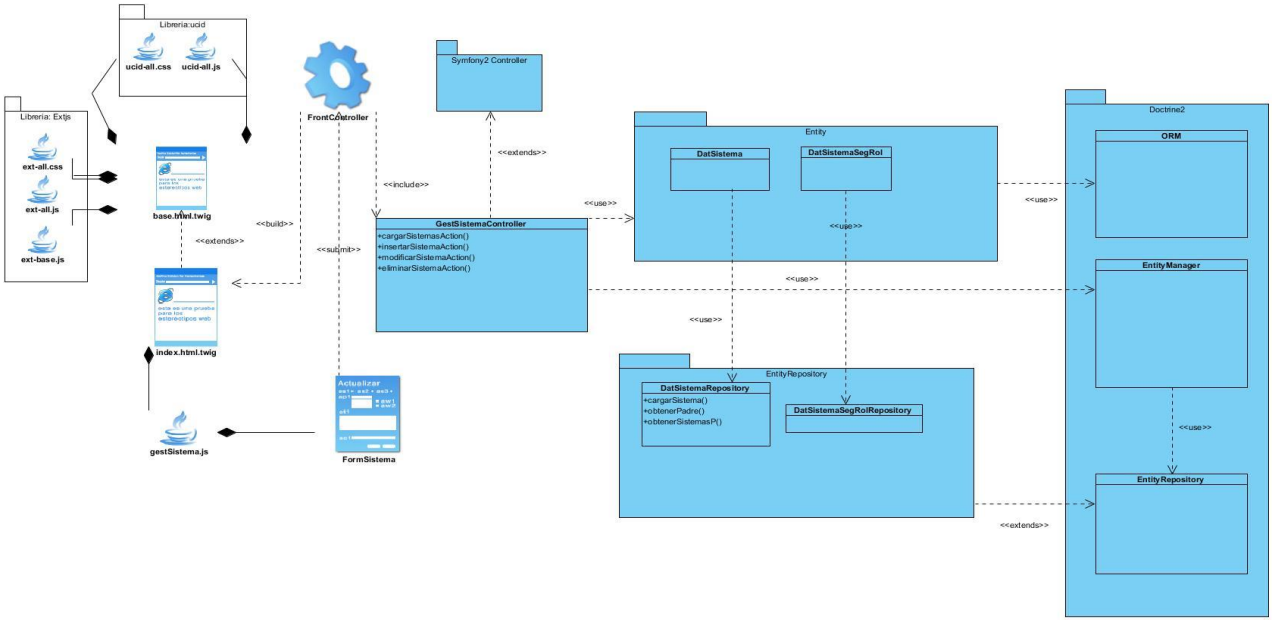


Figura 3. Diagrama de clases Gestionar Sistema

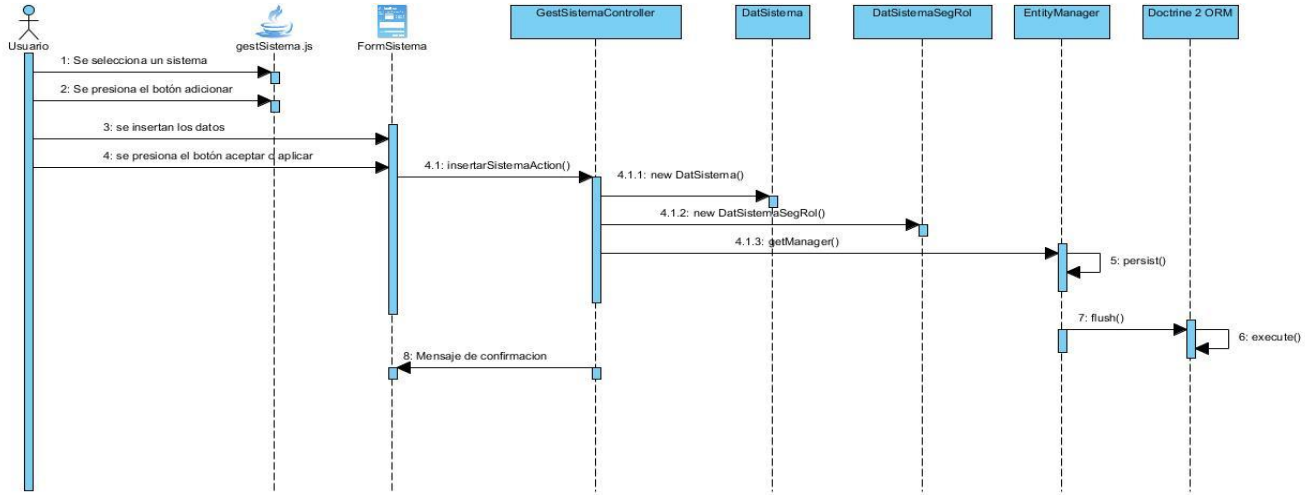


Figura 4. Diagrama de secuencia Adicionar Sistema

2.5 Patrones utilizados

2.5.1 Patrones de diseño

"Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias" (37).

Patrones GRASP

Para el diseño de la solución se tuvieron en cuenta los 5 patrones GRASP: Experto, Creador, Bajo acoplamiento, Alta cohesión y Controlador. El marco de trabajo busca un máximo rendimiento y flexibilidad en sus soluciones y pone en práctica estos patrones para lograr un sistema reusable y flexible.

- **Experto:** Las clases pertenecientes al paquete Entity poseen la información necesaria para cumplir con su responsabilidad. Algunas de estas clases son: *SegUsuario*, *DatSistema*,*DatServidor* y *DatFuncionalidad*.
- **Creador:** Este patrón es adaptable a la clase EntityManager, la cual es la encargada de crear consultas, para permitir el acceso a la información almacenada a nivel de datos.
- **Controlador:** La clase controladora definida: *GestUsuarioController*, es un ejemplo de la aplicación de este patrón, la misma tendrá a cargo la responsabilidad de manejar los eventos referentes a los usuarios dentro del componente.
- **Alta cohesión:** Este patrón fue utilizado en el diseño del componente de manera general; agrupando las clases en dependencia de los requerimientos, según la premisa de que la información almacenada en una clase debe ser coherente y estar relacionada con ésta en mayor medida, enfocada en sus responsabilidades.
- **Bajo acoplamiento:** Las clases se encuentran lo menos relacionadas entre sí, para que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión en las demás, potenciando la reutilización y disminuyendo la dependencia entre las clases.

Patrones GOF (Gang Of Four)

Los patrones GOF describen soluciones simples y elegantes a problemas específicos en el diseño de software. Se clasifican en 3 grandes categorías basadas en su propósito: creacionales, estructurales y de comportamiento.

En la solución propuesta fue utilizado el siguiente patrón GOF:

- **Fachada (Estructural):** Este patrón proporciona una interfaz unificada de alto nivel para un subsistema, que oculta las interfaces de bajo nivel de las clases que lo implementan simplificando así la interacción con el subsistema. Se utiliza para proporcionar un fácil acceso a subsistemas complejos. En el diseño de la solución que se propone, la clase ***Service Container*** es la que se utiliza como fachada, la cual provee Symfony2 para acceder a los servicios de otros componentes.

2.5.2 Patrón arquitectónico

Los patrones arquitectónicos definen la estructura de un sistema o software, los cuales a su vez están compuestos por subsistemas con sus responsabilidades, también tienen una serie de directivas para organizar los componentes del mismo sistema, con el objetivo de facilitar la tarea del diseño de tal sistema (38).

Modelo-Vista-Controlador (MVC)

Al diseñar las clases del componente se aplicó el patrón arquitectónico MVC. Éste provee a la solución de flexibilidad y facilidad a la hora de hacer futuros cambios.

El modelo está representado por las clases que se encuentran en el paquete ***Entity***.

La vista es la información presentada al cliente, a través de las páginas ***.html.twig*** y los ficheros ***.js.***

El controlador está representado por la clase que se encuentran dentro del paquete ***Controller*** que son las responsables de realizar las diferentes funcionalidades de la aplicación.

2.6 Diagrama de componentes

El diagrama de componentes muestra los elementos de diseño de un sistema de software, permitiendo visualizar con mayor facilidad la estructura general del sistema y el comportamiento de los servicios que estos componentes proporcionan y utilizan. La propuesta de solución cuenta con una serie de componentes que interactúan entre sí. La ilustración siguiente muestra el diagrama de componentes elaborado.

