Universidad de las Ciencias Informáticas

FACULTAD 6

Título: Módulo de monitorización de usuarios para aplicaciones desarrolladas con Symfony2

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autora:

Graciela Gutiérrez Rodríguez

Tutores:

MsC. Nara Lidia Pérez Solá

MsC. Adonis Ricardo Rosales García

La Habana, junio de 2013

"Año 55 de la Revolución"

"El genio se hace con un 1% de talento, y un 99% de trabajo."

Albert Einstein

DECLARACIÓN DE AUTORÍA

Declaro ser autora de la presente tesis y reconozco 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 _____.

Graciela Gutiérrez Rodríguez

_____

Firma de la Autora

MsC. Nara Lidia Pérez Solá

_____

Firma del Tutor

MsC. Adonis Ricardo Rosales García

_____

Firma del Tutor

DATOS DE CONTACTO DE LOS TUTORES

Tutora:

MsC. Nara Lidia Pérez Solá

Universidad de las Ciencias Informáticas, La Habana, Cuba

e-mail: nara@uci.cu

Tutor:

MsC. Adonis Ricardo Rosales García

Universidad de las Ciencias Informáticas, La Habana, Cuba

e-mail: arrosales@uci.cu

AGRADECIMIENTOS

Agradezco a mi cuñadito Jorge por hacer a mi hermana la mujer más feliz del mundo e inventarme este tema de tesis que me ha obligado a programar.

Agradezco a los tutores Nara Lidia Pérez Solá, Yusdenis Sánchez y Adonis Ricardo Rosales García por su paciencia, dedicación y optimismo. Aunque quisiera agradecer especialmente a Narín por estar batallando durante estos arduos 10 meses la tarea profesional más importante de mi vida.

A Osay González por siempre estar ahí para todas mis dudas y ser mi oponente, tribunal, y sobre todo tutor.

A mis dúos de tesis Yuned y Rolando por responder a todas mis preguntas en el laboratorio más cálido y confortable de la universidad.

A mis dos amigas Dalia y Lisbeth por demostarme que la amistad puede más que 700km.

A Yailema, Ariannis, Katiuska, Dane, Yanet, Luis, Marvel y Rene que me aportaron su apoyo y ayuda.

A mis amigos de la vocacional Leidis, Luly, Linda, Karen, Lisset y Yaniel que aunque no estén aquí sé que nunca se han olvidado de mí.

No sé si por casualidad o causalidad pero le agradezco a esta universidad la posibilidad de encontrar a dos personitas que se han convertido en más que amigos: Yanet y Yosbel. Gracias por hacerme reír, por hacerme feliz, por levantarme cada vez que pienso que he caído, a los dos: gracias por demostrarme que la palabra amistad vale más que 7 letras.

A mi cochi por ser más que mi pareja. Por hacerme feliz en el momento más duro de mi carrera. Por soportar todos mis dolores de cabeza, malcriadeces y malas noches. Gracias por existir y estar ahí siempre para mí.

En fin, a todos aquellos que de una forma u otra hicieron este sueño posible.

DEDICATORIA

Cinco años de esfuerzo, sacrificio, abnegación hoy dan a luz el resultado gracias a la familia, lo amigos y las personas que siempre confiaron en que el camino no era en vano. Cuando, como dijera Silvio, "El camino se llena de sillas que te invitan a parar", la motivación son ustedes, los que en consecuencia con el amor que me profesan, trasmiten siempre la fe de que bien vale el resultado las tormentas que cuesta. No importa cuánto desfallezca, cuantas veces haya caído en el intento, hoy después de muchos años de estudio estoy aquí compartiendo el resultado de tanta añoranza. Pasamos mucho tiempo esperando este momento, dibujándolo en hojas blancas, imaginando cada detalle y a unos días de que llegue estoy segura que va a ser impredecible, porque los sueños siempre son inesperados y la vocación solo tiene meta en momentos como este.

Dedico este trabajo a las dos personas más importantes de mi vida, mi mamá y mi papá. Gracias por demostrarme una vez más, mediante sus consejos que el genio no se hace con 1% de talento sino con 99% de trabajo. Y sobre todo gracias a Dios por darme la oportunidad de compartir este sueño con ustedes: Mi familia.

A mis dos hermanas Irmi y Nelin por ser mis guías en todo momento. Aunque quisiera agradecer especialmente a mi gigante Nelin por ser mi segunda mamá, por pelearme y cuidarme a cada instante aunque ahora es que me doy cuenta de lo fuerte que me has hecho.

A mis abuelos por ser mis segundos padres, abuelita Pilar, Irma, abuelitos Pucho, Baldo y abuelito Nené que me observa desde el cielo y me da fuerzas para continuar. Aunque quisiera agradecer al viejito que me ha hecho reír tantas veces desde que era niña Mi Monguín, por demostrarme que el amor vale más que cualquier obsequio material.

Resumen

El departamento de Integración de Soluciones perteneciente al Centro de Tecnologías de Gestión de Datos en la Universidad de las Ciencias Informáticas, cuenta actualmente con diferentes proyectos implementados con el framework de desarrollo Symfony2. Una de las prioridades que se tienen en cuenta durante el desarrollo de estos proyectos, es el aseguramiento de la seguridad, mediante los diferentes componentes: autenticación, autorización y auditoría. La presente investigación enmarca la concepción e implementación de un módulo de monitorización para aplicaciones desarrolladas con Symfony2, enfocándose en el componente más débil: la auditoría. El mismo brinda el control de monitorizar, a los administradores, las acciones realizadas por todos los usuarios, en cualquier sistema implementado con este framework. Además, posibilita visualizar las trazas realizadas mediante: gráficas de barras y de series, con el objetivo de contribuir a la toma de decisiones. El sistema se integra al Sistema Integral para la Gestión de Datos (SIGDAT) y puede ser utilizado por otras aplicaciones.

Palabras claves: auditoría, framework de desarrollo, seguridad.

Contenido

Introducción

La informática está inmersa en la gestión integral de cada organización, teniendo asegurado un papel protagónico en el futuro de la humanidad, definida como una de las materias más importantes en la actualidad, ya que por medio de esta se vive en una sociedad comandada por las nuevas tecnologías. Es muy difícil pensar en cambios, transformaciones e innovaciones y dejar atrás la informática, esto se debe al avance tecnológico en la transmisión de datos y a las nuevas facilidades de comunicación, ambos impensables sin la evolución de las computadoras y dispositivos.

Las Tecnologías de la Información y las Comunicaciones (TIC) se presentan cada vez más como una necesidad en el contexto de la sociedad, donde los rápidos cambios, el aumento de los conocimientos y las demandas de una educación de alto nivel constantemente actualizada, se convierten en una exigencia permanente. En el 2002 el Comandante en Jefe Fidel Castro Ruz crea la Universidad de las Ciencias Informáticas (UCI) para producir aplicaciones y servicios informáticos, a partir de la vinculación estudio-trabajo como modelo de formación y servir de soporte a la industria cubana del software.

La UCI está dirigida a la creación del Ingeniero en Ciencias Informáticas, con conocimientos, habilidades y valores sólidos, sustentados en una concepción científica y dialéctico-materialista del mundo, que estén comprometidos con su Patria para así poder llevar a cabo, importantes tareas dentro y fuera de las fronteras del país. Uno de los centros de la Universidad inmerso en el proceso de desarrollo de software es el Centro de Tecnologías de Gestión de Datos (DATEC). Este centro tiene como objetivo proveer soluciones integrales, productos y servicios relacionados con las tecnologías de gestión de datos, dentro del mismo se encuentra el departamento de Integración de Soluciones donde su misión fundamental es: desarrollar soluciones que permitan la captura, análisis, procesamiento y presentación de la información.

Este departamento se encuentra a la vanguardia en el uso de las tecnologías de punta, donde disponer de información continua, confiable y en tiempo, constituye una tarea fundamental. Hoy más que nunca no se puede permitir que merme su productividad y se paralice el desarrollo de aplicaciones por la necesidad imperante que existe de disponer de un plan de seguridad que evite riesgos potenciales.

Gran parte de los proyectos desarrollados por este departamento, utilizan como framework de desarrollo Symfony2, garantizando que la seguridad del mismo sea fehaciente pues incluye numerosas estrategias y utilidades para hacer frente a diversos ataques. Su arquitectura está compuesta por tres componentes esenciales en términos de seguridad: autenticación, autorización y auditoría. Aunque este último es demasiado genérico, ya que carece de componentes que permitan realizar la monitorización de las acciones realizadas por los usuarios, dejando la solución a esta necesidad en manos de los desarrolladores de cada equipo de trabajo.

Los proyectos desarrollados en este framework garantizan la autenticación y la autorización eliminando cada vez más los posibles riesgos de pérdida o robo de información. Aunque estos dos componentes están destinados a conseguir un sistema de información seguro y confiable, no completa la seguridad, ya que los administradores no poseen los elementos necesarios para monitorizar todas las acciones realizadas por los usuarios. Por lo tanto, no se pueden establecer estadísticas de lo que sucede realmente en el sistema, ni enfatizar en los recursos más explotados por los clientes, contribuyendo así, a la falta de conocimiento de cuáles zonas son las más cotizadas y a su vez más vulnerables para su futuro mantenimiento.

Aunque a simple vista parezca un proceso engorroso, la auditoría es una técnica fundamental para el desarrollo y crecimiento de cualquier empresa, dado que otorga interesantes posibilidades de cambio y perfeccionamiento, contribuyendo así a la toma de decisiones. Por tal motivo en las soluciones desarrolladas por el departamento Integración de Soluciones de DATEC crece la necesidad de auditar las acciones de los usuarios de algunos productos e incrementar, así, la calidad y la seguridad de los mismos.

De la situación problemática planteada anteriormente, se identifica como **problema de la investigación**: ¿Cómo monitorizar las acciones de los usuarios en aplicaciones desarrolladas con Symfony2 para contribuir al fortalecimiento de la seguridad ofrecida por este framework?

Considerando lo anterior el **objeto de estudio** es: La seguridad en aplicaciones web, enmarcado en el **campo de acción**: El proceso de auditoría en aplicaciones web.

En correspondencia con el problema planteado, el **objetivo general** de esta investigación consiste en: Desarrollar un módulo de monitorización de usuarios para aplicaciones desarrolladas con Symfony2 que permita auditar las acciones de los usuarios. En concordancia con el objetivo general se derivan los siguientes **objetivos específicos:**

1. Realizar el análisis y diseño del módulo de monitorización para las aplicaciones desarrolladas con Symfony2.
2. Implementar el módulo de monitorización para las aplicaciones desarrolladas con Symfony2.
3. Realizar pruebas al módulo de monitorización para validar su correcto funcionamiento.

Para dar cumplimiento a estos objetivos se han trazado las siguientes tareas:

1. Revisión bibliográfica sobre las herramientas, metodología y mecanismos de auditorías, para fundamentar su selección.
2. Caracterización de la arquitectura de Symfony2 y sus mecanismos de seguridad para identificar las vulnerabilidades en la seguridad del framework.
3. Definición de los requisitos para gestionar las necesidades del módulo de monitorización de usuarios para aplicaciones desarrolladas con Symfony2.
4. Confección del modelo de casos de uso del sistema del módulo de monitorización de usuarios para aplicaciones desarrolladas con Symfony2, para visualizar y especificar el comportamiento del sistema.
5. Elaboración del modelo de diseño para describir cómo se debe implementar el módulo de monitorización de usuarios para aplicaciones desarrolladas con Symfony2.
6. Implementación del módulo para la monitorización usuarios en aplicaciones desarrolladas con Symfony2.
7. Integración del módulo de monitorización de usuarios a la arquitectura del framework Symfony2.
8. Realización de pruebas funcionales, integración, carga y estrés que evidencian el correcto funcionamiento del módulo de monitorización de usuarios para aplicaciones desarrolladas con Symfony2.

El presente trabajo está estructurado en un volumen de 69 páginas, compuesto por varias secciones que forman tres capítulos, además de la introducción, conclusiones, referencias bibliográficas, bibliografía, recomendaciones y anexos.

Capítulo1. Fundamento teórico de la investigación.
En este capítulo se realiza una revisión bibliográfica del estado actual de la temática en estudio, con el objetivo de caracterizar y profundizar en las herramientas, tecnologías y metodología que se necesita utilizar para llevar a cabo dicha investigación.

Capítulo2. Análisis y diseño del sistema.
En este capítulo se describen los materiales y métodos usados, así como, la arquitectura que tendrá el proceso a desarrollar generando todos los artefactos necesarios según la metodología utilizada como: el diagrama de despliegue y el modelo de clases del diseño, este último imprescindible para comprender la lógica del negocio.

Capítulo3. Implementación y pruebas.
En este capítulo se presentan los artefactos asociados a las disciplinas de implementación y prueba. Se elabora el modelo de implementación a través de los diagramas de componentes. Además se realizan las pruebas para garantizar el correcto funcionamiento del módulo propuesto.

Capítulo I: Fundamento teórico de la investigación

Introducción

El objetivo principal de este capítulo se centra en definir, explicar y abordar todos los conceptos necesarios para llevar a cabo la investigación, enfocándose fundamentalmente en los mecanismos de seguridad que ofrece el framework de desarrollo Symfony2. Comprende un estudio de las herramientas, tecnologías y metodología a utilizar, para aprovechar al máximo todas sus potencialidades.

1.1 Seguridad en aplicaciones web

La seguridad en los sistemas informáticos se ha convertido en uno de los eslabones fundamentales de cualquier sistema, debido al aumento de los delitos informáticos, por tanto, existen cuatro pautas que son imprescindibles para la implementación de la seguridad de cualquier sistema:

- Prevención: En esta etapa se toman las acciones necesarias para prevenir una posible intrusión.
- Detección: En caso de que se produzca una intrusión en el sistema, es recomendable detectar el momento en que se produce y tomar las medidas necesarias para que no pueda ocasionar más daño.
- Restauración: Una vez que el sistema ha sido atacado será necesario restaurarlo con las copias de seguridad realizadas anteriormente.
- Análisis forense: Este último se asegura de investigar las acciones del atacante para que así no ocurran posibles ataques futuros. (1)

A criterio de la autora se puede definir la seguridad, como los mecanismos de prevención, detección, restauración y análisis que se llevan a cabo para garantizar la protección de sistemas informáticos. Es importante destacar que la seguridad de la información, en todo sistema, depende en gran medida de la seguridad informática, pues es aquí donde se deben definir claramente los objetivos a partir de los cuales desarrollar las políticas y procedimientos, que definan el marco en el que situar las medidas a implantar. Recalcar que, la seguridad absoluta no existe, debido a que es un proceso lento y continuo, que exige aprender sobre las propias experiencias.

