Universidad de las Ciencias Informáticas

Centro de Tecnologías para la Formación (FORTES)

Facultad 4

Arquitectura de referencia para el desarrollo de aplicaciones web del centro FORTES (XALIX 2.0)

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

Autor: Hermes Rodríguez Quesada

Tutores: MSc. Yerandy Manso Guerra

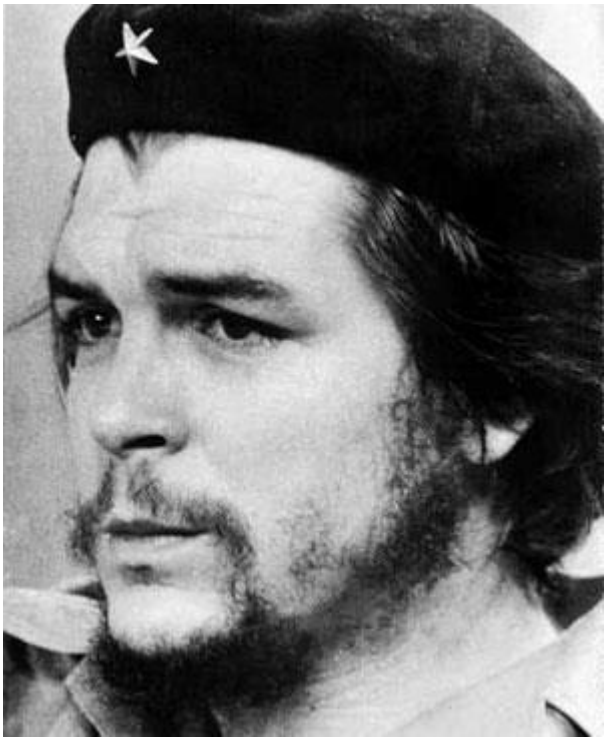
Ing. Ernesto Gutierrez Ramos

Co-tutor: Ing. César Andrés González Rodríguez

“Año 60 de la Revolución”

La Habana, Cuba

Junio 2018



"Seamos la pesadilla de los que pretenden arrebatarnos los sueños."

Ernesto Guevara de la Serna

Declaración de autoría

Declaro por este medio que yo Hermes Rodríguez Quesada, soy el autor principal del trabajo de diploma “Arquitectura de referencia para el desarrollo de aplicaciones web del centro FORTES (XALIX 2.0)” y que autorizo a la Universidad de las Ciencias Informáticas a hacer uso de la misma en su beneficio, así como los derechos patrimoniales con carácter exclusivo.
Para que así conste firmo la presente a los 25 días del mes de junio del año 2018.

Hermes Rodríguez Quesada
Autor

MSc. Yerandy Manso Guerra
Tutor

Ing. Ernesto Gutierrez Ramos
Tutor

Ing. César Andrés González Rodríguez
Co-Tutor

Datos de contacto

MSc. Yerandy Manso Guerra

El profesor es Máster en Ciencias. Posee la categoría docente Profesor Instructor. Se desempeña como Director del centro de desarrollo FORTES, en el cual se gestionan variados proyectos de software relacionados con la tecnología educativa. Actualmente se encuentra matriculado en el doctorado de Informática de la UCI, en el cual desarrolla el tema de investigación “Sistema recomendador de recursos según contextos en entornos ubicuos”. Ha impartido docencia en el pregrado específicamente en las asignaturas de la disciplina Práctica Profesional. Puede ser contactado por vía correo electrónico en la dirección ymanso@uci.cu.

Ing. Ernesto Gutierrez Ramos: Graduado de Ingeniero en Ciencias Informáticas en la novena graduación de la Universidad de las Ciencias Informáticas (UCI), en julio de 2015. Actualmente trabaja como especialista en el Departamento de Construcción de Componentes del Centro de Tecnologías para la Formación (FORTES) de la UCI. Puede ser contactado por vía correo electrónico en la dirección egutierrezr@uci.cu.

Ing. Cesar A González Rodriguez: Graduado de Ingeniero en Ciencias Informáticas en la décima graduación de la Universidad de las Ciencias Informáticas (UCI), en julio de 2016. Actualmente trabaja como RGA en el Departamento de Construcción de Componentes del Centro de Tecnologías para la Formación (FORTES) de la UCI. Puede ser contactado por vía correo electrónico en la dirección caglez@uci.cu.

Dedicatoria

Dedico mi tesis a mi tia y mi abuela que son merecedoras de este humilde obsequio que me respaldaron y apoyaron desde el primer dia, que estuvieron siempre cuando las necesité.

Agradecimientos

Agradezco a mi familia y amistades.

Resumen

La arquitectura XALIX es el pilar fundamental del Centro de Tecnologías para la Formación adscrito a la facultad 4 de la Universidad de las Ciencias Informáticas; sobre la cual se encuentran desarrollados diferentes proyectos del centro. Esta arquitectura está compuesta por el *framework* (marco de trabajo) Symfony en su versión 2.7.16 y un conjunto de *bundles* (paquetes) que hacen posible su funcionamiento. Estos elementos necesitan ser actualizados para evitar problemas de soportes y de obsolescencia tecnológica. Por ello se procede a desarrollar una nueva versión de la arquitectura XALIX, teniendo en cuenta buenas prácticas y las actualizaciones de los componentes de terceros, para permitir su evolución a las nuevas versiones de Symfony y mejorar su mantenibilidad. En esta investigación se logró desarrollar la arquitectura XALIX 2.0. Con esta arquitectura se actualiza el núcleo de Symfony a la versión 3.4, garantizando soporte a largo plazo, modernización de los lenguajes utilizados, entre otras mejoras. Además, XALIX 2.0 viene a suplir las limitantes de su predecesora, la falta de documentación y la mantenibilidad.

Palabras Clave:

Arquitectura, Framework, Symfony, XALIX.

Índice de contenidos:

Índice de Ilustraciones

Índice de tablas

INTRODUCCIÓN

La arquitectura de software es una disciplina que permite diseñar a un alto nivel la organización de un sistema. Este término es a menudo el mismo para sistemas con requerimientos similares y, por tanto, pueden soportar reutilización a gran escala (SOMMERVILLE, 2011). Cuando se trata de precisar una arquitectura que presente todas las características deseables en un dominio de aplicación, entonces es necesario definir un modelo de referencia. Estos tipos de arquitecturas representan un mecanismo de información para los arquitectos de aplicaciones del dominio seleccionado (TORRES *et al.*, 2016).

Una organización desarrolladora de software que pretenda llegar al máximo nivel de productividad, debe formalizar los procesos de su dominio de solución desde el punto de vista tecnológico. Para formalizarlos, se parte de los requisitos tecnológicos comunes en el entorno de las soluciones y se definen las estructuras necesarias para actuar como base de los sistemas. Esta definición se puede institucionalizar con la creación de una arquitectura de referencia para cada plataforma tecnológica en que desarrolle la entidad (TORRES *et al.*, 2016).

Asociado a lo anterior, en el Centro de Tecnologías para la Formación (FORTES) adscrito a la Facultad 4 de la Universidad de las Ciencias Informáticas (UCI), fue desarrollada la arquitectura XALIX 1.0. Esta arquitectura emplea al framework de PHP Symfony en su versión 2.7.16, junto a un grupo de paquetes o *bundles*, que organizan y facilitan el trabajo en producción. XALIX es la base de desarrollo de varias de las aplicaciones web de FORTES, lo que la convierte en un pilar fundamental para el Centro. Entre los productos desarrollados que emplean esta arquitectura se destacan la Plataforma Educativa ZERA, la Plataforma para la Editorial Félix Varela y el Sistema de Gestión de Ingreso a la Educación Superior (SIGIES). Sin embargo, con el decursar del tiempo se han apreciado vulnerabilidades y limitaciones en su mantenibilidad, lo que inciden en la necesidad de incorporar buenas prácticas a la arquitectura XALIX.

Una buena práctica en el desarrollo de software es mantener los sistemas y tecnologías actualizados. En el caso del framework Symfony, la última versión publicada y estable es la 3.4 lanzada en noviembre de 2017 y que cuenta con soporte y mantenimiento a largo plazo, definido hasta 2021. Además, la versión 2.7 de este framework cierra su ciclo de soporte y mantenimiento en 2018 para dar paso al soporte de las versiones 3.x. Esta es una vulnerabilidad de la actual arquitectura que afecta a su mantenibilidad[1] y genera a corto plazo la pérdida del soporte oficial.

[1] La mantenibilidad representa la capacidad del producto software para ser modificado efectiva y eficientemente, debido a necesidades evolutivas, correctivas o perfectivas.

Sumado a lo anterior, entre las principales limitantes detectadas en XALIX está la no existencia de documentación de los componentes que la integran, provocando retrasos y dudas en los equipos de desarrollo al incorporar nuevos miembros. Además, debido al condicionamiento impuesto por XALIX en cuanto a las tecnologías necesarias para su funcionamiento (versión del framework Symfony, del lenguaje, del servidor web, del servidor de base de datos, entre otros) existe una desactualización de la base tecnológica, lo que provoca que no se puedan aprovechar al máximo las potencialidades y mejoras de las tecnologías actualizadas. Así mismo, aparecen problemas de incompatibilidad para agregar *bundles* desarrollados por terceros o la difícil integración de estos con la arquitectura.

Teniendo en cuenta dicha situación, se presenta el siguiente **problema de la investigación:** Las limitaciones en el diseño de la arquitectura XALIX 1.0 impiden su mantenibilidad.

Se define como **objeto de estudio** la arquitectura XALIX 1.0, y como **campo de acción** el framework Symfony y los bundles de terceros de la arquitectura XALIX 1.0.

Para dar solución al problema científico de la investigación se plantea el siguiente **objetivo general:** Implementar la arquitectura de referencia XALIX 2.0 para el desarrollo de aplicaciones web en el Centro FORTES.