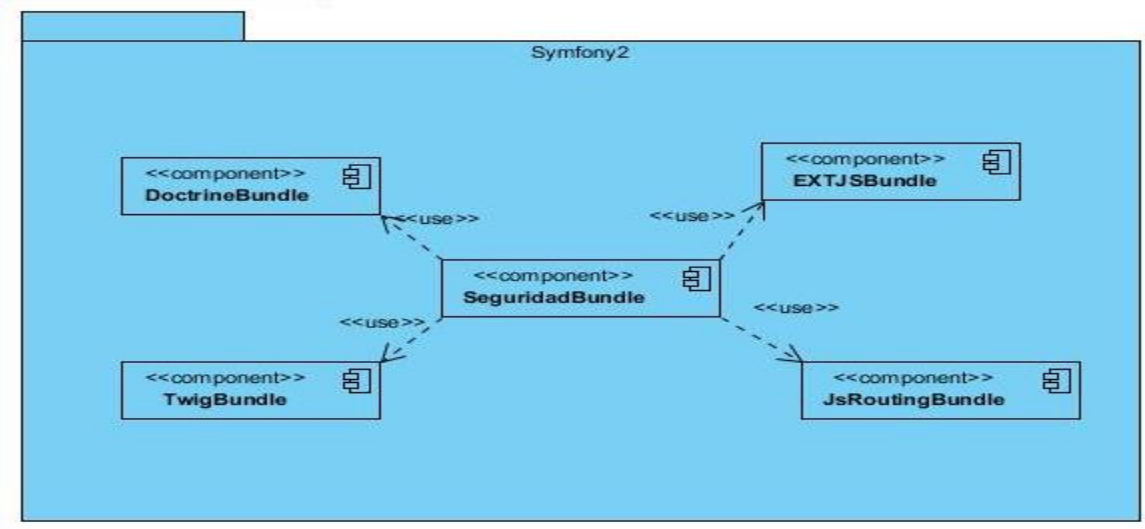


Figura 5. Diagrama de componentes

2.7 Modelo de datos

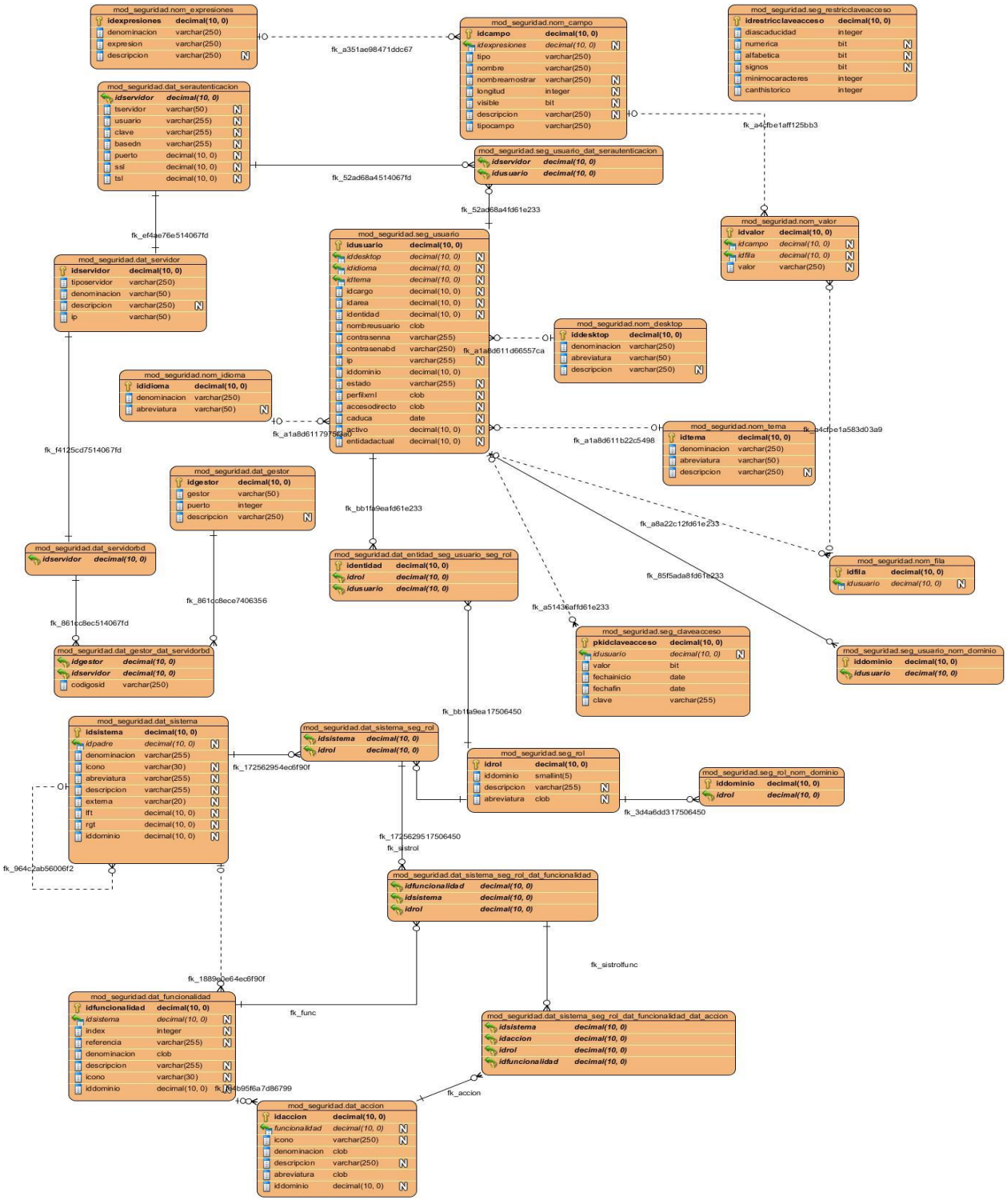


Figura 6. Modelo de datos del Módulo de Seguridad para Symfony2

2.8 Conclusiones del capítulo

Se realizó el análisis y diseño del componente, permitiendo un mejor entendimiento para la fase de implementación al tener los principales artefactos para el desarrollo. Se obtuvo como resultado los diagramas de clases del diseño para una mejor comprensión de cómo están estructuradas las clases que conforman la solución y la relación entre ellas, así como los patrones arquitectónicos y de diseño propuestos. Esto posibilita conocer las clases principales del negocio para luego diseñar el modelo de datos correspondiente a la solución propuesta. La aplicación del patrón Modelo- Vista Controlador permite un mejor flujo de información entre las capas de presentación, negocio y datos, lo que posibilitó separar cada una de estas capas para que cumplan con sus funcionalidades independientemente.

Capítulo 3 Implementación y prueba.

3.1 Introducción

En el presente capítulo se abarcan los elementos referentes a la implementación de la solución. Se define la estructura física de la misma, la definición de los estándares de codificación de las clases, funcionalidades y comentarios. Se valida el diseño y la implementación de la solución mediante métricas y pruebas de caja blanca y caja negra.

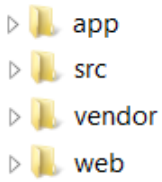
3.2 Implementación de la solución

La implementación de la solución comienza con el resultado alcanzado del diseño y proporcionará como resultado un producto que cumpla las exigencias y necesidades existentes en el departamento de tecnología del centro CEIGE.

En los próximos puntos se mostrará la estructura física de la solución, las métricas utilizadas para validar el diseño y las pruebas realizadas a la solución para su validación en la implementación.

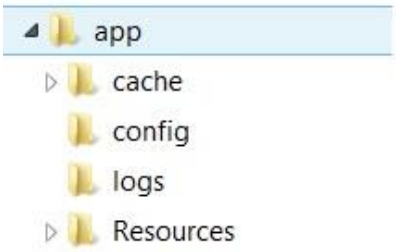
3.2.1 Estructura física de la solución

La estructura física de la solución para el desarrollo del módulo de seguridad del marco de trabajo Symfony2 se rige por la estructura definida para dicho marco.



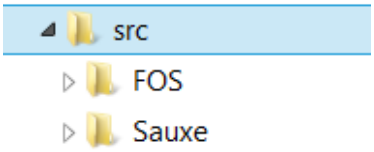
A continuación se presenta la especificación de la estructura de la solución para almacenar y recuperar la información perteneciente al Subsistema de Seguridad.

Dentro de la carpeta raíz de Symfony2 se encuentra la carpeta **app/** la cual aloja los scripts PHP encargados de los procesos de carga del marco de trabajo (lo que se conoce como **bootstraping**) y a todo lo que tenga que ver con la configuración general de la aplicación. Los archivos de este directorio son los encargados de unir y dar cohesión a los distintos componentes del marco de trabajo.



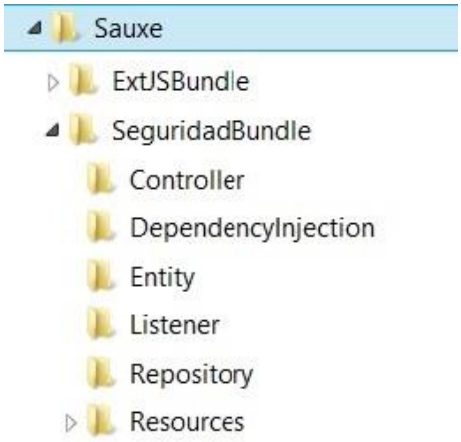
Dentro de la carpeta **app/** se encuentra la carpeta **cache/** en donde se aloja toda la información de los distintos entornos. La carpeta **config/** contiene los archivos de configuración de la aplicación, la conexión a la base de datos, las rutas y la seguridad. La carpeta **logs/** contiene los registros de la aplicación y por último, la carpeta **Resources/** incluye una carpeta **views/,** donde se configura la plantilla base de la aplicación.

Otra de las carpetas que se encuentran en el directorio raíz es la carpeta **src/.** Esta es la carpeta en donde irá todo el código y es aquí donde residen los Bundles, que básicamente son carpetas que representan nuestras aplicaciones.



Dentro de la carpeta **src/** se encuentra la carpeta **FOS/** la cual contiene el bundle denominado **JsRoutingBundle/** cuyo objetivo es simplificar el mecanismo de ruteo.