1.1.1 Elementos de seguridad

El surgimiento de aplicaciones informáticas seguras lleva consigo el proceso de administrar el acceso y el uso de sus recursos. En otras palabras, conocer con precisión la identidad de cada persona en la organización, los servicios a los que puede acceder y delimitar la información que cada individuo puede ver o manipular, son procesos fundamentales en cualquier sistema. Con el objetivo de obtener un producto de software que contribuya a la automatización de los procesos es de vital importancia el uso de la autenticación, autorización y la auditoría.

Autenticación

Proceso utilizado en los mecanismos de control de acceso con el objetivo de verificar la identidad de un usuario, dispositivo o sistema mediante la comprobación de credenciales de acceso. (2)

La demanda por soluciones de autenticación ha provenido que cada empresa necesite incrementar el nivel de seguridad lo más alto posible debido al aumento de la piratería informática. A medida que nuevas arquitecturas de autenticación están siendo desarrolladas, la privacidad de los individuos está siendo protegida a un ritmo impresionante.

Autorización

La autorización es la acción o especie de permiso que consiste en otorgar el consentimiento para realizar las acciones necesarias según los permisos otorgados. Este proceso determina luego de la autenticación, a qué recursos de un sistema tiene acceso una entidad o persona. (3) En el caso de los datos, la autorización debe asegurar la confidencialidad e integridad, ya sea otorgando o denegando el acceso de lectura, modificación, creación o borrado de los datos.

Auditoría

La auditoría no es más que la inspección interna o externa, de una institución o programa, que se utiliza con el objetivo de comprobar y evaluar el comportamiento del trabajo realizado. Es un término que se incorpora del mundo empresarial y judicial. (4)

La adecuada planeación de una institución propicia el espacio para sus evaluaciones, que permite dimensionar sus principales características, dando la oportunidad de organización a través de una auditoría contable para garantizar su buen desempeño. La auditoría proporciona a su vez un mejor control

en su implementación sistemática, rentabilidad, eficiencia y seguridad en el procesamiento de la información, posibilitando un excelente manejo para la toma de decisiones.

En un mundo tecnológico como el actual, los sistemas de seguridad evolucionan con rapidez y ofrecen más funciones y un mayor nivel de integración. En realidad, los diferentes subsistemas de seguridad funcionan mejor si se comportan como un único sistema capaz de responder a todos los riesgos específicos de una instalación.

1.2 Aplicaciones del Centro de Tecnologías de Gestión de Datos

Como centro de desarrollo de software, DATEC, tiene entre sus principales objetivos desarrollar sistemas informáticos que contribuyan al proceso de informatización del país, así como aportar el máximo a la economía. Muchos de los sistemas son implementados utilizando el joven framework de desarrollo Symfony2, entre ellos: Sistema Integral para la Gestión de Datos (SIGDAT) y Generador Dinámico de Reportes (GDR 2.0).

Una de las funcionalidades requeridas por varios clientes ha sido la de poder realizar un estudio sobre las acciones de los usuarios en el sistema; hasta el momento la monitorización no se ha convertido en una solución reutilizable, pues la forma en la que el framework realiza este proceso es difícil de personalizar, lo que ha provocado el desarrollo de soluciones a la medida en cada uno de los casos. Ha sido política del centro desde su fundación la vieja frase de “no reinventar la rueda”, entiéndase por elevar al máximo el grado de reutilización de las funcionalidades ya desarrolladas, así como el diseño de arquitecturas donde su grado de modularidad sea óptimo para contribuir al desarrollo orientado a componentes, lo que acorta en la mayoría de los casos el cronograma para la obtención de productos.

1.3 Metodologías de desarrollo de software

Las metodologías de desarrollo de software constituyen una colección de documentación formal referente a los procesos, las políticas y los procedimientos que intervienen en el desarrollo del software. El núcleo de cualquier metodología de desarrollo se encuentra constituido por documentos escritos que detallan cada uno de los puntos expuestos. (5) Si bien es importante formalizar una metodología de desarrollo, para minimizar riesgos e incrementar las posibilidades de éxito en el desarrollo de aplicaciones

informáticas, también es necesario establecer mecanismos de mejora continua sobre los procesos que componen la metodología.

OpenUP

El uso de una metodología constituye un elemento vital dentro del desarrollo de software, de modo tal que se alcance un producto de gran valor para los clientes. OpenUP es una de las metodologías que se aplica de forma iterativa e incremental dentro de un ciclo de vida estructurado. Se utiliza generalmente para proyectos pequeños, generando de esta forma sólo lo necesario para el desarrollo del software sin dejar de ser completo y extensible.

La estructura del ciclo OpenUP se divide en cuatro fases:

- Concepción: Lograr concordancia entre todos los stakeholders[1] de los objetivos del ciclo de vida del proyecto.
- Elaboración: Establecer una línea base arquitectónica del sistema y proveer las bases para el grueso del esfuerzo de desarrollo de la siguiente fase.
- Construcción: Completar el desarrollo del sistema basado en la arquitectura definida.
- Transición: Determinar si se lograron las expectativas propuestas. (6)

Esta metodología contempla un conjunto de disciplinas, a continuación se muestran las aplicadas durante el desarrollo de este trabajo:

- Modelado del negocio
- Requisitos
- Análisis y diseño
- Implementación
- Prueba
- Despliegue
- Configuración

[1] stakeholders: es una persona u organización que tiene influencia directa o indirecta o se ve influenciado por un proceso de software.

1.4 Herramientas y tecnologías

El uso de herramientas en el mundo de la informática más que necesario, es imprescindible, pues permiten realizar programas, rutinas y sistemas para el avance de este campo en la sociedad. Con el objetivo de realizar buenas prácticas en el uso de herramientas, es necesario contar con una tecnología adecuada para así guiar de una forma organizada todo el proceso de investigación.

1.4.1 Lenguajes de programación

Las computadoras operan sobre bits y los hombres por medio de idiomas, este fenómeno provoca que exista una gran dificultad para la comunicación entre hombre-computadora. Los lenguajes de programación brindan la posibilidad de que ambos se comuniquen de una forma factible. Un lenguaje de programación es un conjunto de símbolos, reglas sintácticas y semánticas que definen su estructura y que a su vez es utilizado para controlar el comportamiento físico y lógico de una máquina. (7) Este le permite al programador especificar de manera precisa: sobre qué datos una computadora debe operar, cómo deben ser estos almacenados y transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias.

PHP 5.3.10

Hypertext Preprocessor (PHP) es un lenguaje script (no se compila para conseguir códigos máquina sino que existe un intérprete que lee el código y se encarga de ejecutar las instrucciones que contiene éste código), para el desarrollo de páginas web dinámicas del lado del servidor, cuyos fragmentos de código se intercalan fácilmente en páginas HTML, debido a esto, y a que es de código abierto, es el más popular y extendido en la web. (8)

Se presenta como una alternativa de fácil acceso para todos, pues se encuentra publicado bajo la PHP License, que está considerada como software libre. Es un lenguaje de una sintaxis muy simple y fácil de aprender que contiene documentación amplia, segura y factible. Permite aplicar técnicas de programación orientada a objetos y está diseñado para establecer conexión a numerosas bases de datos.

Debido a su amplia distribución, PHP está perfectamente soportado por una gran comunidad de desarrolladores, facilitando así su perfeccionamiento en cada una de sus versiones. Para el desarrollo de

este módulo se utiliza la versión 5.3.10, destacando de esta nueva versión, la corrección de un grave fallo de seguridad, recomendando así la actualización inmediata de todas las versiones anteriores de PHP.

JavaScript 1.8

JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios. (9)

Características de JavaScript:

- Es multiplataforma.
- Sencillo de aprender y utilizar.
- Es visual: permite la moderna "programación visual" (ventanas, botones, colores, formularios).
- Es dinámico, responde a eventos en tiempo real.
- Maneja objetos dentro de una página web y sobre ese objeto se pueden definir diferentes eventos.

1.4.2 Entorno de desarrollo

Un entorno de desarrollo integrado (IDE) es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien puede utilizarse para varios. Los IDE´s pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. (10) En pocas palabras no es más que una aplicación de software que proporciona servicios integrales a los programadores de software para el desarrollo del mismo.

NetBeans 7.2

NetBeans es un entorno de desarrollo integrado, modular, de base estándar (normalizado), escrito en el lenguaje de programación Java. El proyecto NetBeans consiste en un IDE de código abierto y una plataforma de aplicación, las cuales pueden ser usadas como una estructura de soporte general (framework) para compilar cualquier tipo de aplicación. (11) Destacar que uno de los detalles más interesante de este IDE es la integración de los diferentes módulos que pueden ser activados para permitir la creación de programas en lenguajes como Java, PHP, C++, Python, Ruby, entre otros.

NetBeans IDE 7.2 ofrece un rendimiento mejorado significativamente, la experiencia de codificación, con las nuevas capacidades de análisis de código estático en el editor de Java y la exploración de proyectos son cada vez más inteligentes. Incluye características notables, como el apoyo a marcos de PHP múltiples y se encuentra disponible en varios idiomas.

Para la realización del módulo se decidió utilizar NetBeans debido a todas las características expuestas anteriormente. Además, es apoyado por una gran comunidad de desarrolladores y ofrece una amplia documentación y recursos de capacitación. También se tuvo en cuenta que esta es una de las herramientas usadas por la Universidad y DATEC para el desarrollo de sus productos.

1.4.3 Lenguaje de modelado

Cualquier aplicación requiere: que cada una de las partes que comprende su desarrollo se visualice, especifique y documente con un lenguaje común. Un lenguaje que sea gráfico con el fin de especificar y documentar de un modo estándar lo que se desea crear, incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema.

Lenguaje de Modelado Unificado

El Lenguaje de Modelado Unificado (UML - Unified Modeling Language) es un lenguaje gráfico para visualizar, especificar, construir y documentar cada una de las partes que comprende el desarrollo de software. UML entrega más características visuales que programáticas, que facilitan la comunicación entre analistas, arquitectos y programadores. Es importante destacar que entre más complejo es el sistema que se desea crear, más beneficios presenta el uso de UML, pues resulta más fácil detectar los posibles errores en la etapa inicial que en cualquier otra etapa del sistema, como sería la fase intensiva de codificación.

1.4.4 Herramientas de modelado

Ingeniería de Software Asistida por Computadora (CASE - Computer Aided Software Engineering) proporciona un conjunto de herramientas para automatizar o apoyar una o más fases del ciclo de vida del desarrollo de sistemas informáticos. Las herramientas de modelado permiten plasmar de forma tangible todo el proceso de software facilitando y garantizando la calidad requerida. Estas, sirven de guía para

verificar que se comprenda correctamente el ambiente del usuario y que esté respaldado con información documental para que los diseñadores y programadores puedan construir el sistema.

Visual Paradigm 6.4

Visual Paradigm es una herramienta que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite realizar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Es una herramienta privada disponible en varias ediciones, que contempla varios idiomas y posee disponibilidad en múltiples plataformas (Windows, Linux).

Visual Paradigm ofrece:

- Entorno de creación de diagramas para UML 2.1.
- Diseño centrado en casos de uso y enfocado al negocio que generan un software de mayor calidad.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Capacidades de ingeniería directa (versión profesional) e inversa.
- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- Disponibilidad de múltiples versiones, para cada necesidad.
- Disponibilidad de integrarse en los principales IDE´s. (12)

Se decidió usar Visual Paradigm debido a todas las características mencionadas anteriormente. Además, es una herramienta CASE profesional, la cual soporta todo el ciclo de vida de desarrollo de software. También se tuvo en cuenta que esta es una de las herramientas usadas por la Universidad y DATEC. La misma posee una licencia comercial y la UCI cuenta con esta para su uso.

1.4.5 Framework de desarrollo

Un framework simplifica el desarrollo de una aplicación mediante la automatización de algunos de los patrones utilizados para resolver las tareas comunes. Proporciona estructura al código fuente, forzando al

desarrollador a crear código más legible y más fácil de mantener. (13) Por último, facilita la programación de aplicaciones, ya que encapsula operaciones complejas en instrucciones sencillas.

Symfony2

Symfony es un completo framework diseñado para optimizar, gracias a sus características, el desarrollo de las aplicaciones web. Symfony se diseñó para que se ajustara a los siguientes requisitos:

- Fácil de instalar y configurar en la mayoría de plataformas.
- Independiente del sistema gestor de bases de datos.
- Sencillo de usar en la mayoría de casos, pero lo suficientemente flexible como para adaptarse a los casos más complejos.
- Basado en la premisa de *"convenir en vez de configurar"*, en la que el desarrollador sólo debe configurar aquello que no es convencional.
- Sigue la mayoría de las mejores prácticas y patrones de diseño para la web.
- Preparado para aplicaciones empresariales y adaptables a las políticas y arquitecturas propias de cada empresa, además de ser lo suficientemente estable como para desarrollar aplicaciones a largo plazo.
- Código fácil de leer que incluye comentarios de phpDocumentor y que permite un mantenimiento muy sencillo.
- Fácil de extender, lo que permite su integración con librerías desarrolladas por terceros. (14)

Arquitectura de Symfony

Symfony es un framework de PHP basado en la arquitectura MVC (Modelo-Vista-Controlador). El Modelo-Vista-Controlador es un patrón o modelo de abstracción de desarrollo de software que separa los datos de una aplicación, la interfaz de usuario y la lógica de negocio en tres componentes distintos. MVC desacopla el concepto de interfaz de usuario y lógica de negocio incrementando la flexibilidad del software.

- Modelo: son los objetos de la aplicación, también conocida como lógica de negocio, o lógica de aplicación.
- Vista: especifica la visualización de los datos, algunas veces conocida como lógica de presentación.
- Controlador: es el coordinador entre estos dos últimos, es decir, define la forma en que la interfaz de usuario reacciona ante la entrada de usuario.