Para asegurar el cumplimiento del objetivo general se definen los siguientes **objetivos específicos**:

1. Elaborar el marco teórico de la investigación a través del estudio de la arquitectura XALIX 1.0 y sus componentes, para identificar los elementos que pueden incluirse en la solución propuesta.
2. Caracterizar la arquitectura XALIX 2.0 y sus componentes.
3. Analizar e implementar la arquitectura de referencia XALIX 2.0, que brinda la solución al problema planteado.
4. Realizar la validación la arquitectura XALIX 2.0.

Para el correcto desarrollo de la investigación, se emplearán los siguientes **métodos científicos:**

Métodos teóricos:

- **Análisis histórico-lógico:** para el análisis del histórico de versiones del framework Symfony, así como en los bundles, entre otros.

- **Analítico-sintético:** para analizar desde diferentes aristas los conceptos asociados a la arquitectura XALIX y sintetizar la información recopilada, permitiendo describir las características generales y las relaciones esenciales entre estas.
- **Hipotético-deductivo:** se utiliza en la elaboración de la problemática a resolver en la investigación y la propuesta de nuevas soluciones.
- **Análisis documental:** para realizar consultas a la literatura especializada en el tema abordado en la presente investigación.
- **Método de la modelación:** para crear abstracciones con el objetivo de explicar la realidad, se utilizará para la modelación de los diagramas de la presente investigación.

Métodos empíricos:

- El método **observación científica** se emplea con el objetivo de observar el funcionamiento de la arquitectura XALIX 1.0, para obtener un registro visual de las características que deben formar parte de la solución.

CAPÍTULO 1: Fundamentos teóricos de la arquitectura XALIX 1.0

1.1 Introducción

La Arquitectura XALIX 1.0 constituye la base del diseño arquitectónico del software en varios de los proyectos desarrollados por el Centro FORTES. En la actualidad está compuesto por varios elementos y bundles de terceros que garantizan el correcto funcionamiento de la arquitectura, de los cuales algunos se verán afectados cuando la versión de Symfony utilizada (2.7) sea actualizada por la (3.4). En este capítulo se realiza un estudio de dicha arquitectura y se describen los componentes que la integran.

1.2 Términos relacionados con la investigación

En la tesis se trabajó con un conjunto de términos y conceptos relacionados con el objeto de la investigación que relacionamos a continuación:

Framework

Es una estructura de software formada por componentes personalizables e intercambiables para el desarrollo de una aplicación. Se puede considerar además como una aplicación genérica incompleta y configurable a la que podemos añadirle las últimas piezas para construir una aplicación concreta. Los objetivos principales que persigue un *framework* son acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el empleo de patrones. Un *framework* web, por tanto, se define como un conjunto de componentes (clases, descriptores, archivos de configuración, entre otros) que componen un diseño reutilizable que facilita y agiliza el desarrollo de sistemas web (GUTIERREZ, JAVIER.J, 2014).

Symfony

Es un *framework* que simplifica el desarrollo de una aplicación web mediante la automatización de algunos de los patrones utilizados para resolver las tareas comunes. Está basado en el clásico patrón de diseño web conocido como arquitectura MVC (Modelo-Vista-Controlador). Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. También, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación (POTENCIER, 2006).

Symfony está desarrollado completamente con PHP y es compatible con la mayoría de gestores de bases de datos (*MySQL, PostgreSQL, Oracle, SQL Server*, entre otros). Se puede ejecutar tanto en

plataformas **nix* (*Unix, Linux*, entre otras.) como en plataformas *Windows*. Además, desde su primera versión, publicada en octubre de 2005, fue liberado bajo la licencia de *software* libre por su creador Fabien Potencier (POTENCIER, 2006).

Debido a la calidad del código fuente, la amplia documentación, la gran comunidad de colaboradores entre otros aspectos, este *framework* es utilizado en grandes redes sociales como: *Delicious*[2], *Dailymotion*[3] o *Yahoo! Answers*[4], así como en medianos y pequeños sitios. Con Symfony se pueden encontrar incluso personalizaciones del *framework* y arquitecturas que facilitan el trabajo ágil con esta herramienta (VILLA LÓPEZ y RODRIGUEZ GUAMAN, 2015).

Arquitectura de software

El objetivo principal de la arquitectura de *software* es aportar elementos que ayuden a la toma de decisiones y, al mismo tiempo, proporcionar conceptos y un lenguaje común que permitan la comunicación entre los equipos que participen en un proyecto. Además, analizar e identificar riesgos potenciales en su estructura y sus propiedades, que puedan afectar al sistema de *software* resultante, verificar que los requerimientos no funcionales estén presentes en la arquitectura, así como habilitar dichos requerimientos, atributos de calidad y restricciones para asegurar que el sistema a ser construido cumple con las necesidades de los *stakeholders* (personas que están relacionadas, de cierta forma, con el sistema, ya sea un desarrollador, usuario o administrador, entre otros) (PUEBLA *et al.*, 2016).

En la actualidad no existe una definición única de arquitectura de *software*, por lo que es posible encontrar gran variedad de conceptos y definiciones por diferentes autores y ninguno de estos es respaldado unánimemente. Durante la investigación se analizan algunos de estos conceptos:

- Una definición reconocida es la de Paul Clements del Instituto de Ingeniería de *Software* de la Universidad de Carnegie Mellon: “La arquitectura de *software* de un programa o sistema de cómputo es la estructura o estructuras del sistema, que comprende los componentes de software, sus propiedades externamente visibles, y las relaciones entre ellos”.

[2] https://delicious.com/

[3] http://www.dailymotion.com/

[4] https://mx.answers.yahoo.com/

- Por otra parte, la definición más reconocida internacionalmente por muchos arquitectos tributa a la industria y pertenece a la IEEE[5], plantea lo siguiente: “La Arquitectura de *Software* es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución" (IEEE, 2000).
- La versión de la guía oficial de la corporación IBM sobre el Proceso Unificado de Racional del 2007 plantea que la arquitectura de *software* puede entenderse como aquella estructura del programa que cohesiona las funcionalidades más críticas y relevantes (necesarias para el sistema), y que sirve de soporte al resto de funcionalidades finales (necesarias para el usuario) (IBM, 2015).

Después del análisis realizado sobre las principales definiciones que existen sobre arquitectura de *software*, se utilizará para la presente investigación el concepto perteneciente a la IEEE por ser la que mejor se ajusta a esta investigación.

XALIX

Un ejemplo de arquitectura de *software* basada en el *framework* Symfony es XALIX. Esta arquitectura fue desarrollada por el Centro FORTES para agilizar el proceso de desarrollo de *software*. Está compuesta por un conjunto de tecnologías y bundles que satisfacen las necesidades de desarrollado web en esta entidad. Entre las tecnologías que integra XALIX están:

- Symfony 2.7.16 como *framework*.
- PHP 5.
- PostgreSQL 9 para la base de datos.
- *Bootstrap* 3.
- *jQuery* 1.10.3.
- HTML 5.
- Nginx como servidor web.

A pesar de la correcta estructuración de la arquitectura y su empleo en proyectos reales, no se cuenta con documentación y sus componentes distan de las versiones actuales lo que provoca limitaciones para su mantenibilidad.

[5] IEEE, del inglés Institute of Electrical and Electronics Engineers (Instituto de Ingenieros Eléctricos y Electrónicos)

1.3 Descripción de la arquitectura XALIX 1.0

Debido a la importancia que reviste para FORTES el empleo de la arquitectura XALIX, se hace necesario el análisis de los bundles que la componen. XALIX divide sus componentes en cuatro clasificaciones, los bundles del *framework*, los bundles de terceros, los desarrollados en FORTES y los desarrollados en la Universidad de las Ciencias Informáticas (UCI). En aras de una mejor compresión, a continuación, se describen cada uno de estos bundles empleando una estructura en la que se refleja el nombre del bundle, la versión integrada en XALIX 1.0 y una breve descripción de su función.

Bundles del framework.

"php": 5.3.3. Propio del *framework* de desarrollo.

"symfony/symfony": 2.7. Propio del *framework* de desarrollo.

"doctrine/orm": 2.2. Es un mapeador relacional de objetos para PHP 7.0 que proporciona persistencia transparente para objetos PHP. Se encuentra en la parte superior de la capa de abstracción de base de datos (DBAL). Una de sus características principales es la opción de escribir consultas en SQL llamado *Doctrine Query Language* (DQL). Esto proporciona a los desarrolladores una alternativa a SQL que mantiene la flexibilidad sin requerir la duplicidad de código innecesario (DUNGLAS, 2013).

La información asociada a los bundles que pertenecen al núcleo de Symfony se extraen de la documentación oficial del *framework* (SENSIOLABS, 2018) y a continuación se detallan:

"doctrine/doctrine-bundle": 1.2. Integra los proyectos ORM y DBAL de *Doctrine* en las aplicaciones de Symfony. Proporciona opciones de configuración, comandos de consola e incluso un recopilador de barra de herramientas de depuración web.

"twig/extensions": 1.0. *Twig* es el motor de plantillas predeterminado para Symfony. Este bundle agrega funciones adicionales, filtros, etiquetas y pruebas para integrar sin problemas los diversos componentes de Symfony con plantillas de *Twig*.

"symfony/assetic-bundle": 2.3. Proporciona la integración de la biblioteca de *Assetic* en el marco de Symfony. El uso de *Assetic* para administrar activos web en aplicaciones Symfony ya no es factible, en su lugar, se recomienda utilizar *Webpack Encore*, que conecta las aplicaciones de Symfony con las herramientas modernas basadas en *JavaScript* para administrar los activos de la web.

"symfony/swiftmailer-bundle": 2.3. El *swiftmailer* permite configurar la integración de Symfony con *Swift Mailer*, que es responsable de crear y entregar mensajes de correo electrónico.