La otra carpeta que reside en **src/** es la carpeta **Sauxe/,** la cual refleja el nombre del sistema y va a contener todos los bundles generados para desarrollar la solución. Dichos bundles se muestran a continuación:

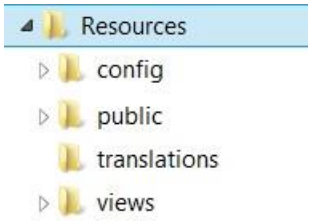


La carpeta **ExtJsBundle/** contiene las librerías de ExtJs 4.2, cuyo objetivo es utilizar los componentes que brindan dichas librerías para el desarrollo e integración de la capa de presentación con Symfony2.

La carpeta **SeguridadBundle/** contiene toda la lógica del negocio, a continuación se explica la estructura interna del bundle:

- **Controller/:** contiene los controladores del bundle, clases encargadas de gestionar las funcionalidades del sistema.
- **DependencyInjection/:** contiene elementos relacionados con el contenedor de inyección de dependencias, además contiene las extensiones para las clases de inyección de dependencias, la configuración que importan los servicios y registra uno o más pases del compilador.
- **Listener/:** contiene los listeners utilizados para el desarrollo de la solución, dichos listeners se utilizan en el mecanismo de autenticación para manipular determinados eventos.
- **Repository/:** contiene los repositorios del bundle, clases utilizadas para realizar consultas complejas para la obtención de información.

A continuación se muestra el conjunto de subcarpetas que contiene la carpeta **Resources**/:



Dentro de la carpeta **Resources/** se encuentra la carpeta **config/**, donde se realizan todas las configuraciones específicas del bundle. Otra de las carpetas es **public/,** donde se alojan las siguientes subcarpetas:

- **css/:** contiene las clases css necesarias para darle la estructura gráfica al formulario de autenticación, separando así el estilo del contenido.
- **Images/:** En esta carpeta se encuentran las imágenes e iconos que se utilizan en el desarrollo de la solución.
- **js/:** almacena por cada funcionalidad los ficheros JavaScript necesarios para que el usuario interactúe con el sistema.

La carpeta **translations/,** alberga las traducciones de los mensajes del inglés al español. Por último la carpeta **views/,** la cual contiene todas las plantillas Twig específicas del bundle por cada funcionalidad de la solución.

Las restantes carpetas que se encuentran en el directorio raíz se explican a continuación:

- **Vendor/:** En este directorio se alojan los bundles de terceros. Entre otras cosas, el directorio contiene los componentes de Symfony2, el ORM *Doctrine2* y el motor de plantillas Twig.
- **Web/:** En este directorio contiene el controlador frontal y todos los *assets (significado)* de la aplicación: CSS's, Javascripts e imágenes.

3.2.2 Estándares de codificación

Un estándar de código se basa en la organización y aspecto físico de un software con el objetivo de facilitar la lectura, entendimiento, mantenimiento del código, reutilización a lo largo del proceso de desarrollo y no en la lógica del programa.

Un estándar de programación no busca únicamente definir la nomenclatura de las variables, objetos, métodos y funciones, sino que además incluye reglas para el orden y legibilidad del código escrito. Teniendo como punto de partida lo anteriormente expuesto, se definen tres partes principales dentro de un estándar de programación:

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*, esta especifica que los identificadores y nombres de variables, métodos y funciones que están compuestos por múltiples palabras juntas, deben iniciar cada palabra con letra mayúscula, lo que posibilita que con sólo leerlo se reconozca el propósito de la misma.

Ejemplo: *SegUsuario* en este caso el nombre de clase está compuesto por dos palabras iniciadas cada una con letra mayúscula.

Nomenclatura según el tipo de clases:

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

Ejemplo: GestSistemaController

Repositorios: Las clases que se encuentran dentro de *Repository* después del nombre llevan la palabra: "Repository".

Ejemplo DatAccionRepository.

Entidades: Las clases que se encuentran dentro de *Entity* reciben el nombre de la tabla en la base de datos.

Ejemplo: SegRol.

Nomenclatura de las funcionalidades y atributos:

El nombre a emplear para las funciones y los atributos se escribe con la inicial del identificador en minúscula, en caso de que sea un nombre compuesto se empleará notación *CamelCasing* que es similar a la antes mencionada: *PascalCasing* con la excepción de la primera letra.

Ejemplo de método: *cargarUsuarios*. El nombre de método está compuesto por dos palabras, la primera en minúsculas y la siguiente iniciando con letra mayúscula.

Las principales funcionalidades de las clases controladoras se les asigna el nombre y seguidamente la palabra: "*Action".*

Ejemplo: *cargarUsuariosAction().*

Nomenclatura de los comentarios:

Los comentarios deben ser lo suficientemente claros y concisos para que se entienda el propósito de lo que se está desarrollando. En caso de ser una función complicada se debe comentar su interior para lograr una mejor comprensión del código.

3.3 Interfaces de usuario de la solución Módulo de seguridad para el marco de trabajo Symfony2.

A continuación se muestran diferentes vistas con las que podrá interactuar el usuario al hacer uso del Módulo de seguridad para el marco de trabajo Symfony2.