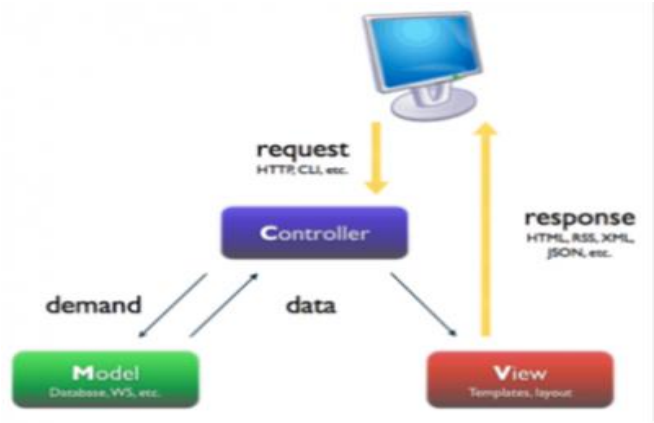


Figura 1. Arquitectura Modelo-Vista-Controlador (15)

Una de las tareas fundamentales para cualquier aplicación consiste en la persistencia y lectura de información hacia y desde una base de datos. Symfony integra plenamente uno de los frameworks ORM (Object-Relational Mapping o Mapeo entre Objetos y Relaciones) más importantes dentro de los existentes para PHP llamado Doctrine, el cual es el encargado de la comunicación con la base de datos. Doctrine permite un control casi total de los datos sin importar el gestor (MySQL, PostgreSQL, SQL server, Oracle), pues la mayoría de las sentencias SQL no son generadas por el programador sino por él mismo.

El correcto funcionamiento del framework es garantizado a partir de la utilización de varios componentes entre los que se destacan:

- HttpFoundation: Contiene las clases petición y respuesta, así como otras clases para manejar sesiones y cargar archivos.
- Routing: Potente y rápido sistema de enrutado que permite asignar una URI específica (por ejemplo /contacto) a cierta información acerca de cómo se debe manejar dicha petición (por ejemplo, ejecutar el método contactoAction()).
- Form: Una completa y flexible plataforma para crear formularios y procesar la presentación de los mismos.
- Validator: Un sistema para crear reglas sobre datos y entonces, cuando el usuario presenta los datos se puede comprobar si son válidos o no siguiendo esas reglas.
- ClassLoader: Una biblioteca para carga automática que permite utilizar clases PHP sin necesidad de requerir manualmente los archivos que contienen esas clases.

- Templating: Un juego de herramientas para reproducir plantillas, la cual gestiona la herencia de plantillas (es decir, una plantilla está decorada con un diseño) y realiza otras tareas de plantilla comunes.
- Security: Una poderosa biblioteca para manejar todo tipo de seguridad dentro de una aplicación.
- Translation: Una plataforma para traducir cadenas en la aplicación.
- El núcleo: Symfony2 cuenta con un micro-kernel altamente optimizado. El núcleo es la pieza central del framework y es el responsable de inicializar la configuración de la aplicación e iniciar (bootear) los paquetes (bundles).
- Bundles: Un bundle es un conjunto estructurado de archivos que implementan una característica única y que puede ser fácilmente compartido con otros desarrolladores. En Symfony2 todo es un bundle. Los bundles permiten una forma limpia de configurar y personalizar el sistema.
- El contenedor de inyección de dependencias (The Dependency Injector Container):Symfony2 está construido utilizando un contenedor de inyección de dependencias, inspirado en el framework Spring de Java. En un proyecto, el desarrollador no interactúa directamente con el contenedor. Todos los detalles de implementación están ocultos detrás de un buen sistema de configuración que permiten personalizar todo a través de archivos YAML o XML, o incluso a través de código PHP.
- El manejador de peticiones (The Request Handler): El manejador de peticiones es la columna vertebral del ciclo de petición/respuesta del protocolo HTTP. Encapsula la petición del usuario y devuelve una respuesta. Este manejador no funciona como la mayoría de manejadores de otros frameworks. Es un manejador (dispatcher) que notifica eventos en espera a que un escuchador (listener) se haga cargo de ellos.
- El despachador de eventos (The Event Dispatcher): El núcleo y el manejador de peticiones están basados en el despachador de eventos. Una implementación del patrón de diseño Observador (Observer). Una de las mejores maneras de desacoplar el código y hacerlo más flexible.

Además de estos componentes, Symfony2 brinda un conjunto de herramientas que son muy potentes y útiles para los desarrolladores, entre las cuales se encuentran:

- La barra de depuración web: El mejor amigo en el desarrollo de aplicaciones con Symfony2 es la barra de depuración web. Aparece en la parte inferior de todas las páginas en el entorno de desarrollo ofreciendo una gran cantidad de información.

- Profiler: Recoge información de las peticiones y las almacena para su posterior análisis. Este profiler es una herramienta para visualizar datos que se utiliza para depurar las aplicaciones y mejorar el rendimiento, pero también puede ser utilizado para buscar los problemas que ocurren en producción.
- Útiles mensajes de error: Cuando algo va mal durante el desarrollo, Symfony2 intenta hacer todo lo posible para proporcionar toda la información que se necesita para depurarlo. Además de la barra de depuración, ofrece páginas de error que muestran con todo detalle el error producido. En lugar de tener que cambiar a la consola para ver los errores del log de Apache, Symfony pone todo al alcance, analiza la excepción ocurrida y traza la ruta en busca del error producido.
- La línea de comandos: Symfony incluye una Interfaz de Línea de Comando (CLI) que hace todo el trabajo de forma automatizada.

La seguridad en Symfony2 es un proceso de dos etapas, cuyo objetivo es evitar que un usuario acceda a un recurso al cual no debería tener acceso. En el primer paso del proceso, el sistema de seguridad identifica quién es el usuario obligándolo a presentar algún tipo de identificación, a este procedimiento se le denomina autenticación.

Una vez que el sistema identifica el usuario, el siguiente paso es determinar si debería tener acceso a un determinado recurso. Esta parte del proceso se llama autorización, y significa que el sistema está comprobando para ver si se tienen los suficientes privilegios para realizar una determinada acción.

Durante las dos etapas anteriores se encuentra involucrado el proceso de auditoría, pues Symfony2 guarda mucha información de sus propios eventos en archivos de tipo log[2], creando un archivo por cada aplicación y cada entorno. Dentro de estos dos archivos se guarda cada evento realizado, donde cada línea incluye la fecha y hora, el tipo de evento, el objeto que ha sido procesado y otros detalles relevantes que dependen de cada tipo de evento u objeto procesado.

[2] log: es un registro de actividad que se guarda en un fichero de texto, al que se le van añadiendo líneas a medida que se realizan acciones sobre el sistema.

Ext JS 4.1.1

Ext JS es una biblioteca JavaScript de alto rendimiento, compatible con la mayoría de los navegadores para crear páginas web y aplicaciones dinámicas. El modelo de Ext JS mantiene su código bien estructurado por lo que incluso las aplicaciones más grandes se pueden mantener fácilmente. (16)

Ext JS es uno de los frameworks que además de flexibilizar el manejo de componentes de la página como el Modelo de Objetos del Documento (DOM), y las peticiones AJAX, tiene la gran funcionalidad de crear interfaces de usuario bastante funcionales. Una de sus características principales es que contiene componentes de interfaz de usuario con buen diseño y documentación.

Para el desarrollo del módulo se decidió utilizar Ext JS 4.1.1 por las mejoras en velocidad, facilidad de uso y estabilidad respecto a la versión anterior. Además, contiene una nueva librería para crear gráficas sin usar Flash a través de los estándares HTML5, elemento importante para las gráficas de barras y de series que se realizarán durante la implementación.

1.4.6 Apache JMeter

JMeter es una herramienta de carga para llevar acabo simulaciones sobre cualquier recurso de software. Inicialmente diseñada para pruebas de estrés en aplicaciones web, hoy en día, su arquitectura ha evolucionado no sólo para llevar a cabo pruebas en componentes habilitados en Internet (HTTP), sino además en bases de datos, programas en Perl, peticiones FTP y prácticamente cualquier otro medio.

Además, posee la capacidad de realizar desde una solicitud sencilla hasta secuencias de peticiones que permiten diagnosticar el comportamiento de una aplicación en condiciones de producción. En este sentido, simula todas las funcionalidades de un navegador, o de cualquier otro cliente, siendo capaz de manipular y reutilizar resultados para ser empleados en una nueva secuencia. El componente principal de JMeter es denominado *Plan de Prueba* o *Test Plan*, en él se definen todos los aspectos relacionados con una prueba de carga, como: parámetros empleados por petición, tipo de reportes a generarse con los resultados obtenidos, la posible reutilización de peticiones compuestas por usuarios, entre otros aspectos. (17)

1.5 Conclusiones parciales

El estudio de los conceptos necesarios acerca de los mecanismos de seguridad en el framework de desarrollo Symfony2 permiten obtener un mayor entendimiento del funcionamiento del área de auditoría, otorgando una visón específica de cómo está estructurado la seguridad del framework. El estudio de las metodologías, tecnologías y herramientas permitió definir cuáles son las que serán empleadas en el desarrollo de la solución, cumpliendo con los requisitos solicitados por el departamento de Integración de Soluciones. Este estudio y selección hizo posible: cubrir las etapas por las que transita el desarrollo de la solución con la metodología seleccionada y sentar las bases para mantener disponibles las estructuras físicas con el objetivo del correcto almacenamiento de la información.

Capítulo II: Análisis y diseño del sistema

Introducción

En el siguiente capítulo se modelan los conceptos necesarios para comprender el entorno en que se desarrollará el sistema. Para ello se definen los requisitos funcionales y no funcionales, facilitando así, la determinación de actores, casos de uso y las relaciones existentes entre ellos, con el objetivo de realizar el diagrama de casos de uso del sistema. Como plano de construcción para guiar la implementación se utilizarán el diagrama de clases del diseño y los diagramas de interacción. Además, se describen los patrones de diseño que utiliza el framework de desarrollo Symfony2 y se realiza, para modelar el hardware empleado, el modelo de despliegue.

2.1 Modelo de Dominio

El modelo de dominio o conceptual es un subconjunto del modelo de negocio y se realiza cuando no están claros los procesos o cuando no se identifican claramente los actores y trabajadores del negocio. Un modelo del dominio captura los tipos de objetos más importantes que existen, o los eventos que suceden en el entorno donde estará el sistema, se identifican y se definen conceptos que se unen o relacionan en un diagrama de clases UML. (18)

El objetivo fundamental de este modelo es comprender y describir los conceptos más importantes dentro del contexto del sistema. Para analizar el módulo de monitorización de usuarios se presenta como modelo de dominio (ver figura 2).

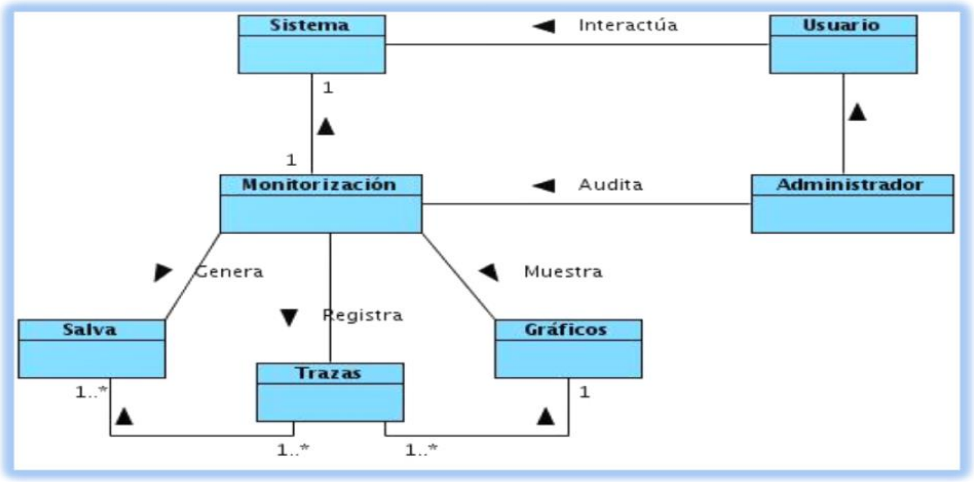


Figura 2. Modelo de Dominio.

La figura 2 muestra la clase Sistema que representa cualquier aplicación desarrollada con Symfony2 donde el usuario y el administrador interactúan con ella. Destacar que, el módulo de monitorización es el encargado de proporcionar al administrador las funcionalidades necesarias para generar salvas, así como mostrar reportes a través de gráficos haciendo uso de las trazas registradas.

Descripción de las clases del dominio

Usuario: Persona que interactúa con el sistema.

Administrador: Individuo que interactúa y controla todas las acciones a realizar con el sistema. **Sistema:** Cualquier sistema implementado con el framework de desarrollo Symfony2.

Monitorización: Módulo de monitorización para aplicaciones desarrolladas con Symfony2 que añade la nueva funcionalidad de auditar las acciones de los usuarios.

Trazas: Entidad que almacena los datos relacionados con cada una de las acciones monitorizadas.

Salvas: Fichero que almacena la información de las trazas.

Gráficos: Permite mostrar la información de las trazas almacenadas para la toma de decisiones.

2.2 Requisitos del sistema

Los requisitos del sistema permiten gestionar las necesidades del proyecto de forma estructurada. Por este motivo, se han convertido en un punto clave en el desarrollo de las aplicaciones informáticas, pues un gran número de proyectos de software naufragan a menudo debido a una mala definición, especificación o

administración de requisitos. Estos se dividen generalmente en dos grupos: funcionales y no funcionales.

2.2.1 Requisitos funcionales

Los requisitos funcionales de un sistema describen lo que el sistema debe hacer, y dependen del tipo de software que se desarrolle, de los posibles usuarios y del enfoque general tomado por la organización al redactarlos. Estos se pueden expresar de diferentes formas adecuándose al entorno en que se desarrolle, debido a esto el módulo de monitorización de usuarios para aplicaciones desarrolladas con Symfony2 presenta los siguientes requisitos:

- **RF1 Configurar la monitorización**

  Descripción: Se seleccionan las entidades a monitorizar en cualquier aplicación desarrollada con Symfony2 mediante la anotación @Traza.

  Entrada: @Traza en las clases entidades.

- **RF2 Registrar datos de las trazas**

  Descripción: Se registra en el sistema cada acción realizada por cualquier usuario, siempre y cuando el administrador decida si es monitorizada o no.

  Entrada: usuario, fecha, hora, ip, acción y detalles de las entidades monitorizadas.

  Salida: Se almacenan los datos en la base de datos.