"symfony/monolog-bundle": 2.4. El *MonologBundle* proporciona integración de la biblioteca de Monolog en el marco Symfony. A partir de la versión 2.4 del paquete, el ciclo de publicación se desincroniza desde el marco. Significa que solo puede requerir "*symfony / monolog-bundle*": "~ 2.4" en el *composer.json* y Composer selecciona automáticamente la última versión del paquete que funcione con la versión actual de Symfony.

"sensio/distribution-bundle": 3.0. *SensioDistributionBundle* proporciona funciones útiles que se pueden reutilizar entre varias distribuciones de Symfony. Symfony *Flex* es un reemplazo total para este paquete.

"sensio/framework-extra-bundle": 3.0. Este paquete proporciona una forma de configurar sus controladores con anotaciones.

"incenteev/composer-parameter-handler": 2.0. Este bundle mantiene la versión en Symfony 3. Esta herramienta permite administrar los parámetros ignorados cuando se ejecuta una instalación o actualización a través del comando composer. Realiza la comprobación de los requerimientos necesarios para la instalación de un paquete y sus dependencias.

"doctrine/data-fixtures": 1.3. Este paquete facilita la creación de dispositivos y es compatible con (*MySQL, PostgreSQL, SQLite*, entre otros). Tiene como objetivo proporcionar una forma sencilla de administrar y ejecutar la carga de dispositivos de datos para *Doctrine ORM o ODM*. Puede escribir clases de dispositivos implementando la interfaz *Doctrine \ Common \ DataFixtures \ FixtureInterface*.

"doctrine/migrations": 1.2. El paquete usa una biblioteca externa de *Doctrine Database Migrations*. La función de las migraciones de base de datos es una extensión de la capa de abstracción y le ofrece la capacidad de implementar mediante programación nuevas versiones de su esquema de base de datos de una manera segura, fácil y estandarizada. Las migraciones de *Doctrine* para Symfony se mantienen en *DoctrineMigrationsBundle*.

"doctrine/doctrine-fixtures-bundle": 2.2. Este paquete integra la biblioteca *Doctrine2 Data Fixtures* en Symfony para que pueda cargar accesorios de datos mediante programación en *Doctrine ORM u ODM.*

"sensio/generator-bundle": 2.3. Este paquete proporciona comandos para formularios, controladores e incluso *backends* basados en CRUD. El código repetitivo proporcionado por estos generadores de códigos ahorra gran cantidad de tiempo y trabajo. *SensioGeneratorBundle* amplía la interfaz de línea de comandos predeterminada de Symfony al proporcionar nuevos comandos interactivos e intuitivos para generar esqueletos de código como paquetes, clases de formulario o controladores CRUD basados en un esquema de Doctrine 2.

"symfony/phpunit-bridge": 3.0. Proporciona utilidades para PHPUnit, especialmente administración de avisos de desaprobación del usuario. Además, presenta potentes características para informes de pruebas heredadas y la utilización de código obsoleto.

Bundles de terceros.

"jms/di-extra-bundle": 1.5.0. *JMSDiExtraBundle* agrega funciones de inyección de dependencia a Symfony. Además, permite configurar la inyección de dependencia a través de anotaciones y la basada en convenciones en controladores. Así mismo, ofrece capacidades de programación orientadas a aspectos para controladores.

"friendsofsymfony/user-bundle": 1.3. Proporciona un marco flexible para la administración de usuarios que tiene como objetivo manejar tareas comunes tales como el registro de usuarios y la recuperación de contraseñas. Este paquete no proporciona un sistema de autenticación, pero puede proporcionarle el proveedor de usuario para el SecurityBundle. Algunas de sus características son:

- Los usuarios pueden ser almacenados a través de *Doctrine ORM o MongoDB / CouchDB ODM.*
- Soporte de registro, con una confirmación opcional por correo electrónico.
- Soporte de restablecimiento de contraseña.

"gedmo/doctrine-extensions": 2.3. Este bundle es tratado actualmente con el nombre *stof/doctrine-extensions-bundle*. El nombre cambia debido a la integración de este bundle con *gedmo/doctrine-extensions*. La versión estable en la que se encuentra es la 1.3. Este paquete proporciona integración para *DoctrineExtensions* en su proyecto Symfony.

"hwi/oauth-bundle": 0.4. Agrega soporte para autenticar usuarios a través de OAuth1.0a u OAuth2 en Symfony. Este paquete contiene soporte para 58 proveedores diferentes, entre los que se encuentran, *Facebook, GitHub, Google, LinkedIn, PayPal, Yahoo!, Twitter, Windows Live, Wordpress, Yandex, YouTube*.

"javiereguiluz/easyadmin-bundle": 1.5.5. Permite crear backends de administración. Entre sus principales características de encuentran las operaciones de CRUD (*Create, Read, Update, Delete*, por sus siglas en inglés) sobre entidades de *Doctrine*. Además, permite la búsqueda de texto completo, paginación y clasificación de columnas. Este paquete se abandona y ya no se mantiene. El autor sugiere utilizar el paquete *easycorp/easyadmin-bundle* en su lugar.

"knplabs/knp-menu-bundle": 2.0. Permite integrar la biblioteca PHP de *KnpMenu* con Symfony, lo que permite crear menús fáciles de implementar y ricos en funciones.

"fp/jsformvalidator-bundle": 1.2. Este módulo permite la validación de *JavaScript* para formularios en Symfony. Convierte las restricciones de tipo de formulario en reglas de validación de *JavaScript*.

"craue/formflow-bundle": 2.1. Proporciona una instalación para construir y manejar formularios de varios pasos en su proyecto Symfony. Entre sus características encontramos navegación (siguiente, atrás, volver a empezar), paso de etiquetas omisión de pasos, diferente grupo de validación para cada paso, manejo de cargas de archivos, navegación dinámica por pasos (opcional), redirigir después de enviar ("Publicar / redirigir / obtener", opcional).

"willdurand/js-translation-bundle": 2.5.0. Se encarga de exponer de manera agradable los mensajes de traducción de Symfony a las aplicaciones clientes.

"kevintweber/ktw-database-menu-bundle": 0.4.2. Este paquete es una extensión simple de la biblioteca KnpMenu, donde se pueden almacenar los elementos de menú en una base de datos.

"gos/web-socket-bundle": 1.6. Es un paquete de Symfony diseñado para reunir la funcionalidad de WebSocket en una arquitectura de aplicación fácil de usar. Al igual que Socket.IO, proporciona tanto el lado del servidor como el código del lado del cliente.

"lexik/maintenance-bundle": 1.0. Este paquete permite colocar las aplicaciones en modo de mantenimiento llamando a dos comandos en la consola. A los usuarios se le muestra una página con el código de estado 503.

"gregwar/captcha-bundle": 1.1. El complemento es compatible con un tipo de formulario captcha para el componente de formulario Symfony.

"vich/uploader-bundle": 0.14. Es un paquete de Symfony que facilita cargar archivos que se adjuntan a entidades ORM, documentos MongoDB ODM, documentos PHPCR ODM o modelos Propel. Sus funciones son configurables para permitir la personalización específica de la aplicación.

"stfalcon/tinymce-bundle": 0.4. Este paquete integra el editor TinyMCE[6] WYSIWYG en un proyecto Symfony.

"friendsofsymfony/comment-bundle": 2.0. Este paquete agrega soporte para un sistema de comentarios para aplicaciones Symfony. Algunas de las características que incluye son:

[6] TinyMCE es un editor de texto WYSIWYG para HTML de código abierto que funciona completamente en JavaScript y se distribuye gratuitamente bajo licencia LGP

- Administra árboles de comentarios.
- Puede incluir hilos de comentarios en cualquier página.
- Compatible con servidores de persistencia.
- Clasificación configurable del árbol de comentarios REST API.
- Extensible a través de eventos disparados durante el ciclo de vida de comentarios.
- Uso opcional de Symfony ACL para proteger los comentarios.
- Integración opcional con *FOS\UserBundle*.

"phpdocumentor/reflection-docblock": 2.0. El componente *ReflectionDocBlock* de *phpDocumentor* proporciona un analizador *DocBlock* que es 100% compatible con el estándar *PHPDoc*. Con este componente, una biblioteca puede proporcionar soporte para anotaciones a través de *DocBlocks* o recuperar información que esté incorporada en un *DocBlock.*

"mockery/mockery": 0.9. Mockery es un *framework* de objetos simulados de PHP simple pero flexible para usar en pruebas unitarias con PHPUnit, PHPSpec o cualquier otro marco de prueba. Su objetivo principal es ofrecer un marco doble de prueba con una API capaz de definir las operaciones e interacciones de objetos utilizando un Lenguaje Específico de Dominio (DSL) legible por humanos. Diseñado como una alternativa a la biblioteca *phpunit-mock-objects* de PHPUnit, Mockery es fácil de integrar con PHPUnit y puede operar junto con *phpunit-mock-objects*.

"mikey179/vfsstream": 1.5. Es un contenedor de flujo para un sistema de archivos virtual que puede ser útil en pruebas unitarias. Se puede utilizar con cualquier marco de prueba de unidad, como PHPUnit o *SimpleTest*.

"fzaninotto/faker": 1.5. Es una biblioteca PHP que genera datos falsos para usted. Ya sea que necesite iniciar su base de datos, crear documentos XML atractivos, completar su persistencia o anonimizar los datos tomados de un servicio de producción.

"raulfraile/ladybug-bundle": 1.0. Paquete Symfony2 para la biblioteca Ladybug (Descargador PHP simple y extensible). Este paquete proporciona un reemplazo *var_dump / print_*r fácil y extensible para proyectos Symfony2, tanto en controladores como en plantillas *Twig*.

"aerialship/saml-sp-bundle": 1.0.0. Este bundle agrega soporte para SAML 2.0 *Service Provider* en Symfony2. Proporciona un detector de seguridad que se puede configurar para autenticar a los usuarios contra uno o más proveedores de identidad de SAML. Entre las funciones incluidas se encuentran:

- Metadatos de federación XML.