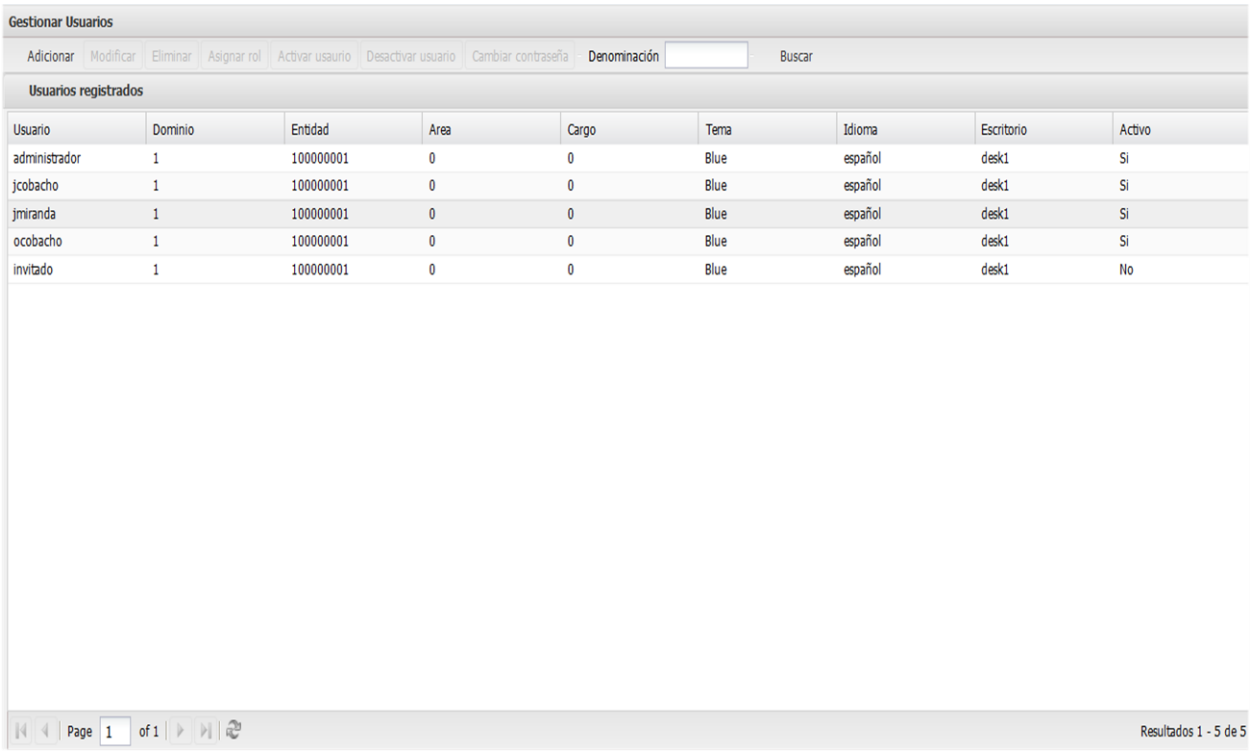


Figura 7. Interfaz de usuario: Gestionar usuario

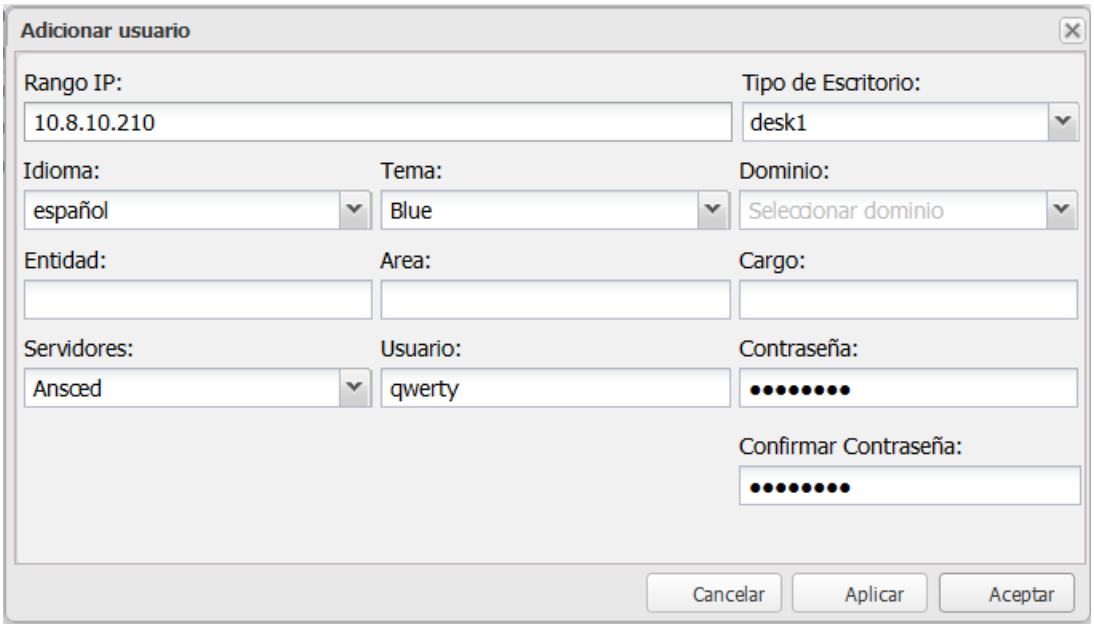


Figura 8. Interfaz de usuario: Adicionar usuario

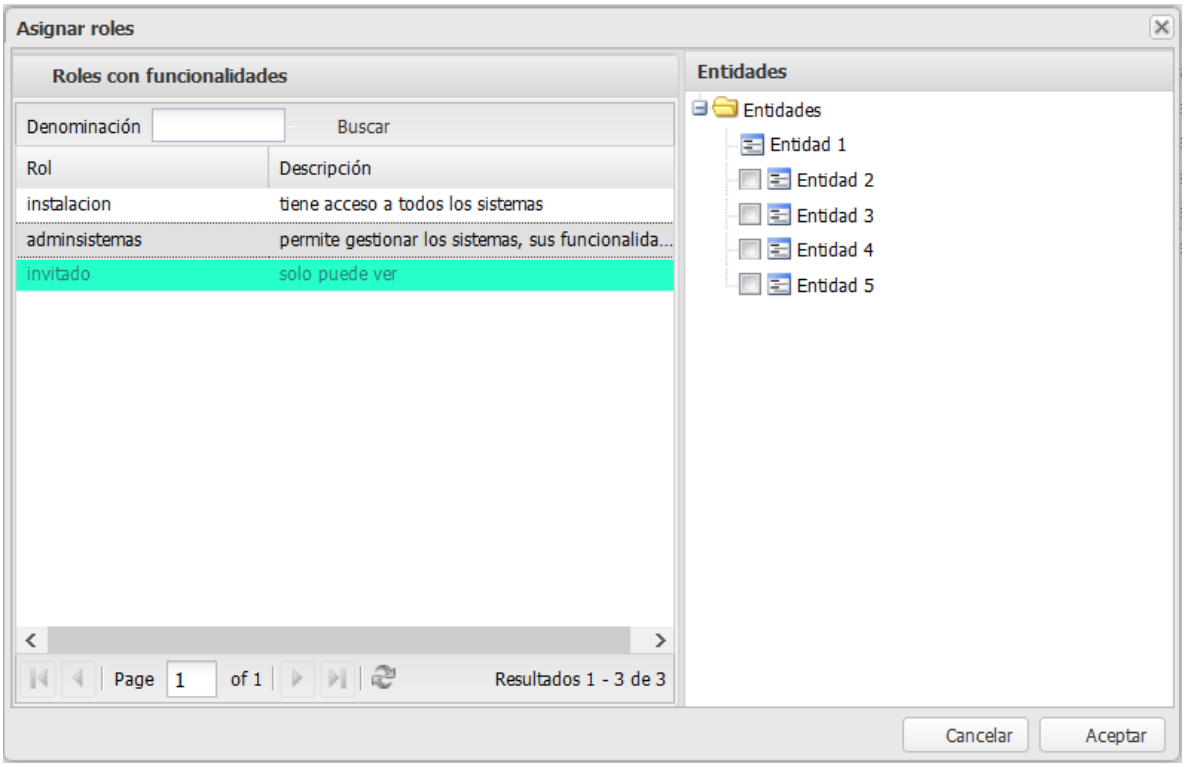


Figura 9. Interfaz de usuario: Asignar roles

3.4 Validación de la solución propuesta

A continuación se muestran los procedimientos empleados para realizar la validación del diseño propuesto. Se muestran las métricas de diseño utilizadas, las cuales permiten estimar la calidad del diseño de la solución obtenida. Posteriormente se especifican las pruebas de aplicación realizadas al mismo, pruebas que fueron hechas con el objetivo de revelar y corregir el máximo de errores posibles.

3.4.1 Validación del diseño propuesto

Para realizar la validación a las clases del diseño se seleccionaron las métricas Tamaño operacional de clases (TOC) y Relación entre clases (RC). Estas métricas validan que los patrones Alta Cohesión y Bajo Acoplamiento fueron bien utilizados.

La aplicación de las métricas TOC y RC contienen medidas de los siguientes atributos de calidad:

- **Responsabilidad**: consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.
- **Complejidad de implementación**: consiste en el grado de dificultad que tiene implementar un diseño de clases determinado.

- **Reutilización**: consiste en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.
- **Acoplamiento**: dependencia o interconexión de una clase o estructura de clase respecto a otras.
- **Cantidad de pruebas**: número o grado de esfuerzo necesario para realizar las pruebas de calidad al producto diseñado.
- **Complejidad del mantenimiento**: nivel de esfuerzo necesario para sustentar, mejorar o corregir el diseño de software propuesto. Puede influir significativamente en los costos y la planificación del proyecto.

Métrica TOC: Tamaño operacional de clase.

Métrica que describe el número de procedimientos (métodos) asignados a una clase. Además establece una relación entre los atributos Responsabilidad y Complejidad de implementación, sin embargo determina una relación inversa entre estos últimos y el atributo Reutilización.

Como resultado de la evaluación de la métrica TOC se obtuvo lo siguiente:

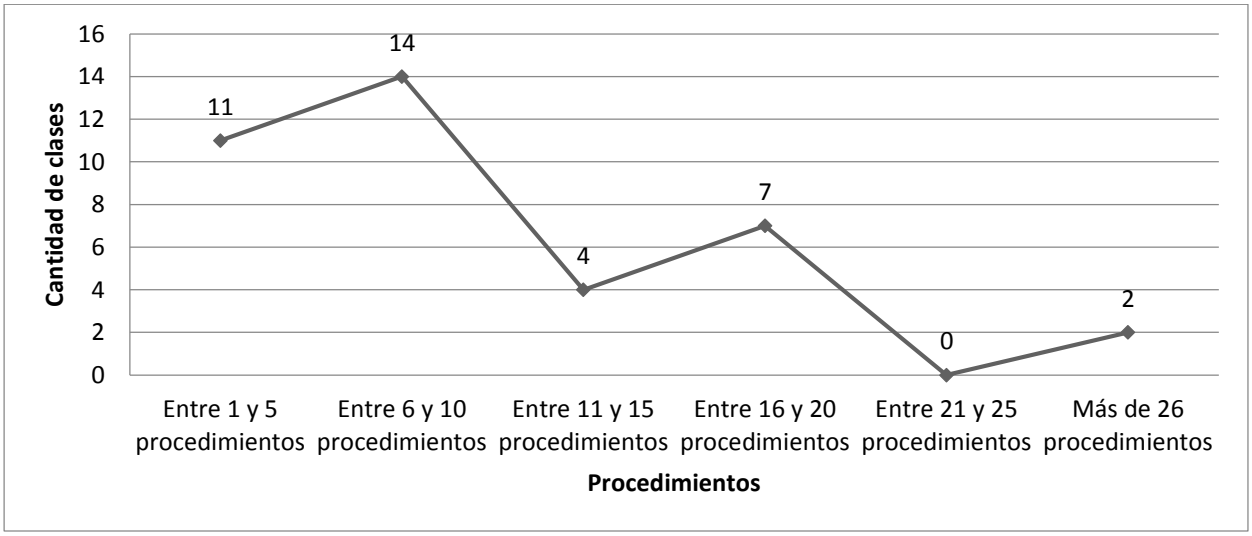


Figura 10. Representación de la cantidad de clase y el número de procedimientos que contienen.

Representación en porciento de los resultados obtenidos en el instrumento agrupados en los intervalos definidos:

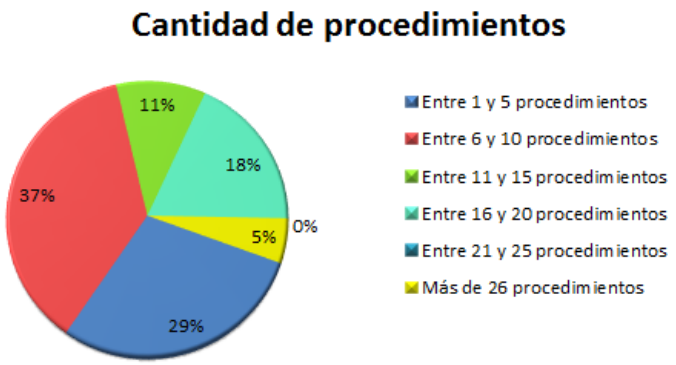


Figura 11. Representación en porciento de la cantidad de clase y el número de procedimientos que contienen.

Representación en porciento de la ocurrencia de los resultados obtenidos en el atributo **Complejidad de Implementación**:

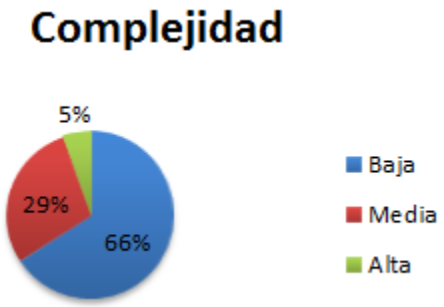


Figura 12. Representación del valor en porciento del atributo Complejidad de Implementación.

