Universidad de las Ciencias Informáticas

Facultad 3

Título: *Componente para la integración de los servicios de seguridad del Sistema Integral de Seguridad Acaxia con aplicaciones desarrolladas en Symfony2*

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

Autor(es): *Dannelis Melgarejo Pozo*

Ernesto Alonso Betancourt

Tutor(es): *Ing. Damián Pérez Alfonso*

Ing. René Bauta Camejo

Junio 2014

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 _____.

_____
Dannelis Melgarejo Pozo
Firma del Autor

_____
Ernesto Alonso Betancourt
Firma del Autor

_____
Ing. Damián Pérez Alfonso
Firma del Tutor

_____
Ing. René R. Bauta Camejo
Firma del Tutor

DATOS DE CONTACTO

Damián Pérez Alfonso.

Ingeniero en Ciencias Informáticas, graduado en el año 2010.

René Bauta Camejo.

Ingeniero en Ciencias Informáticas, graduado en 2009 en la Universidad de las Ciencias Informáticas. Instructor. Cinco años de experiencia en el desarrollo de software.

AGRADECIMIENTOS

De Dannelis:

Todo agradecimiento es pequeño comparado con el que se merece el responsable de que esta tesis se haya realizado, Dios, Él movió todos los hilos para que así sucediese, estuvo a mi lado siempre que me desanimé y pensé que no podía, aun cuando no me acordaba de Él.

A papá Tomás, quien en los primeros años de mi vida fue el único padre que conocí y que sin jamás levantarme una mano me enseñó lo que es el respeto, convirtiéndome en la mujer que hoy soy. Estoy convencida de que este hubiera sido el día más feliz de su vida.

A mancita porque siempre ha tratado de darme lo mejor de sí, gracias por confiar en mí y estar siempre con los ojos llenos de brillo, orgullosa de ser mi madre, aun cuando a veces hago cosas que no te gustan. Ahora es mi turno de retribuirte todo lo que has hecho por mí, esa es la principal razón por la que he llegado hasta aquí.

A papá por romperte el lomo todos los días para que yo solo me preocupara por estudiar y por enseñarme los valores que hoy conforman mi persona.

A mamá, porque has sido realmente una madre para mí. Eres la excepción de la regla de los cuentos de princesas y me alegro que así sea. Gracias porque has sido una excelente amiga y una extraordinaria madre, aun cuando no llevo tu sangre.

A mima, por ser como una madre para mí y dejarme dormir entre ti y papá.

A tata, porque ha jugado muchos roles en mi vida, has sabido ser padre para regañarme y protegerme, has sido un hermano para conversar y compartir cosas lindas y un tío súper chulo para complacer caprichos.

A Magela y Mario por ser unos hermanos tan tranquilos, con los que nunca discuto, ahí también está la excepción de una regla.

A Daimelis y Daimel porque ustedes han sido una de las mayores razones que me han mantenido firme en estos años de la carrera, cuando me daban deseos de abandonar pensaba en ustedes y la idea desaparecía.

A toda mi familia en general por la confianza que siempre me han tenido y por el apoyo que me han dado. Especialmente a esa familia bella que me ha adoptado como miembro y que sé me adoran. Es recíproco.

A Enmanuel, por ser lo más maravilloso que ha llegado a mi vida. Eres la confirmación de que el amor verdadero existe y tengo la sensación de que es para siempre.

A todos mis amigos, a los hermanos que hice en la vocacional Yanet, Yari, Anyi, Felipe, el Costa, Julito, Yovany, Glendy y especialmente a Lianet, que en estos cinco años se volvió indispensable para mí. A los amigos que me regaló la UCI y que van a ser para siempre Lisy, Noe, Yeni, Luis, Roci e Ivaniet, que realmente en estos últimos tiempos no sé qué hubiera sido de mí sin él.

A la gente de Comité Primario, primeramente por darme la oportunidad de compartir con ustedes u luego por no arrepentirse. Aprecio mucho como seguían mis ideas locas aun cuando parecían irrealizables.

A todos los profesores que de una forma u otra han puesto su granito de arena. A mis tutores, especialmente a Damián, porque además, me ayudó a confiar en mí y a descubrir que había cosas para las que era buena y no me había dado cuenta.

A mi compañero de tesis.

A todos, de corazón ***Muchas Gracias****.*

De Ernesto:

A la Revolución, por darme esta oportunidad.

A mi mama Grisel, por complacerme en todos mis antojos, por ser mi profe preferida y la mejor madre del mundo.

A mi papa Alpidio por sus constantes consejos y por saber despertar en mí las ganas de triunfar. Eres el principal promotor de que hoy sea ingeniero.

A mi hermano José Carlos, por siempre apoyarme todo. Te quiero mucho.

A mis abuelos, en especial a Coralio que a pesar de no estar presente entre nosotros, siempre estará presente para mí, la única persona que en momentos críticos, y de mucha tensión nunca dudo de mí. Te extraño tanto.

A mis tíos Luis Orlando, Argelio, Anita, Any y Ramon que se lo mucho que disfrutan esta ocasión,

A mis tutores René Bauta Camejo y Damián Pérez Alfonso, por su cariño y apoyo incondicional.

A la profesora Rosalina, por su comprensión y paciencia.

A mi amigo Tony por su ayuda durante estos cinco años.

A los compañeros del aula, en especial a Pablo, Andrew, Roberto, Artime, José, Yaniel, Osbel, Barreto, Yunier, Andrés y Rene, gracias por todo.

A mi novia Lisandra, por estar presente en esta etapa tan difícil de mi vida.

A mi compañera de tesis Dannelis. Eres la mejor compañera del mundo.

A todos los que de una forma u otra contribuyeron con mi formación universitaria y humana.

DEDICATORIA

De Dannelis:

A papá Tomás, por ser la persona que más confió en mi.

De Ernesto:

A mi papá Alpidio y a mi abuelo Coralio.

RESUMEN

La gestión de la seguridad en aplicaciones web es un proceso costoso y relevante para las mismas. Actualmente cada una de las aplicaciones gestiona su propia seguridad, esto ocasiona pérdida de tiempo en el desarrollo de productos de software. El marco de trabajo Sauxe es una aplicación desarrollada con el objetivo de servir como base para el desarrollo de aplicaciones web de gestión. Sauxe incorpora el Sistema Integral de Gestión de Seguridad Acaxia, el cual tiene como objetivo brindar servicios de este tipo a las aplicaciones que se suscriban a él. Symfony2 es un marco de trabajo muy utilizado en el desarrollo de aplicaciones informáticas. La integración de Acaxia al marco de trabajo Symfony2 permitirá el desarrollo de aplicaciones web de gestión donde se obtiene un soporte más robusto a los procesos de seguridad. Con la presente investigación se propone desarrollar un componente para la gestión de los procesos de seguridad en la integración. Esta se basa en la utilización de conceptos definidos por el componente de seguridad de Symfony2 unido al consumo de servicios brindados por Acaxia. Los procesos soportados por la integración son los de Identificación y autenticación, Autorización y Auditoría.

PALABRAS CLAVE

Seguridad, entornos multidominios, integración, componente.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Los constantes ataques que tienen lugar en las redes informáticas con fines económicos, políticos, militares y sociales han provocado que la seguridad se convierta en un tema de permanentes análisis en las organizaciones. Este problema puede agravarse en escenarios donde sea necesario lograr la interoperabilidad entre dominios heterogéneos con diferentes políticas de seguridad, manteniendo la capacidad de evolucionar ante las necesidades de cambios operacionales y de los servicios que se ofrecen. (Baryolo, 2012)

La seguridad se encarga de cubrir cuatro aspectos fundamentales (confidencialidad, integridad, disponibilidad y trazabilidad). La confidencialidad consiste en que la información solo pueda ser accedida por las personas autorizadas. La integridad está encaminada a salvaguardar la exactitud y totalidad de la información y los métodos de procesamiento (Baryolo et al, 2011). Por su parte, la disponibilidad está dada por la posibilidad de que el usuario pueda acceder al sistema, de la forma autorizada, siempre que lo necesite. Finalmente la trazabilidad permite lograr que los Sistemas de Información (SI en lo adelante) sean auditables a través del registro de datos que conforman un histórico que posibilita conocer la ubicación y trayectoria de un recurso.

Para garantizar el cumplimiento de los aspectos de la seguridad, se definen cuatro procesos: Identificación, Autenticación, Autorización y Auditoría. La Identificación es la acción que realiza el usuario de presentar su identidad al SI; mientras que la Autorización consiste en verificar que el usuario que se está identificando es válido para el sistema; estos dos procesos en la mayoría de los casos se desarrollan unidos, por lo que en lo adelante serán tratados como un proceso único. La Autorización consiste en restringir el acceso de los usuarios sobre los recursos y funcionalidades. La Auditoría establece el registro de las acciones que realiza el usuario en cada momento. Generalmente cada SI gestiona estos procesos de forma independiente, lo cual constituye una limitante en entornos multidominios, por lo que los mecanismos existentes para ello no cubren los requisitos que demanda el control de acceso en este tipo de escenarios (Baryolo, 2012).

PHP es uno de los lenguajes más utilizados para el desarrollo de SI, así lo demuestran varios estudios realizados en los últimos años. Según W3techs[1](ver Anexos 1 y 2), resultó el lenguaje del lado del servidor más popular (W3techs, 2013). Esta afirmación se basa en la utilización del

[1] Servicio web provisto por Q-Success, empresa miembro de la Cámara Económica Federal de Austria, sección Consultas de Negocios y Tecnologías de la Información.

mismo en el 81.8 % de los sitios web hosteados en Internet. Recientemente Google Trends[2] (ver Anexo 3) lo ubica en el segundo lugar de popularidad mundial entre los lenguajes de programación de propósito general. Para fundamentar esta afirmación se tiene en cuenta la frecuencia con que se realizan búsquedas relacionadas con él en varias regiones del mundo y en múltiples idiomas (GoogleTrends, 2014).

El desarrollo de SI está determinado por el uso de marcos de trabajo que permiten al desarrollador reducir tiempo, costo y esfuerzos. Según estudios de Google Trends en el 2014, Symfony2 es el marco de trabajo de PHP con mayor promedio de búsquedas en internet (ver Anexo 4). Esto queda demostrado en la suma del promedio de búsquedas al usar las palabras Symfony (95) y Symfony2 (54), donde aparece situado en el primer lugar con 149 puntos, seguido de Cakephp con 105 y ZendFramework con 39, lo que constituye una expresión de su popularidad.

Symfony2 cuenta con un componente encargado de soportar los procesos de seguridad que permite la autenticación HTTP[3] y realiza la Autorización mediante un sistema de Listas de Control de Acceso (ACL por sus siglas en inglés). Brinda además la posibilidad de que el programador implemente su propia estrategia de seguridad. Este soporte es limitado, pues no permite el acceso a varios recursos o aplicaciones con un único proceso de Autenticación, por lo que en entornos multidominios los usuarios deben presentar sus credenciales en cada una de las aplicaciones. Además, las ACL no resultan factibles en aplicaciones de alta envergadura y muchas funcionalidades, donde los permisos varían con frecuencia. Resulta importante resaltar que el componente no brinda soporte al proceso de auditoría (SensioLabs, 2013a). Estas limitaciones traen consigo que los desarrolladores, además de implementar una aplicación que responda a sus necesidades, necesiten centrar también su atención en la implementación de una estrategia de seguridad robusta.

El Sistema de Gestión Integral de Seguridad Acaxia, constituye una alternativa fiable para el soporte a la seguridad en los SI, de modo que el desarrollador puede abstraerse de estos procesos y centrar su atención en el negocio que desea informatizar. Acaxia fue desarrollada con el marco para la creación de aplicaciones web de gestión Sauxe, creado en el Centro para la Gestión e Informatización de las Entidades (CEIGE en lo adelante), perteneciente a la Universidad de las Ciencias Informáticas. Constituye una alternativa que soporta los procesos de

[2]Herramienta de Google Labs que muestra los términos de búsqueda más populares del pasado reciente.

[3]Protocolo de Transferencia de Hipertexto (HTTP por sus siglas en inglés). Es el protocolo para la trasferencia de hipertexto usado en cada transacción de la Red informática mundial (WWW por sus siglas en inglés).

seguridad de las aplicaciones que estén suscritas a él. Constituye una implementación del Modelo de Control de Acceso para Entornos Multidominios (CAEM en lo adelante). Este modelo permite gestionar la seguridad de varios sistemas de forma centralizada y su capacidad para adaptarse a entornos heterogéneos como los multidominios, ha facilitado su aplicación en múltiples entornos reales con buenos resultados (Baryolo, 2012).

En la investigación “Modelo de Control de Acceso para Entornos Multidominios” (Baryolo, 2012), se aplican técnicas de validación usando determinados indicadores relacionados con la fortaleza de los procesos de seguridad. Estos valores para Acaxia son mayores que para los demás estándares y protocolos estudiados por el autor, por lo que se infiere que la integración de Acaxia a Symfony2 permitiría el fortalecimiento de los procesos de seguridad para las aplicaciones desarrolladas usando este marco de trabajo. Se debe tener en cuenta que los mecanismos de integración de Sauxe (y por tanto el mecanismo de integración de Acaxia) y Symfony2 difieren entre ellos en cuanto al conjunto de reglas que establecen la definición e integración de los componentes.

Symfony2 define los componentes en forma de bundles que permiten la reutilización de funcionalidades creadas por terceros. Facilitan, además, la activación y desactivación de determinadas características dentro de la aplicación. La integración de los bundles se realiza mediante la aplicación del patrón de diseño contenedor de dependencias, a través del cual se configuran y obtienen los servicios relacionados al componente. Un componente en Sauxe debe contener en su directorio raíz un fichero en el cual está contenida su descripción, dada por algunos datos generales necesarios para el control de su ciclo de vida, los servicios que brinda y las dependencias que necesita para su funcionamiento. Los servicios y las dependencias se encuentran especificados en las interfaces.

El problema fundamental consiste en que el contenedor de dependencias de Symfony2 no reconoce los componentes de Sauxe, y a su vez, el mecanismo de integración de Sauxe no reconoce los bundles de Symfony2, debido a que cada uno de estos marcos de trabajo posee diferentes contextos de ejecución, configuraciones, estructura física de directorios y convención para espacios de nombres. Por otra parte, la seguridad es un proceso que se realiza de forma transversal, lo cual plantea retos adicionales para la integración.

Por ello, se define como **problema a resolver**: ¿Cómo integrar el Sistema de Gestión Integral de Seguridad Acaxia al marco de trabajo Symfony2 para fortalecer los procesos de Identificación y autenticación, Autorización y Auditoría?

La problemática descrita posee como **objeto de estudio:** seguridad de SI en PHP, enmarcado en el **campo de acción:** componentes de seguridad para SI.

Para darle solución al problema definido se tiene como **objetivo general:** desarrollar un componente para la integración del Sistema de Gestión Integral de Seguridad Acaxia al marco de trabajo Symfony2 para fortalecer los procesos de Identificación y autenticación, Autorización y Auditoría.

El objetivo general, para su cumplimiento, se ha desglosado en los siguientes **objetivos específicos:**

1. Realizar un estudio acerca del soporte a los procesos de seguridad en los marcos de trabajo Sauxe y Symfony2.
2. Realizar un análisis de los mecanismos de integración entre componentes de los marcos de trabajo Sauxe y Symfony2.
3. Desarrollar componentes que integren el marco de trabajo Symfony2 al Sistema de Gestión Integral de Seguridad Acaxia para garantizar los procesos de Identificación y autenticación, Autorización y Auditoría.
4. Evaluar en un caso de estudio el cumplimiento de indicadores de seguridad de una aplicación antes y después de la integración de la solución propuesta para validar la misma.

Se espera obtener un componente para la integración de los servicios de seguridad del Sistema Integral de Seguridad Acaxia con aplicaciones desarrolladas en Symfony2.

El desarrollo de la investigación parte de la **idea a defender**: Si se desarrolla un componente para la integración de los servicios de seguridad del Sistema Integral de Seguridad Acaxia con aplicaciones desarrolladas en Symfony2, se logrará fortalecer el soporte de Symfony2 a los procesos de Identificación y autenticación, Autorización y Auditoría.

Definición de las variables:

Variable independiente: componente para la integración de los servicios de seguridad del Sistema Integral de Seguridad Acaxia con aplicaciones desarrolladas en Symfony2.

Variable dependiente: el soporte a los procesos de Identificación y autenticación, Autorización y Auditoría.

Operacionalización

Variable independiente	Indicador	Escala de medición	Descripción de la evaluación de la variable indicador
Componente para la integración de los servicios de seguridad del Sistema Integral de Seguridad Acaxia con aplicaciones desarrolladas en Symfony2.	Presencia: La variable independiente, es una variable dicotómica, o sea, de ella se mide su presencia o ausencia.	Nominal Si No	Se realiza un pre-experimento, en el cual se evalúa el soporte a los procesos de seguridad a una aplicación desarrollada con Symfony2 antes y después de la integración. **Valores del indicador:** **No:** antes de la integración **Sí:** después de la integración.
Variable dependiente	**Indicador**	**Escala de medición**	**Descripción de la evaluación de la variable indicador**
El soporte a los procesos de Identificación y autenticación, Autorización y Auditoría.	Fortalecimiento	Nominal Si No	Se realiza un pre-experimento, en el cual se evalúa el soporte a los procesos de seguridad a una aplicación desarrollada con Symfony2 antes y después de la integración. **Valores del indicador:** **Si:** cuando los valores obtenidos luego de la integración son mayores que los obtenidos antes de integrar. **No:** los valores obtenidos luego de la integración son menores o iguales a los obtenidos antes de integrar.

Tabla 1. Operacionalización de las variables (elaboración propia).

Métodos de la investigación científica a aplicar:

Métodos teóricos:

- **Histórico-lógico:** se empleará para estudiar las tendencias actuales de la seguridad de los SI, así como las principales tecnologías y herramientas usadas en el proceso de desarrollo del componente.
- **Analítico-Sintético:** se utilizará para el estudio del soporte a los procesos de seguridad de Symfony2 y Acaxia.

- **Inductivo-Deductivo:** permitirá asumir que la integración fortalece los procesos de seguridad de las aplicaciones desarrolladas con Symfony2, teniendo en cuenta que los resultados arrojados por la evaluación de los indicadores que tributan a la fortaleza de estos procesos es mayor para Acaxia que para los demás estándares y protocolos existentes.

Métodos empíricos:

- **Entrevista:** se explotará en dos momentos, primero para la recolección de las características que debe cumplir el componente y al final de la investigación para la evaluación de los indicadores que tributan a la fortaleza de los procesos de seguridad, en este último momento a través del empleo de un cuestionario de preguntas cerradas.
- **Experimento Científico:** posibilitará demostrar la utilidad del componente obtenido.