- Servicio de descubrimiento.
- *AuthnRequest* / Inicio de sesión único.
- *LogoutRequest / Single Logout*.
- HTTP *Post* y HTTP *Redirect Bindings*.

Bundles desarrollados en FORTES

BackEndBundle: 1.0. Un paquete de Sulu para facilitar la creación de *backend* personalizado. Inspirado por *Doctrine Generator* CRUD.

FrontEndBundle: 1.0. Es el paquete para la interfaz de las aplicaciones.

NotificationBundle: 1.0. Es el paquete encargado de la gestión de las notificaciones.

ReportsBundle: 1.0. Es el paquete encargado de la generación de reportes.

TraceBundle: 1.0. Es el rastreador de paquetes.

UserBundle: 1.0. Es el paquete de usuarios.

Bundles desarrollados en la UCI

Boson: 1.0. Paquete encargado del control de la seguridad.

1.4 Caracterización de las versiones de Symfony

Debido a que el núcleo de la arquitectura XALIX es Symfony, es necesario analizar a profundidad este *framework*, así como los cambios que incluye en sus actualizaciones. A continuación, se realiza una breve reseña de algunas de estas versiones.

Versión 2.7

La versión de Symfony 2.7 representa la segunda versión de largo soporte (LTS o *"long term support*" en inglés) de la rama 2.x. Esta versión incluyó más de 100 modificaciones con respecto a su predecesora, la mayoría pequeñas, pero muy importantes. Entre las principales modificaciones se

encuentra la facilidad de añadir mensajes de aviso cuando la aplicación utiliza funcionalidades declaradas obsoletas. Estos avisos se muestran tanto en el navegador como en los archivos de log (EGUILUZ, 2015a).

Debido a la forma de trabajar en el proyecto Symfony, todas las novedades de la versión 3.0 deben introducirse primero en las versiones 2.x. Pero al Symfony 2.7 encontrarse en el período de *feature-freeze* donde ya no es posible introducir nuevas funcionalidades, trae consigo que 3.0 ya no pueda recibir ninguna novedad de sus antecesores. Esto significa que el paso de la versión 2.x a 3.x resulta complejo. Para solucionarlo, se lanza la versión 2.8 que permite añadir nuevas funcionalidades que se especifican con mayor detalle a continuación (EGUILUZ, 2015b).

Versión 2.8

Entre las mejoras que presenta Symfony 2.8 se encuentra la compatibilidad con PHP 7 así como el rediseño de todas las páginas, que consiste en la modernización de la apariencia del *profiler*, sus contenidos y funcionalidades. Además, la barra de depuración web (WDT) se modifica y trae como una de las principales ventajas que ocupa menos espacio, permite así ver todos los paneles que incluyen los bundles de terceros que utiliza la aplicación (EGUILUZ, 2016).

Versiones 3.0 y 3.1

Symfony presenta nuevas características en sus versiones 3.0 y 3.1. A continuación se listan algunas de las áreas que fueron actualizadas (POTENCIER, 2016a):

- Symfony introdujo el Protocolo Ligero de Acceso a Directorios (LDAP) como un componente en la versión 3.0, que proporciona la integración con el componente de seguridad Symfony. En la versión 3.1, LDAP obtiene algunas actualizaciones importantes como la opción para configurar todas las opciones de LDAP disponibles para la conexión, comprobar si existe un atributo de entrada, nombre de servicio predeterminado agregado para los componentes de seguridad, entre otras.

- Mejoras en el YAML, utilizado en Symfony para convertir cadenas YAML a matrices PHP y viceversa. YAML se utiliza principalmente para los archivos de configuración, planos y en la configuración de páginas. YAML ha recibido algunas actualizaciones importantes como la introducción de banderas para personalizar el comportamiento de análisis, se agrega una opción para volcar objetos como mapas y objetos *DateTime* y el soporte para analizar etiquetas binarias.

- Mejoras en el componente de proceso de Symfony, el cual ejecuta comandos en subprocesos. Se ocupa de las diferencias sutiles entre las diferentes plataformas cuando se ejecuta un comando. *Process* proporciona una abstracción orientada a objetos sobre las funciones *proc_ * para ejecutar procesos independientes en PHP.

- Mejoras en el serializador que convierte objetos en formatos específicos como JSON, XML, YAML, entre otros. El serializador permite que los codificadores solo se ocupen de la codificación de formatos específicos en matrices y viceversa. El servicio de serializador no está disponible de manera predeterminada y se debe activar desde el archivo de configuración. Entre las modificaciones que recibe están la detección de reenvío y redireccionamiento adicional, la adición de más información en el perfil de seguridad y se agrega un código de retorno HTTP en la tabla de la lista de solicitudes de *AJAX*.

- Mejoras en la consola que facilita la creación de interfaces de línea de comandos. Antes de Symfony 3.1, los comandos no se podían registrar de forma privada. Sin embargo, en esta versión, el componente recibe varias actualizaciones como que los comandos privados pueden registrarse de forma privada, se agrega compatibilidad con *ApplicationTester* para probar *stdout* y *stderr, s*e añade la funcionalidad de ancho de columna no automática y se agrega el método truncado a *FormatterHelper.*

- El componente formulario se enfoca en procesar datos hacia y desde el cliente y la aplicación. Este componente tiene algunas actualizaciones que incluyen optimización a los componentes *LazyChoiceList* y *DoctrineChoiceLoader* y la anulación de la opción "*choices_as_values*" de *ChoiceType*.

- El componente de seguridad proporciona un firewall dentro y fuera de la Symfony para protegerlo de amenazas de seguridad. Una opción importante de este componente es la creación de políticas personalizadas y reglas de seguridad e incorpora cuatro subcomponentes; *security-core*, *security-HTTP, security-csrf y security-acl*. Algunas de las actualizaciones para el componente de seguridad son la utilización del *auth trust resolver* para determinar anónimos en *ContextListener* y se agrega un nuevo *TargetPathTrait* para obtener o establecer la autenticación *"target_path"*.

Versión 3.2 y 3.3

Esta versión mantiene la compatibilidad con las versiones anteriores lo que permite actualizar de manera fácil sin cambiar nada en el código. Entre los componentes actualizados se encuentran

YAML, Procesor, Serializer, Web Profiler, Console, HTTPKernell, Form, Twig y Security. Todos estos componentes garantizan un mayor rendimiento (POTENCIER, 2016b). Al igual que su predecesora, la versión 3.3 sigue la misma línea en cuanto al mantenimiento en la compatibilidad con versiones anteriores. Los cambios más significativos son que el componente de Inyección de Dependencia permite importar directorios enteros y archivos de configuración desde otros archivos de configuración de diferentes formatos, permite activar automáticamente los log para los comandos de la consola y permite registrar las excepciones y emplear *subscriber* para la escucha del evento *console.terminate*, lanzando un log para cualquier comando cuyo código de salida no sea el correcto (EGUILUZ, 2017a).

Versión 3.4

Entre las principales mejoras que se incorporan en la versión 3.4 de Symfony se destacan el mantenimiento y soporte a largo plazo hasta el año 2021, los indicadores sobre cobertura de código, las variables de entorno avanzadas, las mejoras de los validadores de comparación, las respuestas HTTP inmutables, el soporte para grupos de validaciones, excepciones en la consola mejoradas y mejoras en los *security.listeners* (EGUILUZ, 2017b).

1.5 Tecnologías que componen XALIX 1.0

La arquitectura XALIX, además del núcleo basado en el framework Symfony, necesita de una serie de tecnologías para su correcto funcionamiento y despliegue en producción, como se explicará a continuación.

Servidor web NGINX

Un servidor web es un programa diseñado para permitir la interacción entre ordenadores y funciona permaneciendo a la espera de peticiones utilizando el protocolo HTTP[7]. NGINX es un servidor HTTP libre, de código abierto, de alto rendimiento y proxy inverso, así como un servidor proxy IMAP[8] / POP3[9]. Es conocido por su alto rendimiento, estabilidad, conjunto de características ricas, configuración sencilla y bajo consumo de recursos (INC, 2017).

A diferencia de los servidores tradicionales, NGINX no se basa en los hilos para manejar las solicitudes, en su lugar, utiliza una arquitectura asincrónica basada en eventos, mucho más

[7] Protocolo de Transferencia de Hipertexto

[8] Protocolo de accesos a mensajes de Internet

[9] Protocolo de oficina de correo o protocolo de oficina postal

escalable. Además, escala en todas las direcciones, desde el más pequeño VPS[10] hasta grandes grupos de servidores. Utiliza módulos externos que se pueden agregar al servidor, lo cual lo hace mucho más liviano y ágil que otros servidores web. Además, es multiplataforma, fácil de instalar y es compatible con la mayoría de los frameworks y sistemas de gestión de contenidos (CMS, del inglés, *Content Management System*) existentes (INC, 2017).

Sistema Gestor de Bases de Datos PostgreSQL

Un Sistema Gestor de Bases de Datos (SGBD) es una colección de programas cuyo objetivo es servir de interfaz entre la base de datos, el usuario y las aplicaciones. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta. Un SGBD permite definir los datos a distintos niveles de abstracción y manipularlos, garantizando su seguridad e integridad (ALVAREZ, 2007).

PostgreSQL es un SGBD objeto-relacional liberado bajo licencia BSD. Además, es extensible, multiplataforma y presenta modelos de negocios rentables con instalaciones a gran escala. Tiene ahorros considerables en costos de operación. Ha sido diseñado para tener un mantenimiento y ajuste mucho menor que los productos de los proveedores comerciales, conservando todas las características de rendimiento. Se destaca por su gran estabilidad, potencia, robustez, ejecución en diferentes plataformas y facilidad de administración. Además, se desempeña correctamente con grandes cantidades de datos y una alta concurrencia de usuarios (POSTGRESQL, 2012).