- **RF3 Listar datos de las trazas**

  Descripción: El sistema muestra todos los datos de las trazas.

  Salida: Se listan todas las trazas almacenadas, con los siguientes datos: id, ip, usuario, fecha, hora, acción y detalles.

- **RF4 Filtrar datos de las trazas**

  Descripción: El sistema debe brindar la posibilidad de seleccionar según el tipo de filtro de búsqueda que desee como por ejemplo: usuarios, fecha (desde-hasta), ip y por las acciones realizadas.

  Entrada: Se introducen los datos necesarios para la búsqueda.

  Salida: Se muestran las trazas que cumplen con el criterio establecido por el usuario.

- **RF5 Graficar datos asociados a las trazas**

  Descripción: Se muestran diferentes gráficos con los datos de las trazas.

  Entrada: Se selecciona el criterio por el cual se desea graficar, que pueden ser:

  1- Tipos de trazas (gráfico de barras).

  2- Trazas por ip (gráfico de barras).

  3- Acciones por hora (gráfico de series).

  Salida: Se muestra un gráfico con las características seleccionadas.

- **RF6 Realizar salvas de las trazas**

  Descripción: El sistema debe guardar en un fichero las trazas que se encuentran almacenadas en la base de datos.

  Entrada: ip, usuario, fecha, hora, acción y detalles.

  Salida: Un fichero de tipo log con la siguiente estructura Traza fecha (desde-hasta) por ejemplo: Traza 05-04-2013 09-04-13.log.

2.2.2 Requisitos no funcionales

Los requisitos no funcionales son características que hacen al producto atractivo, usable, rápido y confiable. El incumplimiento de uno de los requisitos no funcionales puede significar que el sistema entero sea inutilizable, por lo cual es de suma importancia especificar los mismos. Seguidamente se describen los requisitos no funcionales del módulo de monitorización de usuarios para aplicaciones desarrolladas con Symfony2.

Usabilidad

- El módulo deberá presentar facilidades al usuario para el manejo de la información, se pretende una vista descriptiva para la realización de todas las operaciones, con el objetivo de propiciar un buen entendimiento del mismo a los usuarios finales. Debe ser sencillo, manejable y fácil de usar.

Portabilidad

- Será un sistema multiplataforma, lo que permitirá poder disponer del mismo en cualquier sistema operativo.

Soporte

- Se garantizará la instalación del módulo y se impartirá capacitaciones a los administradores interesados. Además, se contará con un manual de instalación que guía paso a paso las acciones que se deben realizar para utilizar el módulo propuesto.

Restricciones de diseño e implementación

- El sistema deberá ser implementado en los lenguajes de programación PHP 5.3.10 y JavaScript 1.8 o superior.
- Se utilizarán los frameworks de desarrollo Symfony 2.2.4 y Ext JS 4.1.1.
- Se empleará la herramienta de desarrollo NetBeans 7.2.

Software

El servidor donde se instalará el módulo debe cumplir con los siguientes requisitos:

- Sistema Operativo: GNU/Linux preferentemente Ubuntu GNU/Linux 10.10 o superior, Windows.
- Paquetes: apache2, php5, libapache2-mod-php5, php5-cli, php5-mysql, php5-pgsql, php5-sqlite, php5-sybase, php5-xsl, php5-gd, php-apc.
- Bundle: FOSJsRoutingBundle[3].

Hardware

Las PC´s clientes deben cumplir con los siguientes requisitos de hardware:

- Ordenador Pentium IV o superior, con 1.7 GHz de velocidad de microprocesador.
- Memoria RAM mínimo 256 MB.

La PC Servidor debe cumplir con los siguientes requisitos de hardware:

- Ordenador Pentium IV o superior, con 1.7 GHz de velocidad de microprocesador.
- Memoria RAM mínimo 1Gb.
- Disco Duro con 10Gb mínimo de capacidad, para instalar el sistema.

[3] FOSJsRoutingBundle: es un bundle encargado de interpretar las rutas de Symfony desde JavaScript.

2.3 Diagrama de casos de uso del sistema

Los diagramas de casos de uso del sistema (DCUS) son importantes para visualizar y especificar el comportamiento del sistema y está compuesto por un conjunto de casos de uso, actores y sus relaciones. Los casos de uso engloban los requisitos funcionales de un sistema facilitando su interpretación, lo que hace que sean especialmente útiles para la comunicación entre el equipo de desarrollo y el cliente. Además, permiten definir los límites del sistema y las relaciones entre el sistema y su entorno. A continuación se presenta el DCUS del módulo de monitorización de usuarios para aplicaciones desarrolladas con Symfony2 (ver figura 3).

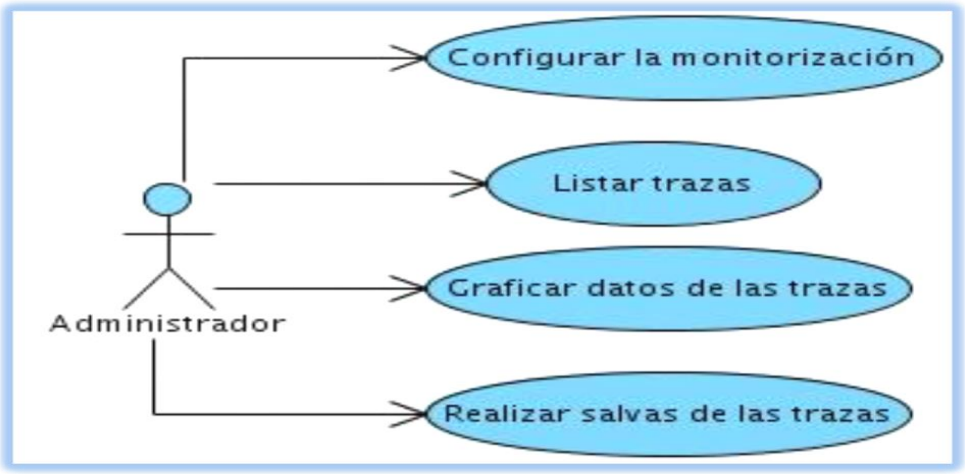


Figura 3. Diagrama de casos de uso del sistema.

Descripción de los casos de uso:

- **Configurar la monitorización:** Permite seleccionar las entidades que serán monitorizadas mediante la anotación @Traza y de esta forma se almacenan los datos de todas las acciones (ejemplo: crear, actualizar, eliminar y autenticar) realizadas por los usuarios de cualquier aplicación desarrollada con el framework de desarrollo Symfony2.
- **Listar trazas:** Permite listar las trazas almacenadas y buscarlas según los criterios seleccionados por los usuarios.
- **Graficar datos de las trazas:** Permite graficar datos específicos de las trazas almacenadas, en los tipos de gráficos de barras y de series.

- **Realizar salvas de las trazas:** Brinda la posibilidad de guardar las trazas registradas en archivos de tipo log.

Actor	Descripción
Administrador	Responsable de administrar los temas de seguridad del sistema, específicamente la auditoría, configurando lo que se desea monitorizar mediante la anotación @Traza. Además de listar las trazas realizadas por los usuarios, realizar gráficos de los datos almacenados y las salvas pertinentes.

Tabla1. Descripción del actor del sistema

Descripción textual de los casos de uso del sistema

Cada caso de uso se detalla mediante una descripción textual que describe la interacción entre el actor y el sistema. A continuación se muestra un ejemplo:

Descripción del caso de uso Listar trazas

Caso de Uso:	Listar trazas
Actores:	Administrador.
Resumen:	El caso de uso inicia cuando el administrador decide las entidades que se desean monitorizar. Permite listar todas las trazas almacenadas en la base de datos. El caso de uso finaliza satisfactoriamente cuando son listadas todas las trazas almacenadas.
Precondiciones:	El administrador sitúa @Traza en las clases entidades que desea monitorizar. El sistema debe estar conectado a la base de datos.
Referencias	**RF2**
Prioridad	**Crítico**
Flujo Normal de Eventos	

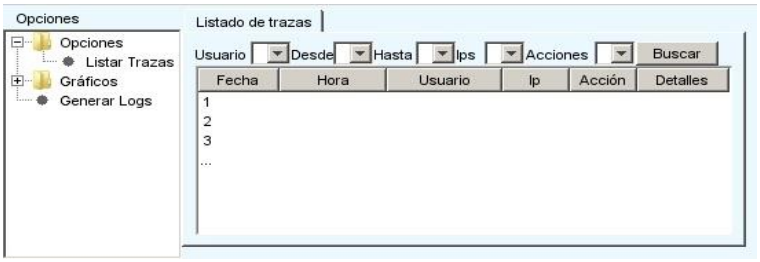
Acción del Actor	Respuesta del Sistema
1- El administrador selecciona la opción “Listar trazas”.	2- Busca las trazas almacenadas en la base de datos. 3- Muestra las trazas almacenadas.
Prototipo de Interfaz 	
Poscondiciones	Se listan las trazas almacenadas en la base de datos.

Tabla 2. Descripción del CU Listar trazas

2.4 Modelo de Diseño

El modelo de diseño es un modelo de objetos que describe la realización de los casos de uso, y se utiliza como una abstracción del modelo de implementación y el código fuente. Además, se utiliza para concebir y documentar el diseño del sistema de software, pues abarca todas las clases del diseño, subsistemas, paquetes, colaboraciones y las relaciones entre ellos.

2.4.1 Diagrama de clases del diseño

Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas y describen gráficamente las especificaciones de las clases de software y las interfaces en una aplicación. Un diagrama de clases está compuesto por los siguientes elementos:

- Clases: atributos, métodos y visibilidad.
- Relaciones: herencia, composición, agregación, asociación y uso.

La figura 4 muestra el diagrama de clases del diseño del módulo de monitorización de usuarios para aplicaciones desarrolladas con Symfony2, agrupándose en dos subsistemas y tres paquetes que contienen siete clases para el caso de uso Listar trazas.

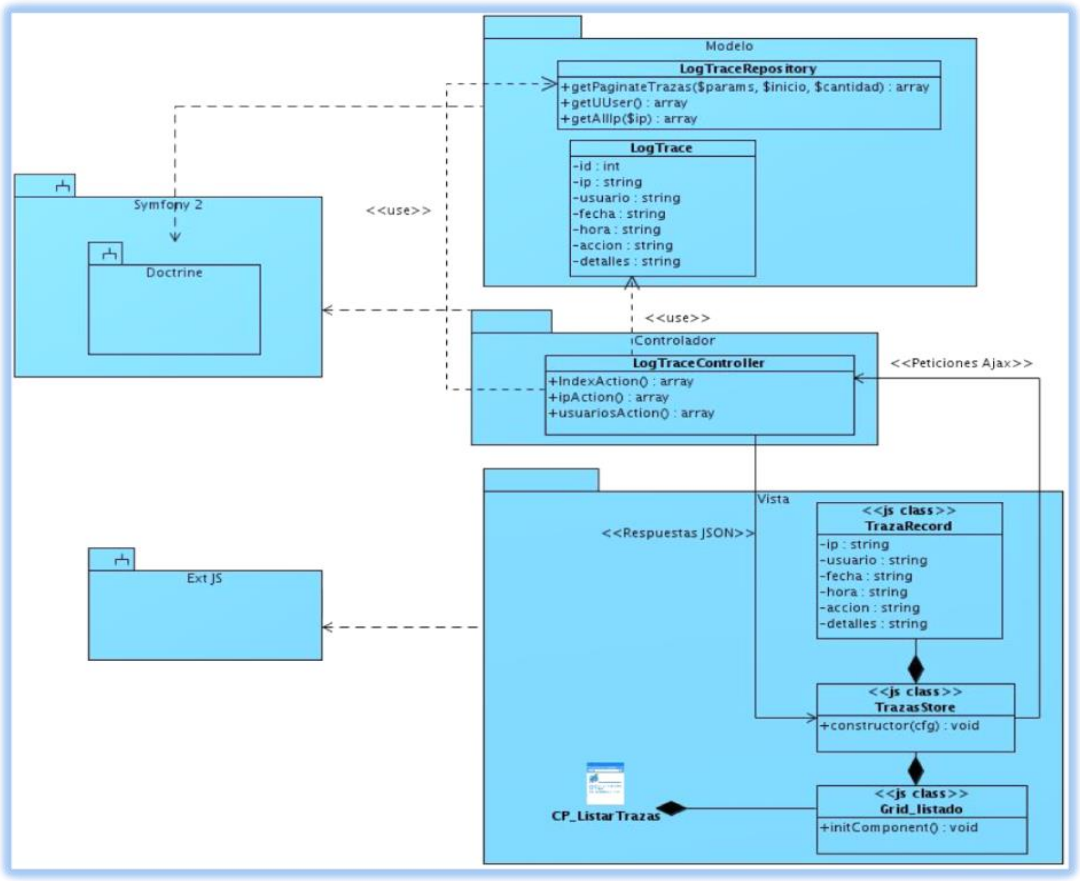


Figura 4. Diagrama de clases del diseño del CU Listar trazas.

Para simplificar el desarrollo de la aplicación se empleó uno de los patrones más utilizados en el framework Symfony2: el patrón Modelo-Vista-Controlador (MVC), que está formado por tres niveles. El Modelo representa la información con la que trabaja la aplicación, es decir, su lógica de negocio. La Vista transforma el modelo en una página web que permite al usuario interactuar con ella y el Controlador se encarga de procesar las interacciones del usuario y realiza los cambios apropiados en el modelo o en la vista.

Como se trata de una aplicación enriquecida para la web, se utiliza como framework de desarrollo del lado del cliente Ext JS 4.1.1. Para listar la información de las trazas almacenadas se utilizó el componente grid del framework anteriormente mencionado. Dicho componente utiliza un store el cual se encarga de cargar los datos que devuelve el servidor a través de una petición AJAX. Las peticiones entrantes son manejadas

por el controlador LogTraceController el cual para su correcto funcionamiento hace uso de la clase LogTrace que representa la tabla existente en la base de datos y de la clase LogTraceRepository donde se implementan los métodos necesarios para proveer los datos al controlador.

Mediante el framework de desarrollo Symfony2 se utiliza la clase LogTraceController, la cual brinda la posibilidad al controlador del módulo, de acceder fácilmente a los servicios que ofrece este framework. Además, se emplea la clase LogTraceRepository perteneciente a Doctrine, facilitando una mayor flexibilidad en el manejo de la base de datos.