La presente investigación se encuentra estructurada en tres capítulos:

Capítulo 1: Fundamentación teórica

Se describe el marco conceptual que existe alrededor de la seguridad en SI. Se brinda una descripción general del marco de trabajo Symfony2 y del Sistema de Gestión Integral de Seguridad Acaxia. Posteriormente se describe el soporte de ambos a los procesos de seguridad y sus mecanismos de integración. Además, se enumeran las principales características del modelo de desarrollo de software seleccionado y el flujo de trabajo que propone. Finalmente se presenta el estudio de las tecnologías y herramientas empleadas en el desarrollo de la solución.

Capítulo 2: Propuesta de solución

Se muestra el modelo conceptual del componente para la integración de los servicios de seguridad del Sistema Integral de Seguridad Acaxia con aplicaciones desarrolladas en Symfony2. Se exponen los requisitos funcionales y no funcionales de la solución. Se define la arquitectura del sistema y finalmente se seleccionó uno de los requisitos más críticos, del cual se muestran su diagrama de clases y de secuencia.

Capítulo 3: Implementación y pruebas

Se aplican las métricas Tamaño Operacional de Clases y Relaciones entre Clases al diseño elaborado. Se exponen los estándares de codificación utilizados para lograr un buen entendimiento y legibilidad del código. Se describe el diagrama de componentes generado. Se aplica la prueba de caja blanca a través de la técnica del camino básico. Se evidencia el cumplimiento de la idea a defender planteada con la aplicación de un cuestionario respondido por dos expertos en el desarrollo de aplicaciones con Symfony2.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

La seguridad es un factor fundamental a tener en cuenta en la implementación de un SI, debido a que estos contienen la información sensible de las organizaciones y del personal que los utiliza. El marco de trabajo Symfony2 presenta un soporte limitado en cuanto a este tema, una posible solución para ello sería su integración al Sistema Integral de Seguridad Acaxia. Es por ello que en el presente capítulo se presenta un estudio sobre los principales conceptos asociados a la seguridad en los SI, así como el soporte de estos en Acaxia y Symfony2. Además, se describen sus mecanismos de integración entre componentes.

1.1 Seguridad en Sistemas de información

Un SI constituye un conjunto organizado de personas, procesos y recursos, incluyendo la información sensible y sus tecnologías asociadas, que interactúan de forma dinámica para satisfacer las necesidades informativas que posibilitan alcanzar los objetivos de una o varias organizaciones (Baryolo 2012). En la mayoría de los casos sus recursos materiales están constituidos casi en su totalidad por sistemas informáticos; por lo que la seguridad debe ser un factor elemental a tener en cuenta en su implementación. Entendiéndose este término como una disciplina que integra un conjunto de políticas, procesos, procedimientos, estructuras organizacionales y funciones de software y hardware para ayudar a proteger la confidencialidad, integridad, disponibilidad y trazabilidad de los recursos gestionados por los sistemas de información en las organizaciones (Achiary, 2005). La seguridad comprende tres procesos fundamentales que se describen a continuación.

Identificación y autenticación

La Identificación no es más que la acción, por parte de un usuario, de presentar su identidad a un sistema, para lo que normalmente se utiliza un identificador de usuario. Establece, además, que el usuario es responsable de las acciones que lleva a cabo en el sistema (Baryolo et al, 2011). La Autenticación consiste en establecer un elemento como auténtico, en el ámbito de la seguridad se traduce en la verificación de que el usuario que trata de identificarse es válido. Normalmente se implementa con una contraseña en el momento de iniciar una sesión. Los métodos para realizar la Autenticación se dividen en tres categorías, atendiendo a los elementos que utilizan para realizar el proceso:

- Sistemas basados en algo conocido (contraseña).

- Sistemas basados en algo poseído (tarjeta inteligente, dispositivo usb).
- Sistemas basados en una característica física o en un acto involuntario del usuario (voz, huellas, escritura).

Para realizar el proceso de autenticación se pueden tener en cuenta determinados factores, que se clasifican en cuatro categorías:

- Algo que el usuario **es** (huella digital, patrón retiniano, secuencia de ADN, patrón de voz, reconocimiento de la firma, patrones biométricos).
- Algo que el usuario **tiene** (tarjeta de la identificación, símbolo de la seguridad, símbolo del software o teléfono celular).
- Algo que el usuario **sabe** (contraseña, frase, número de identificación personal).
- Algo que el usuario **hace** (reconocimiento de voz, firma).

La selección de un método en específico y de los factores a emplear para realizar la Autenticación está determinada por la relevancia de la información que se gestiona, las posibilidades de estar sometida a ataques, así como la solvencia económica de la organización. Estos factores se pueden emplear separados o realizar varias combinaciones, a lo que se le conoce como Autenticación multifactor.

Autorización

Es el proceso mediante el cual se comprueba que el usuario con identidad válida (previamente identificado y autenticado) solo tenga acceso en el sistema a los recursos sobre los que tiene permisos asignados. Los recursos incluyen archivos y otros objetos de datos, programas, dispositivos y funcionalidades provistas por aplicaciones.

Auditoría

Es el proceso encargado del registro y análisis de las acciones que realizan los sujetos en todo momento sobre los recursos del SI (Baryolo, 2012). La definición anterior se traduce en la recopilación permanente de datos relacionados con las acciones que realizan los usuarios en el sistema, que pueden ser utilizados para la asignación de responsabilidades y contribuir a la toma de decisiones en las organizaciones. Este proceso está estrechamente relacionado con la Identificación y autenticación y la Autorización, pues su funcionamiento correcto depende en gran medida de la implementación de los dos anteriores.

1.2 Marco de trabajo Symfony2

Symfony2 es un marco de trabajo PHP construido con varios componentes independientes creados por el proyecto Symfony. Su código, y el de todos los componentes y librerías que incluye, se publican bajo la licencia MIT[4] de software libre. Incluye varias herramientas gráficas y de consola para depurar fácilmente los errores que se produzcan en las aplicaciones. Permite establecer los parámetros de configuración de las aplicaciones a través de variables de entorno del propio servidor. Emplea la herramienta Composer, que simplifica la instalación y gestión de las dependencias de las aplicaciones PHP, creada también por varios miembros de la comunidad Symfony (ANON, 2013).

El centro del componente de seguridad de Symfony2 es el Contexto de Seguridad, este contiene la información generada una vez que el proceso de Autenticación concluye satisfactoriamente. Posteriormente puede ser consultado para verificar si un usuario tiene acceso a una acción o recurso dentro de la aplicación (SensioLabs, 2013a).

1.3 Sistema de Gestión Integral de Seguridad Acaxia

Acaxia controla la información de todos los sistemas que se suscriban a él para gestionar la seguridad de forma centralizada. De cada uno se controlan las funcionalidades que brindan y de cada una de ellas las acciones que se realizan. Los usuarios cuentan con un perfil configurable donde se recogen los datos que sean de interés para las entidades. Un aspecto importante que incorpora el sistema es la administración de conexiones donde se restringe el acceso de los sistemas y usuarios a los servidores, bases de datos, esquemas y otras estructuras de datos; con esto no se tendrá que almacenar las configuraciones de conexión en ficheros físicos que pueden ser objetos de ataques. Cuenta con un componente capaz de registrar todas las acciones realizadas en un sistema. Posee cuatro módulos fundamentales que se muestran en la Figura 1.

[4]Licencia de Software empleada por el Instituto Tecnológico de Massachusetts (MIT por sus siglas en inglés), que permite reutilizar el software así licenciado tanto para ser software libre como para ser software no libre.

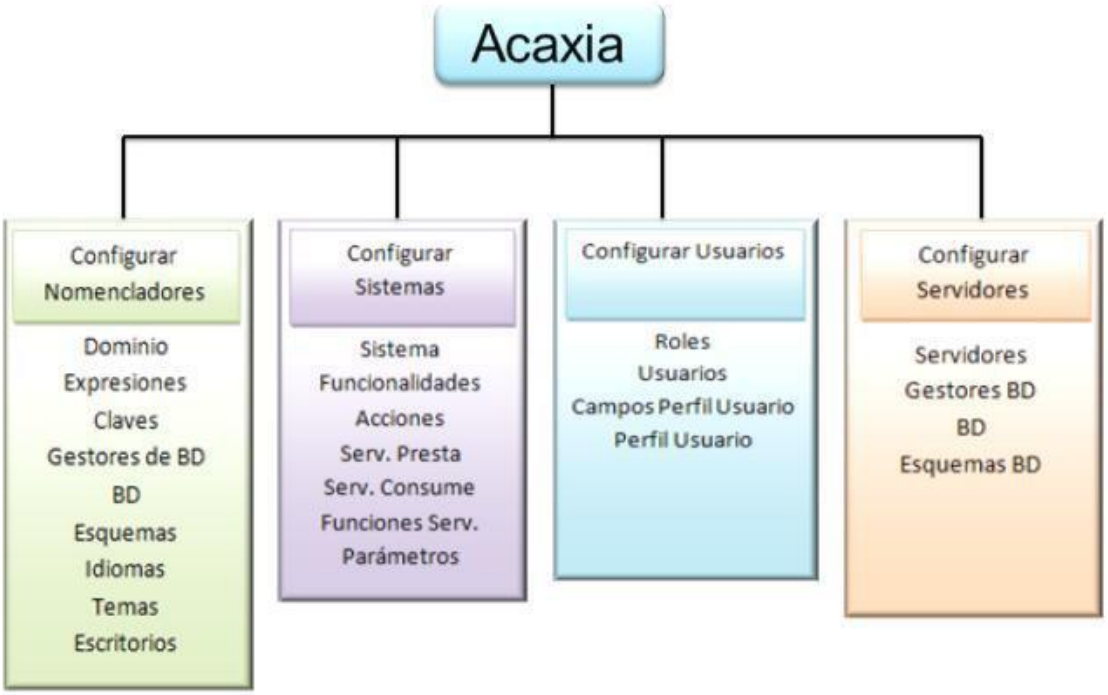


Figura 1. Módulos de Acaxia (Baryolo et al, 2011).

1.4 Proceso de Identificación y autenticación

La Identificación es el proceso en el cual el usuario se da a conocer al sistema, mientras que la Autenticación es la verificación que realiza el sistema sobre esta identidad presentada. Estos dos conceptos, en términos de implementación, generalmente se tratan como un único proceso; sus resultados son necesarios para el control del acceso de los usuarios a los diversos recursos que gestiona el SI. Existen cuatro tipos de técnicas que permiten realizar la Identificación y autenticación del usuario, las cuales pueden ser utilizadas individualmente o combinadas (autenticación de varios factores).

1.4.1 Soporte de Acaxia

El componente de Identificación y autenticación del Sistema de Gestión Integral de Seguridad Acaxia emplea el Lenguaje de Enmarcado de Aserciones de Seguridad (SAML por sus siglas en inglés), estándar que implementa un protocolo de petición-respuesta, donde los sistemas aceptan o rechazan a los usuarios mediante aserciones de seguridad. Posee dos componentes fundamentales: el proveedor de servicios (PS en lo adelante) y el proveedor de identidad (PI en lo adelante), a través de los cuales implementa una arquitectura Single-Sing-On (SSO), que permite unificar los procesos de Identificación y autenticación, Autorización y Auditoría para evitar que el usuario tenga que recordar sus credenciales constantemente.

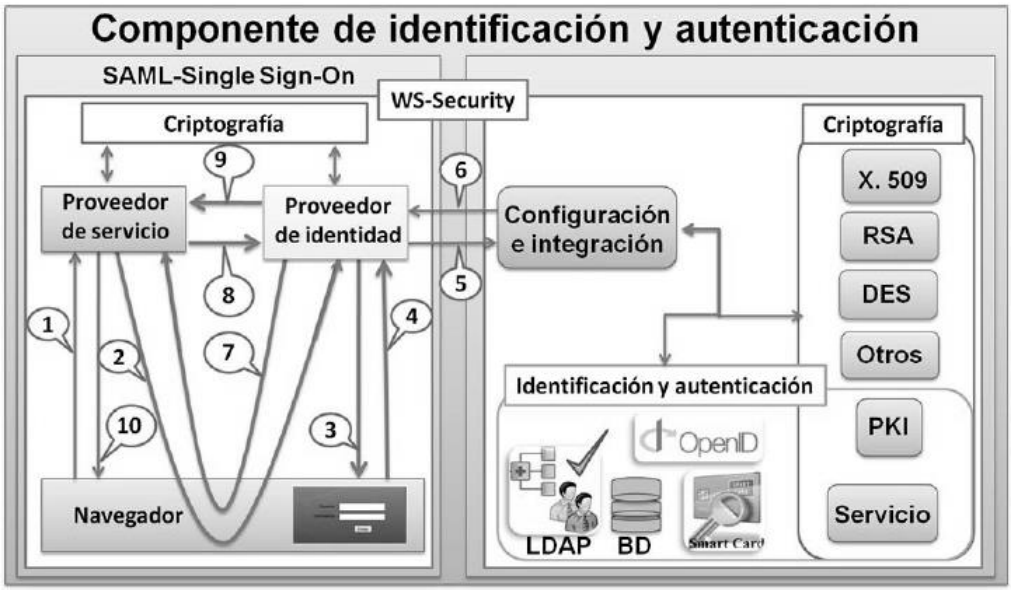


Figura 2. Componente de Identificación y autenticación (Baryolo, 2012).

1.4.2 Soporte de Symfony2

Symfony2 emplea la autenticación de acceso básica o autenticación HTTP, método diseñado para permitir a un navegador web, u otro programa cliente, proveer credenciales en la forma de usuario y contraseña cuando se le solicita una página al servidor. Las credenciales que se envían desde el cliente al servidor, solo son codificadas en Base64[5], lo que hace que se puedan obtener fácilmente debido a que es perfectamente reversible, es decir, una vez que se posee el texto codificado, es posible obtener la cadena original sin ningún problema, por lo que la información enviada no es cifrada ni segura. Una vez introducidas las credenciales del usuario, en el proceso intervienen los siguientes elementos: el cortafuegos, el mapa de áreas seguras, los escuchadores, el Gestor de Autenticación, el Proveedor de Autenticación, el Proveedor de Usuarios, el token de seguridad, y la Fábrica de Seguridad.

La configuración requerida por el componente de seguridad para la Identificación y autenticación comprende un conjunto de áreas seguras, que son almacenadas en un mapa que es usado por el cortafuegos. Cuando una petición realizada por el usuario apunta a una de estas áreas seguras, sus escuchadores se encargan de verificar si la petición contiene la información necesaria para realizar la Autenticación. Una vez que el escuchador extrae de la petición las credenciales del usuario, crea un token de seguridad para almacenarlas y le solicita al Gestor de Autenticación

[5]Sistema de numeración posicional que usa 64 como base.

que valide el token. Finalmente, envía al Contexto de Seguridad un token autenticado, si las credenciales son válidas.

El Gestor de Autenticación contiene un conjunto de Proveedores de Autenticación que soportan diferentes tipos de tokens de seguridad. Específicamente el Proveedor de Autenticación se encarga de implementar la estrategia de Autenticación, para lo cual usa el Proveedor de Usuarios. Este último garantiza el proceso de lectura de los usuarios ya sean usuarios estáticos, definidos en la configuración del marco de trabajo o provenientes de una base de datos previamente mapeada con el Mapeador Relacional de Objetos (ORM por sus siglas en inglés) usado por el marco de trabajo. (SensioLabs, 2013b)

Una de las potencialidades del componente es que todos estos elementos pueden ser personalizados por el desarrollador para implementar su propia estrategia de seguridad. Para que esta personalización sea reconocida, debe construir una Fábrica de Seguridad donde se especifican los elementos personalizados.

1.5 Proceso de Autorización

La autorización es el procedimiento para determinar si el usuario o proceso previamente identificado y autenticado tiene permitido el acceso a los recursos (Baryolo, 2012). Se refiere a conceder o no servicios específicos a un usuario, basándose para ello en su propia autenticación, los servicios que solicita y el estado actual del sistema.

1.5.1 Soporte de Acaxia

El componente de Autorización de Acaxia extiende las especificaciones del modelo de Control de Acceso Basado en Roles (RBAC por sus siglas en inglés), con lo que incorpora conceptos que contribuyen a la granularidad en el establecimiento de las políticas de acceso (Figura 3). Estos conceptos son definidos por Baryolo:

- Recursos: es el concepto que agrupa los activos asociados de un SI sobre los que se establecen políticas de acceso.
- Criterios: son atributos que identifican únicamente a un recurso.
- Reglas: representan restricciones que evalúan uno o varios criterios.
- Operaciones: son acciones que se realizan sobre uno o varios recursos, pueden iniciarse debido a la petición de un usuario o por configuraciones internas del SI.

- Permisos: es un concepto que agrupa las operaciones que se pueden realizar sobre uno o varios recursos que cumplen con una o varias reglas.
- Roles: son funciones de trabajo en el contexto de una organización con una semántica asociada a la autoridad y la responsabilidad conferida a una persona.
- Organizaciones: son estructuras organizativas que forman parte del entorno donde se despliega un SI.
- Dominios: constituyen agrupaciones de organizaciones que tienen en común algún criterio.
- Usuarios: pueden tratarse de personas o sistemas a los que se le asignan privilegios sobre los diferentes recursos de una o varias organizaciones.
- Sesiones: es un concepto que mapea los privilegios que tiene un usuario en un espacio de tiempo, sobre los recursos de una o varias organizaciones debido a los roles que desempeña o por privilegios asignados directamente a él.
- Restricciones: son condiciones impuestas por el negocio para evitar violaciones de seguridad.