PHP 5

Acrónimo de *Hypertext Preprocessor*, PHP es un lenguaje de código abierto interpretado, de alto nivel, embebido en páginas HTML y ejecutado en el servidor. Puede ser utilizado en cualquiera de los principales sistemas operativos del mercado actual, incluyendo *Linux, UNIX, Microsoft Windows, Mac OS X, RISC OS*. Soporta la mayoría de servidores *web*, incluyendo *Apache, Microsoft Internet Information Server, Personal Web Server, Netscape e iPlanet*. Tiene la posibilidad de soporte para una gran cantidad de bases de datos, incluye *InterBase, FrontBase, mSQL, Direct MS-SQL, MySQL, Oracle (OCI7 and OCI8), Ovrimos, PostgreSQL* (ACHOUR *et al.*, 2014).

Las cuatro características más importantes de PHP son simplicidad, seguridad, estabilidad y velocidad. En cuanto a la velocidad, se integra muy bien a otro *software* y requiere pocos recursos de sistema. Garantiza la estabilidad pues utiliza su propio sistema de administración de recursos y posee

[10] Servidor virtual privado (VPS, por sus siglas en inglés) es un método para dividir un servidor físico en varios servidores de tal forma que todo funcione como si se estuviese ejecutando en una única máquina.

un sofisticado método para manejar variables, lo que lo hace un sistema robusto. Posee protección contra ataques, proveyendo diferentes niveles de seguridad. La simplicidad es la posibilidad brindada al desarrollador para generar código rápidamente, para esto, el lenguaje incluye gran cantidad de funciones predefinidas (MARIÑO, 2008).

HTML 5

"HTML siglas de *Hypertext Markup Language* (lenguaje de marcado de hipertexto), es un lenguaje diseñado para crear páginas *web* y otros documentos que sean posibles visualizar en el navegador web, siendo esencial para la creación de páginas y mostrar el contenido en ellas. El HTML se encarga de desarrollar una descripción sobre los contenidos que aparecen como textos y sobre su estructura, complementando dicho texto con diversos objetos (como imágenes, animaciones, videos)" (LAWSON y SHARP, 2011).

JavaScript

JavaScript es un lenguaje de *script* multiplataforma, diseñado para una fácil incrustación en otros productos y aplicaciones, tales como los navegadores *web*. JavaScript puede ser conectado a los objetos de su entorno para proveer un control programable sobre estos (ZELDMAN, 2010). El empleo de JavaScript para complementar el diseño de interfaz de usuario ofrece efectos de animación que mejoran la visualización de los elementos que componen la arquitectura. Como medio de optimización para JavaScript se puede utilizar JQuery, que realiza funciones de *script* frecuentes y utiliza menos líneas de código.

CSS3

Cascading Style Sheets u Hojas de Estilo en Cascada: es un lenguaje que permite a los desarrolladores describir la presentación de las páginas web. Son incluidos los colores, el diseño, y las fuentes, permitiendo adaptar la presentación a los diferentes tipos de dispositivos, tales como diferentes resoluciones de pantallas o impresoras (PÉREZ, 2007). La separación del código HTML de CSS hace que sea fácil mantener sitios, compartir hojas de estilos a través de páginas y adaptarlas a entornos distintos ((W3C), 2012). Para optimizar CSS3, se utiliza *Bootstrap*, que permite que la arquitectura se adapte a las distintas resoluciones de pantalla, aumente el atractivo visual, incorpore las nuevas tendencias del diseño web y garantice una fácil actualización del diseño visual.

1.6 Vistas arquitectónicas

Desde los inicios de la arquitectura de software se planteó la necesidad de contar con varias vistas para separar a los diferentes elementos del *software*. El arquitecto del *software* necesita un número

de vistas diferentes de la arquitectura de *software* para varios usos y usuarios, las que son requeridas para enfatizar y entender diferentes aspectos de la arquitectura (KROLL y KRUCHTEN, 2003). Existen diferentes definiciones para el término vista:

- "Un subconjunto resultante de practicar una selección o abstracción sobre una realidad, desde un punto de vista determinado" (KROLL y KRUCHTEN, 2003).
- De acuerdo a la definición del estándar 1471 de la IEEE: "Una vista define un patrón o plantilla para representar un conjunto de incumbencias (concerns) relativo a una arquitectura" (HILLIARD, 2000).

En conclusión, una vista no es más que una temática desde la que se enfoca el proyecto, ya sea desde el punto de vista de los datos, integración, entre otros y que hace posible enfrentar el diseño de la arquitectura desde diversas perspectivas reduciendo la complejidad presente en su desarrollo.

Las vistas arquitectónicas constituyen las estructuras fundamentales de la arquitectura. El diseño de las vistas arquitectónicas implica establecer una sincronización entre sus elementos para conformar una arquitectura integrada, y de poder ser capaz de mantener la consistencia entre sus elementos. Cada vista agrupa a elementos que pertenecen a diferentes intereses. Además, se llegan a establecer relaciones de correspondencia entre los elementos de distintas vistas, puesto que, al formar parte de un mismo sistema, de alguna manera se llegan a relacionar. Estas relaciones de correspondencia constituyen el adhesivo que mantiene el enlace a las vistas que integran a las AS (REYNOSO y KICILLOF, 2004).

Las arquitecturas de *software* se enfrentan a la abstracción, descomposición y composición usando estilos y patrones. Para describir estas arquitecturas se utiliza un modelo compuesto de múltiples vistas o perspectivas. Con el fin de resumir estos modelos, Philippe Kruchten propone cinco vistas (KRUCHTEN, 1995):

- **La vista lógica:** es el objeto del diseño (cuando se usa el modelo orientado a objetos).
- **La vista de procesos:** captura la concurrencia y sincronización como aspectos del diseño.
- **La vista física:** describe el mapeo del *software* en el *hardware* y refleja su aspecto distribuido.
- **La vista de desarrollo:** describe la organización estática del *software* en su entorno de desarrollo.
- **Escenarios:** donde se ensamblan los elementos mostrados en las vistas anteriores. La descripción de una arquitectura (las decisiones tomadas) puede ser organizada alrededor de

las vistas lógica, de procesos, física y de desarrollo, mostrándose en los escenarios. La arquitectura, de hecho, evoluciona a partir de estos escenarios.

Por otra parte, (SONI *et al.*, 1995) realizaron un estudio para conocer en una arquitectura, las estructuras que son de mayor importancia y el empleo de éstas. El estudio se efectuó sobre varios sistemas de *software* de ámbito industrial tales como, sistemas de procesamiento de señales e imágenes, sistemas operativos en tiempo real, sistemas de comunicaciones y sistemas de control e instrumentación. Tras el estudio realizado, propusieron cuatro categorías o "vistas" para agrupar las estructuras principales de una arquitectura. Dentro de cada una, se describen las estructuras principales del sistema desde una perspectiva en particular. A continuación, se explica brevemente cada una de éstas (GÓMEZ, 2007):

- **Vista conceptual:** Se describe el sistema en términos de sus elementos principales de diseño y las relaciones entre éstos, dentro de un dominio determinado. Esta vista es independiente de las decisiones de implementación y enfatiza en los protocolos de interacción entre los elementos de diseño.
- **Vista de módulos:** Se captura la descomposición funcional y las capas del sistema. El sistema es descompuesto lógicamente en subsistemas, módulos, y unidades abstractas. Cada capa representa las distintas interfaces de comunicación permitidas entre los módulos.
- **Vista de ejecución:** Se describe la estructura dinámica del sistema en términos de sus elementos en tiempo de ejecución. Por ejemplo, se modela las tareas operativas del sistema, procesos, mecanismos de comunicación y asignación de recursos. Algunos de los aspectos que se consideran en esta vista son, el desempeño y el entorno de ejecución.
- **Vista de código:** Se organiza el código fuente en directorios, archivos y bibliotecas. Algunos de los aspectos que se incluyen son: los lenguajes de programación a utilizar, herramientas de desarrollo, la administración de la configuración, la estructura y organización del proyecto.

Entre las principales características de la arquitectura propuesta se destacan que no se modela un negocio ni se enfoca en la realización de una aplicación web específica. A partir de lo anterior, no es factible utilizar la propuesta de las vistas 4 + 1 de Kruchten. Este autor describe las vistas a través de una especificación de requisitos y casos de uso, así como la generación de un cúmulo de diagramas los cuales aumentarían considerablemente la complejidad a la hora de describir la propuesta. Debido a lo anterior, el enfoque presentado por (SONI *et al.*, 1995) fue seleccionado para este trabajo.

1.7 Lenguaje Unificado de Modelado

El Lenguaje Unificado de Modelado (en lo adelante UML, por sus siglas en inglés, *Unified Modeling Language*) se utiliza para especificar, visualizar, construir y documentar artefactos de un sistema de *software*. Se emplea para diseñar, hojear, configurar, mantener y controlar la información sobre sistemas. Está pensado para emplearse con todos los métodos de desarrollo, etapas del ciclo de vida y dominios de aplicación. UML incluye conceptos semánticos, notación y principios generales. Tiene partes estáticas, dinámicas, de entorno y organizativas. Está pensado para ser utilizado en herramientas interactivas de modelado visual que tengan generadores de código así como generadores de informes (JACOBSON *et al.*, 1999).

Es importante resaltar que UML es un "lenguaje" para especificar y no para describir métodos o procesos. Se utiliza para definir un sistema de *software*, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo. Se puede aplicar en una gran variedad de formas para dar soporte a una metodología de desarrollo de *software*, pero no especifica en sí mismo qué metodología o proceso utilizar.

1.8 Herramienta de ingeniería de *software* asistida por computadoras

Las herramientas de ingeniería de *software* asistida por computadoras (en lo adelante CASE, por sus siglas del inglés, *Computer Aided Software Engineering*) brindan ayuda a los analistas, ingenieros de *software* y desarrolladores. Por ejemplo, permiten el modelado de los sistemas mediante diferentes diagramas, generación de código a partir de estos y viceversa. Las herramientas CASE de modelado con UML permiten aplicar la metodología de análisis y diseño orientados a objetos. Además permiten abstraerse del código fuente, en un nivel donde la arquitectura y el diseño se tornan más fáciles de entender (THAMES, 2011).