Representación en porciento de la ocurrencia de los resultados alcanzados en el atributo **Responsabilidad**:

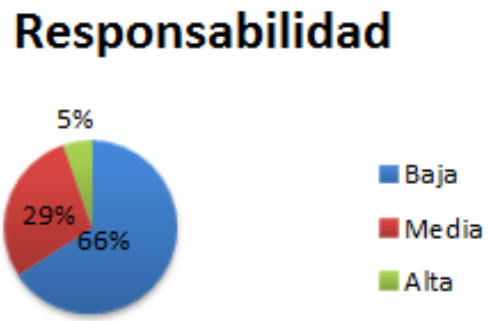


Figura 13. Representación del valor en porciento del atributo Responsabilidad.

Representación en porciento de la ocurrencia de los resultados alcanzados en el atributo **Reutilización**:

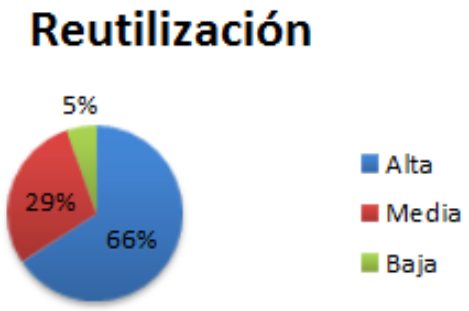


Figura 14. Representación del valor en porciento del atributo Reutilización.

Análisis de los resultados obtenidos en la evaluación de la métrica TOC

Luego de aplicarse la métrica de diseño **TOC** se puede afirmar que el diseño de clases elaborado se encuentra dentro de los límites de calidad, es aceptable teniendo en cuenta que el 66 % de las clases empleadas en el sistema poseen entre 1 y 10 operaciones.

Los atributos de calidad se encuentran en un nivel satisfactorio en el 66 % de las clases; de esta forma se puede observar cómo se fomenta la Reutilización (elemento clave en el proceso de desarrollo de software) y cómo están reducidas en menor grado la Responsabilidad y la Complejidad de implementación.

Métrica RC: Relaciones entre clases.

La métrica RC describe el número de relaciones de uso de una clase. Está determinada por los atributos: Acoplamiento, Complejidad de mantenimiento, Cantidad de pruebas y Reutilización existiendo una relación directa entre los tres primeros e inversa con el último antes mencionado. Al aplicarse la métrica Relación entre clases se obtuvo el siguiente resultado:

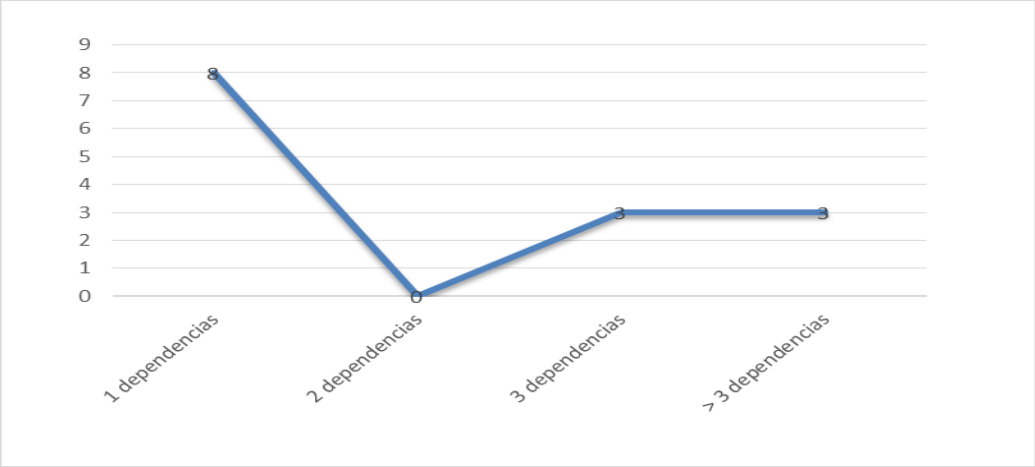


Figura 15. Representación de las asociaciones de uso por cantidad a de clases.

Representación en porciento de los resultados obtenidos en el instrumento agrupados en los intervalos definidos:

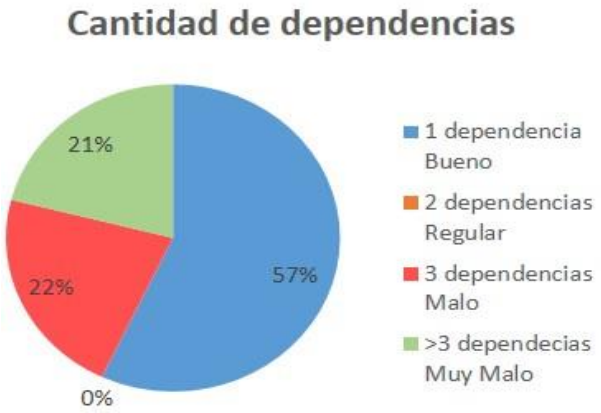


Figura 16. Representación en porciento de las asociaciones de uso por cantidad de clases.

Representación en porciento de la ocurrencia de los resultados obtenidos en el atributo **Acoplamiento**:

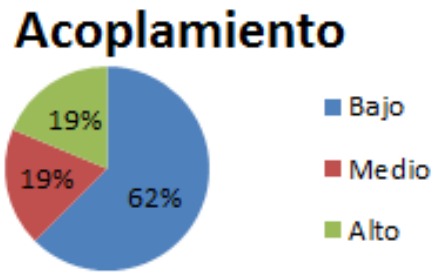


Figura 17. Representación del valor en porciento del atributo Acoplamiento.

Representación en porciento de la incidencia de los resultados obtenidos en el atributo **Complejidad de Mantenimiento**:

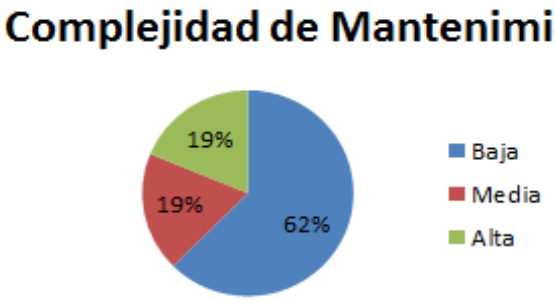


Figura 18. Representación del valor en porciento del atributo Complejidad de Mantenimiento.

Representación en porciento de la incidencia de los resultados obtenidos en el atributo **Cantidad de Pruebas.**

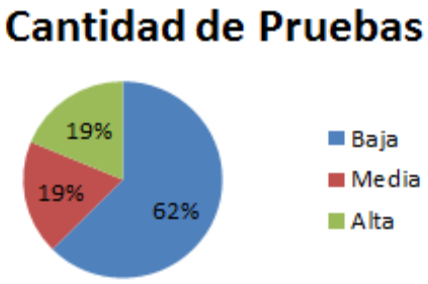


Figura 19. Representación del valor en porciento del atributo Cantidad de Pruebas.

Representación en porciento de la incidencia de los resultados obtenidos en el atributo **Reutilización**.

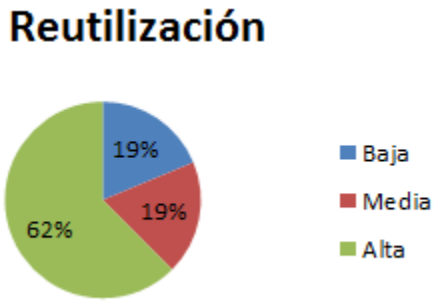


Figura 20. Representación del valor en porciento del atributo Reutilización.

Análisis de los resultados obtenidos en la evaluación de la métrica RC.

Luego de aplicarse la métrica de diseño RC se obtuvieron resultados que permiten evaluar el diseño propuesto con una calidad aceptable teniendo en cuenta que un 57 % de las clases utilizadas en el sistema poseen 1 dependencia, otro 22 % presentan 3 dependencias, y el otro 21 % restante presenta más de 3 dependencias.

Los atributos de calidad se encuentran en un nivel satisfactorio; ya que los atributos Complejidad de Mantenimiento, la Cantidad de Pruebas y la Reutilización se comportan favorablemente para un 62 % de las clases.

Diagrama de despliegue

A continuación se describe como queda distribuido físicamente el sistema entre los diferentes nodos de cómputo. Existen dos posibles escenarios para el despliegue de la solución: El escenario para PC cliente con disco y el escenario para PC cliente sin disco, también conocido como cliente ligero.

Figura 21. Diagrama de despliegue de escenario para PC cliente con disco.

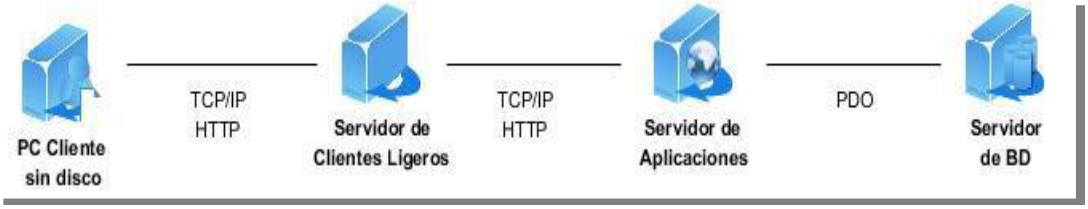


Figura 22. Diagrama de despliegue de escenario para PC cliente sin disco.