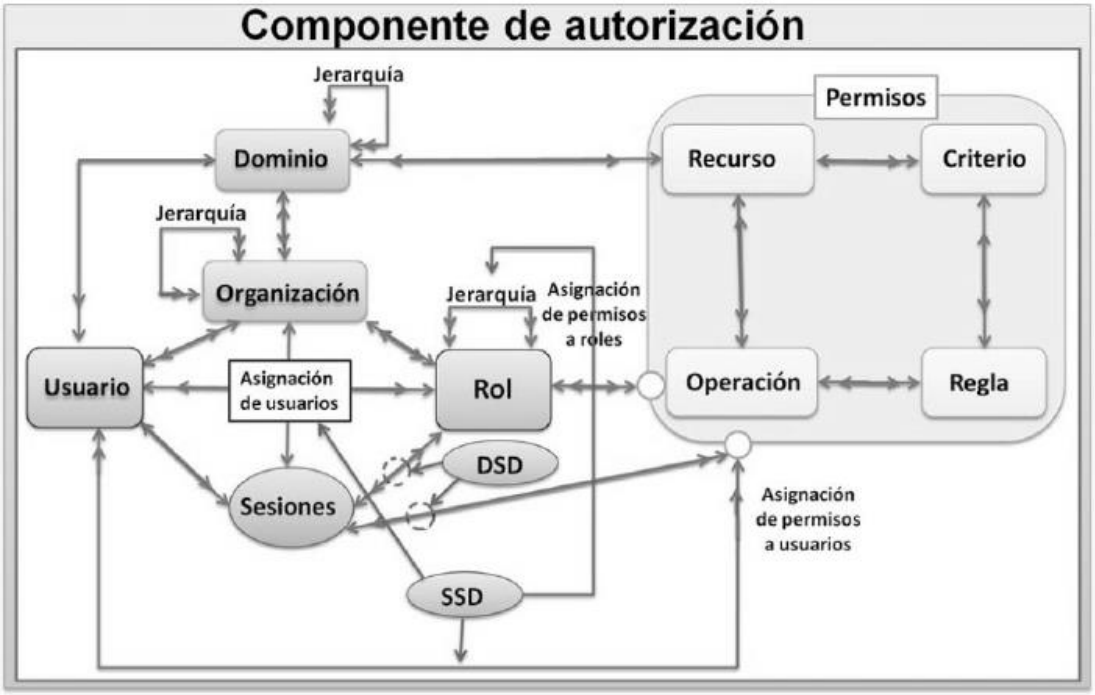


Figura 3. Componente de Autorización de Acaxia (Baryolo, 2012).

El componente se divide en cuatro niveles: el núcleo, herencia de roles, SSD[6] y DSD[7]. El núcleo del componente define nueve conjuntos de elementos de datos básicos: usuarios, dominios, organizaciones, roles, operaciones, recursos, criterios, reglas y permisos, y establece las relaciones entre estos para el funcionamiento del componente. El registro de los recursos se realiza en correspondencia con el dominio y la organización a la que pertenece el usuario, lo que permite compartir los recursos entre las organizaciones y los dominios; a estos recursos compartidos tienen acceso los usuarios con roles administrativos.

Se definen, además, las restricciones estáticas y dinámicas, útiles a la hora de hacer cumplir políticas de conflicto de intereses, manteniendo la formalización que se propone en el modelo RBAC para la asignación de roles a un usuario, y la jerarquía de roles añade una restricción para cada tipo, con el objetivo de restringir los privilegios asignados directamente al usuario de forma dinámica o estática. (Baryolo, 2012)

1.5.2 Soporte de Symfony2

Una vez que el usuario se ha autenticado, da inicio el proceso de Autorización, el cual proporciona un mecanismo estándar para decidir si este puede acceder a algún recurso. Los recursos pueden ser tanto un archivo, como objetos de dato, programas, dispositivos o funcionalidades provistas por aplicaciones. El funcionamiento de la Autorización se basa en asignar roles específicos a cada usuario y, después, hacer que las distintas partes de la aplicación requieran cada una de roles diferentes para poder acceder.

El proceso de Autorización tiene, por tanto, dos componentes principales:

- El usuario dispone de uno o más roles.
- Un recurso requiere de un rol específico para poder acceder a él.

El componente de seguridad de Symfony2 gestiona el proceso de Autorización a través de un mecanismo denominado Listas de Control de Acceso (ACL por sus siglas en inglés). Este mecanismo permite tener un control eficiente sobre los permisos asignados a cada usuario en el sistema. Las Listas de Control de Acceso definidas en Symfony2 operan en dos ámbitos diferentes:

[6]Separación Estática de Deberes (SSD por sus siglas en inglés). Restricción que se establece sobre los usuarios y roles.
[7]Separación Dinámica de Deberes (DSD por sus siglas en inglés). Restricción que se establece sobre los usuarios y roles.

1. Class-Scope: Este ámbito se aplica a todas las instancias creadas de una clase determinada.
2. Object-Scope: Este ámbito se aplica a un objeto determinado.

Para definir una ACL el desarrollador necesita poseer un control de los permisos asignados a cada usuario, de esta forma cada vez que estos permisos varían es necesario acceder al código de la ACL y agregar las modificaciones pertinentes. Además, resulta obligatorio poseer al menos una conexión configurada a la base de datos con el ORM Doctrine; esto se debe a que internamente los permisos asociados a los usuarios se almacenan en una base de datos.

1.6 Proceso de Auditoría

Proceso encargado del registro y análisis de las acciones que realizan los usuarios del sistema en todo momento sobre los recursos del SI (Baryolo, 2012). Garantiza la existencia de un registro histórico de la interacción de los usuarios con el sistema y con los objetos que este maneja. Está fuertemente ligado a los dos procesos anteriores, pues las salidas de estos constituyen parte de sus entradas.

1.6.1 Soporte de Acaxia

El componente para el soporte del proceso de auditoría de Acaxia permite realizar análisis con tres objetivos fundamentales: cumplimiento de las normas de seguridad, funcionamiento del SI y minería de procesos de negocio informatizados. Posee dos niveles fundamentales:

1. El Nivel de negocio, constituido por los procesos y las actividades que se deben ejecutar en el sistema. Este permite identificar qué usuarios o roles realizan una actividad determinada y en qué nivel de la jerarquía.
2. El Nivel de sistema, que contiene la estructura del SI. En este se definen las acciones del sistema que ejecutan las actividades del proceso de negocio. Estas generan un conjunto de eventos que constituyen información básica para la auditoría.

Los eventos están divididos en tipos, con el objetivo de garantizar escalabilidad, disminuir la cantidad de información y facilitar su análisis. Existen datos que son comunes para todos los eventos, independientemente de su tipo: el usuario, rol, entidad, sistema, ip, fecha y hora. El componente de auditoría registra los eventos que se describen a continuación:

Acción: Registra los parámetros generales de una acción realizada en el sistema. Incluye parámetros que indican si la acción se realizó o no correctamente. Puede, a su vez, desencadenar otros tipos de eventos.

Rendimiento: Evento que se activa en caso de ser necesario registrar información relacionada con el funcionamiento del sistema. Incluye parámetros como: tiempo de respuesta y memoria consumida.

Integración: Se activa si en la acción se consume algún servicio para realizar un proceso de negocio. Se registran parámetros como: el sistema destino y el servicio solicitado.

Excepciones: Se producen como consecuencia de la violación de reglas impuestas por el negocio o por las restricciones tecnológicas del entorno. Incluye el registro de parámetros como: la regla ejecutada y la información asociada a ella.

Excepciones de integración: Se registra cuando hay algún error en la llamada de un servicio. Se registra quien solicitó el servicio, quien lo brinda, la clase y el método que implementa el servicio.

Datos: Se ejecuta si una acción realiza alguna operación en la base de datos. Incluye parámetros como: servidor, gestor, puerto, rol de base de datos, base de datos, esquema (si aplica), tabla y objeto afectado.

Proceso: Es un evento que se activa si el SI está orientado a proceso, en este caso se incluye el registro de parámetros como: proceso, instancia, actividad y objeto afectado.

1.6.2 Soporte de Symfony2

La Auditoría cubre dos tareas fundamentales, el registro y el análisis. Symfony2 no brinda soporte a este proceso, sin embargo provee funcionalidades que pueden ser empleadas para ejecutar el registro de ciertos datos. En esta área se encuentran las funcionalidades asociadas a la gestión de eventos que provee el marco de trabajo.

La gestión de eventos en Symfony 2 se basa en los conceptos de evento y escuchador. Se pueden definir escuchadores personalizados para eventos determinados de forma tal que permitan a los administradores hacer un resumen de las acciones que se realizan en la aplicación. Estos eventos cubren las capas de presentación y negocio de las aplicaciones.

En la capa de acceso a datos, Symfony2 propone a Doctrine 2 como ORM (Mapeador Relacional de Objetos). Doctrine, a su vez, provee un conjunto de eventos definidos que permiten detectar

cuando se está haciendo uso de los datos almacenados en la base de datos. Esto permite que se puedan implementar escuchadores para estos eventos, que registren las trazas de dichas acciones para su posterior análisis.

1.7 Mecanismos de integración

Un componente es un fragmento de un sistema de software que puede ser ensamblado con otros fragmentos para formar piezas más grandes o aplicaciones completas. Esta definición se basa en tres perspectivas de un componente: la perspectiva de empaquetamiento, que considera un componente como unidad de empaquetamiento; la perspectiva de servicio, que considera un componente como proveedor de servicios y la perspectiva de integridad, que considera un componente como un elemento encapsulado. En consonancia con este concepto, cada proyecto define para su desarrollo su propia estructura de componentes (CEIGE, 2012).

Una interfaz determina tanto las operaciones que el componente implementa como las que precisa utilizar de otros componentes durante su ejecución (Terreros, 2009). La integración de componentes se realiza a través de las interfaces.

Cuando un desarrollador emplea el paradigma de programación orientado a componentes, debe establecer un modelo donde se estandaricen aspectos como: definición, implementación, ensamblaje, estilos de código y convención de nombres de los componentes para facilitar la integración entre estos. (Terreros, 2009)

1.7.1 Sauxe

En Sauxe cada componente tiene una descripción en un fichero de extensión *.scl* que se encuentra en su directorio raíz, estructurada por las etiquetas que se explican a continuación:

<data>: contiene la versión que se especifica por el usuario según la estructura de versionado definida y el estado que es asignado por el marco de trabajo al reconocer el componente; estos elementos se tienen en cuenta para el ciclo de vida del mismo.

<services>: servicios que provee.

<dependencies>: servicios que necesita de otros componentes.

<service>: se emplea para declarar un servicio y contiene una serie de atributos que se deben especificar.

- id: identificador del servicio.

- Interface: dirección física donde se encuentra el fichero PHP con la interfaz donde se definen los métodos que brinda el componente.
- imp: dirección física donde se encuentra el fichero PHP con la implementación de la interfaz.

<dependency>: se emplea para declarar una dependencia; se deben especificar los siguientes atributos:

- id: identificador de la dependencia.
- interface: dirección física donde se encuentra el fichero PHP con la interfaz donde se definen los métodos que necesita el componente.
- version: establece la versión del componente que debe satisfacer la dependencia.
- optional: se especifica si es opcional la resolución de la dependencia.

Las interfaces de los componentes contienen comentarios en PHPNotation para especificar el tipo de dato de cada uno de los parámetros que recibe cada servicio y/o el tipo de dato del retorno, en caso de que existan. A partir de estos comentarios y los métodos definidos, se establecerá la equivalencia entre interfaces. Los nombres de las interfaces deben ser definidos con la siguiente estructura: NombrecomponenteNombreinterfaz.

Un componente tiene un ciclo de vida determinado por dos fases; en la primera se encuentran todos los componentes y se les asigna el estado resuelto si no tiene dependencias, y no resuelto, en caso contrario. Posteriormente se trata de resolver las dependencias de los componentes no resueltos, con los servicios que brindan los resueltos. En la segunda fase se encuentran los componentes activados o desactivados. Están desactivados todos los componentes no resueltos porque no brindan ningún servicio, y los resueltos pasan a ser activados por resueltos; estos se pueden desactivar en caso de que no se desee que brinden sus servicios.

Para el consumo de un servicio de Sauxe este debe estar declarado previamente como una dependencia. Existen dos formas de realizarlo, que difieren entre sí por el lugar del marco de trabajo en el que pueden ser utilizadas. La primera se emplea dentro de un controlador o modelo del sistema. (Ver Figura 4). La segunda se utiliza desde cualquier parte de Sauxe (Ver Figura 5).

```
$seguridad = $this->component->Seguridad;
```

Figura 4. Consumo de servicios desde un controlador (elaboración propia).

```
$this->component->setNewInstance(true);
$seguridad = $this->component->Seguridad;


$seguridad = ZendExt_Component_Factory::getComponent($this, "Seguridad", true);
```

Figura 5 Consumo de servicios desde cualquier parte de Sauxe (elaboración propia).

Finalmente, Sauxe brinda dos opciones para agregar un nuevo componente, una interfaz visual mediante la cual se pueden adicionar componentes que quedan registrados por su ubicación o borrar los registros ficheros bundles.xml y bundles.data que contienen la información de todos los componentes del marco de trabajo, y este se encargará de localizar nuevamente todos los componentes, incluyendo los nuevos.

1.7.2 Symfony2

El mecanismo de integración propuesto por Symfony2 tiene su base conceptual en componentes denominados bundles, que son la base fundamental de su filosofía de trabajo. Su propio código fuente y el de las aplicaciones creadas con él se estructuran de esta forma. Técnicamente un bundle es un directorio que contiene todo tipo de archivos dentro una estructura jerarquizada de directorios. Suelen contener clases PHP y archivos web (JavaScript, CSS e imágenes). El marco de trabajo posee un contenedor de servicios que permite la estandarización y centralización de la construcción de objetos en una aplicación, con el objetivo de proporcionar ayuda en la creación de instancias y organizar y recuperar objetos (servicios) en una aplicación. Este proceso se realiza mediante la inyección de dependencias. (SensioLabs, 2013a)

La inyección de dependencias es la clave para entender el funcionamiento de Symfony2. Consiste en crear y configurar primero un objeto y después crear una instancia para ser empleada por la clase que solicita la dependencia. El contenedor de inyección de dependencias es un objeto que contiene la información necesaria para crear los objetos de una aplicación. Para ello, tiene definidas todas las relaciones entre las clases y la configuración necesaria para instanciar correctamente cada una. Su empleo simplifica el uso de la inyección de dependencias en las aplicaciones.

Para definir servicios en el contenedor solo se debe crear una clase PHP y realizar las configuraciones en un archivo YAML, XML o PHP. Internamente el marco de trabajo transforma automáticamente esa configuración en el código PHP que se ejecuta para instanciar dependencias, cargar opciones de configuración y crear los objetos solicitados. Las

funcionalidades antes mencionadas proporcionan facilidades para el trabajo con una arquitectura que promueve el código reutilizable y disociado.

Los bundles pueden ser configurados por dos vías: a través del archivo config.yml en la carpeta config/ del proyecto, o a través del archivo services.yml que se encuentra en la carpeta config/ dentro del bundle. En esta configuración solo se especifican los servicios que provee un bundle, pudiéndose consumir dichos servicios desde cualquier parte de la aplicación. La configuración se registra en caché desde la primera petición.

Un servicio es cualquier objeto PHP que realiza alguna tarea global, un objeto creado para un propósito determinado. Cada servicio se utiliza en toda una aplicación, siempre que se necesite la funcionalidad específica que este proporciona. Al explicitar un servicio en alguno de los archivos de configuración también se explicitan los parámetros que debe recibir, así como la ubicación de la clase que lo implementa.

Dada la concepción del mecanismo para la integración entre componentes en Symfony2 es posible el consumo de los servicios que estos brindan desde cualquier bundle de una aplicación. Entre los componentes existe un escaso nivel de dependencia que no queda expuesto en la configuración de estos. Una deficiencia notable en Symfony2 es que sus componentes no declaran sus dependencias, siendo necesario para los desarrolladores inspeccionar el código.

1.8 Valoraciones acerca del soporte a los procesos de seguridad y los mecanismos de integración de Symfony2 y Acaxia

Una vez concluido el estudio realizado, se pudo constatar que el mecanismo provisto por Symfony2 para la Identificación y autenticación carece de la fortaleza que requiere el control de acceso para los SI, puesto que emplea la autenticación HTTP, donde las credenciales del usuario son enviadas al servidor codificadas con un algoritmo fácilmente reversible. El uso de las ACL permite un control eficiente de los usuarios y sus permisos asociados, pero cuando se gestionan grandes cantidades de usuarios y permisos, puede resultar engorroso, pues esta información es configurada de forma manual y directamente en el código fuente de la aplicación.

Además, no se brinda soporte al proceso de Auditoría, de modo que se dificulta el control de cada una de las acciones de los usuarios en el SI. Finalmente, se identificó que el componente no permite la realización de un proceso de Autenticación centralizado, por lo que en entornos donde coexistan varios SI, el usuario necesita presentar sus credenciales para cada uno de los sistemas. Es importante destacar que el componente cuenta con elementos que brindan la posibilidad de

personalizar la estrategia de seguridad a seguir, esta potencialidad puede ser aprovechada para la integración.

Una posible solución para fortalecer el soporte a los procesos de seguridad del marco de trabajo Symfony2 sería su integración al Sistema Integral de Seguridad Acaxia, teniendo en cuenta las funcionalidades que este proporciona. Acaxia garantiza la gestión de la seguridad de forma centralizada en un entorno de varias aplicaciones. De esta forma se fortalece una parte importante de la arquitectura de seguridad como es la protección de la información almacenada en los sistemas que utilicen sus servicios. Provee la utilización de estándares internacionales establecidos para los procesos de seguridad: SAML (Identificación y autenticación, Autorización) y RBAC (Autorización). Admite el registro de los eventos generados a través de las diferentes acciones que realizan los usuarios en el sistema.

Los mecanismos de integración de Acaxia y Symfony2 difieren en cuanto a las reglas para la definición en integración de los componentes. Adicionalmente Symfony2 no contempla de forma explícita las dependencias de sus componentes, lo cual puede resultar en un reto para la integración. A pesar de ello, poseen características similares, como el uso de los contextos de ejecución y las variables globales de PHP, que pueden ser utilizadas para la integración.

1.9 Modelo de desarrollo de software

Todo proceso de desarrollo de software necesita ser estructurado, planificado y organizado para su culminación exitosa. La presente investigación se desarrolla en el Departamento de Tecnologías del CEIGE, por lo cual se ajusta a su modelo de desarrollo de software. El modelo describe el ciclo de vida de sus proyectos, con la incorporación de los distintos subprocesos dictados por el Nivel II de CMMI[8] obtenido por la UCI y reconocido por el SEI[9] como aval de la calidad de su proceso de desarrollo de software.

Características

Orientado a componentes: sistema compuesto por varios componentes desarrollados de manera independiente pero que, al unirlos, constituyen dicho sistema.

[8]Integración de Modelos de Madurez de Capacidades (CMMI por sus siglas en inglés). Es un modelo para la mejora y evaluación de procesos para el desarrollo, mantenimiento y operación de sistemas de software.

[9]Instituto de Ingeniería de Software (SEI por sus siglas en inglés). Es un instituto federal estadounidense de investigación y desarrollo, constituye un referente de Ingeniería de Software.

Iterativo e incremental: plantea que ese mismo sistema puede tener tantas iteraciones como sean necesarias y que en cada versión se obtendrá un incremento y mejoramiento de las funcionalidades del sistema.