La herramienta recomendada a utilizar para el modelado de la aplicación es Visual Paradigm, debido fundamentalmente a que es una herramienta multiplataforma que ayuda a una rápida construcción de aplicaciones de calidad. Además, la UCI posee licencia para su utilización. A continuación, se brindan detalles de esta herramienta.

Visual Paradigm

Visual Paradigm para UML es una herramienta CASE profesional que soporta el ciclo de vida completo del desarrollo de *software*: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite dibujar todos los tipos de diagramas de clases, código inverso y generar documentación. También proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML. Brinda la posibilidad de generar código a partir de los diagramas para

lenguajes como Java y PHP, así como obtener diagramas a partir de código (LTD, 2011). Esta es precisamente una gran ventaja puesto que el sistema será desarrollado en PHP.

1.9 Conclusiones del capítulo

El estudio de los principales términos, la descripción de la arquitectura XALIX 1.0 y sus tecnologías asociadas, así como la caracterización del *framework* Symfony permitieron realizar la fundamentación teórica de la investigación, que constituye un sustento de importancia para complementar la escasa documentación existente de la arquitectura y aportan elementos que argumentan la necesidad del desarrollo de una nueva que garantice la mantenibilidad.

CAPÍTULO 2: Arquitectura de referencia XALIX 2.0

2.1 Introducción del capítulo

Para solventar el problema de la presente investigación se procede a desarrollar la arquitectura XALIX 2.0. En este capítulo, se especifican las restricciones arquitectónicas que rigen esta arquitectura, se describen los principales componentes que la integran y se describe la arquitectura siguiendo el enfoque de vistas arquitectónicas propuesto por (SONI *et al.*, 1995) . A modo de organización, el desarrollo fue estructurado en tres etapas que comprenden:

- **La migración del núcleo de Symfony:** se basa en el cambio del núcleo del *framework* Symfony a la versión 3.4, última versión LTS oficial publicada.
- **La revisión de los bundles existentes en XALIX 1.0:** se basa en la comprobación de la compatibilidad de estos bundles con la versión 3.4 de Symfony, así como el chequeo de las últimas versiones estables publicadas. Esta comprobación sirve para identificar los bundles que se mantienen, se actualizan, se eliminan o se sustituyen por otros para su integración a la nueva arquitectura.
- **La incorporación de nuevos bundles:** se basa en la adición de bundles a la nueva arquitectura y que no estaban integrados en la versión 1 de XALIX. El principal objetivo de estos bundles es incorporar mejoras en la gestión de la parte administrativa, los recursos, la documentación, entre otras.

2.2 Descripción de la Arquitectura XALIX 2.0

La arquitectura XALIX 2.0 es la integración y configuración de un conjunto de componentes que hacen posible el desarrollo ágil de los productos web del Centro de Tecnologías para la Formación. Esta versión mantendrá una estructura similar a su predecesora. A continuación, se detallan las características de XALIX 2.0.

Elementos de la arquitectura

XALIX 2.0 está compuesta por 4 elementos fundamentales que garantizan su correcto funcionamiento. Los componentes que la integran son (Ilustración 1):

1. El *framework* Symfony en su versión 3.4: Es el elemento fundamental de la arquitectura, ya que constituye la base en la que son integrados todos los componentes.

2. Los Bundles de Terceros: Son un conjunto de bibliotecas que son requeridas en XALIX para agregar mejoras funcionales.
3. Bundles de FORTES: Son los bundles desarrollados en el Centro FORTES que constituyen elementos identificativos y propios de la arquitectura.

Bundles de XALIX 2.0

A partir del análisis de los bundles de la Arquitectura XALIX 1.0 se decide establecer que las versiones a utilizar de estos por la arquitectura XALIX 2.0 serán sus versiones estables hasta el cierre de esta investigación[11]. A continuación, se muestra el listado de los bundles y sus versiones que complementan la nueva arquitectura:

Bundles del framework.

Nombre	Versión
"php"	7.0
"symfony/symfony"	3.4
"doctrine/orm":	2.5
"doctrine/doctrine-bundle"	1.6
"twig/extensions"	1.5
"symfony/assetic-bundle"	2.8
"symfony/swiftmailer-bundle"	3.1
"symfony/monolog-bundle"	3.1
"sensio/distribution-bundle"	5.0.19
"sensio/framework-extra-bundle"	5.0

[11] Cierre de la investigación: febrero de 2018

"incenteev/composer-parameter-handler"	2.0
"symfony/polyfill-apcu"	1.0
"doctrine/data-fixtures":	1.3
"doctrine/migrations"	1.7
"doctrine/doctrine-fixtures-bundle"	3.0
"sensio/generator-bundle"	3.1.7
"symfony/phpunit-bridge"	4.0.4

Bundles de terceros

Nombre	**Versión**
"jms/di-extra-bundle"	1.9
"friendsofsymfony/user-bundle"	2.1.2
"friendsofsymfony/jsrouting-bundle"	2.2
"stof/doctrine-extensions-bundle"	1.3
"hwi/oauth-bundle"	0.6
"knplabs/knp-menu-bundle"	2.2.0
"craue/formflow-bundle"	3.0
"willdurand/js-translation-bundle"	2.7
"kevintweber/ktw-database-menu-bundle"	0.4.2
"gos/web-socket-bundle"	1.8.11
"lexik/maintenance-bundle"	2.1.2

"gregwar/captcha-bundle"	2.0
"vich/uploader-bundle"	1.7.1
"stfalcon/tinymce-bundle"	2.0
"friendsofsymfony/comment-bundle"	2.0.14
"phpdocumentor/reflection-docblock"	3.0.1
"mockery/mockery"	1.0
"mikey179/vfsstream"	1.6.5
"fzaninotto/faker"	1.8
"raulfraile/ladybug-bundle"	1.0.6

Con el objetivo de garantizar una mayor funcionalidad en la nueva arquitectura se propone adaptar los widgets (componente del formulario) con la versión 3.4 de Symfony debido a que los formularios fueron modificados. Además, se mejora en la instalación de los bundles mediante 4 pasos:

1. Definir el bundle y su versión estable en el archivo Composer.json.
2. Ejecutar el comando ***composer. phar*** en la consola.
3. Declarar la librería instalada en el archivo AppKernel.
4. Realizar las configuraciones necesarias atendiendo a las necesidades de cada desarrollador.

Se incorporan en la propuesta nuevos bundles *(Tabla 1)* para propiciar mayores ventajas a los desarrolladores y ganar en robustez y calidad a la hora de crear las aplicaciones web en el centro FORTES. Los bundles garantizan la creación y el manejo de componentes de gran importancia para la arquitectura; así como la generación de información.

Tabla 1: Descripción de los nuevos bundles de XALIX

Nombre	**Versión**	**Descripción**

"sonata-project/easy-extends-bundle":	2.5	Es un prototipo para generar una estructura de paquete válida a partir de un bundle proveedor(vendor). La herramienta se inicia con la línea de comando simple: sonata: *easy-extends: generate*. La línea de comando genera todos los directorios requeridos para un paquete (controlador, configuración, doctrina, vistas, entre otros), los archivos de mapeo y entidad.
"sonata-project/admin-bundle”	3.31.1	Es un generador de administración Symfony. Es el encargado del manejo de la administración de la librería sonata.
"sonata-project/doctrine-orm-admin-bundle”:	3.4.2	Doctrine ORM Admin proporciona servicios para trabajar con Admin Bundle y Doctrine Project.
"nelmio/api-doc-bundle”	3.2	Este paquete permite generar documentación en el formato OpenAPI (Swagger) a partir de anotaciones y proporciona un entorno para navegar de forma interactiva la documentación de la API(aplicación).
"fp/jsrouting-bundle"	1.5	Este paquete permite exponer rutas de Symfony a JavaScript, por lo que puede generar URL relativas o absolutas en el navegador utilizando las mismas rutas que en el back-end.
"friendsofsymfony/rest-bundle"		Este paquete contiene varias características ligeramente acopladas. Proporciona varias herramientas para utilizar en la construcción de aplicaciones API REST como son: • La capa de vista. • Soporte de escucha. • Soporte ExceptionController.

		• Generación automática de ruta: solo controlador RESTful (para recursos simples). • Definición manual de rutas.

Fueron eliminados diferentes bundles debido a su obsolescencia tecnológica, la no utilización de los mismos en la nueva propuesta de arquitectura o su sustitución por otros que garantizan mayores facilidades y funcionalidades. En la *Tabla 2* se detallan estas modificaciones.

Tabla 2: Bundles eliminados o sustituidos de XALIX

Nombre	**Descripción**
"aerialship/saml-sp-bundle"	No es incluido en la nueva arquitectura debido a que es requerido por el bundle Boson el que es eliminado por no contar con soporte en la actualidad
"aerialship/lightsaml"	No es incluido en la nueva arquitectura por la misma causa que el anterior.
"javiereguiluz/easyadmin-bundle"	Fue sustituido por sonata-project/admin-bundle. Debido a que easyadmin presenta menos funcionalidades que el sonata-admin y entra en obsolescencia tecnológica.
TraceBundle	No es incluido en la nueva arquitectura debido a la dependencia que presenta con el bundle Boson.
Boson	Es eliminado por no contar con soporte, ni mantenimiento en la actualidad.