2.4.2 Diagrama de interacción

Un diagrama de interacción (secuencia o colaboración), representa la forma en que un cliente (actor) y otros objetos (clases) se comunican entre sí, en respuesta a un determinado suceso. (19)

El diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada caso de uso. Una vez que se han identificado los eventos de un caso de uso, se crea una representación de cómo éstos causan un flujo de un evento a otro, como una función del tiempo. Este tipo de diagrama, muestra los objetos que participan en la interacción mediante las líneas de vida y los mensajes que intercambian.

En el presente trabajo se utilizan diagramas de secuencia para indicar cómo los eventos causan transiciones de objeto a objeto. A continuación se representa para el caso de uso Listar trazas (ver figura 5).

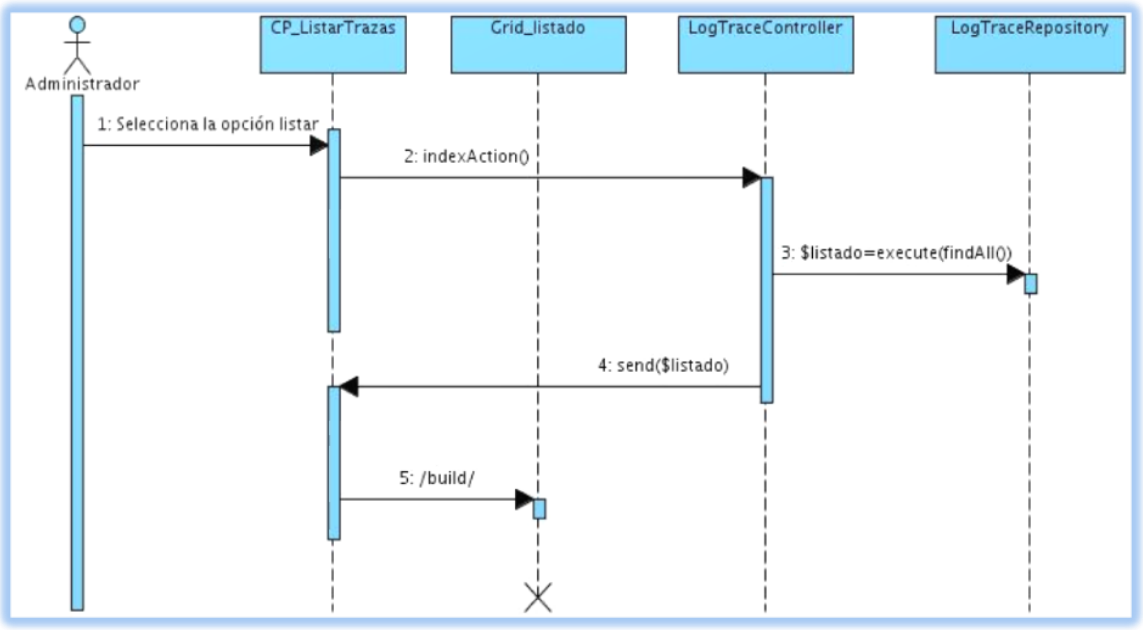


Figura 5. Diagrama de secuencia del CU Listar trazas.

En este diagrama de secuencia el administrador selecciona la opción Listar Trazas en la página cliente "Listar Trazas", mediante una petición AJAX al controlador "LogTraceController", este interactúa con el "LogTraceRepository" ejecutando la acción findAll(). El controlador devuelve una respuesta en formato JSON con todos los datos suministrados por el repositorio. La CP_ListarTrazas recoge estos datos y construye el "Grid_ListarTrazas" a partir de la respuesta del controlador.

2.5. Patrones de diseño

Los patrones de diseño constituyen el esqueleto de las soluciones a problemas comunes en el desarrollo de software. Estos brindan una solución probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares. (20)

Patrones GRASP

GRASP, es un acrónimo que significa General Responsibility Asignment Software Patterns (patrones generales de software para asignar responsabilidades). Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos. El nombre se eligió para indicar la importancia de captar (grasping) estos principios, si se quiere diseñar eficazmente el software orientado a

objetos. (20)

Para la implementación de Symfony se utilizan varios patrones, situándolos en las capas de Modelo y Control, que plantea el patrón arquitectónico MVC. Dentro de los patrones GRASP empleados en la solución del módulo se encuentran: Experto, Creador, Bajo Acoplamiento, Alta Cohesión y Controlador.

Experto: Es uno de los patrones que más se utiliza cuando se trabaja con Symfony, con la inclusión de la biblioteca Doctrine2 para mapear la Base de Datos. Symfony2 utiliza esta librería para realizar su capa de abstracción en el modelo, encapsular toda la lógica de los datos y generar las clases con todas las funcionalidades comunes de las entidades, las clases de abstracción de datos (Entity del Modelo) poseen un grupo de funcionalidades que están relacionadas directamente con la entidad que representan y contienen la información necesaria de la tabla que representan.

Creador: En la clase EntityManager se encuentran las acciones definidas para el sistema y se ejecutan en cada una de ellas. En dichas acciones se crean los objetos de las clases que representan las entidades, lo que evidencia que la clase EntityManager es “creador” de dichas entidades. Ejemplos de algunas funciones utilizadas en la clase EntityManager son: findAll (), findByXXX (), findOneBy ().

Alta Cohesión: Symfony permite la organización del trabajo en cuanto a la estructura del proyecto y la asignación de responsabilidades con una alta cohesión. Un ejemplo de ello es la clase Action, la cual está formada por varias funcionalidades que están estrechamente relacionadas, siendo la misma la responsable de definir las acciones para las plantillas y colaborar con otras para realizar diferentes operaciones, instanciar objetos y acceder a las Properties.

Bajo Acoplamiento: La clase TrazasController hereda únicamente de Controller para alcanzar un bajo acoplamiento de clases. Las clases que implementan la lógica del negocio y de acceso a datos se encuentran en el modelo, las cuales no tienen asociaciones con las de la vista o el controlador, lo que proporciona que la dependencia en este caso sea baja.

Controlador: Todas las peticiones web son manipuladas por un solo controlador frontal (app.php), que es el punto de entrada único de toda la aplicación en un entorno determinado. Este patrón se evidencia en las clases AppKernel, Controller, Request, los “actions” y el app.php del ambiente.

Patrones GOF

Es la abreviación del grupo Gang of Four, compuesto por Erich Gamma, Richard Helm, Ralph Jhonson y John Vlisides, quienes en su publicación “Design Patterns” (década de los 90s), describen 23 patrones de diseño comúnmente utilizados y de gran aplicabilidad en problemas de diseño. Estos patrones se agrupan en las siguientes categorías: Creacionales, Estructurales y de Comportamiento. (21) El framework de desarrollo Symfony2 implementa varios patrones, por lo cual se explicará el más significativo de cada una de las categorías expuestas anteriormente.

Patrones Creacionales: Inicialización y configuración de objetos. Dentro de estos se encuentran numerosos tipos como por ejemplo:
Fábrica Abstracta (Abstract Factory), Constructor (Builder), Instancia Única (Singleton), entre otros.

- Singleton: El principal objetivo de este patrón es asegurarse que sólo pueda haber una instancia de una clase facilitando el punto de acceso global a dicha instancia. Este patrón es frecuentemente implementado en clases que manejan base de datos, controladores y Request/Response object.

Patrones Estructurales: Separan la interfaz de la implementación. Se ocupan de cómo las clases y objetos se agrupan, para formar estructuras más grandes. Dentro de estos se encuentran numerosos tipos como por ejemplo:
Adaptador (Adapter), Decorador (Decorator), Fachada (Facade), Puente (Bridge), Objeto Compuesto (Composite), Peso Ligero (Flyweight), Apoderado (Proxy).

- Decorator: Este patrón añade dinámicamente nuevas responsabilidades a un objeto. Este método pertenece a la clase abstracta View, padre de todas las vistas, que contienen un decorador para permitir agregar funcionalidades dinámicamente. El archivo nombrado layout.html.twig es el que contiene el Layout de la página. Este archivo, conocido también como plantilla global, guarda el código HTML que es usual en todas las páginas del sistema, para que no se repita en cada página. El contenido de la plantilla se integra en el layout, o si se mira desde el otro punto de vista, el layout decora la plantilla.

Patrones de Comportamiento: Describen los objetos o clases y la comunicación entre ellos. Dentro de estos se encuentran numerosos tipos como por ejemplo:

Comando (Command), Mediador (Mediator), Patrón Observador (Observer).

- Observer: El componente Event Dispatcher Symfony2 implementa el patrón Observer en una forma simple para que sus proyectos sean realmente extensible. Un ejemplo sobre esto se evidencia cuando se llama al componente HttpKernel y éste, una vez que el objeto Response ha sido creado, puede ser utilizado en otros componentes del sistema para modificarlo, por ejemplo, agregarle una cabecera de caché antes de que sea utilizado realmente. Para hacer esto posible el núcleo de Symfony2 lanza un evento kernel.response.

2.6 Modelo de Datos

Desde tiempos remotos, los datos han sido registrados por el hombre en algún tipo de soporte como por ejemplo en piedra, papel y madera, a fin de que quedara constancia de un fenómeno o idea. El modelo de datos es una representación abstracta de los datos y las relaciones que existen entre ellos. Por lo general, un modelo de datos permite describir el tipo de datos que incluye la base de datos y la forma en que se relacionan, así como las restricciones de integridad y las operaciones de manipulación de los datos. La siguiente figura (ver figura 6) muestra la estructura del modelo de datos del módulo de monitorización para aplicaciones desarrolladas con Symfony2.

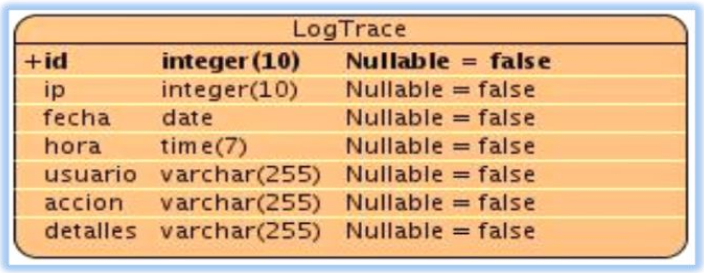


Figura 6. Modelo de Datos.

El modelado es la actividad más delicada e importante en la realización de una aplicación con base de datos. Para la realización de este módulo se obtendrá una tabla LogTrace la cual es la encargada de almacenar el identificador, ip de la PC, usuario, fecha, hora, acción y detalles con el objetivo de la persistencia de estos datos.

2.7 Modelo de Despliegue

El modelo de despliegue muestra la configuración física de un sistema facilitando la comprensión clara del funcionamiento donde se quiera apreciar la forma en que el software y el hardware trabajan juntos. Dentro de la vista, el elemento principal es el nodo, que se conecta a través de canales de comunicación. Un nodo puede contener algún software, ya sea un dispositivo, que puede ser tanto una computadora como alguna otra pieza de hardware conectada a un sistema, o un ambiente de ejecución de un lenguaje de programación. A continuación (ver figura 7) se muestra el modelo de despliegue del módulo de monitorización de usuarios para aplicaciones desarrolladas en Symfony2.

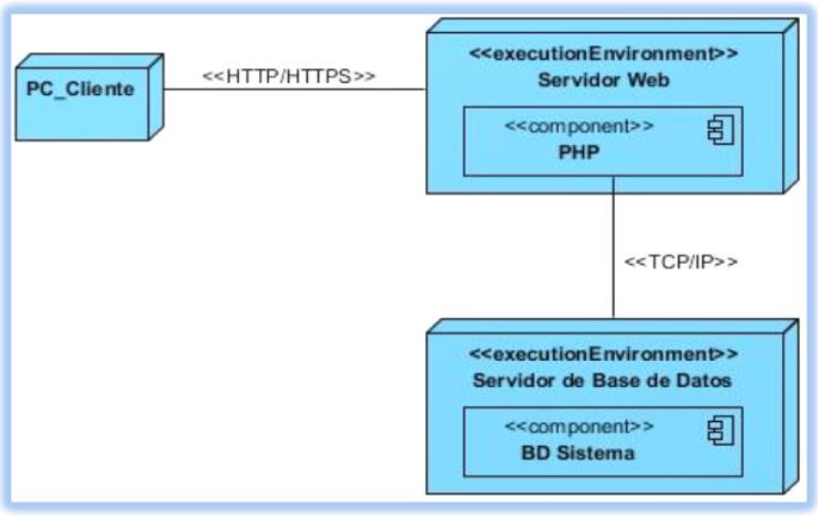


Figura 7. Modelo de Despliegue.

Estaciones de trabajo con la Aplicación Cliente:

Son las estaciones de trabajo usadas por los usuarios para interactuar con cualquier aplicación.

Protocolos de Comunicación:

Los protocolos son, como reglas de comunicación que permiten el flujo de información entre computadoras distintas.

- Conexión HTTP y HTTPS: El propósito de estos dos protocolo HTTP y HTTPS (Protocolo de transferencia de hipertexto y Protocolo de transferencia segura de hipertexto) es permitir la transferencia de archivos entre un navegador (el cliente) y un servidor web. Este último es más

apropiado para garantizar la seguridad durante el flujo de datos.

- Conexión TCP/IP: El protocolo TCP/IP (Transmition Control Protocol/Internet Protocol) hace posible enlazar cualquier tipo de computadoras, sin importar el sistema operativo que éstas usen.

Servidor web:

Es un programa que está diseñado para transferir contenido estático a un navegador, este carga un archivo y lo envía a través de la red al navegador de un usuario. El servidor Web se ejecuta sobre el servidor que escucha las peticiones HTTP que le llegan y las satisface. Dependiendo del tipo de petición, el servidor web buscará una página web o bien ejecutará un programa en el servidor.

Servidor de Base de Datos:

Un servidor es un nodo que forma parte de una red, provee servicios a otros nodos denominados clientes. Provee servicios de base de datos a otros programas u otras computadoras, como es definido por el modelo cliente-servidor.

Los usuarios podrán conectarse a la aplicación (app) desde estaciones de trabajo por medio de un navegador web haciendo uso del protocolo de comunicación HTTP o HTTPS, pues el módulo propuesto es capaz de adecuarse al entorno donde está desplegado el proyecto original. En el servidor se encuentra la app desarrollada en PHP que atenderá las peticiones realizadas por cada uno de los usuarios y que a su vez utiliza una base de datos para satisfacer esas demandas. El servidor de base de datos puede estar desplegado en la misma PC que el servidor web o potencialmente en otra PC. La comunicación entre la app desarrollada y el servidor de base datos se realiza mediante el protocolo TCP/IP.