Flujo de trabajo

El modelo seleccionado establece las diferentes fases por las que se debe transitar durante el desarrollo de un producto de software y los distintos artefactos que se generan en cada una de ellas. Las fases definidas son: Inicio o Estudio preliminar y Desarrollo. La solución que se desea implementar se enmarca en la fase de Desarrollo, que tiene como objetivo obtener un sistema que satisfaga las necesidades de los clientes y usuarios finales (CEIGE, 2012). En el marco de la presente investigación el flujo de trabajo comprende las fases que se muestran en la Figura 6 y se describen a continuación.

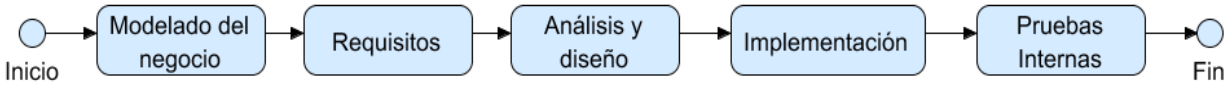


Figura 6. Disciplinas de la fase de Desarrollo (elaboración propia).

La fase **Modelado del Negocio**, es la destinada a comprender los procesos de negocio que se desean informatizar con el fin de garantizar que la solución cumpla las expectativas del cliente. En esta disciplina se puede emplear el modelo de proceso o el modelo de dominio; en la presente solución se emplea este último, debido a que para el sistema que se desea implementar no existen procesos de negocio definidos, por lo que el artefacto generado para esta disciplina es el modelo conceptual. En la fase **Requisitos** se desarrolla un modelo del sistema que se va a construir. Incluye un conjunto de artefactos que describen todas las interacciones que tendrán los usuarios con el software. Se identifican los requisitos funcionales, se describen los mismos y se elaboran los prototipos de interfaces de usuario asociados a ellos. Por último, se identifican los requisitos no funcionales.

Durante la fase **Análisis y diseño** se modela el sistema, de forma tal que satisfaga todos los requisitos definidos. Se generan artefactos más orientados a la implementación, estos son: estilo arquitectónico y patrón arquitectónico, que determinan la estructura fundamental de la herramienta; el modelo de diseño, artefacto conformado por los diagramas de clases del diseño con estereotipos web y los diagramas de secuencia; el modelo de datos, que no es aplicable para la presente solución y por último los diseños de casos de prueba. En la fase **Implementación,** a partir de los resultados del análisis y diseño, se implementa el sistema en términos de componentes, es decir, ficheros de código fuente, scripts y ejecutables. Es donde se elabora el

diagrama de componentes, se definen los estándares de codificación a utilizar y se implementan los requisitos funcionales identificados con sus interfaces correspondientes.

En la fase **Pruebas internas** se identifican posibles errores en la documentación y el software, es decir, requisitos que el producto debería cumplir y que aún no los cumple. En esta etapa se realizan las pruebas de caja blanca y pruebas de negra, y además, se resuelven las no conformidades detectadas durante la ejecución de estas pruebas. Dadas las características de la solución, los investigadores decidieron realizar únicamente pruebas de caja blanca.

1.10 Tecnologías y herramientas para el desarrollo

Las tecnologías que se emplean para el desarrollo de la solución han sido definidas por el Departamento de Tecnologías del CEIGE en el documento Vista Entorno de Desarrollo Tecnológico.(DT, 2013)

1.10.1 Lenguaje para el modelado

UML

"El Lenguaje Unificado de Modelado (UML por sus siglas en inglés) es un lenguaje para especificar, visualizar, construir y documentar los artefactos de los sistemas software, así como para el modelado del negocio y otros sistemas no software" (Larman, 2005) . UML captura decisiones y conocimiento sobre los sistemas que se deben construir (James Rumbaugh et al, 2013). Ayuda al usuario a entender realmente la tecnología y basado en ello, tomar decisiones acerca de si es factible invertir en el proyecto. De este modo se reducen costos y tiempo.

UML, como estándar de modelado, posibilita la concurrencia, ya que es un lenguaje distribuido y adecuado a las necesidades de conectividad en la actualidad. Reemplaza a decenas de notaciones empleadas con otros lenguajes. Modela estructuras complejas. Las estructuras más importantes que soporta tienen su fundamento en las tecnologías orientadas a objetos, tales como objetos, clase, componentes y nodos. Emplea operaciones abstractas como guía para variaciones futuras, añadiendo variables si es necesario. Permite modelar el comportamiento del sistema: casos de uso, diagramas de secuencia y de colaboraciones.

1.10.2 Lenguaje de programación del lado del servidor

PHP 5.3.10

PHP es un lenguaje de scripting de propósito general ampliamente utilizado, que es especialmente adecuado para el desarrollo web y puede ser embebido en páginas HTML. La

mayoría de su sintaxis es similar a C, Java y Perl. Su función es permitir escribir a los creadores de páginas web, páginas dinámicas de una manera rápida y fácil. Permite aplicar principios de la Programación Orientada a Objetos (POO). Presenta interfaces para una gran cantidad de sistemas de base de datos, así como bibliotecas incorporadas para muchas tareas web habituales. A partir de PHP5 no requiere la definición de tipos de variables. (Colectivo de autores, 2005)

1.10.3 Herramientas utilizadas

PostgreSQL 9.1.0

PostgreSQL es un sistema de base de datos objeto-relacional de almacenamiento y manipulación de datos muy común. Permite la creación de tipos propios. Incorpora una estructura de datos array. Incorpora funciones de diversa índole (manejo de fechas, geométricas, orientadas a operaciones con redes). Permite la declaración de funciones propias, así como la definición de disparadores. Soporta el uso de índices, reglas y vistas. Incluye herencia entre tablas (aunque no entre objetos), por lo que a este gestor de bases de datos se le incluye entre los gestores objeto-relacionales. Permite la gestión de diferentes usuarios, como también los permisos asignados a cada uno de ellos. Soporta el manejo de elevadas cantidades de datos y una alta concurrencia de usuarios accediendo a ellos a la vez. (PostgreSQL, 2014)

Apache 2.2.22

Apache es un servidor de aplicaciones cuya adaptabilidad, robustez y estabilidad lo han hecho popular desde 1996. Proporciona un servidor seguro, eficiente y extensible que provee servicios HTTP en sincronía con los estándares actuales. Es una tecnología gratuita de código abierto y multiplataforma. Puede ser adaptado a diferentes entornos y necesidades, lo que denota su modularidad. (The Apache Software Foundation, 2014)

Netbeans 7.3

Netbeans es un Entorno de Desarrollo Integrado (IDE por sus siglas en inglés) libre y de código abierto, con variados módulos. Brinda las herramientas necesarias para el desarrollo de aplicaciones profesionales de escritorio, web y aplicaciones móviles con la plataforma Java, así como con PHP y otras. Simplifica la gestión de grandes proyectos con el uso de diferentes vistas, asistentes de ayuda, y estructurando la visualización de manera ordenada. (Oracle Corporation and its affiliates, 2014)

1.10.4 Herramienta de Ingeniería de Software Asistida por Computación

Visual Paradigm 8.0

Visual Paradigm es una herramienta de Ingeniería de Software Asistida por Computadora (CASE por sus siglas en inglés) que utilizando UML como lenguaje de modelado, permite la captura, diseño, gestión y documentación de los artefactos generados durante el proceso de desarrollo de software. Es multiplataforma y posee facilidades de interoperabilidad con otras aplicaciones. (VisualParadigm, 2011)

1.10.5 Marcos de trabajo y librerías para el desarrollo

Marco de trabajo Sauxe 2.3

Sauxe es un marco de trabajo que contiene un conjunto de componentes reutilizables. Provee la estructura genérica y el comportamiento para una familia de abstracciones, con lo que se logra una mayor estandarización, flexibilidad, integración y agilidad en el proceso de construcción del software. Centra su desarrollo en los requerimientos funcionales, las interfaces de usuario y la lógica del negocio. El marco de trabajo brinda funcionalidades de seguridad, auditoría, interoperabilidad, concurrencia, administración de transacciones, entre otras. Es una solución que sigue el principio de soberanía tecnológica, facilidad de mantenimiento, reusabilidad e integración. (Torres, 2013)

Marco de Trabajo Symfony2

El marco de trabajo Symfony2 es una completa librería de clases escritas en PHP. Provee una arquitectura, componentes y herramientas que le permiten a los desarrolladores construir aplicaciones de forma sencilla y rápida. Está basado en la experiencia acumulada, utilizando las mejores prácticas de desarrollo y reutiliza librerías de terceros. (Figueredo, 2009)

1.11 Conclusiones del capítulo.

El estudio realizado en el presente capítulo permitió identificar los principales conceptos asociados a la seguridad en los SI, principalmente los procesos de Identificación y autenticación, Autorización y Auditoría.

El análisis del componente de seguridad de Symfony2 y de Acaxia mostró que en Symfony2 el soporte a los procesos de Identificación y autenticación y Autorización es limitado y no se brinda soporte a la Auditoría; mientras que Acaxia soporta los tres procesos de forma integral.

El análisis de los mecanismos de integración de ambos contribuyó a identificar las reglas que cada uno establece para la definición e integración de los componentes y finalmente a determinar los elementos necesarios para realizar la integración.

La adopción del modelo de desarrollo del CEIGE unido a la selección de las herramientas según lo establecido en el documento Vista Entorno Desarrollo Tecnológico sirvió de guía durante el proceso de desarrollo de software.

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

Introducción

En el presente capítulo se realiza una propuesta de solución para la integración, mediante el desarrollo de un componente de software. En el modelado del negocio no se utiliza modelo de procesos debido a que para el sistema a desarrollar no existen procesos de negocio definidos, por lo que se emplea el modelo de dominio. En este modelo se describen los aspectos del dominio, identificándose los principales elementos físicos o lógicos del negocio que ayuden a entender el problema. Se especifican los requisitos funcionales y no funcionales con que contarán los componentes. Se muestran, además, los diferentes artefactos generados durante la fase de análisis y diseño, definiendo el estilo y patrón arquitectónico.

2.1 Descripción de la solución

El proceso de integración basa su funcionamiento en el uso, por parte de ambos marcos de trabajo, de las variables globales de PHP; teniendo en cuenta, además, que cada marco de trabajo tiene su contexto de ejecución manejado por un controlador frontal. Por lo que cada vez que se necesite consumir un servicio de Acaxia, es necesario que Sauxe tome el control de la ejecución para que su controlador frontal interprete la petición. A continuación se describe la solución propuesta para la gestión de los tres procesos de seguridad en la integración.

2.1.1 Identificación y autenticación

En el proceso de Identificación y autenticación se propone utilizar los conceptos definidos por Symfony2, de modo que facilite la familiarización de los desarrolladores con la solución. Partiendo de este principio, se personalizan los elementos que forman parte del componente de seguridad de este marco de trabajo asociados al proceso.

Cuando comienza la ejecución de la aplicación de Symfony2, en su controlador frontal se accede al fichero de configuración desde donde se accede al SAML para realizar el proceso de autenticación centralizada a través del PS y el PI. Posteriormente, el Escuchador será el encargado de consultar al Contexto de Seguridad de Symfony2 para verificar si posee un token autenticado cada vez que el usuario realice una petición. En caso de no estarlo, le solicita al Contexto de Seguridad que lo autentique. Este consulta, entre sus Proveedores de Autenticación, cuál es el que soporta el tipo de token que maneja el Escuchador.

En el Proveedor de Autenticación se realiza la carga de los datos del usuario mediante el Proveedor de Usuarios y finalmente se autentica. La carga de los datos de los usuarios se realiza

mediante el consumo de servicios de Acaxia, para lo que se tiene en cuanta el intercambio de los contextos de ejecución. Una vez culminado el proceso, el Escuchador registra el token autenticado en el Contexto de Seguridad.

2.1.2 Autorización

Una vez concluido satisfactoriamente el proceso anterior, el token autenticado contiene toda la información referida a los roles del usuario definidos en la configuración de seguridad de Acaxia. Este proceso se realizará a través del consumo de un servicio del sistema Acaxia, lo que permite que el componente de seguridad de Symfony 2 se encargue de realizar todas las comprobaciones pertinentes para cada petición que realice dicho usuario.

2.1.3 Auditoría

Para la gestión del proceso de auditoría en la integración se usan diferentes conceptos que provee Symfony2, tales como los eventos y sus respectivos escuchadores. Se definen escuchadores personalizados para los diferentes tipos de eventos que gestiona Symfony2, de forma tal que permitan a los administradores hacer un resumen de las acciones que se realizaron en la aplicación. Cada vez que un usuario realiza una petición al sistema, este registra una traza de evento en la que se recogen datos generales como: nombre de usuario, fecha, hora, ip, rol. A partir de este momento, en correspondencia con la acción que desencadena esta petición, se registran las trazas de funcionalidades, que guardan la información referida a las funcionalidades del sistema que ejecuta el usuario.

En caso de realizarse una operación (inserción, modificación o eliminación) en la base de datos, se registran las trazas de datos. Para toda acción desencadenada se guardan trazas de rendimiento, que están determinadas por el consumo de memoria en la operación realizada por el usuario y el tiempo que tarda el sistema en proveerle una respuesta. Para el consumo de servicios de Acaxia relacionados con las trazas se emplea el mismo mecanismo descrito en el proceso de autenticación.

2.2 Modelo conceptual

Los principales conceptos y relaciones de la solución se presentan en la Figura 7. El presente Modelo conceptual ilustra el funcionamiento de los componentes para la integración del Sistema de Gestión Integral de Seguridad Acaxia al marco de trabajo Symfony2. Uno de los conceptos fundamentales a tener en cuenta es el manejador de contextos, encargado del intercambio de los contextos de ejecución de ambos marcos de trabajo para el consumo de los servicios de

seguridad que brinda Acaxia. Se toman, además, conceptos definidos por Symfony2, como son: el Proveedor de Usuarios, encargado de la lectura de los usuarios, y el Proveedor de Autenticación, responsable del proceso de Autenticación.

Cada usuario tiene asociados certificados de seguridad que lo responsabilizan por sus acciones sobre el sistema. Estas acciones generan determinados eventos, que por parte de Symfony2 son manejados con los Escuchadores, y para ello Sauxe utiliza las trazas. Los eventos pueden ser de varios tipos; para la solución solo se tienen en cuenta los de rendimiento, acción, datos.

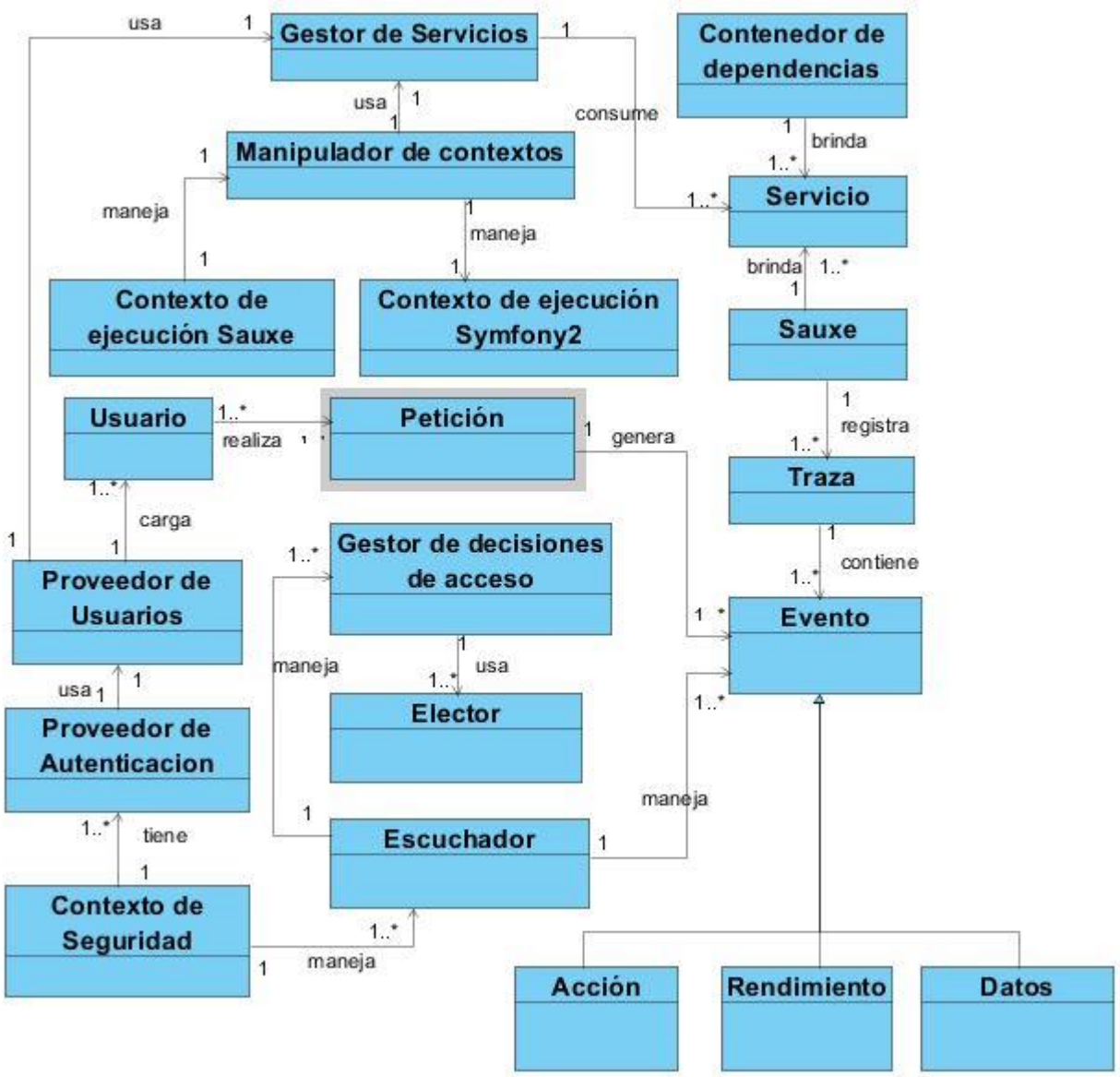


Figura 7. Modelo conceptual (elaboración propia).

2.3 Requisitos de software

La obtención de requisitos es el proceso mediante el cual los interesados en un sistema de software descubren, revelan, articulan y entienden sus requisitos. En muchos casos, se requiere tiempo para llegar a especificar claramente lo que el interesado espera de la aplicación de software, por lo que se hace necesario, por parte de los analistas, el empleo de técnicas que permitan establecer una buena comunicación con los interesados del producto y así lograr la satisfacción del cliente.

A continuación se enuncian las principales técnicas utilizadas durante el proceso de desarrollo para recopilar los requisitos de software.