2.3 Restricciones arquitectónicas

La calidad de un producto de *software* se puede interpretar como el grado en que dicho producto

satisface los requisitos de sus usuarios, aportando de esta manera un valor. Son precisamente estos requisitos los que se encuentran representados en el modelo de calidad, el cual categoriza la calidad del producto en características y subcaracterísticas (ISO/IEC, 2017). La definición de una arquitectura de software debe poseer un conjunto de reglas para el desarrollo de un *software*, en este contexto, las reglas se denominan restricciones arquitectónicas. A continuación, se relacionan las restricciones arquitectónicas que debe cumplir la propuesta de solución:

- La arquitectura debe utilizar el *framework* de PHP Symfony en su última versión LTS.
- Los bundles de terceros que se integren a la arquitectura deben estar en su última versión estable compatible con la versión de Symfony seleccionada.
- La arquitectura debe poseer un diseño adaptativo a fin de garantizar la adecuada visualización en diferentes dispositivos.
- La arquitectura debe garantizar una gestión básica de usuarios y del control de acceso.
- La arquitectura debe ser flexible para incorporar traducción en múltiples idiomas.
- La arquitectura debe ser multiplataforma para permitir su visualización en los diferentes sistemas operativos.
- La arquitectura debe permitir la actualización o incorporación de compontes de manera ágil.
- La arquitectura debe permitir la gestión básica de recursos y medias (subida de archivos).
- La arquitectura debe permitir la gestión de nomencladores.

2.4 Vistas arquitectónicas

Para describir la arquitectura de *software* utilizando el modelo propuesto por (SONI *et al.*, 1995) mencionado en el Capítulo 1 se proponen las siguientes vistas:

2.3.1 Vista conceptual

En esta vista se describe el sistema en términos de sus elementos principales de diseño y las relaciones entre éstos, dentro de un dominio determinado. Esta vista es independiente de las decisiones de implementación y enfatiza en los protocolos de interacción entre los elementos de diseño (SONI *et al.*, 1995). A continuación, se muestra el Modelo Conceptual de la arquitectura XALIX 2.0.

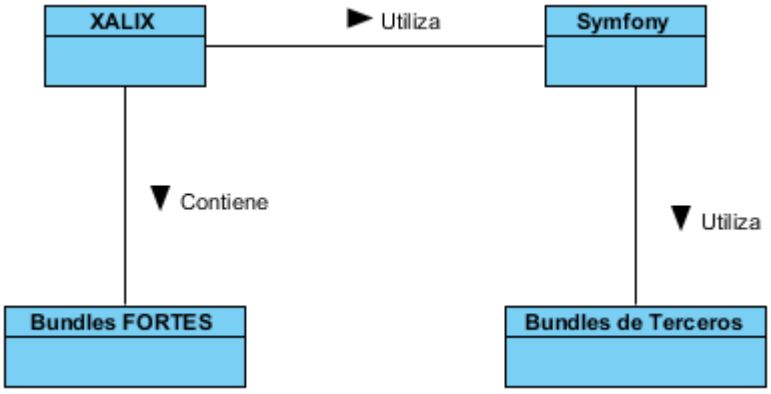


Ilustración 1:Modelo Conceptual

XALIX: Representa la arquitectura que contiene a todos los componentes del sistema.

Symfony: *Framework* de desarrollo que utiliza la arquitectura.

Bundles de Terceros: Grupo de librerías empleadas por la arquitectura.

Bundles FORTES: Representa los componentes desarrollados por el Centro FORTES.

2.3.2 Vista de módulos

En esta vista se captura la descomposición funcional y las capas del sistema. El sistema es descompuesto lógicamente en subsistemas, módulos, y unidades abstractas. Cada capa representa las distintas interfaces de comunicación permitidas entre los módulos (GÓMEZ, 2008).

Diagrama de paquetes

Un paquete es un mecanismo utilizado para agrupar elementos de UML. Contiene elementos del modelo al más alto nivel, tales como clases y sus relaciones, máquinas de estado, diagramas de casos de uso, interacciones y colaboraciones: cualquier elemento que no esté contenido en otro. Los paquetes pueden contener otros paquetes. Las dependencias entre paquetes resumen dependencias entre los elementos internos a ellos, es decir, las dependencias del paquete se derivan a partir de las dependencias entre los elementos individuales (GUTIERREZ, DEMIAN, 2009).

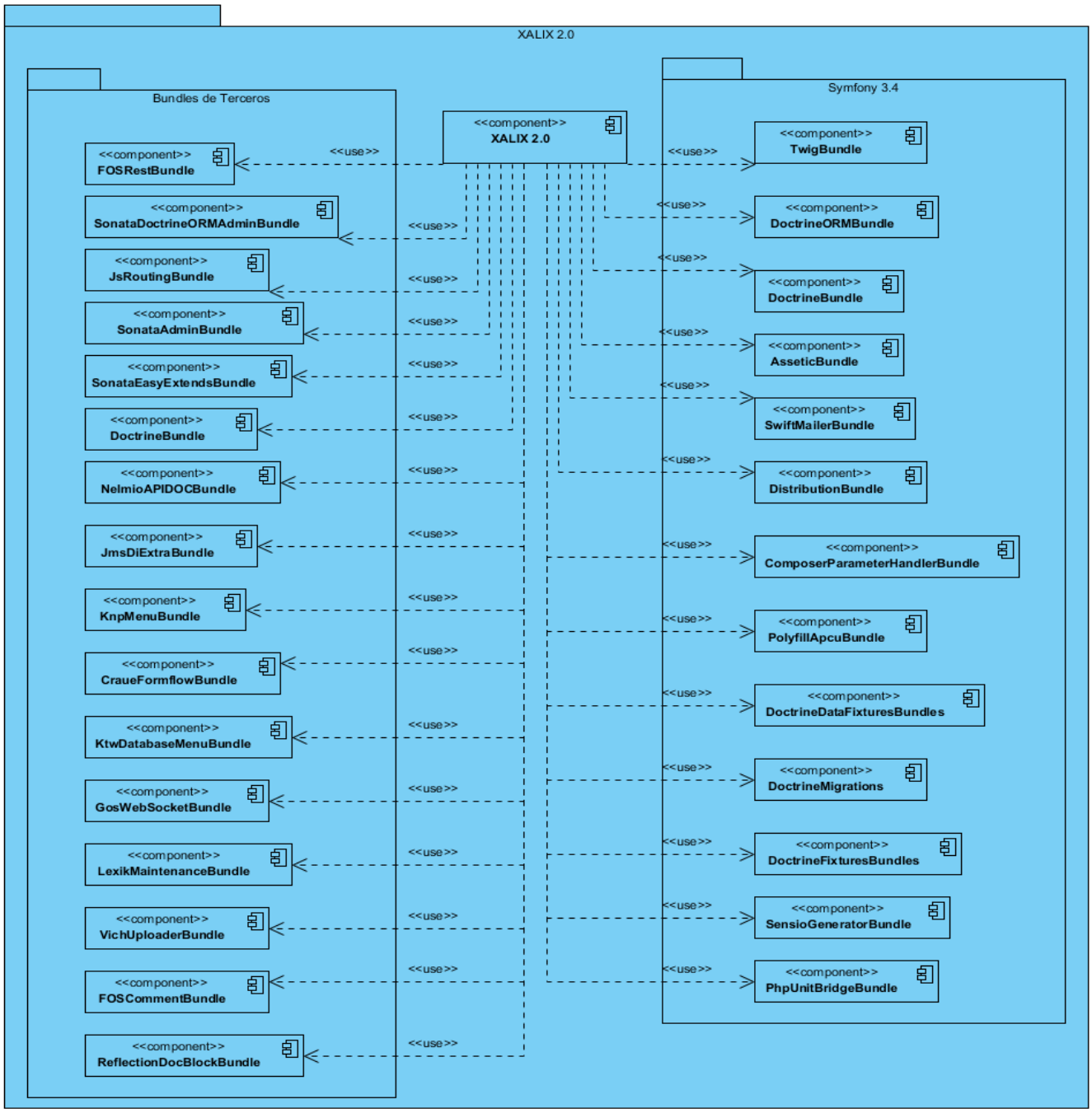


Ilustración 2:Diagrama de Paquetes

2.3.3 Vista de código

La vista de código es una apreciación del sistema en tiempo de diseño (del código), donde se muestra la implementación del sistema organizado dentro del código y componentes de desarrollo (CORDERO y SALAVERT, 2010). A continuación, se detalla esta vista aplicada sobre la arquitectura XALIX 2.0 y para ello se describe el patrón arquitectónico y los patrones diseños utilizados.

Patrón arquitectónico de XALIX 2.0

XALIX, al tener en su núcleo a Symfony, se nutre de la estructura definida por el *framework* para su funcionamiento, utilizando por tanto sus patrones arquitectónicos. Symfony está basado en un patrón clásico del diseño web conocido como arquitectura Modelo-Vista-Controlador (MVC), que está formado por tres niveles (POTENCIER y ZANINOTTO, 2008):

- **Modelo:** es la representación de la información con la cual el sistema opera. Gestiona todos los accesos a dicha información, tanto consultas, como actualizaciones. Además, implementa los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio).
- **Vista:** presenta el modelo (información y lógica de negocio) en un formato adecuado para que un usuario pueda interactuar (interfaz de usuario).
- **Controlador:** es el intermediario entre la vista y el modelo, su función consiste en controlar el flujo de datos, responder a eventos (generalmente provocados por los usuarios) e invocar peticiones al modelo.

Patrones de diseño

Los patrones de diseño tratan los problemas que se repiten y que se presentan en situaciones particulares del diseño, con el fin de proponer soluciones a ellas. Por lo tanto, los patrones de diseño son soluciones exitosas a problemas comunes en el desarrollo de *software* y otros ámbitos referentes al diseño de interacción o interfaces. Los patrones dan nombre y forma a heurísticas abstractas, reglas y buenas prácticas de técnicas orientadas a objetos (LARMAN, 1999).