2.8 Conclusiones parciales

En el desarrollo del presente capítulo se describieron los seis requisitos funcionales que el módulo debe cumplir, agrupados en cuatro casos de uso, los que a través del DCUS permitieron un mejor entendimiento de la lógica del módulo de monitorización. Además, con el propósito de describir cómo estará estructurado el mismo, se realizó el modelo de diseño y se representó la estructura estática y dinámica, a través del diagrama de clases del diseño y de los diagramas de interacción (secuencia) respectivamente. Apoyado en los seis requisitos no funcionales se realizó el modelo de despliegue,

exponiendo las restricciones del diseño. Para minimizar el riesgo de generar un mal diseño y con el propósito de obtener un módulo de calidad se identificaron los patrones GRASP y GOF.

Capítulo III: Implementación y pruebas

Introducción

Una vez realizado el modelo del diseño, este capítulo tiene como objetivo tomar los elementos obtenidos para implementar el módulo, siguiendo una secuencia lógica de pasos. Se realiza el modelo de implementación, lo que permite obtener los diagramas de componentes. Además, se muestran los principales fragmentos de código y la descripción de los estándares de codificación utilizados para llevar a cabo el desarrollo de la solución y mediante los diferentes tipos de pruebas aplicadas se verifica el correcto funcionamiento del módulo, mostrando además sus resultados.

3.1 Modelo de implementación

El modelo de implementación es comprendido por un conjunto de componentes y subsistemas que constituyen la composición física de la implementación del sistema. Entre los componentes se pueden encontrar datos, archivos, ejecutables, código fuente y los directorios. Fundamentalmente, se describe la relación que existe entre los paquetes, las clases del modelo de diseño, los subsistemas y componentes físicos. (22)

3.1.1 Diagrama de componentes

En los diagramas de componentes se muestran los elementos de diseño de un sistema de software. Un diagrama de componentes permite visualizar con facilidad la estructura general del sistema y el comportamiento del servicio que estos componentes proporcionan y utilizan a través de las interfaces.

Se puede usar un diagrama de componentes para describir un diseño que se implemente en cualquier lenguaje o estilo. Solo es necesario identificar los elementos del diseño que interactúan con otros elementos del diseño a través de un conjunto restringido de entradas y salidas. (23)

El módulo que motiva el presente trabajo, contará con tres paquetes de implementación básicos. Estos paquetes son:

- El Paquete de Clases Vista, que agrupa los componentes que permiten la interacción directa con el usuario final del módulo, mostrando y recogiendo información.

- El Paquete de Clases Controlador, que contendrá la clase que manipula los eventos del usuario y realiza peticiones al subsistema modelo para mostrarlas en las vistas.
- El Paquete de Clases Modelo, que agrupa las clases que interactúan con la base de datos y velan por el cumplimento de las reglas del negocio. (ver figura 8)

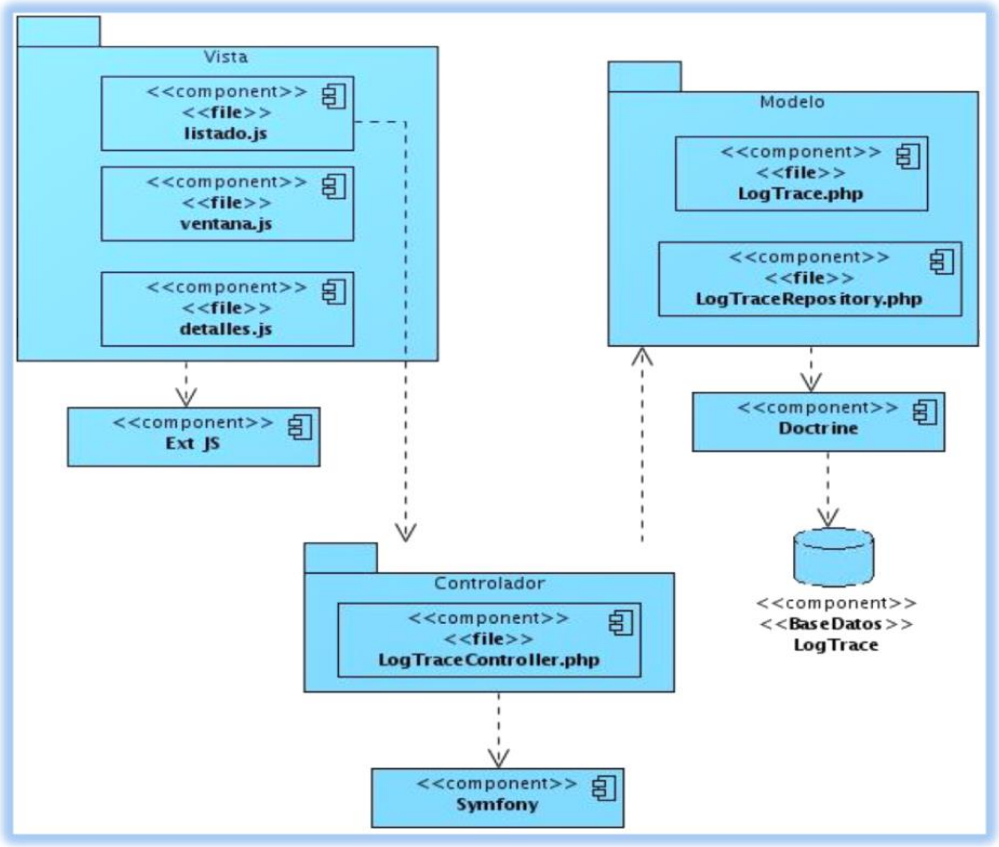


Figura 8. Diagrama de componentes del CU Listar Trazas.

3.2 Código fuente

El código fuente, también llamado código base, es un texto que se ha escrito en un lenguaje de programación concreto, y que sólo puede ser leído por un experto o programador. Este código está escrito en un lenguaje de programación que consiste en un conjunto de símbolos, reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones.

3.2.1 Estándares de codificación

Los estándares de codificación son pautas que se deben seguir para permitir que el código fuente de la solución posea mayor calidad, además de posibilitar el entendimiento por parte de otros programadores con el objetivo de facilitar el futuro mantenimiento de la aplicación informática.

Estilo de Codificación Utilizado:

1- Todas las etiquetas php deben ser completas (<?php?>), no reducidas (<? ?>).
2- Un solo espacio después de cada delimitador coma.
3- Un solo espacio alrededor de los operadores. Por ejemplo: ==, &&.
4- Todas las funciones y clases deben estar comentadas. Los comentarios deben ser añadidos de forma que resulten prácticos, para explicar el flujo del código y el propósito de las funciones o variables.
5- Mayúsculas intercaladas —sin guiones bajos— en nombres de variable, función, método o argumentos.
6- Guiones bajos para nombres de opción y nombres de parámetro.
7- Llaves para indicar la estructura del cuerpo de control, independientemente del número de declaraciones que contenga.

Ejemplo de código fuente:

A continuación se muestra el fragmento de código del método: “Listar trazas”, indispensable para el desarrollo del módulo de monitorización de usuarios para aplicaciones desarrolladas con Symfony2 (ver figura 9).

```php
class LogTraceController extends Controller
{
    /** IP para validar el stacked ...*/
    private $ip = '::1';

    /**
     * Listado de todas las trazas del sistema.
     * @return JsonResponse
     *
     */
    public function indexAction()
    {
        $em = $this->getDoctrine()->getManager();
        $inicio = $this->getRequest()->query->get('page');
        $fin = $this->getRequest()->query->get('limit');
        $cantidad = $em->getRepository('AuditoriaBundle:LogTrace')->findAll();
        $entities = $em->getRepository('AuditoriaBundle:LogTrace')->getPaginateTrazas($inicio, $fin);
        $respuesta = array();
        $respuesta['success'] = true;
        $respuesta['total'] = count($cantidad);
        $respuesta['trazas'] = array();
        foreach ($entities as $entity) {
            $data = array(
                'id' => $entity->getId(),
                'fecha' => $entity->getFecha()->format('d-m-Y'),
                'hora' => $entity->getHora()->format('H:i:s'),
                'usuario' => $entity->getUsuario(),
                'ip' => $entity->getIp(),
                'accion' => $entity->getAccion(),
                'detalles' => $entity->getDetalles(),
            );
            array_push($respuesta['trazas'], $data);
        }
        return new JsonResponse($respuesta);
    }
```

Figura 9. Ejemplo de código fuente. Método "Listar trazas".

3.3 Pruebas de software

El desarrollo de un software implica una serie de actividades de producción donde la probabilidad de aparecer al menos una falla, es muy común. La prueba de software es un elemento substancial para el aseguramiento de la calidad del sistema y a su vez, representa una revisión final de las especificaciones, del diseño y de la codificación.

La creciente inclusión del software como un elemento más de muchos sistemas y la importancia de los costos asociados a un fallo del mismo, han motivado a la creación de pruebas más minuciosas y bien planificadas. La prueba es un proceso de ejecución de un programa con la intención de descubrir errores, donde se puede demostrar la existencia de defectos en el software.

Las pruebas del software en un proceso automatizado son utilizadas para conseguir minimizar el impacto del error humano y facilitar la mejora continua y la verdadera calidad del software. Además, son utilizadas para identificar posibles fallos de implementación, calidad o usabilidad de un programa.

3.3.1 Niveles de pruebas

Cuando se aplican pruebas a un software, se toman en cuenta una serie de objetivos en diferentes escenarios y niveles de trabajo, debido a que las pruebas son agrupadas por niveles que se encuentran en distintas etapas del proceso de desarrollo. Existen diferentes niveles de pruebas, cada una de ellas se realiza en determinados momentos del ciclo de vida del software como por ejemplo: desarrollador, independiente, unidad, integración, sistema y aceptación. Para el desarrollo del módulo de seguridad se seleccionaron de estos seis niveles, dos de ellos: desarrollador e integración.

Tipos de pruebas

Existen diferentes tipos de pruebas que se pueden aplicar para verificar que el módulo de monitorización de usuarios, cumple con todas los requisitos identificados. Dentro de los tipos de pruebas que se realizan se encuentran:

- Pruebas funcionales: Las pruebas funcionales son aquellas que tienen por objetivo probar que los sistemas desarrollados, cumplan con las funciones específicas para los cuales han sido creados, es común que este tipo de pruebas sean desarrolladas por analistas de pruebas con apoyo de algunos usuarios finales; esta etapa suele ser la última y al dar conformidad sobre esta, el paso siguiente es el pase a producción. (24) Las pruebas funcionales tienen como objetivo asegurar el trabajo apropiado de los requisitos funcionales, incluyendo la navegación, entrada de datos, procesamiento y obtención de resultados.

- Pruebas de integración: Las pruebas de integración involucran un número creciente de módulos y terminan probando el sistema como un conjunto. Cuando se procede a añadir un nuevo módulo a

un software como parte de la integración, el software cambia, pues se establecen nuevos caminos lógicos de flujos de datos, por lo que esto puede traer consigo errores en funciones que antes funcionaban correctamente.

- Pruebas de Carga: Las pruebas de carga consisten en someter la aplicación a un régimen de carga de trabajo similar al esperado en su explotación real.

- Pruebas de Estrés: Las pruebas de estrés pudieran verse como un caso particular de prueba de carga, sometiendo al sistema a un régimen de carga de trabajo superior al esperado.

Método de prueba

Caja negra: Esta prueba básicamente se centra en los requisitos funcionales del software, lo que le permite al probador a través de un conjunto de entradas ejecutar los requisitos funcionales y así obtener posibles errores que puedan ser corregidos. Esta técnica no es una alternativa a la prueba de caja blanca, más bien se trata de un nuevo enfoque que detecta diferentes errores de los métodos de caja blanca. Esta técnica intenta descubrir errores de las siguientes categorías.

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores de estructura de datos o de conexión a bases de datos externas.
- Errores de rendimiento.

3.3.2 Diseño de casos de pruebas

Los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada, que se produce un resultado correcto y que la integridad de la información externa se mantiene.

A continuación se detallan las variables que se encuentran asociadas al caso de uso Listar trazas y dentro de este, la sección “Filtrar datos de las trazas”.

No	Nombre del campo	Clasificación	Valor nulo	Descripción
1	Usuarios	Campo de selección	No	Se selecciona el usuario.
2	Desde	Campo de selección	No	Se selecciona la fecha de inicio de la traza a buscar.
3	Hasta	Campo de selección	No	Se selecciona la fecha de fin de la traza a buscar.
4	IPs	Campo de selección	No	Se selecciona el IP.
5	Acción	Campo de selección	No	Se selecciona la acción ejecutada que puede ser: crear, actualizar, eliminar y autenticar.

Tabla 3. Descripción de las variables.

A partir de la descripción de variables se realizó la matriz de datos, donde se evaluó y probó la validez de cada uno de los datos seleccionados en el sistema, utilizando un juego de datos válido e inválido, mediante el empleo de la técnica de partición de equivalencia. Esta técnica divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.

Escenario	Variables (enumeradas según descripción de las variables)					Respuesta del sistema	Resultado de la prueba	Flujo central
	1	2	3	4	5			
Filtrar datos de las trazas	V	V	V	V	V	El sistema realiza la búsqueda según el criterio seleccionado, enviando los	Satisfactorio	El sistema envía los datos codificados en formato JSON, retornando al administrador

						datos encontrados al administrador.		los datos existentes.
	V	I	V	V	V	El sistema devuelve un mensaje tipo Exception debido a que ha ocurrido un error con la fecha de inicio.	Satisfactorio	

Tabla 4. Sección filtrar datos.

3.3.3 Resultados de las pruebas

Pruebas funcionales

Después de realizar las pruebas funcionales mediante el método de caja negra, utilizando los casos de prueba asociados a cada caso de uso, se comprobó el correcto funcionamiento del módulo. Durante las tres iteraciones realizadas, se detectaron un total de seis no conformidades, quedando todas resueltas, como se muestra sobre la figura 10 y la tabla 5.