- ✓ **Tormenta de ideas:** Para ello se estructuró un equipo de trabajo conformado por:
  - Jefe del Departamento de Tecnologías.
  - Arquitecto de software del Departamento Tecnologías.
  - Jefe del proyecto Acaxia.
  - Analista principal del proyecto Acaxia.
  - Los estudiantes involucrados en el desarrollo de los componentes.
- ✓ **Entrevista:** Se consultaron a personas que usan el marco de trabajo Symfony2:
  - Arquitecto de software del Sistema de Informatización de las Fiscalías (SIGEF en lo adelante).
  - Arquitecto de software del Sistema de Informatización de los Tribunales Populares de Cuba (SITPC en lo adelante).
- ✓ **Revisión de investigaciones:** Se estudiaron textos que abordan la seguridad en Symfony2 y la integración de este con el Sistema de Gestión Integral de Seguridad Acaxia.

2.3.1 Requisitos funcionales

Durante la fase de captura de requisitos se identificaron siete requisitos funcionales con los que debe contar la aplicación para satisfacer con eficiencia las necesidades requeridas por el cliente. A continuación se muestra el listado de los mismos.

RF1- Autenticar usuario en el sistema

RF 2- Restringir acceso a los usuarios sobre las funcionalidades del sistema.

RF 3- Restringir acceso a los usuarios sobre las acciones del sistema.

RF 4 - Registrar la información asociada a las funcionalidades ejecutadas en los sistemas.

RF 5 - Registrar la información asociada al inicio y cierre de sesión de los usuarios en los sistemas.

RF 6 - Registrar la información asociada a las operaciones ejecutadas en las bases de datos.

RF 7 - Registrar la información asociada al rendimiento por cada acción ejecutada.

Con el propósito de lograr un mayor entendimiento de los requisitos, se realizó una descripción de cada uno. La Tabla 2 muestra la descripción de uno de ellos. Para consultar el resto de las descripciones, el interesado puede dirigirse al Expediente de Proyecto Acaxia v2.0.

RF1: Autenticar usuario en el sistema.

Precondiciones	Debe existir al menos un usuario registrado en la base de datos.
Flujo de eventos	
Flujo básico	
1	El usuario realiza una petición.
2	El sistema muestra el formulario de Autenticación.
3	El usuario introduce el nombre de usuario y la clave de acceso y presiona el botón **Entrar**.
4	El sistema valida los datos registrados por el usuario (Ver validación 1).
5	El sistema muestra la página principal de la aplicación.
6	Concluye el requisito.
Pos-condiciones	
1	Comienza el proceso de Autorización
Flujos alternativos	
Flujo alternativo 3a. Los datos introducidos por el usuario son incorrectos.	
1.	El sistema muestra el siguiente mensaje de error: “El usuario o la contraseña son incorrectos”.

2.	Regresa al paso 1 del flujo básico.	
Flujo alternativo 3.b. El usuario está accediendo desde una dirección ip no válida.		
1.	El sistema muestra el siguiente mensaje de error: “Usted no tiene acceso al sistema desde esta estación de trabajo, contacte al administrador”.	
2.	Concluye el requisito.	
Flujo alternativo 3.c. El usuario está deshabilitado.		
3.	El sistema muestra el siguiente mensaje de error: “Su usuario está deshabilitado, contacte al administrador”.	
4.	Concluye el requisito.	
Flujo alternativo 3.d. La clave del usuario ha expirado.		
5.	El sistema muestra el siguiente mensaje de error “Su clave ha expirado, contacte al administrador”.	
6.	Concluye el requisito	
Validación		
1	El nombre de usuario solo admite letras y números.	
Relaciones	**Requisitos Incluidos**	N/A
	Extensiones	N/A
Conceptos	**Usuario**	Visibles en la interfaz: Nombre de usuario Clave de usuario Utilizados internamente: N/A
Requisitos especiales	N/A	

Asuntos pendientes	N/A

Tabla 2. RF 1: Autenticar usuario en el sistema (elaboración propia).

Prototipo elemental de interfaz gráfica de usuario RF 1: Autenticar usuario en el sistema.

Figura 8. Interfaz del RF 1: Autenticar usuario en el sistema (elaboración propia).

2.3.2 Requisitos no funcionales

Los requisitos no funcionales del componente para la integración de los servicios de seguridad del Sistema Integral de Seguridad Acaxia con aplicaciones desarrolladas en Symfony2 parten de los definidos por el CEIGE. De estos últimos se tomaron los que eran aplicables para la solución y se adaptaron a las condiciones del componente.

Rendimiento (REN)

Los tiempos de respuesta y velocidad de procesamiento de la información serán rápidos, no mayores de tres segundos para el direccionamiento del usuario a una ruta y dos segundos para la Autenticación del usuario en el sistema.

Seguridad (SEG)

El proceso de Autenticación se realizará mediante un nombre y una clave única para cada usuario. Una vez autenticado el usuario se podrá restringir su acceso a determinados recursos y funcionalidades. La restricción de acceso evita que se realicen acciones no autorizadas o que puedan afectar la integridad de los datos. Además, se contará con un registro de las acciones de los usuarios en el sistema.

Soporte (SOP)

El componente contará antes de su puesta en marcha con un período de pruebas, se le dará mantenimiento, configuración y se brindará el servicio de instalación.

2.4 Arquitectura de software.

La Arquitectura de Software es una disciplina que intenta contrarrestar efectos negativos, como, presiones en los tiempos de ejecución de un proyecto, ausencia de métodos controlados para cumplir con los objetivos planteados, continuos cambios en las funcionalidades requeridas y documentación pobre o inexistente. Esta disciplina ocupa un rol significativo en la estrategia de negocio de una organización que basa su funcionamiento en un SI. Define el estilo y el patrón arquitectónico a utilizar en la implementación del software.

Para el desarrollo del componente se tuvo en cuenta que el Sistema de Gestión Integral de Seguridad Acaxia y el marco de trabajo Symfony2 poseen una arquitectura en capas e implementan el patrón arquitectónico Modelo Vista Controlador, estos se describen a continuación.

Estilo arquitectónico:

- ✓ **Capa de presentación:** esta es la capa encargada de elaborar y mostrar las interfaces de los usuarios. La capa de presentación está compuesta principalmente por el marco de trabajo ExtJS y centra su desarrollo en tres componentes fundamentales archivos JS, archivos CSS, páginas clientes y páginas servidoras.
- ✓ **Capa de control o negocio:** En esta capa se encuentra ubicado el marco de trabajo Zend Framework y es en la que se implementan las clases controladoras.
- ✓ **Capa de acceso a datos:** En esta capa se implementan las clases del modelo, también es la encargada de comunicarse con el gestor de base de datos mediante el marco de trabajo para la persistencia de datos Doctrine.
- ✓ **Capa de datos:** en esta capa se encuentran los archivos de configuración XML y el gestor de base de datos.

En la solución solo se incide en la capa de control, pues así lo demanda la implementación de cada uno de los procesos, debido a que todos los elementos utilizados para el desarrollo del componente pertenecen a ella. Para la Identificación y autenticación junto a la Autorización se emplea el controlador frontal de Symfony2. En la Autorización se tienen en cuenta, además, los componentes definidos por el marco de trabajo para la restricción del control de

acceso. La Auditoría se implementa mediante la escucha de los eventos generados por los elementos pertenecientes al negocio.

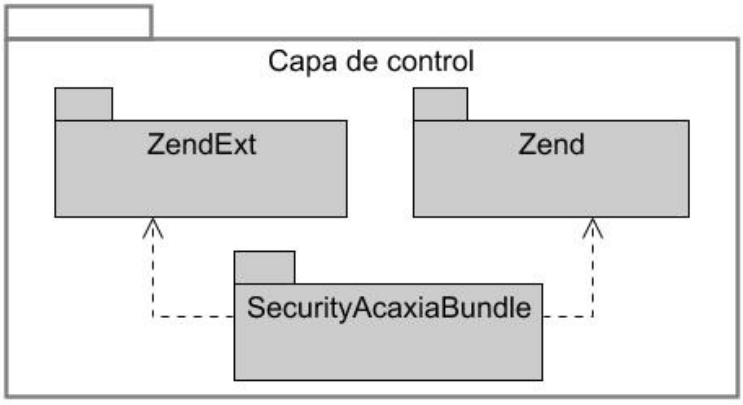


Figura 9. Capa de la arquitectura en que incide la solución (elaboración propia).

Patrón arquitectónico:

El patrón arquitectónico Modelo-Vista-Controlador se emplea en las dos primeras capas de la arquitectura descrita anteriormente. Este patrón es el encargado de separar los datos de la aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos.

Modelo: está compuesto por datos, reglas de negocio y las funcionalidades correspondientes para la comunicación con marco de trabajo para la persistencia de datos Doctrine.

Vista: muestra la información del modelo al usuario.

Controlador: gestiona las entradas y las respuestas del sistema al usuario.

2.5 Patrones de diseño:

A continuación se enuncian los diferentes patrones de diseño utilizados durante el proceso de desarrollo de la solución.

Patrones GRASP:

Experto: se evidencia el uso de este patrón en todas las clases a utilizar en el componente, pues cada clase conoce su información y es la encargada de implementar las funcionalidades que les corresponde, como por ejemplo, la clase ServiceUserProvider.

Creador: su uso se evidencia en las clases controladoras, pues cada una de ellas se encarga de la creación de objetos de varias clases, como AutenticationProvider y SauxeService, entre otras.

Controlador: las diferentes clases controladoras se encargan de llevar el control de todos los eventos relacionados con el negocio. Implementan las funcionalidades que dan respuesta a las peticiones del usuario, como por ejemplo, la clase controladora Seguridad Services.

Alta Cohesión: el uso de este patrón indica que la información almacenada en las clases debe ser coherente y relacionada con lo que se maneja en dicha clase; se evidencia su uso en todas las clases, como por ejemplo, en Sauxe.

Bajo Acoplamiento: el uso de éste patrón se evidencia en la poca relación existente entre las clases que conforman el componente, por ejemplo, en la clase UserToken.

Patrones GoF:

Mediador: es un patrón de comportamiento, encargado de definir un objeto que encapsula cómo interactúan un conjunto de objetos. Estimula la pérdida de acoplamiento ocultando las referencias explícitas entre los objetos, lo que permite variar su interacción de forma independiente. Éste patrón se puede evidenciar en la clase Access Listener, la cual garantiza la comunicación entre varios objetos de distintas clases, como por ejemplo, Access Decisión Manager, Security Context, UserToken.

El uso de estos patrones garantizó asignar a cada clase la responsabilidad que le corresponde, obtener el menor número de relaciones y dependencias entre clases, y aumentar las posibilidades de reusabilidad de las mismas. Todos estos elementos ayudaron a la confección de un diseño de la solución de gran claridad y rendimiento.

2.6 Modelo de diseño.

2.6.1 Diagramas de clases del diseño

Estos diagramas especifican las diferentes clases que serán utilizadas en el sistema y las relaciones que existen entre ellas. La Figura 10 muestra una versión simplificada del diagrama de clases del requisito Autenticar usuario en el sistema, la versión ampliada se encuentra en el Anexo 7.

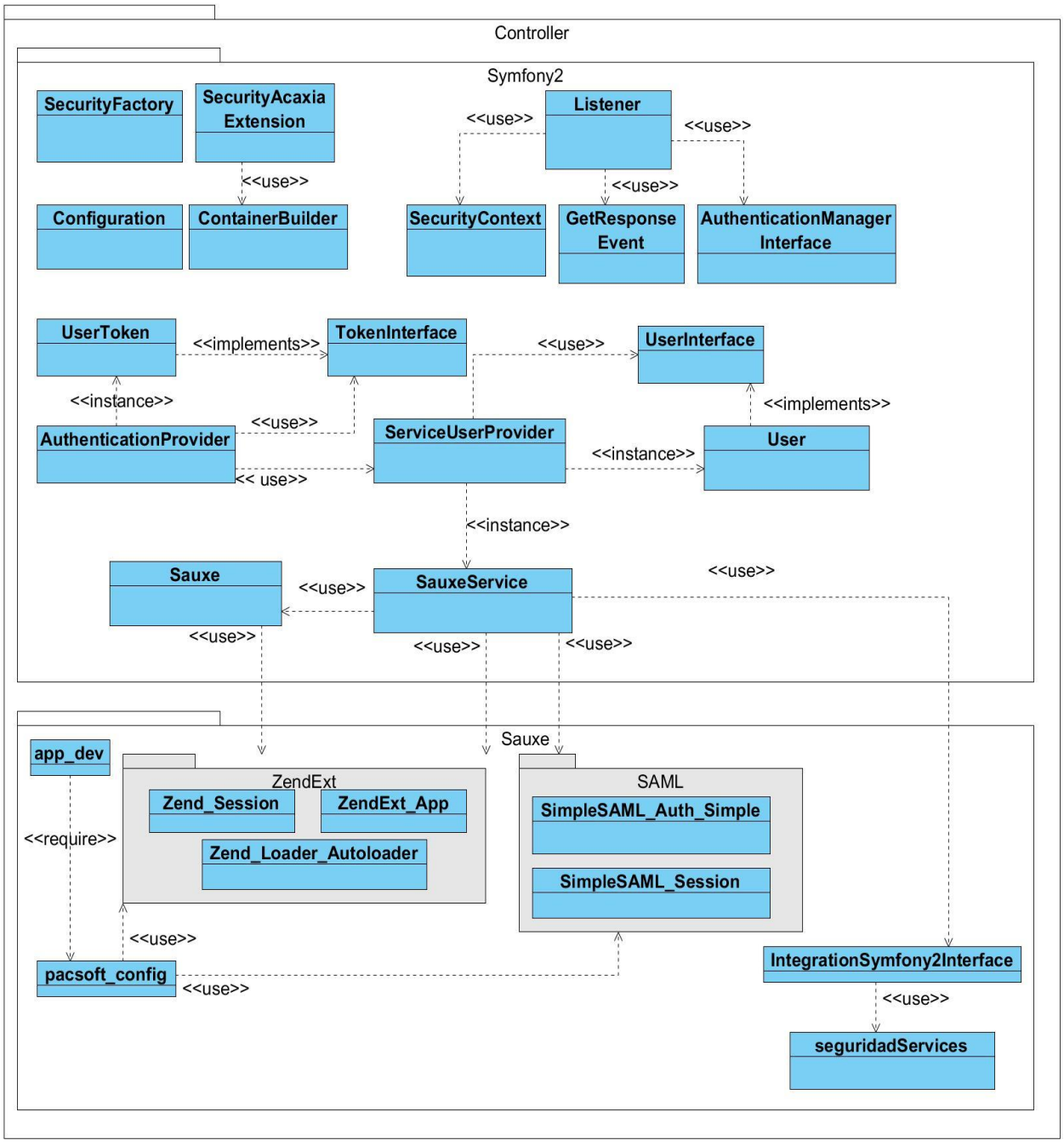


Figura 10. Diagrama de clases RF1.2: Autenticar usuario en el sistema (elaboración propia).

2.6.2 Diagrama de secuencia:

Los diagramas de secuencia describen la interacción entre los objetos de una aplicación y los mensajes recibidos y enviados por los objetos. La Figura 11 representa el diagrama de secuencia del requisito Autenticar usuario en el sistema.

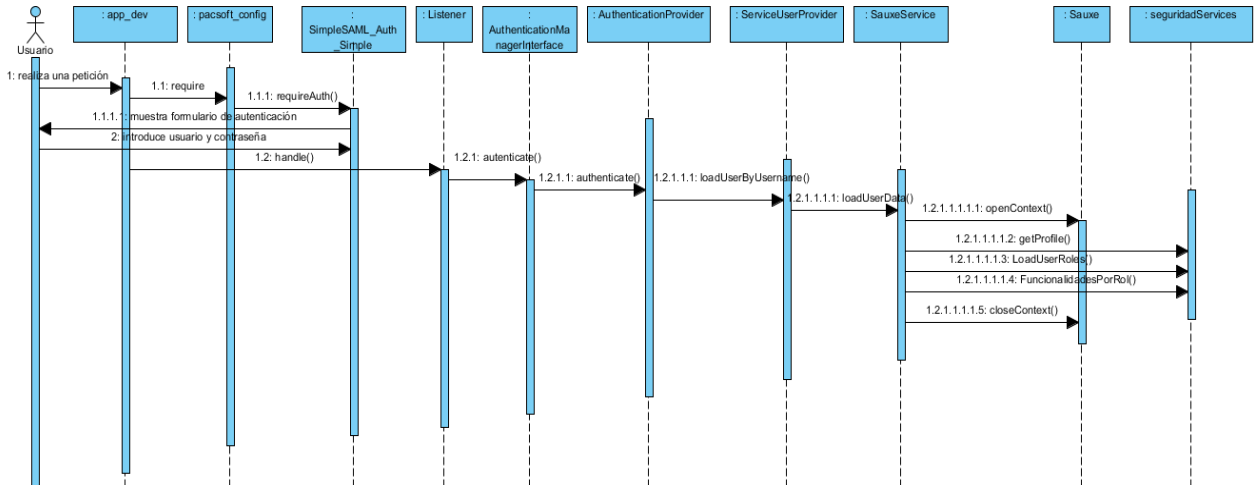


Figura 11. Diagrama de secuencia RF1.2: Autenticar usuario (elaboración propia).

Conclusiones del capítulo.

La descripción de la estrategia a seguir para la implementación del componente a desarrollar, de forma tal que cubra los procesos de seguridad, permitió establecer como vía de solución el empleo de los elementos que intervienen en el componente de seguridad de Symfony2, desde los que se consumirán los servicios brindados por Acaxia.

La elaboración del modelo conceptual de la solución mostró todos los elementos en términos de conceptos que intervienen en la solución. El empleo de técnicas para la obtención de requisitos favoreció la obtención de 7 requisitos funcionales que permiten una mayor comprensión del componente a desarrollar. La definición de los requisitos no funcionales que garantizó el aseguramiento del buen desempeño del componente, según los aspectos definidos para los componentes desarrollados en Sauxe. La selección de una arquitectura de software y el empleo de patrones de diseño permite garantizar que la posterior implementación del componente sea robusta y reutilizable.

La construcción de los diagramas de clases del diseño permitió visualizar las diferentes clases que conforman el componente y las relaciones entre ellas. A través de la creación de los diagramas de secuencia se obtuvo una representación del flujo e interacción existente entre los objetos de la solución.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBA

3.1 Introducción

En el presente capítulo se define el modelo de implementación para poner en práctica el análisis y diseño del componente para la integración de los servicios de seguridad del Sistema Integral de Seguridad Acaxia con aplicaciones desarrolladas en Symfony2 realizado en el capítulo anterior. Se describen los estándares de codificación utilizados para garantizar un buen entendimiento del código. Se muestran los resultados de la aplicación de las métricas Tamaño Operacional de Clases y Relaciones entre Clases al diseño elaborado en el capítulo anterior.