3.4.2 Pruebas de software

Las pruebas de software constituyen una etapa imprescindible durante el proceso de desarrollo del software. Su objetivo principal es asegurar que el software cumpla con las especificaciones requeridas y eliminar los posibles defectos que este pudiera tener. De forma más específica los propósitos de las pruebas son:

- Realizar la validación del software desarrollado, entendiendo como validación el proceso que determina si el software satisface los requisitos.
- Realizar la verificación del software desarrollado, entendiendo como verificación el proceso que determina si los productos de una fase satisfacen las condiciones de la misma.

Es importante considerar que las pruebas de software no garantizan que un sistema esté libre de errores, sino que se detecten la mayor cantidad de defectos posibles para su debida corrección (39).

Métodos de prueba:

Un método de prueba es un enfoque sistemático, independiente del nivel en que se enmarque la prueba, que ayuda a encontrar buenos conjuntos de casos de prueba para detectar diferentes tipos de errores.

- **Pruebas de caja blanca:** comprueban los componentes internos. Comprueba los caminos lógicos del software proponiendo casos de prueba que ejerciten condiciones específicas. Se

puede examinar el estado de la aplicación en varios puntos para determinar si el estado real coincide con el esperado o mencionado.

Este método de casos de prueba usa los detalles procedimentales de la aplicación. Se busca obtener casos de prueba que:

- Garanticen que se ejecuta por lo menos una vez todos los caminos independientes de cada módulo.
- Verificar las decisiones lógicas (V/F).
- Ejecutar las estructuras internas de datos para asegurar su validez.

Dentro del método de caja blanca se incluyen varias de pruebas tales como:

- **La prueba del camino básico:** permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución.

  Los pasos que se siguen para aplicar esta técnica son:

  a) A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.
  b) Se calcula la complejidad ciclomática del grafo.
  c) Se determina un conjunto básico de caminos independientes.
  d) Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

- **La prueba de condición:** es un método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa.
- **La prueba de flujo de datos:** se selecciona caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.
- **La prueba de bucles:** es una técnica de prueba de caja blanca que se centra exclusivamente en la validez de las construcciones de bucles.

➢ **Pruebas de caja negra:** comprueban las funcionalidades sin tener en cuenta la estructura interna. Se refiere a las pruebas que se llevan a cabo sobre la interfaz del software. O sea, a través de los casos de prueba se demuestra que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene.

Este tipo de pruebas permite encontrar:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores de rendimiento.

Dentro de la prueba de caja negra se incluyen varias técnicas de pruebas tales como:

- **Partición de equivalencia:** divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.
- **Análisis de valores límites:** prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.
- **Grafos de causa-efecto:** permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

Pruebas realizadas a la solución

Para verificar el correcto funcionamiento de la solución para la recuperación de la información se realizaron pruebas funcionales aplicando los métodos de caja negra, específicamente la técnica de partición de equivalencia y caja blanca con la técnica de camino básico.

A continuación se presenta el caso de prueba asociado a la prueba de caja negra generado al aplicar dicha técnica.

Tabla 2 Caso de prueba "Gestionar autenticación de usuario"

Nombre del requisito	Descripción general	Escenarios de pruebas	Flujo del escenario
Gestionar autenticación de usuario.	Permite la autenticación de un usuario a un sistema	EP 1.1: Gestionar autenticación de usuario insertando usuario y contraseña correctos.	1 El usuario realiza la petición de un recurso (sistema) utilizando un navegador web. 2 Se activa el proceso de autenticación. 3 Se muestra la ventana de autenticación. 4 El usuario introduce sus credenciales de autenticación (usuario y contraseña). 5 El sistema valida que el usuario exista en la base de datos y tenga algún rol asignado, que tenga asignada la contraseña especificada, que esté accediendo desde una dirección ip válida y que esté activo.

		6	Se crea la sesión y el certificado digital (token de seguridad).
		7	Se brinda acceso al recurso solicitado.
	EP 1.2: Gestionar autenticación de usuario insertando usuario o contraseña incorrectos	1	El usuario realiza la petición de un recurso (sistema) utilizando un navegador web.
		2	Se activa el proceso de autenticación.
		3	Se muestra la ventana de autenticación.
		4	El usuario introduce un usuario o contraseña inválido o deja algún campo vacío.
		5	El sistema valida que el usuario exista en la base de datos y tenga algún rol asignado, que tenga asignada la contraseña especificada, que esté accediendo desde una dirección ip válida y que esté activo.
		6	El sistema muestra un mensaje de error: Credenciales no válidas.
	EP 1.3: Usuario desactivado.	1	El usuario realiza la petición de un recurso (sistema) utilizando un navegador web.
		2	Se activa el proceso de autenticación.
		3	Se muestra la ventana de autenticación.
		4	El usuario introduce sus credenciales de autenticación (usuario y contraseña).

		5	El sistema valida que el usuario exista en la base de datos y tenga algún rol asignado, que tenga asignada la contraseña especificada, que esté accediendo desde una dirección ip válida y que esté activo.
		6	El sistema muestra un mensaje de error: “El usuario está deshabilitado, contacte al administrador”
	EP 1.4 Dirección ip denegada	1	El usuario realiza la petición de un recurso (sistema) utilizando un navegador web.
		2	Se activa el proceso de autenticación.
		3	Se muestra la ventana de autenticación.
		4	El usuario introduce un usuario o contraseña inválido o deja algún campo vacío.
		5	El sistema valida que el usuario exista en la base de datos y tenga algún rol asignado, que tenga asignada la contraseña especificada, que esté accediendo desde una dirección ip válida y que esté activo.
		6	El sistema muestra un mensaje de error: El usuario no puede acceder desde la dirección ip actual.

Véase Anexo 1. Juego de datos a probar.

Resultados de las pruebas de caja negra

Luego de aplicados los métodos de prueba a las funcionalidades implementadas se pudo concluir que los resultados obtenidos hasta el momento han sido satisfactorios desde el punto de vista funcional. Las no conformidades detectadas fueron debidamente atendidas en aras de lograr el correcto comportamiento de la solución desarrollada ante diferentes situaciones. A continuación se muestra la cantidad de no conformidades detectadas en cada iteración de pruebas realizada:

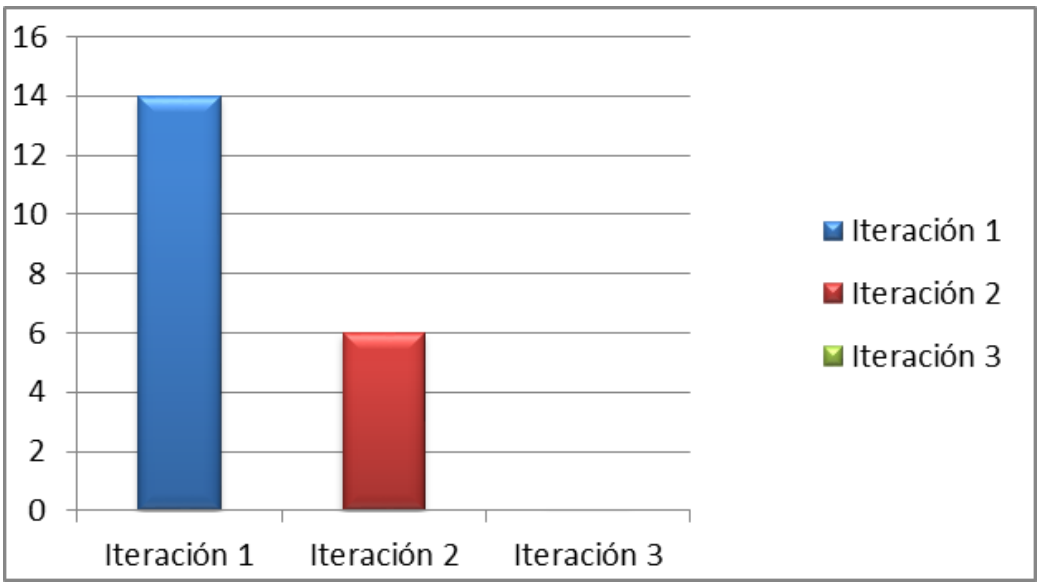


Figura 23. No conformidades por iteración detectadas durante las pruebas realizadas.