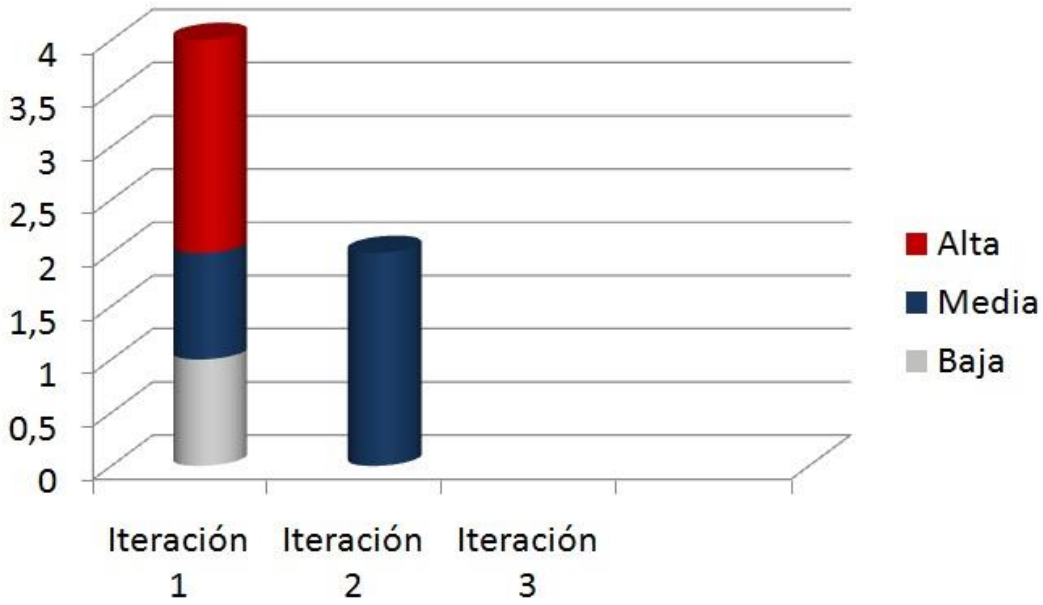


Figura 10. Resultados de las pruebas funcionales según la complejidad.

Iteraciones	*Baja*	*Media*	*Alta*
1	Ortografía	Validación	Programación
2	-	Validación	-
3	-	-	-

Tabla 5. Descripción de las iteraciones realizadas durante las pruebas.

Pruebas de integración

Luego de realizar las pruebas funcionales, se integró el módulo al proyecto SIGDAT y se pudo comprobar que funciona correctamente. En el anexo 3 se evidencia un aval que corrobora la correcta integración del módulo al proyecto antes mencionado.

Pasos para realizar la integración:

1- Instalar el bundle FOSJsRoutingBundle, pues este es el encargado de interpretar las rutas de Symfony desde JavaScript.
2- Se procede a instalar el módulo de monitorización de usuarios, como aparece en el fichero config.txt que están dentro de la raíz del proyecto.

Pruebas de carga y estrés

Para realizar las pruebas de carga y estrés se integró el módulo propuesto al proyecto SIGDAT, conectándose un número determinado de usuarios a la misma. Esto fue posible gracias a la facilidad que brinda la herramienta JMeter para realizar peticiones sobre un servidor de base de datos y después repetirlas, simulando la conexión de un número de usuarios tantas veces como se desee.

- Carga

De cada prueba se tomaron los tiempos de respuestas y algunos datos estadísticos que proporciona el JMeter en el informe agregado (ver figura 11). Informe que contiene por las filas, cada una de las solicitudes realizadas al servidor y por las columnas los elementos medidos en la prueba.

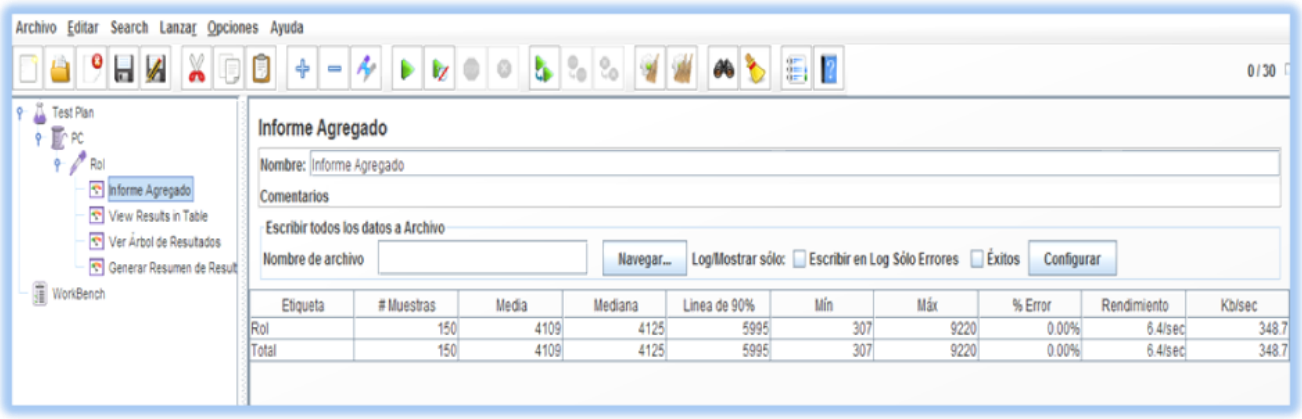


Figura 11. Resultados del informe agregado para 30 usuarios con 5 bucles.

Para esta prueba, el tiempo promedio de ejecución de una petición es de 4 segundos, realizándose un total de 150 solicitudes al servidor sin errores, lo que significa que todas las peticiones fueron servidas correctamente. Además se puede observar un rendimiento de 6,4 peticiones por segundos y un tiempo máximo de ejecución de petición que no sobrepasan los 10 segundos. Por lo que se puede afirmar que el tiempo de respuesta del módulo es aceptable para 150 solicitudes que se realicen desde 30 máquinas que se conecten.

Se continuó realizando las mismas transacciones de carga de trabajo, pero para una cantidad variada de usuarios, obteniéndose los resultados expuestos en la tabla 6.

Usuarios	*Cantidad de solicitudes*	*Tiempo de respuesta (s)*	*% de errores*	*Rendimiento (petición/seg)*	*Tiempo máximo de una petición (s)*
30	150	4.1	0	6.4	9.2
60	300	14.4	0	3.8	25.1
120	600	36.7	0	3.0	53.3

Tabla 6. Resultados generales del informe agregado.

- Estrés

Esta prueba de realizó con un total de 200 usuarios conectados y 5 bucles[4], arrojando un total de 1000 peticiones. Para esta muestra el sistema demostró poca solidez pues el rendimiento demoró más de una hora.

3.4 Interfaces del módulo

A continuación se muestran algunas secciones de la interfaz gráfica del módulo, encaminadas a lograr una visión agradable, sencilla y atractiva mediante la utilización del framework de presentación Ext JS.

[4] bucles: es la cantidad de ejecuciones que desarrolla una estación de trabajo.

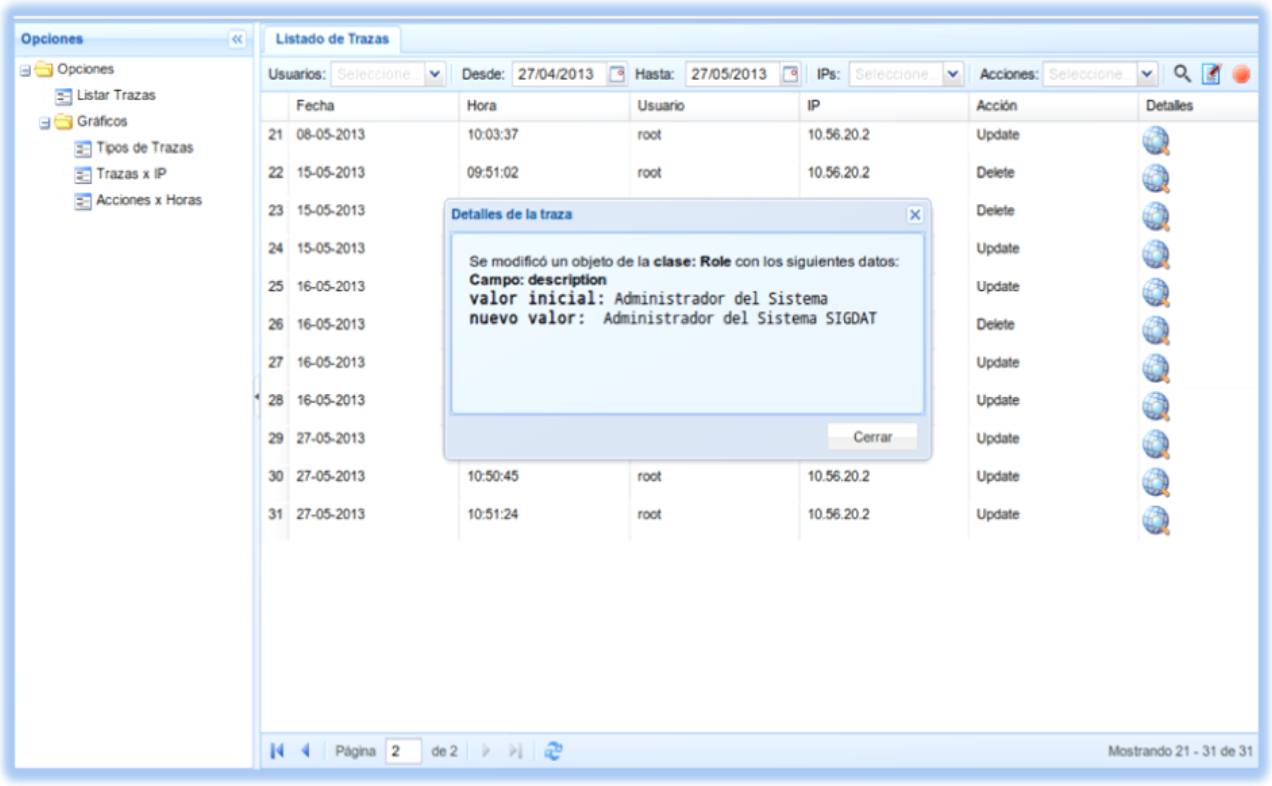


Figura 12. Interfaz: Listado de las trazas.

La figura 12 muestra la interfaz gráfica correspondiente al caso de uso “Listar trazas”. El administrador puede realizar un filtro para la búsqueda, según los siguientes criterios: usuarios, fecha (desde-hasta), ip y acción realizada. Además, si desea examinar los datos, selecciona la opción “Detalles”, la cual mostrará los últimos datos que estaban en la base de datos y los datos vigentes ingresados en el caso particularmente del “Update”.

3.5 Conclusiones parciales

Como resultado de este capítulo se estructuraron las clases del diseño en paquetes de componentes mostrando la organización y las dependencias entre los componentes que conforman el módulo. Se detalló el uso de un estándar de codificación, el cual es de gran importancia para lograr mayor organización y claridad, contribuyendo a realizar una programación explícita. Dando solución a los requisitos especificados en el capítulo anterior, se obtuvo la implementación de todas las funcionalidades, garantizando el correcto funcionamiento mediante las pruebas funcionales y de integración, utilizando el

método de caja negra basado en la técnica de partición de equivalencia. Mediante las pruebas de carga y estrés se evidenció a través de resultados concretos el rendimiento del módulo, permitiendo acotar el número de usuarios y peticiones realizadas al mismo. Se identificaron seis no conformidades durante las tres iteraciones, las cuales fueron corregidas.

Conclusiones generales

Una vez culminado el trabajo es posible afirmar que se le dio cumplimiento a los objetivos trazados para el mismo. Durante su realización se arribaron a las siguientes conclusiones:

- El estudio de los principales mecanismos de seguridad en Symfony2, evidenció la necesidad de incorporar la auditoría como una funcionalidad más de dicho framework, mediante la monitorización de las trazas.
- La obtención de los artefactos necesarios durante el proceso de análisis y diseño permitió obtener como resultado un mayor entendimiento de cómo se debe implementar el módulo de monitorización.
- La correcta especificación de los requisitos y la realización del diseño del módulo permitieron la implementación del mismo.
- La realización de las pruebas funcionales, integración, carga y estrés así como la resolución de las seis no conformidades detectadas durante este proceso, permitieron consolidar la seguridad del producto implementado.

Recomendaciones

Luego de haber analizado los resultados del presente trabajo de diploma, se recomienda:

- Incorporar mecanismos de alarmas al módulo de monitorización.
- Integrar el módulo propuesto al proyecto Generador Dinámico de Reportes GDR 2.0.

Referencias Bibliográficas

1. **Gómez López, Julio.** Administración de Sistemas Operativos. SFGS. [En línea] 2011. [Citado el: 11 de octubre de 2012] Disponible en: <www.adminso.es/index.php/4._Medidas_de_seguridad_en_los_sistemas_inform%C3%A1ticos>.

2. **Centro Criptológico Nacional.** Manual de Seguridad de las TIC. 2006.

3. **Wolter, Christian; Schaad, Andreas.** Modeling of Task-Based Authorization Constraints in BPMN [En línea] 2007. [Citado el: 20 de octubre de 2012] Disponible en: <link.springer.com/chapter/10.1007%2F978-3-540-75183-0_5?LI=true>.

4. **Fincowsky, Franklin; Benjamin, Enrique.** Auditoría Administrativa: gestión estratégica del cambio. 2007.

5. **Anaya, Raquel.** Una visión de la enseñanza de la Ingeniería de Software como apoyo al mejoramiento de las empresas de software. 2012.

6. **Garcilaso, Jordana.** Introducción Open UP. 2008.

7. **J.Sierra, Antonio.** Lenguajes de programación. 2011.

8. **php.net** php.net. [En línea] [Citado el: 20 de diciembre de 2012]. Disponible en: <www.php.net>.

9. **Eguíluz Pérez, Javier.** Introducción a Java Script. 2009.

10. **Maldonado, Daniel.** El CoDiGo K. Qué son los IDE de programación. 2007.

11. **netbeans.org.** netbeans.org. [En línea] [Citado el: 2 de diciembre de 2012]. Disponible en: <www.netbeans.org>.

12. **Sierra, María.** Trabajando con Visual Paradigm for UML. [En línea] 2010. [Citado el: 4de diciembre de 2012] Disponible en: <www.scribd.com/doc/65619842/Visual-Paradigm>.

13. **Gutiérrez J, Javier.** Framework web. 2008.

14. **symfony2.es** symfony2.es. [En línea] [Citado el: 6 de diciembre de 2012]. Disponible en: <www.symfony.es>.