Se utiliza la técnica del camino básico para obtener la complejidad lógica de los procedimientos. Se muestran los resultados de las pruebas funcionales realizadas al componente, para validar que los requisitos identificados fueron implementados correctamente y satisfacen las expectativas del cliente. Además, se evidencia el cumplimiento de la idea a defender planteada, a través de un cuestionario aplicado a dos expertos en el desarrollo de aplicaciones con Symfony2.

3.2 Modelo de implementación

El modelo de implementación describe cómo los elementos del modelo de diseño se implementan en términos de componentes. Describe también cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje de programación utilizados; y cómo dependen los componentes unos de otros. (Acuña, 2013)

3.2.1 Diagrama de componentes

Los diagramas de componentes permiten visualizar con más facilidad la estructura general del sistema y el comportamiento del servicio que estos componentes proporcionan y consumen a través de las interfaces (ANON, 2013a). El siguiente diagrama de componentes permite visualizar la estructura general del sistema y la relación entre los componentes. En el mismo se muestra la integración de los marcos de trabajo, así como el consumo de los diferentes servicios. El consumo de servicios se realiza mediante la clase IoC. Para los procesos de Autorización e Identificación y autenticación el componente desarrollado en Symfony2 (SecurityAcaxiaBundle) consume los servicios del componente de Seguridad. Adicionalmente, en el proceso de Identificación y autenticación utiliza el componente SAML. Por otra parte, para el proceso de Autorización se emplea, además, el componente Session. Con el objetivo de garantizar el proceso de Auditoria, el componente Security Acaxia Bundle se integra con el componente Trazas, que a su vez utiliza

el módulo Trace, que forma parte de Zend Ext, el núcleo de Sauxe. El componente App, se invoca desde SecurityAcaxiaBundle para la instanciación de Sauxe.

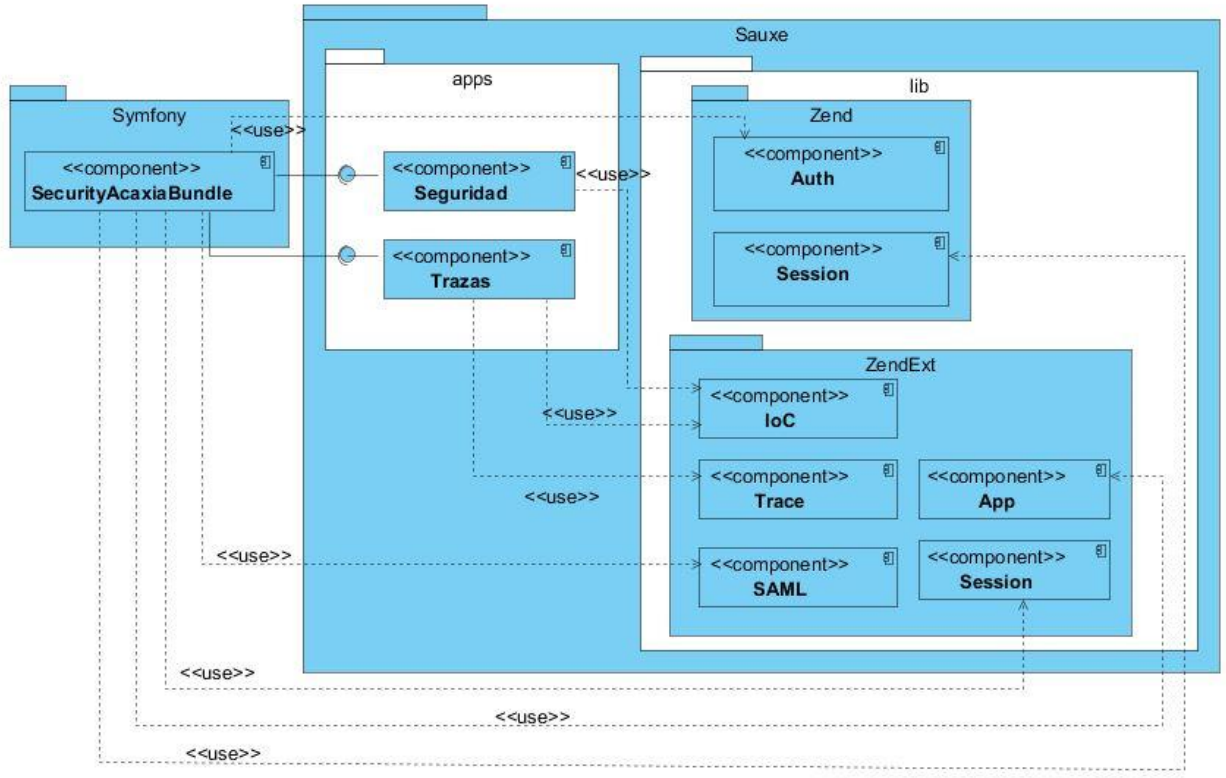


Figura 12. Diagrama de componentes (elaboración propia).

3.3 Estándares de codificación

La armonía en el código fuente de una aplicación es un factor fundamental a tener en cuenta en su implementación, dado que no en todos los casos es el propio programador quien realiza el mantenimiento del software. Este además es un aspecto a tener en cuenta para facilitar la reutilización del código. Durante el desarrollo de la solución propuesta, se emplean algunos de los estándares y normas de codificación propuestos como parte de la Línea de Arquitectura determinada para el proyecto ERP[10] Cuba, los cuales se muestran a continuación.

- **Notación Húngara:** definir prefijos para cada tipo de datos y según el ámbito de las variables. La idea de esta notación es la de dar mayor información al nombre de la variable,

[10]Sistema de Planificación de Recursos Empresariales (por sus siglas en inglés ERP). Son sistemas informáticos destinados a la administración de recursos en una organización.

método o función, definiendo en ella un prefijo que indique su tipo de dato o ámbito. (Sperberg, 2013)

- **Notación Pascal Casing:** Establece que el primer carácter de todas las palabras se expresa en mayúscula y el resto de los caracteres en minúscula. (ANON, 2013)
- **Notación Camel Casing:** Define que el primer carácter de todas las palabras, excepto la primera, se expresa en mayúscula y el resto de los caracteres en minúscula. (ANON, 2013b)

3.4 Resultados obtenidos de la aplicación de las métricas para validar el diseño

Para comprobar el adecuado diseño de las clases y el nivel de reutilización de las mismas se aplicaron las métricas Tamaño operacional de clase (TOC) y Relaciones entre clases (RC), propuestas por Lorenz y Kid.

Tamaño Operacional de Clase**:** esta métrica se determina por el número total de operaciones que están encapsuladas dentro de la clase. Grandes valores de esta medida muestran que la clase puede tener demasiada responsabilidad, lo cual reducirá la reusabilidad de la misma y complicará su implementación. Por otro lado, en cuanto menor sea el valor medio para el tamaño, más probable es que las clases tengan menos responsabilidad y complejidad y más nivel de reutilización. (Vázquez et al, 2014)

Nombre de la clases	Cantidad de operaciones	Responsabilidad	Complejidad	Reutilización
SecurityFactory	1	Baja	Baja	Alta
Listener	1	Baja	Baja	Alta
Configuration	1	Baja	Baja	Alta
SecurityAcaxiaExtension	1	Baja	Baja	Alta
seguridadServices	3	Baja	Baja	Alta
User	1	Baja	Baja	Alta
ServiceUserProvider	3	Baja	Baja	Alta
Sauxe	3	Baja	Baja	Alta
SauxeService	1	Baja	Baja	Alta

BaseDatosListener	4	Baja	Baja	Alta
FuncionalidadesListener	3	Baja	Baja	Alta
Services	5	Baja	Baja	Alta

Tabla 3. Aplicación de la técnica TOC (elaboración propia).

De un total de 12 clases analizadas se obtuvo un promedio de 2.25 operaciones. A continuación se exponen los valores de los umbrales necesarios para evaluar las métricas.

Clasificación	Valores de los umbrales
Bajo	<=Promedio de operaciones(PO)
Medio	>PO y <=2*PO
Alto	>2*PO

Tabla 4. Umbrales para evaluar la métrica TOC (elaboración propia).

Al analizar los resultados obtenidos luego de aplicar el instrumento de medición de la métrica TOC, se puede concluir que el diseño propuesto para el desarrollo del componente posee una calidad aceptable. Los atributos de calidad se encuentran en un nivel satisfactorio en la mayoría de las clases; de manera que se puede observar la alta reutilización (elemento clave en el proceso de desarrollo de software), y baja responsabilidad y complejidad. Este resultado promueve el bajo acoplamiento entre las clases, facilitando la implementación al desarrollador.

Relaciones entre Clases**:** se calcula a partir de la cantidad de relaciones existentes entre las clases contenidas en el diseño. El número de dependencias es directamente proporcional al nivel de acoplamiento, a la complejidad del mantenimiento y a la cantidad de pruebas a realizar sobre las clases. Además, el valor de esta métrica es inversamente proporcional al grado de reutilización de las clases evaluadas. (Vázquez et al, 2014)

Nombre de la clases	Cantidad de Relaciones de Uso	Acoplamiento	Complejidad Mantenimiento	Reutilización	Cantidad de Pruebas
SecurityFactory	0	Ninguno	Baja	Alta	Baja
Listener	3	Alto	Media	Media	Media

Configuration	0	Ninguno	Baja	Alta	Baja
SecurityAcaxiaExtension	1	Bajo	Baja	Alta	Baja
seguridadServices	1	Bajo	Baja	Alta	Baja
User	2	Medio	Baja	Alta	Baja
ServiceUserProvider	4	Alto	Media	Media	Media
Sauxe	2	Medio	Baja	Alta	Baja
SauxeService	5	Alto	Alta	Baja	Alta
BaseDatosListener	1	Bajo	Baja	Alta	Baja
FuncionalidadesListener	1	Bajo	Baja	Alta	Baja
Services	1	Bajo	Baja	Alta	Baja

Tabla 5. Aplicación de la métrica RC (elaboración propia).

De un total de 12 clases analizadas se obtuvo un promedio de 1.75 asociaciones de uso. A continuación se exponen los valores de los umbrales necesarios para evaluar las métricas.

Clasificación	Valores de los umbrales
Bajo	<=Promedio de asociaciones de uso(PAU)
Medio	> PAU y <=2* PAU
Alto	>2* PAU

Tabla 6. Umbrales para evaluar la métrica RC (elaboración propia).

De los resultados obtenidos en la evaluación de la métrica RC se puede deducir que el diseño del componente tiene una calidad aceptable, teniendo en cuenta que el 41% de las clases presentes en el diseño cuentan con una baja cantidad de relaciones de uso. Se puede observar que las clases promueven un bajo nivel de acoplamiento, así como de la complejidad de mantenimiento; igualmente la cantidad de pruebas a realizar representan un bajo por ciento. En consecuencia, el grado de reusabilidad es mayor. En sentido general el resultado de la aplicación de estas métricas demuestra que las clases no se encuentran sobrecargadas en responsabilidad, existe, además, bajo acoplamiento entre las mismas y presentan un alto nivel de reutilización.

Indican, también, que el diseño no es complejo, pues la complejidad de mantenimiento es baja, así como la cantidad de pruebas a realizar. De esta forma, los resultados arrojados sustentan la calidad del diseño.

3.5 Certificación de los artefactos generados

La Revisión Técnica Formal (RTF en lo adelante): es el filtro más efectivo para garantizar la calidad. Se puede definir como una actividad llevada a cabo por los ingenieros del software con el propósito de buscar y reparar errores en la actividad actual antes de pasar a la siguiente actividad dentro de la planificación (García 2005). Se define como un proceso manual que involucra al cliente y el equipo de desarrollo, cuyo objetivo fundamental es detectar errores que afecten el buen funcionamiento del sistema (Sommerville 2005).

Con el objetivo de validar los artefactos generados a lo largo del proceso de desarrollo de software, que forman parte de las salidas de cada una de las fases que define el modelo de desarrollo adoptado por los autores, se aplicó la técnica RTF. Para ello la analista principal del proyecto Sauxe, perteneciente al Departamento de Tecnologías, emitió una carta de aceptación donde certifica que los artefactos cumplen con todos los requerimientos establecidos para el Departamento.

3.6 Pruebas de software

Las pruebas de software son empleadas para determinar el comportamiento de un sistema ante determinados escenarios que pueden ocurrir en tiempo de ejecución. Constituyen una serie de actividades que se realizan con el propósito de encontrar los posibles fallos de implementación, calidad o usabilidad de una solución informática. Existen varios tipos de pruebas que se pueden realizar con este fin, entre ellas se encuentran las de caja blanca, que fueron las elegidas para ser aplicadas al componente desarrollado. Con su aplicación es posible asegurar que las operaciones internas se ajustan a las especificaciones, a través de la elaboración de casos de prueba basados en el diseño procedimental del componente.

En la presente investigación para realizar las pruebas de caja blanca se emplea la técnica del camino básico, es una técnica que 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 casos de prueba obtenidos garantizarán que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa. (González, 2012)

Para utilizar esta técnica es necesario el cálculo de la complejidad ciclomática del código fuente que vaya a ser analizado, para lo cual se deben enumerar las sentencias de código del mismo y elaborar el grafo de flujo de la funcionalidad (Figura 13). Las sentencias enumeradas del método handle se presentan en la Figura 14.

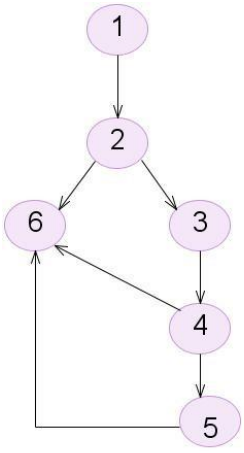


Figura 13. Grafo de flujo del método handle (elaboración propia).

```
public function handle(\Symfony\Component\HttpKernel\Event\GetResponseEvent $event) {
    $request = $event->getRequest();
    if (null !== $token = $this->securityContext->getToken()) {
        $token = new \UCI\SecurityAcaxiaBundle\Security\Authentication\Token\UserToken();
        try {
            $authToken = $this->authenticationManager->authenticate($token);
            $this->securityContext->setToken($authToken);
        } catch (AuthenticationException $failed) {
            $entryPoint = new \UCI\SecurityAcaxiaBundle\Security\Firewall\EntryPoint();
            $entryPoint->start($request, $failed);
            $this->securityContext->setToken(null);
        }
    }
}
```

Figura 14. Código del método handle (elaboración propia).

Fórmulas para calcular la complejidad ciclomática:

V (G) = (A - N) + 2

Donde “A” es la cantidad de aristas y “N” la cantidad de nodos.

V (G) = (7- 6) + 2

V (G) = 3

V (G) = P + 1

Siendo “P” la cantidad de nodos predicados (son los nodos de los cuales parten dos o más aristas).

V (G) = 2 + 1

V (G) = 3

V (G) = R

Donde “R” representa la cantidad de regiones en el grafo.

V (G) = 3

Con el cálculo efectuado mediante las fórmulas se obtiene el mismo valor, concluyendo que la complejidad ciclomática del código es 3, esto significa que existen 3 posibles caminos por donde el flujo puede circular; este valor representa el límite mínimo del número total de casos de pruebas para el procedimiento tratado.

No	Caminos
1	1 , 2 ,6
2	1 , 2 , 3, 4 , 5 , 6
3	1 , 2 ,3 , 4 ,6

Tabla 7. Posibles caminos que puede tomar el flujo (elaboración propia).

Posteriormente, de haber determinado los caminos básicos, se procede a ejecutar los casos de pruebas para cada uno de estos. Para definir los casos de prueba es necesario tener en cuenta:

- **Descripción:** se describe el caso de prueba y de forma general se tratan los aspectos fundamentales de los datos de entrada.
- **Condición de ejecución:** se especifica cada parámetro para que cumpla una condición deseada y así ver el funcionamiento del procedimiento.
- **Entrada:** se muestran los parámetros que serán la entrada al procedimiento.
- **Resultado esperado**: se expone el resultado esperado que debe devolver el procedimiento después de efectuado el caso de prueba.

Caso de prueba para el camino básico # 1.

Descripción: se recibe una petición de un usuario ya autenticado.

Condición de ejecución: en el contexto de seguridad existe un token de seguridad.

Entrada: objeto de tipo GetResponseEvent.

Resultado esperado: no se realiza ninguna operación durante el manejo de la petición.

Caso de prueba para el camino básico # 2.

Descripción: se recibe una petición de un usuario ya autenticado.

Condición de ejecución: en el contexto de seguridad no existe un token de seguridad y el usuario ya se ha autenticado con el SAML.

Entrada: objeto de tipo GetResponseEvent.

Resultado esperado: se añade un token de seguridad al contexto de seguridad que refleja la realización exitosa del proceso de autenticación.

Caso de prueba para el camino básico # 3.

Descripción: se recibe una petición de un usuario ya autenticado.

Condición de ejecución: en el contexto de seguridad no existe un token de seguridad y el usuario no se ha autenticado con el SAML.

Entrada: objeto de tipo GetResponseEvent.

Resultado esperado: la petición se redirecciona al EntryPoint para la autenticación contra el SAML.

3.7 Validación de la solución

Con el objetivo de validar la pertinencia de la solución propuesta y asegurar que el componente satisface las expectativas del cliente, se realizó un pre experimento, el cual permitió validar si para cada uno de los procesos la evaluación del nivel de seguridad luego de la integración es mayor que antes de efectuarla.

Para ello se determinaron diferentes indicadores para de una manera operativa comprobar su estado en la práctica. En su investigación (Baryolo, 2012) emplea un conjunto de indicadores para la validación del CAEM que cubren los requisitos necesarios para evaluar el nivel de

seguridad que proveen las soluciones de control de acceso y está alineado con las regulaciones vigentes en materia de seguridad informática en el país. Es una propuesta generalizable, adecuada a las nuevas exigencias en materia tecnológica y adaptada a las condiciones propias del escenario nacional (Muñoz Dussac 2014). Los autores de esta investigación asumen estos indicadores para evaluar la fortaleza de los procesos de seguridad.

Para su mejor comprensión se pueden dividir en tres grupos en correspondencia con el proceso al que tributan. Cada uno tiene asociado un valor de criticidad, relacionado con el impacto que tienen en la seguridad de un SI y que por tanto evidencia la importancia de su implementación para el fortalecimiento de los procesos de seguridad (Ver Tabla 8).