Prueba de caja blanca

La figura muestra las sentencias de código del método *modificarUsuarioAction(),* la cual se encuentra enumerada y posteriormente, el grafo de flujo correspondiente a este método.

```php
public function modificarUsuarioAction(){//1

        $request=$this->getRequest()->request;//1
        $em = $this->getDoctrine()->getManager();//1
        $usuario = $em->getRepository('SauxeSeguridadBundle:SegUsuario')->find($request->get('idusuario'));//1
              $temp1=$em->getRepository('SauxeSeguridadBundle:NomTema')->findOneBy(array('idtema'=>$request->get('idtema')));//1
        $temp2=$em->getRepository('SauxeSeguridadBundle:NomIdioma')->findOneBy(array('ididioma'=>$request->get('ididioma')));//1
        $temp3=$em->getRepository('SauxeSeguridadBundle:NomDesktop')->findOneBy(array('iddesktop'=>$request->get('iddesktop')));//1
        $usuario->setIp($request->get('ip'));//1
        $usuario->setIdentidad(request->get('entidad'));//1
        $usuario->setIdarea(request->get('area'));//1
        $usuario->setIdcargo(request->get('cargo'));//1

        $usuario->setIddominio($this->getUser()->getIddominio());//1
        $usuario->setIdtema($temp1);//1
        $usuario->setIdidioma($temp2);//1
        $usuario->setIddesktop($temp3);//1
        $idservidorauth=$request->get('idservidor');//1
        $usuario->setNombreusuario($request->get('nombreusuario'));//1
        $result=array('success' => false, 'message' => 'Existe un usuario que tiene esta denominación.')//1

        if(!existeNombreUsuario($request->get('nombreusuario'))){//2
             if($idservidorauth==$usuario->getIdservidor()->toArray()[0]->getIdservidor()->getIdservidor()){//3
                $em->flush();//4
                $result=array('success' => true, 'message' => 'El usuario fue modificado satisfactoriamente.') //4
              }
             else{//5
                  $servidor = $usuario->getIdservidor()->toArray();//6
                  if(count($servidor[0]->getIdservidor())){//7
                     $servidor[0]->removeIdusuario($usuario);//8
                    }
                  $servidor=$em->getRepository('SauxeSeguridadBundle:SegUsuario')->cargarServidorAutAdd($idservidorauth,$em);//9
                  $servidor[0]->addIdusuario($usuario);//9
                  $em->flush();//9
                  $result=array('success' => true, 'message' => 'El usuario fue modificado satisfactoriamente.')//9
                }
        }
       return new Response(json_encode(result));//10
    }
```

Figura 24. Método que modifica un usuario registrado en el sistema.

Grafo de flujo

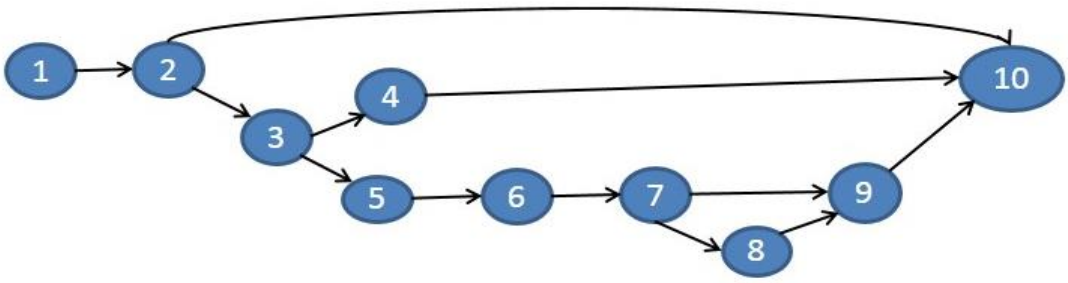


Figura 25. Grafo de flujo asociado al método.

Una vez construido el grafo de flujo asociado al procedimiento anterior se determina la complejidad ciclomática, la cual garantiza la menor cantidad de casos de pruebas a realizar, para que se ejecute al menos una vez cada sentencia de código. El cálculo de la complejidad ciclomática es necesario efectuarlo mediante tres vías o fórmulas, se debe utilizar el mismo grafo en cada caso:

Fórmula 1.V (G)= (A-N)+2 **A:** cantidad total de aristas del grafo**.**

Resultado. **N:** cantidad total de nodos del grafo.

V (G) = (12-10) + 2

V (G) =4

Fórmula 2.V (G)= P+1 **P:** cantidad total de nodos predicados.

Resultado. **Nota:** Un nodo es predicado cuando parten del mismo dos o más aristas.

V (G) = 3+1

V (G) = 4

Fórmula 3.V (G)=R **R:** cantidad total de regiones existentes en el grafo, se incluye el área exterior del grafo, como una región más.

Resultado V (G) = 4

Se especifican los caminos básicos que puede tomar el algoritmo durante su ejecución. Luego de calculada la complejidad ciclomática a través de la fórmulas, se concluye que existen cuatro caminos básicos. A continuación los caminos básicos:

Tabla 3 *Caminos básicos*

Número	Caminos básicos
1	1-2-3-4-10
2	1-2-3-5-6-7-8-9-10
3	1-2-3-5-6-7-9-10
4	1-2-10

Después de haber extraído los caminos básicos del flujo, se procede a ejecutar los casos de pruebas para este procedimiento. Para realizarlos es necesario cumplir con las siguientes exigencias:

Descripción: se hace la entrada de datos necesaria, validando que ningún parámetro obligatorio pase nulo al procedimiento o no se entre algún dato erróneo.

Condición de ejecución: se especifica cada parámetro para que cumpla una condición deseada para ver el funcionamiento del procedimiento.

Entrada: se muestran los parámetros que entran al procedimiento.

Resultados esperados: se expone el resultado que se espera que devuelva el procedimiento.

Resultados: se muestra el resultado obtenido.

***Tabla 4**. Caso de prueba para el camino básico #1*

Camino básico 1: **1-2-3-4-10**	
Descripción	Se modifica el usuario.
Condición de ejecución	Se selecciona el mismo servidor que tiene el usuario.
Entrada	$idservidorauth=24
Resultado esperado	El sistema muestra el siguiente mensaje “El usuario fue modificado satisfactoriamente.”
Resultado	Satisfactorio

***Tabla 5.**Caso de prueba para el camino básico #2*

Camino básico 2: **1-2-3-5-6-7-8-9-10**	
Descripción	Se modifica el usuario.
Condición de ejecución	Se selecciona un servidor diferente al actual.
Entrada	$idservidorauth=25
Resultado esperado	El sistema muestra el siguiente mensaje “El usuario fue modificado satisfactoriamente.”
Resultado	Satisfactorio

***Tabla 6**. Caso de prueba para el camino básico #3*

Camino básico 3: **1-2-3-5-6-7-9-10**	
Descripción	Se modifica el usuario.
Condición de ejecución	No se cambia el campo del servidor.
Entrada	$idservidorauth=26,

Resultado esperado	El sistema muestra el siguiente mensaje “El usuario fue modificado satisfactoriamente.”
Resultado	Satisfactorio**.**

Tabla 7. *Caso de prueba para el camino básico #3*

Camino básico 4: **1-2-10**	
Descripción	No se modifica el usuario.
Condición de ejecución	Se introduce un nombre de usuario ya asignado.
Entrada	$nombreUsuario=’administrador’
Resultado esperado	El sistema muestra el siguiente mensaje “Existe un usuario que tiene esta denominación.”
Resultado	Satisfactorio**.**

Resultados de las pruebas de caja blanca

Luego de aplicados los métodos de prueba internas se pudo concluir que de los cuatro diseños de casos de prueba aplicados, un 100 % fue satisfactorio, por lo cual queda evidenciado que los resultados obtenidos son satisfactorios.

3.5 Conclusiones del capítulo

El actual capítulo fue definitivo para precisar algunos elementos que intervienen en el desarrollo de la solución propuesta. De esta forma se presentó la estructura física de la solución, se definieron los estándares de código para su implementación y los elementos necesarios para su despliegue. Se aplicó del mismo modo las métricas: Relaciones entre clases y Tamaño operacional de clase, métrica que permitió evaluar especialmente la complejidad en la implementación de la solución en la actual investigación.

Se realizaron pruebas de caja negra y caja blanca para comprobar el correcto funcionamiento de la solución propuesta. Luego de realizar 3 iteraciones de pruebas se detectaron un total de 14 no conformidades en la primera iteración, en la segunda se detectaron 6 no conformidades, las cuales fueron resueltas permitiendo cumplir con los requisitos capturados en la primera fase de desarrollo.

Conclusiones Generales

El desarrollo de la reciente investigación y los resultados que generó, han permitido llegar a las siguientes conclusiones:

1. Mediante la elaboración del marco teórico y el estudio del estado del arte se evidenció la necesidad de desarrollar un módulo de seguridad para el marco de trabajo Symfony2 que soporte entornos multi-sistema y multi-dominio.
2. Mediante la elaboración de los artefactos definidos en el modelo de desarrollo del CEIGE, se realizó el análisis y diseño de la solución, teniendo en cuenta las necesidades del cliente.
3. Se desarrolló un módulo para el marco de trabajo Symfony2 que permite gestionar la seguridad. Además es flexible, adaptable y soporta entornos multi-sistema y multi-dominio.
4. Se validó la solución implementada a través de métodos pruebas de software efectuadas obteniéndose resultados positivos.

Recomendaciones

Al concluir el presente trabajo de diploma, considerando cumplidos los objetivos trazados en el mismo, se recomienda:

- Integrar el sistema con el módulo Portal para facilitar la navegación de la solución.
- Integrar el sistema con el módulo Trazas para fortalecer la seguridad de las aplicaciones desarrolladas sobre el marco de trabajo Symfony2.

Bibliografía

1. OINER GÓMEZ BARYOLO. *MODELO DE CONTROL DE ACCESO PARA SISTEMAS DE INFORMACIÓN EN ENTORNOS MULTIDOMINIOS*. 2012.

2. OINER GÓMEZ BARYOLO, YOEL HERNÁNDEZ MENDOZA and MILEIDY M. SARDUY PÉREZ. *Extensión del modelo RBAC para sistemas ERP en entornos multientidad*. 15 September 2011.