15. **php.uci.cu** php.uci.cu. [En línea] [Citado el: 5 de enero del 2013]. Disponible en: <php.uci.cu/articles.php?article_id=87>.

16. **Frederick, Shea; Ramsay, Colin; Blades, Steve.** Learning Ext JS. 2008.

17. **JMeter.** [En línea]. [Citado el: 6 de enero de 2010.] Disponible en: <www.osmosislatina.com/jmeter/basico.htm>.

18. **Ortiz, Kadir Hector.** Eumed. [En línea] [Citado el: 20 de febrero del 2013]. Disponible en: <www.eumed.net/libros/2009c/583/Representacion%20del%20Modelo%20de%20Objetos%20de%20Dominio.htm>.

19. **Cáceres Tello, Jesús.** DIAGRAMAS DE SECUENCIA. Citado el: 1 de abril del 2013.] Disponible en: <www2.uah.es/jcaceres/capsulas/DiagramaSecuencia.pdf>.

20. **Reynoso, Carlos. Kiccillof, Nicolas.** Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. Versión 1.0. 2010.

21. **Veloso Hernández, Pedro.** Uso de patrones de arquitectura.2007.

22. **Rodríguez, Jose.** Modelo de Implementación. [En línea] [Citado el: 10 de marzo del 2013.] Disponible en: <merinde.rinde.gob.ve/index.php?option=com_content&task=view&id=96&Itemid=297>.

23. **Microsoft.Msdn.** MSDN. Diagramas de componentes de UML: Referencia. [En línea] [Citado el: 12 de marzo del 2013.] Disponible en: <msdn.microsoft.com/es-es/library/dd409390.aspx>.

24. **Oré B, Alexander**. FUNCTIONAL TESTING - PRUEBAS FUNCIONALES. [Citado el: 1 de abril del 2013.] Disponible en: <www.calidadysoftware.com/testing/pruebas_funcionales.php>.

Bibliografía

1. **Gomez Lopez, Julio.** Administración de Sistemas Operativos. SFGS. [En línea] 2011. [Citado el: 11 de octubre de 2012] Disponible en : <www.adminso.es/index.php/4._Medidas_de_seguridad_en_los_sistemas_inform%C3%A1ticos>.

2.**Centro Criptológico Nacional**. Manual de Seguridad de las TIC. 2006.

3. **Fincowsky, Franklin; Benjamin, Enrique.** Auditoría Administrativa: gestión estratégica del cambio. 2007.

4. **Wolter, Christian; Schaad, Andreas.** Modeling of Task-Based Authorization Constraints in BPMN [En línea] 2007. [Citado el: 20 de octubre de 2012] Disponible en: <link.springer.com/chapter/10.1007%2F978-3-540-75183-0_5?LI=true>.

5. **php.net** php.net. [En línea] [Citado el: 20 de diciembre de 2012]. Disponible en: <www.php.net>.

6.**Maldonado, Daniel.** El CoDiGo K. Qué son los IDE de programación. 2007.

7. **netbeans.org**. netbeans.org. [En línea] [Citado el: 2 de diciembre de 2012]. Disponible en: <www.netbeans.org>.

8. **Sierra, María.** Trabajando con Visual Paradigm for UML. [En línea] 2010. [Citado el: 4de diciembre de 2012] Disponible en : <www.scribd.com/doc/65619842/Visual-Paradigm>.

9. **Gutiérrez J, Javier.** Framework web. 2008.

10. **symfony2.es** symfony2.es. [En línea] [Citado el: 6 de diciembre de 2012]. Disponible en: <www.symfony.es>.

11. **php.uci.cu** php.uci.cu. [En línea] [Citado el: 5 de enero del 2013]. Disponible en: <php.uci.cu/articles.php?article_id=87>.

12. **Garcilaso, Jordana.** Introducción Open UP. 2008.

13. **Frederick, Shea; Ramsay, Colin; Blades, Steve.** Learning Ext JS. 2008.

14. **Eguíluz Pérez, Javier.** Introducción a Java Script. 2009.

15. **Mendez, Justo.** Lenguajes de programación. [En línea] 2011. [Citado el: 11 de noviembre del 2012] Disponible en : <www.monografias.com/trabajos/lengprog/lengprog.shtml>.

16. **Colombo, Clelia.** Innovación Democrática Y Tic, ¿Hacia Una Democracia Participativa?. 2006.

17. **Weitzenfeld, Alfredo.** Ingeniería de software orientada a objetos con UML, Java e Internet. 2005.

18. **Ardissone, Juan.** Symfony. Framework PHP orientado a objetos. [En línea] 2012. [Citado el: 20 de diciembre del 2012] Disponible en : <www.maestrosdelweb.com/editorial/curso-symfony2-introduccion-instalacion>.

19. **Ortiz, Kadir Hector.** eumed. [En línea] [Citado el: 5 de febrero del 2013]. Disponible en: <www.eumed.net/libros/2009c/583/Representacion%20del%20Modelo%20de%20Objetos%20de%20Dominio.htm>.

20. **Mendoza Navarro, Javier.** Diseño del Sistema de Tarjeta de Crédito Con UML. 2007.

21. **Reynoso, Carlos. Kiccillof, Nicolas.** Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. Versión 1.0. 2010.

22. **Veloso Hernández, Pedro.** Uso de patrones de arquitectura.2009.

23. **Rodriguez, Jose.** Modelo de Implementación. [En línea] [Citado el: 10 de marzo del 2013.] Disponible en: <merinde.rinde.gob.ve/index.php?option=com_content&task=view&id=96&Itemid=297>.

24. **Microsoft.Msdn.** MSDN. Diagramas de componentes de UML: Referencia. [En línea] [Citado el: 20 de febrero del 2013.] Disponible en: <msdn.microsoft.com/es-es/library/dd409390.aspx>.

25. **Oré B, Alexander**. FUNCTIONAL TESTING - PRUEBAS FUNCIONALES. [Citado el: 1 de abril del 2013.] Disponible en: <www.calidadysoftware.com/testing/pruebas_funcionales.php>.

26. **Anaya, Raquel.** Una visión de la enseñanza de la Ingeniería de Software como apoyo al mejoramiento de las empresas de software. 2012.

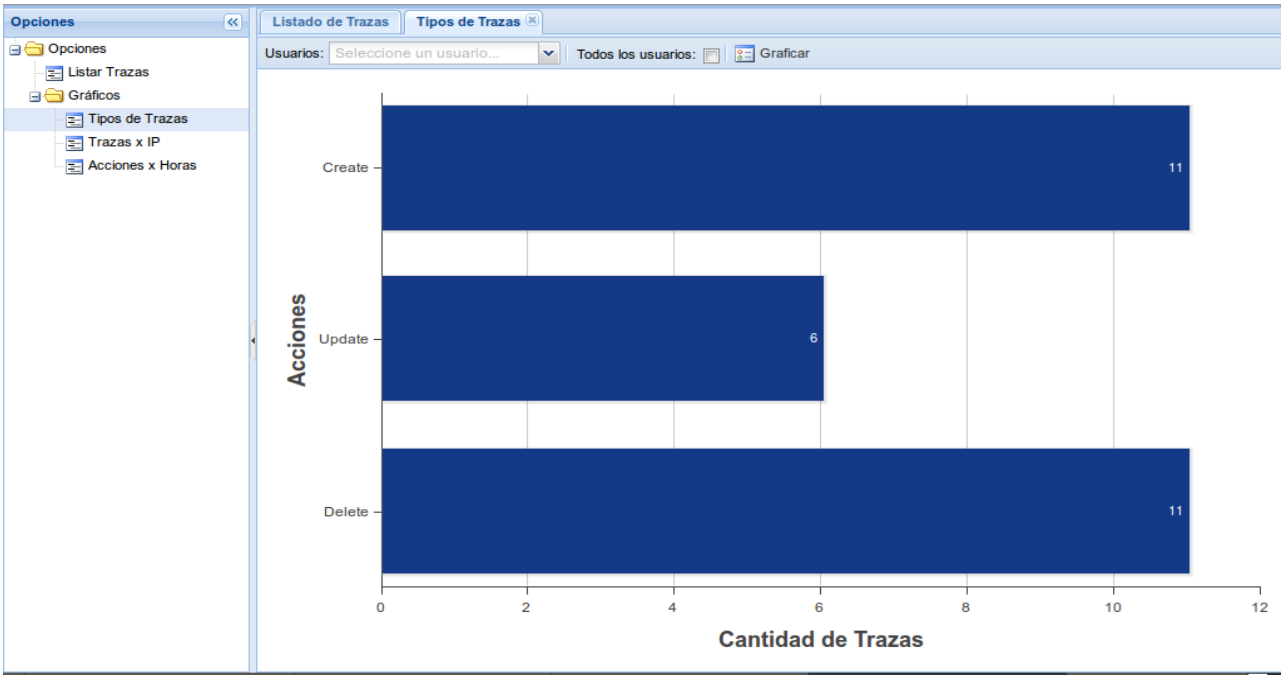
27. **J.Sierra, Antonio.** Lenguajes de programación. 2011.

28. **JMeter.** [En línea]. [Citado el: 6 de enero de 2010.] Disponible en: <www.osmosislatina.com/jmeter/basico.htm>.

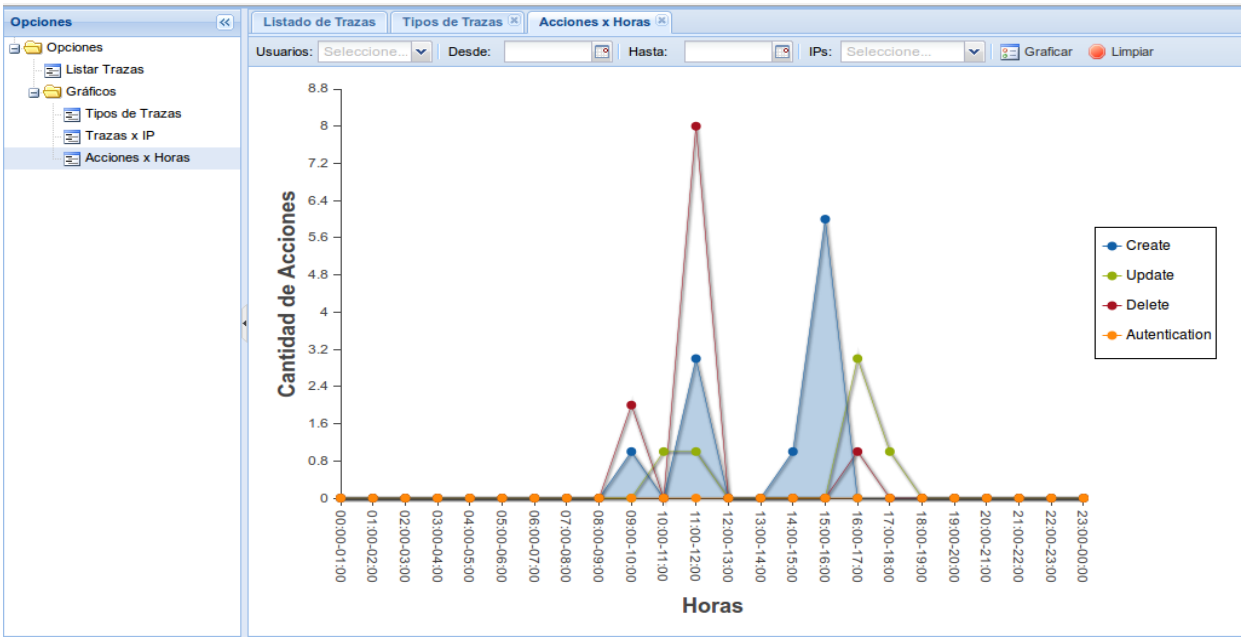
29. **Cáceres Tello, Jesús.** DIAGRAMAS DE SECUENCIA. Citado el: 1 de abril del 2013.] Disponible en: <www2.uah.es/jcaceres/capsulas/DiagramaSecuencia.pdf>.

Anexos

Anexo 1: Interfaz de la aplicación. Gráfico de barras referente a la cantidad de trazas contra acciones.



Anexo 2: Interfaz de la aplicación. Gráfico de series referente a cantidad de acciones contra horas.



Anexo 3: Acta de aceptación. Módulo de monitorización de usuarios para aplicaciones desarrolladas con Symfony2 integrado al proyecto SIGDAT.

UCI
Universidad de las Ciencias Informáticas

Centro de Tecnologías de Gestión de Datos
DATEC

La Habana, 08 de mayo del 2013
"Año 54 de la Revolución".

ACTA DE ACEPTACIÓN

De una parte, el Centro de Tecnologías de Gestión de Datos, en lo sucesivo DATEC, de la Universidad de las Ciencias Informáticas, representado en este acto por: Ing. Aldis Joan Abreu Medina, y de **otra parte** el estudiante: Graniela Gutiérrez Rodríguez.

Primero: Que en cumplimiento de los requisitos funcionales han sido efectuadas las implementaciones correspondientes.

CONSIDERANDO: Que los hitos realizados han sido desarrollados con la calidad requerida y bajo las condiciones pactadas y aprobadas por **Las Partes**.

CONSIDERANDO: Que los hitos que se han ejecutado cumplen con los requerimientos establecidos.

POR TANTO: Las Partes acuerdan formalizar mediante la presente Acta, la aceptación del producto: Módulo de monitorización de usuarios para aplicaciones desarrolladas con Symfony 2

Y para que así conste, se extiende la presenta **Acta** en dos (2) ejemplares, rubricados por **Las Partes**.

Ing. Aldis Joan Abreu Medina
Entrega

UCI Universidad de las Ciencias Informáticas Decanato Facultad 6

Graniela Gutiérrez Rodríguez
Recibe

Glosario de términos

CASE: Ingeniería de Software Asistida por Computación.

DATEC: Centro de Tecnologías de Gestión de Datos.

GRASP: Patrones de Software para la Asignación General de Responsabilidad

IDE: Entorno Desarrollo Integrado.

JSON: Notación de Objetos de JavaScript.

MVC: Modelo – Vista – Controlador.

OpenUp: Metodología para el proceso de desarrollo del software.

ORM: Mapeo de Objetos a Bases de Datos.

ORM: Object Relational Mapper es una técnica de programación que permite generar clases a través de las tablas de la base de datos.

SIGDAT: Sistema Integral para la Gestión de Datos.

UCI: Universidad de las Ciencias Informáticas.

UML: Lenguaje Unificado de Modelado.