Proceso	Indicadores	Criticidad
Identificación y autenticación	1. Solución de identificación y autenticación de los usuarios (personas y sistemas).	5
	2. Implementación de estándares para el intercambio de mensajes de identificación y autenticación.	5
	3. Soporte para nuevas técnicas de identificación y autenticación de los usuarios.	4,3
	4. Gestión de identidades de los usuarios.	5
	5. Implementación de métodos criptográficos y mecanismos seguros de comunicación (envío, recepción y almacenamiento de información sensible).	5
	6. Implementación del patrón Single Sign-On.	4,6
	7. Solución para la federación de identidades entre dominios.	4,6
	8. Solución para gestionar la fortaleza y seguridad de las claves de acceso.	5
	9. Empleo de pruebas desafío-respuesta en los eventos donde se necesite determinar cuando el usuario es una persona o no.	4,3

	10. Restricciones de identificación y autenticación basadas en los atributos que identifican a los host en la red.	4,6
Autorización	11. Solución de autorización.	5
	12. Implementa algún estándar para la asignación de privilegios y el intercambio de mensajes de autorización.	4,9
	13. Gestión de privilegios basado en roles.	5
	14. Gestión de privilegios a nivel de usuario.	4,8
	15. Gestión de las estructuras del nivel de base de datos y su relación con los sistemas.	4,7
	16. Gestión de privilegios a nivel de base de datos.	4,7
	17. Gestión de privilegios utilizando criterios o propiedades que identifican a los recursos (objetos, datos, URL, entre otros) para aplicar reglas sobre ellos.	4,9
	18. Gestión de privilegios a nivel de funcionalidades.	5
	19. Gestión de privilegios teniendo en cuenta las estructuras y los dominios organizacionales (entornos multidominios).	4,9
	20. Gestión centralizada de privilegios en entornos multisistemas.	4,7
	21. Mínimo privilegio.	4,9
	22. Administración de cuentas.	5
	23. Federación de privilegios entre dominios.	4,9
	24. Administración de sesiones de usuarios.	5
Auditoría	25. Solución de auditoría.	5
	26. Implementa estándares para el registro y auditoría de los eventos que se ejecutan en los sistemas.	4,9
	27. Sistema de notificación de uso y funcionamiento de los sistemas.	4,6
	28. Auditoría de eventos de ejecución de funcionalidades.	5
	29. Auditoría de eventos de ocurrencia de errores.	4,8
	30. Auditoría de eventos de inicio y cierre de sesión.	4,9
	31. Auditoría de eventos de integración.	4,2
	32. Auditoría de eventos de operaciones a nivel de base de datos.	5

	33. Auditoría de eventos de rendimiento.	4,2
	34. Nivel de respuesta ante fallos, vulnerabilidades o posibles ataques.	4,5
	35. Protección de los log de eventos.	5
	36. Mecanismo para incluir la auditoría de nuevos atributos y eventos.	4,7
	37. Definición de incidente de seguridad.	4,5
	38. Completitud de los datos contenidos en los log de eventos para aplicar técnicas de descubrimiento o minería de proceso.	5
	39. Completitud de los datos contenidos en los log de eventos para realizar análisis enfocados en las violaciones o ataques que afecten la seguridad.	5
	40. Completitud de los datos contenidos en los log de eventos para realizar análisis enfocados en el desempeño de los sistemas.	4,6
	41. Completitud de los datos contenidos en los log de eventos para realizar análisis enfocados en el recorrido histórico de los recursos o los usuarios en los sistemas.	4,9
	42. Capacidad para gestionar los log de eventos de forma centralizada en entornos multisistema.	4,7

Tabla 8. Indicadores para evaluar la fortaleza de los procesos de seguridad (elaboración propia).

Para ello se ha descrito un caso de estudio al que se le dio solución con el desarrollo de un SI usando Symfony2. En un primer momento se soportan los procesos de seguridad mediante la estrategia que brinda este marco de trabajo y se aplica un cuestionario de preguntas cerradas (ver Anexo 6), donde básicamente se le da a cada uno de los indicadores un valor entre cero y cinco (0 significa que el indicador no se implementa y 5 significa que el indicador se implementa eficientemente). Los valores se asignan según la apreciación de los especialistas que participan en el pre-experimento.

En un segundo momento luego de realizar las configuraciones pertinentes para lograr la integración (definidas en el manual de usuario del componente) se aplica nuevamente el cuestionario. En último lugar las evaluaciones finales obtenidas son comparadas. Si el valor luego

de la integración es mayor que el anterior se puede concluir que se ha fortalecido el soporte a los procesos de seguridad.

Para obtener la evaluación final se realiza una suma aritmética de las evaluaciones obtenidas para Identificación y autenticación (P1), Autorización (P2) y Auditoría (P3), como muestra la ecuación:

$$E_{(Final)} = E_{(P1)} + E_{(P2)} + E_{(P3)}$$

$$E_{(P1)} = \sum_{i=1}^{10} C_i * E_{(ei)}$$

$$E_{(P2)} = \sum_{i=11}^{24} C_i * E_{(ei)}$$

$$E_{(P3)} = \sum_{i=25}^{42} C_i * E_{(ei)}$$

Donde C es la criticidad del indicador i y $E_{(ei)}$ es la evaluación dada por el especialista al indicador i.

Caso de estudio Sistema de Gestión Hospitalaria

El Ministerio de Salud Pública (MINSAP) de la República de Cuba es el organismo rector del Sistema Nacional de Salud, encargado de dirigir, ejecutar y controlar la aplicación de la política del Estado y del gobierno en cuanto a la salud pública, el desarrollo de las Ciencias Médicas y la industria médico-farmacéutica. Cubre las funciones de atención médica, asistencia a ancianos y minusválidos, control higiénico-epidemiológico, formación de profesionales y la producción y distribución de medicamentos.

Este ministerio realiza grandes esfuerzos para que los servicios médicos y asistenciales, satisfagan las necesidades de los pacientes y familiares; basados en la búsqueda de herramientas para la mejora de estos procesos. El caso de estudio aborda, la introducción de herramientas gerenciales de apoyo a la mejora de este sector, desde una concepción más integradora. El Sistema de Gestión Hospitalaria le brinda al país la posibilidad de informatizarse en la rama de la salud.

Este sistema contempla la gestión de los principales procesos que tienen lugar en un hospital. Para ello del hospital se controla: nombre, dirección, fax y los teléfonos. En esta institución se

encuentran varias salas, que se identifican con un número. En cada una de ellas se dispone de varias camas, que también poseen un número. Resulta importante llevar un registro del personal que interactúa en el hospital, que pueden ser médicos, pacientes o personal de servicios. Los datos generales de las personas independientemente de la función que realicen en el hospital comprenden su nombre, apellidos, carnet de identidad, edad, dirección particular, teléfono. Los médicos poseen además su especialidad. Del personal de servicio se especifica el tipo de servicio que brinda en el hospital. En el caso de los pacientes se conoce la fecha de admisión. Estos tienen asociado un tratamiento que contempla la enfermedad que padece y los medicamentos que le son administrados. Para el hospital resulta de especial relevancia poseer la información asociada a los resultados que tienen los tratamientos aplicados a los pacientes.

Se requiere que el sistema sea capaz de brindar los siguientes reportes:

- Hospitales dado un lugar.
- Pacientes ingresados en una sala.
- Médicos que posee un hospital.
- Tratamientos que han sido aplicados a un paciente.
- Pacientes menores de edad.
- Pacientes ingresados en el hospital en una fecha determinada.

Resultados de la validación

Los especialistas seleccionados para realizar el pre-experimento fueron los arquitectos principales de los sistemas SIGEF y SITPC respectivamente. Se tuvo en cuenta para su selección su experiencia en el desarrollo de aplicaciones usando Symfony2 y su interacción previa con el Sistema de Gestión Integral de Seguridad Acaxia.

Según su apreciación ambos otorgaron valores entre cero y cinco en función del nivel de cumplimiento por parte del sistema de los indicadores evaluados en el cuestionario. Se realizaron dos evaluaciones, una antes y otra después de la integración. A continuación se muestran los resultados del cuestionario aplicado, donde se evalúa la fortaleza de los procesos de seguridad antes de la integración.

		Arquitecto SIGEF		**Arquitecto SITPC**	
Indicadores	**(C_i)**	**($E_{(ei)}$)**	**($E_{(P1)}$)**	**($E_{(ei)}$)**	**($E_{(P1)}$)**

In 1	5	3	15	2	10
In 2	5	3	15	0	0
In 3	4,3	5	21,5	4	17,2
In 4	5	4	20	3	15
In 5	5	3	15	3	15
In 6	4,6	0	0	0	0
In 7	4,6	3	13,8	0	0
In 8	5	3	15	3	15
In 9	4,3	5	21,5	0	0
In 10	4,6	4	18,4	0	0
Evaluación final			**310,4**		**144,4**

Tabla 9. Evaluación del proceso de Identificación y autenticación antes de la integración (elaboración propia).

		Arquitecto SIGEF		**Arquitecto SITPC**	
Indicadores	**(C_i)**	**($E_{(ei)}$)**	**($E_{(P2)}$)**	**($E_{(ei)}$)**	**($E_{(P2)}$)**
In 11	5	4	20	1	5
In 12	4,9	4	19,6	3	14,7
In 13	5	3	15	3	15
In 14	4,8	3	14,4	4	19,2
In 15	4,7	4	18,8	0	0
In 16	4,7	4	18,8	0	0
In 17	4,9	4	19,6	3	14,7
In 18	5	4	20	1	5
In 19	4,9	3	14,7	0	0

In 20	4,7	5	23,5	0	0
In 21	4,9	3	14,7	5	24,5
In 22	5	3	15	0	0
In 23	4,9	3	14,7	0	0
In 24	5	3	15	0	0
Evaluación final			**243,8**		**98,1**

Tabla 10. Evaluación del proceso de Autorización antes de la integración (elaboración propia).

		Arquitecto SIGEF		**Arquitecto SITPC**	
Indicadores	**(C_i)**	$(E_{(ei)})$	$(E_{(P3)})$	$(E_{(ei)})$	$(E_{(P3)})$
In 25	5	1	5	5	25
In 26	4,9	0	0	5	24,5
In 27	4,6	0	0	5	23
In 28	5	3	15	5	25
In 29	4,8	2	9,6	5	24
In 30	4,9	3	14,7	5	24,5
In 31	4,2	0	0	5	21
In 32	5	0	0	5	25
In 33	4,2	2	8,4	5	21
In 34	4,5	0	0	5	22,5
In 35	5	0	0	5	25
In 36	4,7	2	9,4	5	23,5
In 37	4,5	0	0	0	0
In 38	5	0	0	0	0

In 39	5	2	10	5	25
In 40	4,6	2	9,2	5	23
In 41	4,9	1	4,9	5	24,5
In 42	4,7	0	0	5	23,5
Evaluación final			**193,3**		**86,2**

Tabla 11. Evaluación del proceso de Auditoría antes de la integración (elaboración propia).

El cuestionario aplicado al arquitecto de SIGEF antes de la integración arroja los siguientes resultados para cada uno de los procesos: $\boldsymbol{E_{(P1)} = 310.4}$, $\boldsymbol{E_{(P2)} = 243.8}$, $\boldsymbol{E_{(P3)} = 193.3}$ con lo que se obtiene la evaluación final $\boldsymbol{E_{(Final)} = 747,5}$.

Mientras que para el arquitecto de SITPC los resultados por procesos son: $E_{(P1)} = 144$**,** $E_{(P2)} = 98.1$**,** $E_{(P3)} = 86.2$ a partir de lo que se obtiene una evaluación final $E_{(final)} = 328.3$.

A continuación se muestran los resultados del cuestionario aplicado, donde se evalúa la fortaleza de los procesos de seguridad luego de la integración.

		Arquitecto SIGEF		**Arquitecto SITPC**	
Indicadores	**(C_i)**	**$(E_{(ei)})$**	**$(E_{(P1)})$**	**$(E_{(ei)})$**	**$(E_{(P1)})$**
In 1	5	5	25	4	20
In 2	5	5	25	4	20
In 3	4,3	4	17,2	4	17,2
In 4	5	5	25	4	20
In 5	5	4	20	3	15
In 6	4,6	5	23	5	23
In 7	4,6	5	23	4	18,4
In 8	5	5	25	4	20
In 9	4,3	5	21,5	5	21,5

In 10	4,6	3	13,8	4	18,4
Evaluación final			**437**		**387**

Tabla 12. Evaluación del proceso de Identificación y autenticación después de la integración (elaboración propia).

		Arquitecto SIGEF		**Arquitecto SITPC**	
Indicadores	**(C_i)**	**$(E_{(ei)})$**	**$(E_{(P2)})$**	**$(E_{(ei)})$**	**$(E_{(P2)})$**
In 11	5	5	25	4	20
In 12	4,9	5	24,5	4	19,6
In 13	5	5	25	5	25
In 14	4,8	5	24	5	24
In 15	4,7	5	23,5	4	18,8
In 16	4,7	5	23,5	4	18,8
In 17	4,9	5	24,5	5	24,5
In 18	5	5	25	4	20
In 19	4,9	5	24,5	5	24,5
In 20	4,7	5	23,5	4	18,8
In 21	4,9	5	24,5	5	24,5
In 22	5	5	25	4	20
In 23	4,9	5	24,5	2	9,8
In 24	5	4	20	5	25
Evaluación final			**337**		**293,3**

Tabla 13. Evaluación del proceso de Autorización después de la integración (elaboración propia).

		Arquitecto SIGEF		Arquitecto SITPC	
Indicadores	**(C_i)**	**$(E_{(ei)})$**	**$(E_{(P3)})$**	**$(E_{(ei)})$**	**$(E_{(P3)})$**
In 25	5	5	25	4	20
In 26	4,9	5	24,5	4	19,6
In 27	4,6	5	23	4	18,4
In 28	5	5	25	4	20
In 29	4,8	5	24	4	19,2
In 30	4,9	5	24,5	5	24,5
In 31	4,2	5	21	4	16,8
In 32	5	5	25	4	20
In 33	4,2	5	21	4	16,8
In 34	4,5	5	22,5	1	4,5
In 35	5	5	25	4	20
In 36	4,7	5	23,5	4	18,8
In 37	4,5	0	0	2	9
In 38	5	0	0	4	20
In 39	5	5	25	4	20
In 40	4,6	5	23	4	18,4
In 41	4,9	5	24,5	4	19,6
In 42	4,7	5	23,5	5	23,5
Evaluación final			**380**		**329,1**

Tabla 14. Evaluación del proceso de Auditoría después de la integración (elaboración propia).

El cuestionario aplicado al arquitecto de SIGEF después de la integración arroja los siguientes resultados para cada uno de los procesos: $\boldsymbol{E_{(P1)} = 437}$, $\boldsymbol{E_{(P2)} = 337}$, $\boldsymbol{E_{(P3)} = 380}$ con lo que se obtiene la evaluación final $\boldsymbol{E_{(Final)} = 1174}$.

Mientras que para el arquitecto de SITPC los resultados por procesos son: $E_{(P1)} = 387$, $E_{(P2)} = 293$, $E_{(P3)} = 329.1$ a partir de lo que se obtiene una evaluación final $E_{(final)} = 1009.1$.Las evaluaciones se resumen en los siguientes gráfico de barras (Figuras 15 y 16).

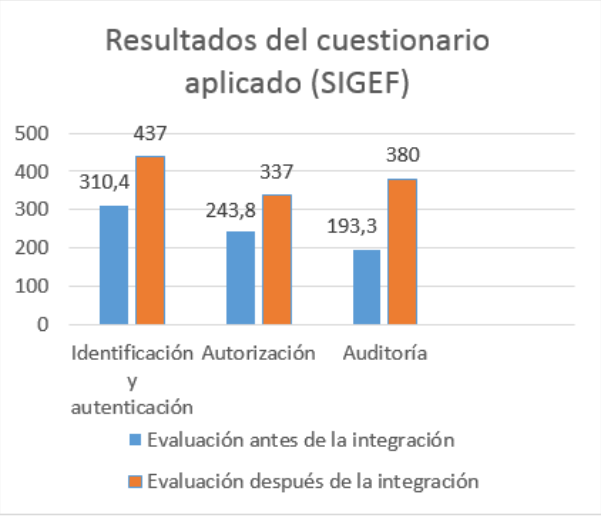


Figura 15. Resultados del cuestionario aplicado al arquitecto de SIGEF (elaboración propia).

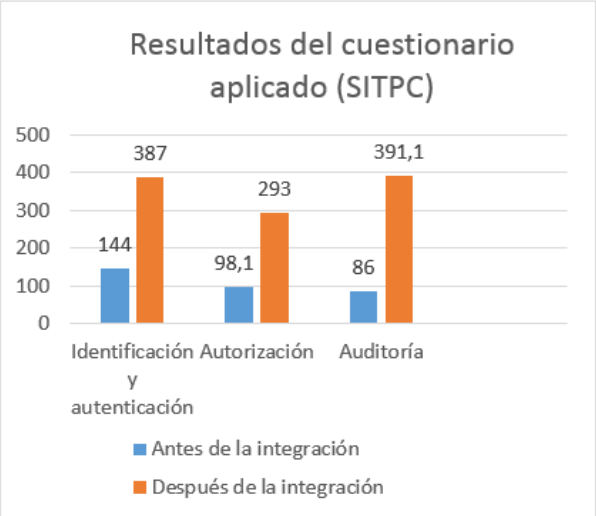


Figura 16. Resultados del cuestionario aplicado al arquitecto de SITPC (elaboración propia).

Una vez aplicados los cuestionarios a los dos especialistas en ambos casos, mostraron una evaluación final posterior a la integración superior a la evaluación final antes de la integración. Por lo que se ha evidenciado que la integración del Sistema de Gestión Integral de Seguridad Acaxia al sistema desarrollado usando Symfony2 que responde al caso de estudio fortalece los procesos de seguridad. Se puede constatar, además, en el gráfico resumen de la Figura 17 que el proceso que más se fortalece es la auditoría.

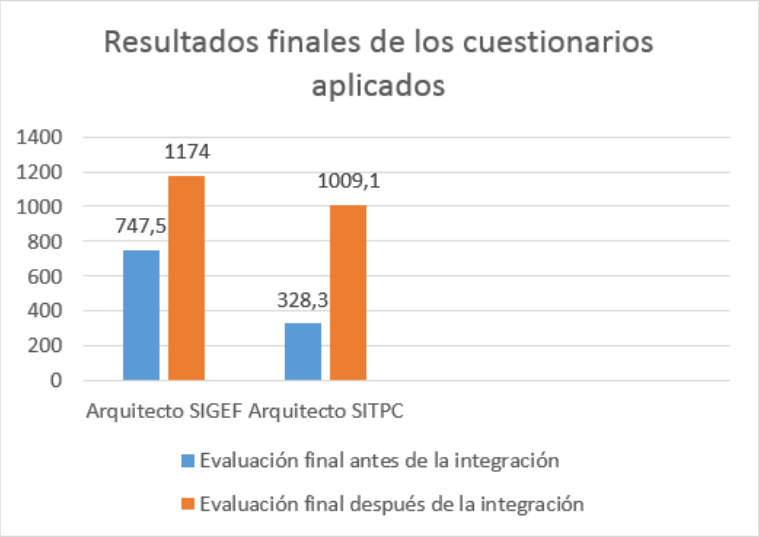


Figura 17. Resumen de evaluaciones finales (elaboración propia).