3. Definicion de Seguridad informática - ¿qué es Seguridad informática? [online]. [Accessed 16 June 2014]. Available from: http://www.alegsa.com.ar/Dic/seguridad%20informatica.php

4. JAVIER J. GUTIÉRREZ. ¿Qué es un framework web? [online]. [Accessed 8 March 2014]. Available from: http://www.lsi.us.es/~javierj/investigacion_ficheros/Framework.pdf

5. The bundle system. In : *Symfony the cookbook. [no date]*.

6. Glosario Informático, Letra R - Marcelo Pedra. [online]. [Accessed 3 June 2014]. Available from: http://www.marcelopedra.com.ar/blog/glosario-informatico-y-de-internet/glosario-informatico-letra-r/

7. Sistemas gestores de bases de datos. [online]. [Accessed 9 March 2014]. Available from: http://www.desarrolloweb.com/articulos/sistemas-gestores-bases-datos.html

8. ProfesorJava: Ambiente de Desarrollo Integrado (IDE). [online]. [Accessed 13 March 2014]. Available from: http://profejavaoramas.blogspot.com/2010/04/ambiente-de-desarrollo-integrado-ide.html

9. Lenguajes de programación. [online]. [Accessed 13 March 2014]. Available from: http://es.kioskea.net/contents/304-lenguajes-de-programacion

10. MOISÉS MAÑAS CARBONELL. Interfaz gráfica de usuario (GUI) | Fundéu BBVA. [online]. [Accessed 9 March 2014]. Available from: http://www.fundeu.es/escritoeninternet/interfaz-grafica-de-usuario-gui/

11. ¿Que es una URL? - Definición de URL. [online]. [Accessed 9 March 2014]. Available from: http://www.masadelante.com/faqs/url

12. Servicios Web - EcuRed. [online]. [Accessed 9 March 2014]. Available from: http://www.ecured.cu/index.php/Servicios_Web

13. Definicion de API - ¿qué es API? [online]. [Accessed 8 March 2014]. Available from: http://www.alegsa.com.ar/Dic/api.php

14. Plataforma Informatica. [online]. [Accessed 8 March 2014]. Available from: http://www.reddeaprendizaje.com/inicio/item/47-plataforma-informatica

15. Servidores Web - EcuRed. [online]. [Accessed 8 March 2014]. Available from: http://www.ecured.cu/index.php/Servidores_Web

16. Definicion de LAMP - ¿qué es LAMP? [online]. [Accessed 8 March 2014]. Available from: http://www.alegsa.com.ar/Dic/lamp.php

17. PHP: WampServer Definicion, Instalación y configuración ~ codegeando. [online]. [Accessed 8 March 2014]. Available from: http://codegeando.blogspot.com/2013/03/php-wampserver-definicion-instalacion-y.html

18. Virtualhost y múltiples dominios con Apache 2 y Debian 5 | Solo Bits. [online]. [Accessed 8 March 2014]. Available from: http://blog.mbonell.com/2011/08/virtual-host-y-multiples-dominios-con-apache-2-y-debian-5/#content

19. Multiplataforma (definición) ~ InformaticandoTf. [online]. [Accessed 8 March 2014]. Available from: http://informaticandotf.blogspot.com/2010/08/multiplataforma-definicion.html

20. Administración electrónica. - Centro de Publicaciones - Google Libros. [online]. [Accessed 15 June 2014]. Available from: http://books.google.com.cu/books

21. Top 10 2013-Top 10 - OWASP. [online]. 8 February 2014. [Accessed 8 February 2014]. Available from: https://www.owasp.org/index.php/Top_10_2013-Top_10

22. RYAN WEAVER, Fabien Potencier. Capítulo 13. Seguridad (Symfony 2.3, el libro oficial). LibrosWeb.es [online]. 9 February 2014. [Accessed 9 February 2014]. Available from: http://librosweb.es/symfony_2_x/capitulo_13.html

23. GARCIA, Adrian Naranjo. *Mecanismo de Control de Acceso y Autenticación para Aplicaciones en Symfony del Departamento de Soluciones para la Aduana.* Ciudad de la Habana : Universidad de las Ciencias Informáticas, 2010.

24. Bundles de Symfony relacionados con Seguridad. [online]. [Accessed 8 March 2014]. Available from: http://symfony.es/bundles/relacionados-con-seguridad

25. OBREGÓN, Ing William González. *Modelo de desarrollo CEIGE*.

26. Introducción, definición y evolución de PHP. ¿Qué es PHP? Definición de PHP. [online]. [Accessed 8 March 2014]. Available from: http://php.ciberaula.com/articulo/introduccion_php

27. Introducción a Symfony 2 |Maestros del Web. [online]. [Accessed 8 March 2014]. Available from: http://www.maestrosdelweb.com/editorial/curso-symfony2-introduccion-instalacion/

28. Twig - EcuRed. [online]. [Accessed 5 June 2014]. Available from: http://www.ecured.cu/index.php/Twig

29. Bases de datos | El rincón de un parásito virtual | Página 5. [online]. [Accessed 10 March 2014]. Available from: http://parasitovirtual.wordpress.com/category/cursos-y-articulos/bases-de-datos/page/5/

30. Sencha Ext JS - EcuRed. [online]. [Accessed 9 March 2014]. Available from: http://www.ecured.cu/index.php/Sencha_Ext_JS#Ventajas_y_Desventajas

31. Sobre PostgreSQL | www.postgresql.org.es. [online]. [Accessed 9 March 2014]. Available from: http://www.postgresql.org.es/sobre_postgresql

32. Análisis y Diseño de Sistemas :: Trabajo Nº 4. [online]. [Accessed 13 March 2014]. Available from: http://www.oocities.org/es/alex_28200/fase3/t4.html

33. Bienvenido a NetBeans y www.netbeans.org, Portal del IDE Java de Código Abierto. [online]. [Accessed 14 March 2014]. Available from: https://netbeans.org/index_es.html?print=yes

34. pgAdmin: PostgreSQL administration and management tools. [online]. [Accessed 4 June 2014]. Available from: http://www.pgadmin.org/

35. Lenguaje unificado de modelado. [online]. [Accessed 14 March 2014]. Available from: http://www.slideshare.net/DarwinGranda/lenguaje-unificado-de-modelado

36. SOMMERVILLE, Ian. *Ingeniería de Software*. 7. [no date].

37. Patrón de diseño. [online]. [Accessed 4 June 2014]. Available from: http://sourcemaking.com/patr%C3%B3n-de-dise%C3%B1o

38. isg3 / Patrones Arquitectónicos. [online]. [Accessed 4 June 2014]. Available from: http://isg3.pbworks.com/w/page/7624479/Patrones%20Arquitect%C3%B3nicos

39. CINTRA, Yordano Yunior Osorio. SISTEMA PARA LA GESTIÓN DEL BANCO DE PROBLEMAS DE LA UCI. September 2013.

40. ABNER GERARDO VALDEZ HUARACA, JORGE LUIS MENDOZA VALDEZ and CARLOS ORLANDO PACHAS LAURA. *Pruebas de Sistemas y Pruebas de Aceptación*. 2013.

Glosario de términos

Ajax:(Asynchronous JavaScript And XML) es una técnica para cargar datos o fragmentos de HTML sin refrescar la ventana del navegador. Esencialmente esta tecnología permite que el cliente JavaScript se comunique con el servidor utilizando el objeto XMLHttpRequest, presente en la mayoría de los navegadores modernos.

Base de datos: es una herramienta para recopilar y organizar información. En las bases de datos, se puede almacenar información sobre personas, productos, pedidos, o cualquier otra cosa.

Herramientas Case: conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del Ciclo de Vida de desarrollo de un Software (investigación Preliminar, Análisis, diseño, Implementación e Instalación).

JavaScript: es un lenguaje interpretado orientado a las páginas web, con una sintaxis semejante a la del lenguaje Java. También es utilizado en páginas web HTML, para realizar tareas y operaciones en el marco de la aplicación cliente.

Open source (Código abierto): Software de código abierto es software cuyo código fuente está disponible para su modificación o mejora por cualquier persona.

ORM (Object Relational Mapping): es una técnica de programación para convertir datos entre el lenguaje de programación orientado a objetos utilizado y el sistema de base de datos relacional utilizados en el desarrollo de nuestra aplicación.

SQL: StructuredQueryLanguage (lenguaje de consulta estructurado), es un lenguaje surgido de un proyecto de investigación de IBM para el acceso a bases de datos relacionales.

SSL: son las siglas en inglés de Secure Socket Layer (en español capa de conexión segura). Es un protocolo criptográfico (un conjunto de reglas a seguir relacionadas a seguridad, aplicando criptografía) empleado para realizar conexiones seguras entre un cliente (como lo es un navegador de Internet) y un servidor (como lo son las computadoras con páginas web).

Anexos

Anexo 1 .Juego de datos a probar.

Tabla 8. Juegos de datos a probar

Id del escenario	Escenario	Usuario	Contraseña	Respuesta del sistema	Resultado de la prueba
EP 1.1	Gestionar autenticación de usuario insertando usuario y contraseña correctos.	V[(administrador]	V[12345q]	Se brinda acceso al recurso solicitado.	.
EP 1.2	Gestionar autenticación de usuario insertando usuario o contraseña incorrectos	V[administrador]	I[vacío]	El sistema muestra un mensaje de error: Credenciales no válidas.	
		I[vacío]	I[vacío]		
		I[pepe]	I[*prueba12]		
EP 1.3	Usuario desactivado.	V[(jcobacho]	V[entrar123]	El sistema muestra un mensaje de error: “El usuario está deshabilitado, contacte al administrador.”	

EP 1.4	Dirección ip denegada	V[invitado]	V[12345q]	El sistema muestra un mensaje de error: “El usuario no puede acceder desde la dirección ip actual”	