3.8 Conclusiones del capítulo

La obtención del diagrama de componentes aportó una representación de la estructura general del sistema, que evidencia la proporción y el consumo de servicios. El establecimiento de estándares de codificación aseguró una homogeneidad en la nomenclatura de las clases y métodos con lo que se facilita su comprensión a otros programadores. La validación realizada al diseño propuesto en el capítulo dos mediante el empleo de métricas permitió demostrar que el componente cuenta con un diseño robusto. La aplicación de las pruebas de caja blanca a la solución demostró que esta respondía correctamente a los requisitos identificados a partir de las necesidades del cliente.

Finalmente el desarrollo de un pre-experimento a partir de un caso de estudio, donde se usó como técnica el cuestionario demostró que la integración fortalece el soporte a los procesos de seguridad en las aplicaciones desarrolladas con Symfony2.

CONCLUSIONES GENERALES

Sobre la base del análisis, interpretación y sistematización de las indagaciones teóricas y empíricas, a continuación se presentan las siguientes conclusiones de la investigación:

1. El estudio teórico de los presupuestos actuales en los que se sustenta la seguridad en los SI, el soporte del macro de trabajo Symfony2 y el Sistema de Gestión Integral de Seguridad Acaxia a los procesos de seguridad Identificación y autenticación, Autorización y Auditoría, así como los mecanismos de integración de ambos permitió a los autores de la investigación fundamentar la elaboración de un componente para fortalecer el soporte de las aplicaciones desarrolladas con Symfony2 a estos procesos.
2. El análisis de un conjunto de herramientas y tecnologías permitió definir el marco tecnológico en ajuste a las políticas de migración a software libre por la cual aboga el país.
3. Se demuestra que el desarrollo y puesta en práctica del componente permitió fortalecer el soporte a los procesos de seguridad de las aplicaciones desarrolladas usando Symfony2.

RECOMENDACIONES

- Incluir en el sistema la compartimentación de los recursos, de tal forma que se limite el acceso de los usuarios a la información registrada en dominios diferentes.
- Realizar la integración para el consumo de los servicios asociados a la autenticación por reconocimiento facial.

REFERENCIAS BIBLIOGRÁFICAS

¿Qué es Symfony?, 2013. *symfony.es* [online], [Accessed 7 June 2014]. Available from: http://symfony.es/que-es-symfony

ACHIARY, C., 2005, *Modelo de Política de Seguridad de la Información para Organismos de la Administración Pública Nacional*. 2005. Oficina Nacional de Tecnologías de Información (ONTI).

ANON, 2013a, Diagramas de componentes de UML. [online]. 2013. [Accessed 8 June 2014]. Available from: http://msdn.microsoft.com/es-es/library/dd409390.aspx

ANON, 2013b, BUENAS PRÁCTICAS EN C#. [online]. 2013. [Accessed 8 June 2014]. Available from: http://www.ba-technology.net/index.php/metodologia/91?start=1

BARYOLO, Oiner, 2012, *CAEM: MODELO DE CONTROL DE ACCESO PARA SISTEMAS DE INFORMACIÓN EN ENTORNOS MULTIDOMINIOS*. Doctorado. La Habana : Universidad de las Ciencias Informáticas.

CALA TORRES, Abraham, 2013, Desarrollo de un mecanismo de integración entre componentes para el marco de trabajo Sauxe. *COMPUMAT*. 2013.

CASAL TERREROS, Julio, 2009, *Desarrollo de software basado en componentes*.

CEIGE, 2012, *Modelo de desarrollo de software*. 2012.

COLECTIVO DE AUTORES, 2005, *PHP 5 Power Programming*. Estados Unidos : Pearson Education, Inc. ISBN 0-131-47149-X.

CRAIG LARMAN, 2005, *UML y patrones*. 2da.

DT, 2013, *Vista Entorno de Desarrollo Tecnológico*. 2013.

ESCUDERO VÁZQUEZ, Pedro Jesús, GARCÍA MORENO, María N. and GARCÍA PEÑALVO, Francisco J., 2014, *Métricas orientadas a objetos*. 2014.

Salamanca

GARCIA, Jose M, 2005, *Ingeniería del Software.Control y garantía del software* [online]. 2005. Available from: http://www.falconmarbella.com/esigranada/dmdocuments/Tema_5.pdf.

GÓMEZ BARYOLO, Oiner, RIVERO PINO, Noel Jesús and LÓPEZ MÉNDEZ, Daniel E., 2011, Sistema de gestión integral de seguridad Acaxia. *15-07-2011*. 2011. No. Serie Científica de la Universidad de las Ciencias Informáticas, p. 23.

GONZÁLEZ, Montoya and ANDREA, Paola, 2012, *Caracterización de un protocolo de pruebas*. 2012. Santiago de Cali . Facultad de Ingeniería, Universidad de San Buenaventura.

GOOGLE TRENDS, 2014, Tendencias de búsqueda de Google - Interés en Búsqueda en la Web - Todo el mundo, 2004 - hoy. [online]. 2014. [Accessed 9 June 2014]. Available from:

http://www.google.com/trends/explore#cmpt=q
JAMES RUMBAUGH, IVAR JACOBSON and GRADY BOOSH, 2013, *El Lenguaje Unificado de Modelado. Manual de referencia*.
KARENNY BRITO ACUÑA, 2013, *Selección de Metodologías de Desarrollo para Aplicaciones Web en la Facultad de Informática de la Universidad de Cienfuegos* [online]. [Accessed 8 June 2014]. ISBN 978-84-692-6641-0. Available from: http://www.eumed.net/libros-gratis/2009c/584/index.htm
http://www.eumed.net/libros-gratis/2009c/584/index.htm

MUÑOZ DUSSAC, Humberto, 2014, *Indicadores para evaluar las soluciones de control de acceso*. 2014.
ORACLE CORPORATION AND ITS AFFILIATES, 2014, NetBeans IDE - Overview. [online]. 2014. [Accessed 10 June 2014]. Available from: https://netbeans.org/features/index.html
POSTGRESQL, 2014, PostgreSQL: About. [online]. 2014. [Accessed 9 June 2014]. Available from: http://www.postgresql.org/about/
RICARDO FIGUEREDO, Renier, 2009, SfAspectPlugin extensión de Symfony para la programación orientada a aspectos. 2009. Vol. 2, no. 10.
SENSIOLABS, 2013a, *The Book for Symfony2.1*.
SENSIOLABS, 2013b, *The Components Book for Symfony 2.1*.
SOMMERVILLE, Ian, 2005, *Ingeniería de Software*. 7ma. Madrid : Pearson Educación, S.A. ISBN 84-7829-074-5.
SPERBERG CAMILO, 2013, Sobre convenciones y notaciones (húngara, CamelCase, etc) « unreal4u's Personal Network. [online]. 2013. [Accessed 8 June 2014]. Available from: http://blog.unreal4u.com/2011/03/sobre-convenciones-y-notaciones-hungara-camelcase-etc/
THE APACHE SOFTWARE FOUNDATION, 2014, Apache Software Foundation - Projects. [online]. 2014. [Accessed 9 June 2014]. Available from: http://projects.apache.org/
VISUAL PARADIGM, 2011, Full-Featured UML Software Design Tool. [online]. 2011. [Accessed 10 June 2014]. Available from: http://www.visual-paradigm.com/features/
W3TECHS, 2013, W3Techs - extensive and reliable web technology surveys. [online]. 20 December 2013. [Accessed 20 December 2013]. Available from: http://w3techs.com/

ANEXOS

Anexo 1. Lenguajes de programación del lado del servidor más populares según W3techs

© W3Techs.com	usage	change since 1 November 2013
1. PHP	81.6%	+0.3%
2. ASP.NET	18.2%	-0.4%
3. Java	2.7%	
4. ColdFusion	0.8%	
5. Perl	0.6%	-0.1%

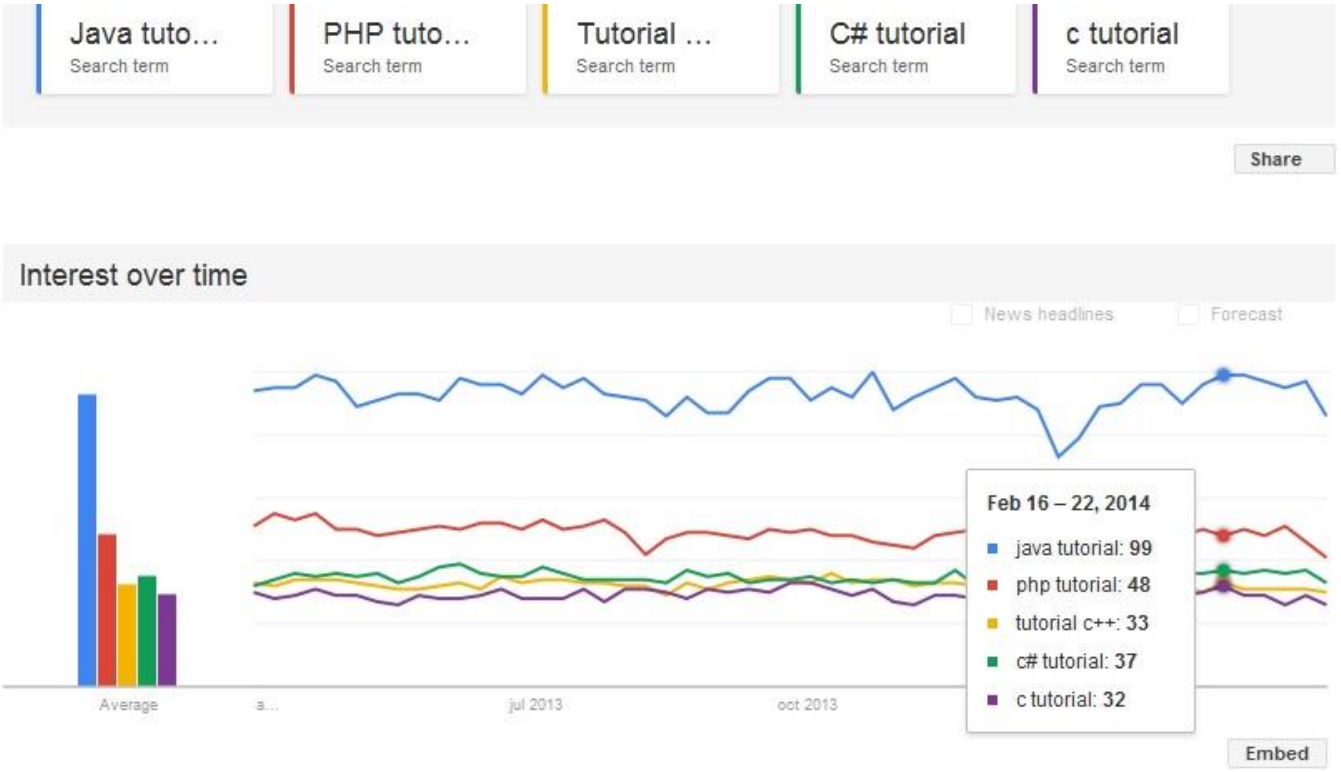
percentages of sites

Anexo 2. Lenguajes de programación del lado del servidor más rápidos según W3techs

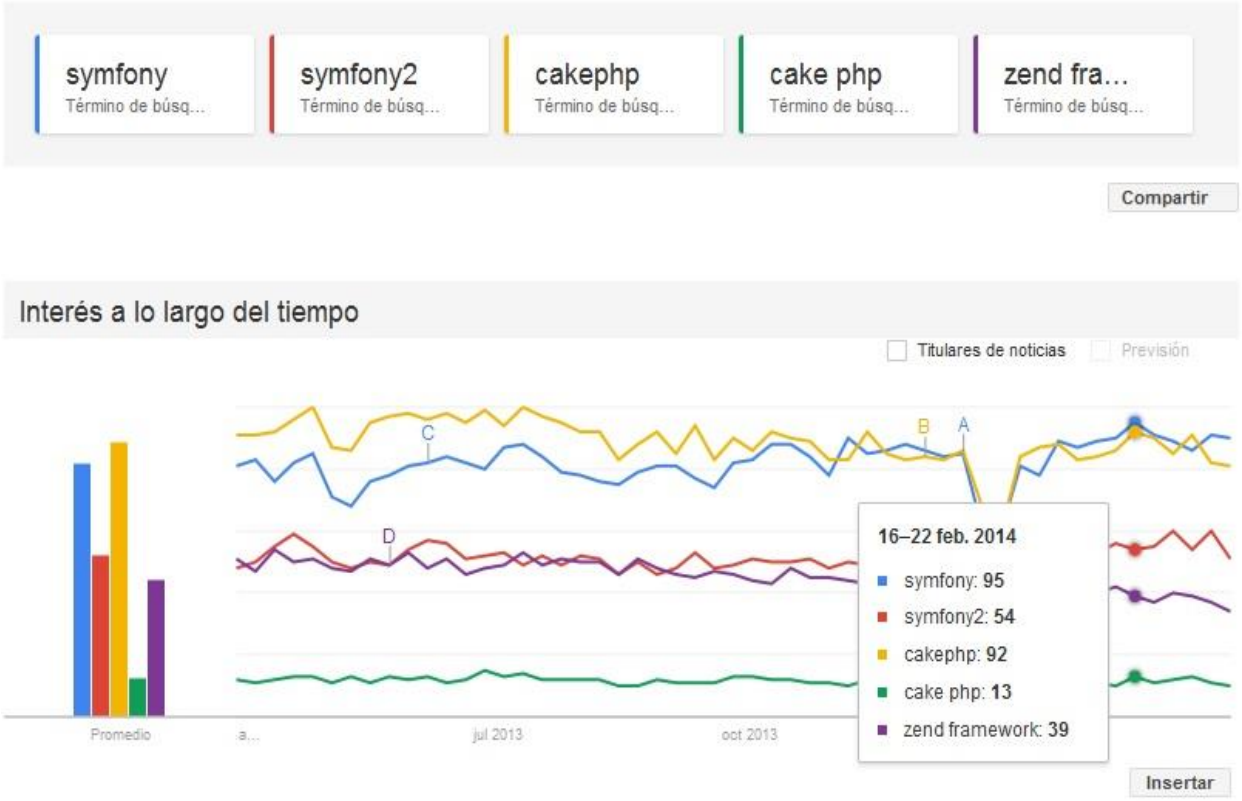
© W3Techs.com	sites
1. PHP	633
2. JavaScript	18
3. Python	6

daily number of additional sites in the top 10 million

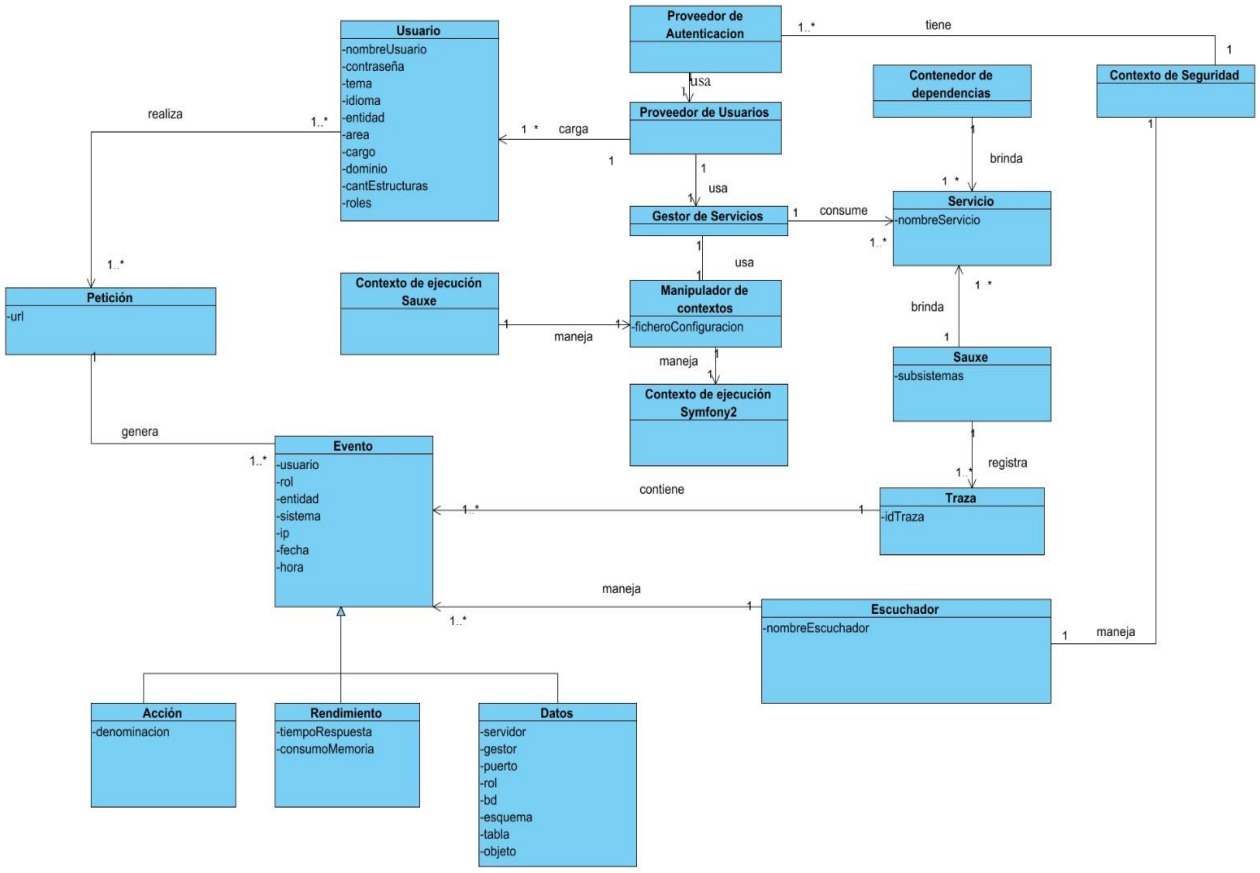
Anexo 3. Estudio de Google Trends de los lenguajes de programación más populares



Anexo 4. Estudio de Google Trends de los marcos de trabajo PHP más populares.



Anexo 5. Modelo conceptual del componente



Anexo 7. Diagrama de clases del requisito Autenticar usuario en el sistema