Los patrones de *software* para la asignación general de responsabilidades (GRASP, por las siglas del inglés *General Responsibility Assignment Software Patterns*) describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en formas de patrones. La arquitectura propuesta se basa en el empleo de los patrones de diseño definidos por el *framework* Symfony, estos son:

- **Experto:** Es uno de los patrones que más se utiliza cuando se trabaja con Symfony, con la inclusión de la librería Doctrine para mapear la Base de Datos. Symfony 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 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 UserAdmin por ejemplo se encuentran las acciones definidas para el sistema en cuanto al manejo de los usuarios y se ejecutan cada una de ellas. En dichas acciones se crean los objetos de las clases que representan las entidades, lo que evidencia que la clase UserAdmin es “creador” de dichas entidades. Ejemplos de algunas funciones utilizadas en la clase son: configureDatagridMapper(), listMapper() y formMapper().

- **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 las clases Admin, 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* (propiedades).

- **Bajo Acoplamiento:** La clase UserAdmin hereda únicamente de AbstractAdmin 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.

A continuación, se detallan algunos de los patrones de la "pandilla de los cuatro" (*Gang - of-Four*) que se utilizaron en el diseño:

- **Singleton:** En el archivo *services.yml* se encuentran declarados todos los servicios que se utilizan en la arquitectura.
- **Builder:** Son empleados por los formularios en su construcción y la de sus elementos a través del método *Builder*.
- **Decorator:** Las plantillas *twig* lo utilizan en el proceso de herencia cuando son enriquecidas con los elementos necesarios para su creación.

2.3.4 Vista de ejecución

En la vista de ejecución se describe la estructura dinámica del sistema en términos de sus elementos en tiempo de ejecución. Por ejemplo, se modela las tareas operativas del sistema, procesos, mecanismos de comunicación y asignación de recursos. Algunos de los aspectos que se consideran en esta vista son, el desempeño y el entorno de ejecución (GÓMEZ, 2008).

El diagrama de despliegue muestra la configuración de los nodos (procesadores y dispositivos) que participan en la ejecución y de los componentes que residen en estos, mostrando la colaboración entre nodos mediante protocolos de comunicación (MARCA y QUISBERT). En el caso de las aplicaciones que se desarrollen empleando la arquitectura XALIX 2.0, la base para su despliegue se aprecia en la ***Ilustración 3***. A continuación, se detallan los elementos que componen este diagrama:

- **PC:** Es la computadora desde donde los usuarios podrán acceder a las futuras aplicaciones y se conecta a través del protocolo HTTPS con el servidor web.
- **Servidor de Aplicación:** Es el nodo intermediario entre las PC clientes que realizan sus peticiones y el nodo donde se encuentra el servidor de BD con las informaciones. Este servidor web toma los datos, realiza sus funciones y presenta la información a las PC. El servidor web se comunica con el nodo donde se encuentra el servidor de BD a través del protocolo TCP/IP.
- **Servidor de Base de Datos (XALIX):** Es el nodo donde se encuentra el servidor de base de datos.
- **Servidor de Media:** Es el nodo encargado de gestionar los recursos medias y establece la comunicación con el Servidor de Aplicación, a través del protocolo NFS.

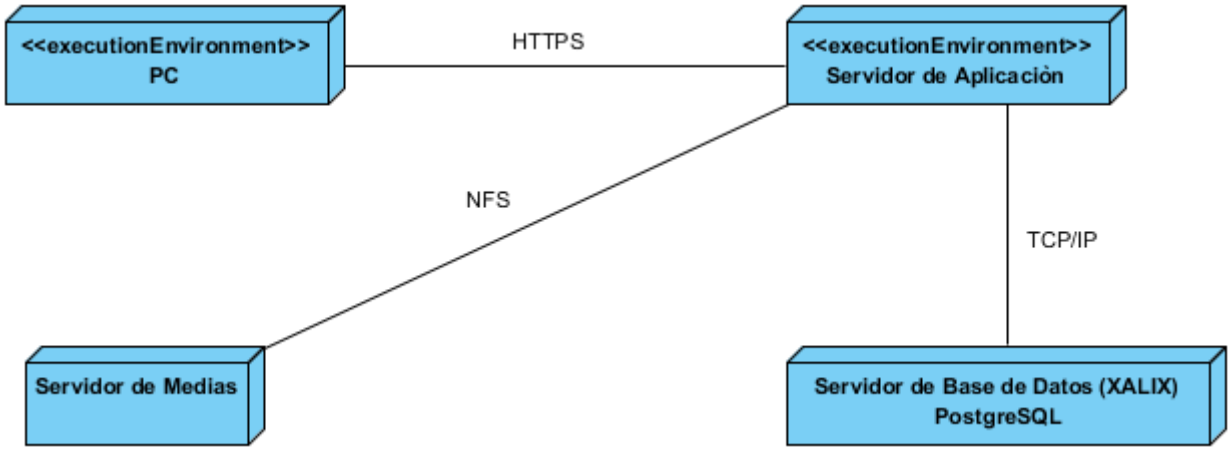


Ilustración 3:Diagrama de Despliegue

2.5 Estándar de codificación

Los estándares de codificación se enfocan en definir la estructura y apariencia física del código a programar, lo que facilita su entendimiento y lectura. En la implementación de la arquitectura se utilizaron los diferentes estándares definidos en el Centro FORTES que se aprecian a continuación:

Definición de clases

Las declaraciones de clases tienen su llave de apertura una línea más abajo de la declaración y el nombre de la clase comienza con mayúscula. A continuación, un ejemplo de la definición de una clase.

```php
<?php

namespace XALIX\CoreBundle\Entity;

use FOS\UserBundle\Model\User as BaseUser;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Validator\Constraints as Assert;
use Vich\UploaderBundle\Mapping\Annotation as Vich;
use Gedmo\Mapping\Annotation as Gedmo;

/**
 *User
 *
 * @ORM\Table(name="fos_user"))
 * @ORM\Entity(repositoryClass="XALIX\CoreBundle\Repository\UserRepository")
 * @Vich\Uploadable
 */
class User extends BaseUser
{
```

Ilustración 4: Ejemplo de definición de una clase

Definiciones de métodos

El nombre de los métodos comienza con una letra minúscula, en caso de ser un nombre compuesto por dos o más palabras, estas comenzarán con letra mayúscula. A continuación, un ejemplo:

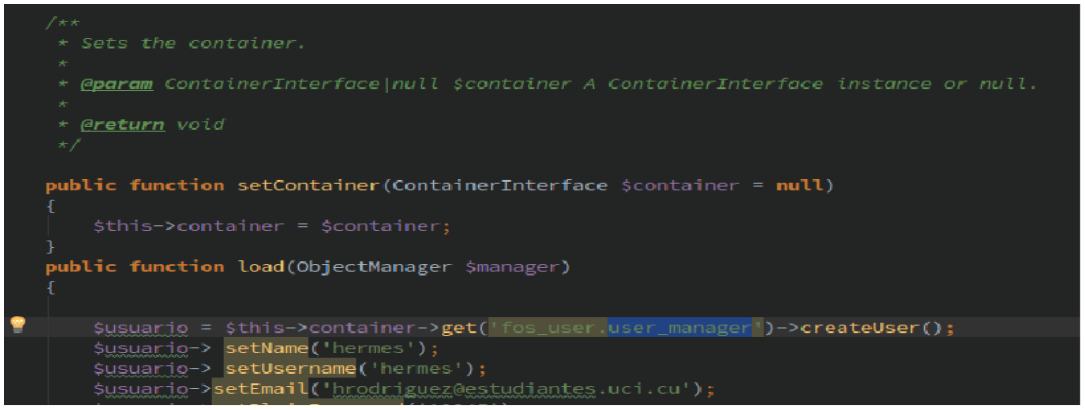


Ilustración 5: Ejemplo de definición de métodos

Llamadas a funciones y asignación de variables

Las llamadas a las funciones de una clase se realizarán sin espacios entre el nombre de la función y los paréntesis. En el caso de utilizar llamadas a funciones de una misma clase, se utilizará *($this->)* antes del nombre de la función. Las asignaciones se realizarán mediante el signo de igualdad (=). A continuación, se muestra un ejemplo:

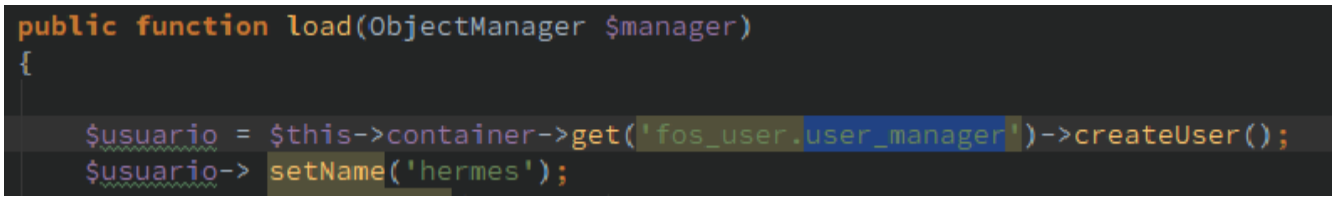


Ilustración 6: Ejemplo de llamadas a funciones y asignación de variables

2.6 Conclusiones del capítulo

En este capítulo se han abordado los aspectos referentes a la concepción de la arquitectura XALIX 2.0. Se detallan los componentes, versiones de bundles, las restricciones arquitectónicas y el patrón arquitectónico que conforman la propuesta de solución de la arquitectura XALIX 2.0, que posibilita salir de la obsolescencia tecnológica que presentaba la versión 1.0. A partir de lo anterior, aparecen factores primordiales para el éxito de los productos de FORTES, la mantenibilidad y la documentación de todos los cambios con respecto a la versión anterior. Ya establecida una propuesta solución y

diseñada e implementada la arquitectura, queda establecida la vía para llevar a cabo la validación de la arquitectura.

CAPÍTULO 3: Evaluación de la arquitectura

3.1 Introducción del capítulo

En el capítulo se abordan los elementos fundamentales para la evaluación de la arquitectura, se expresan los atributos de calidad, las técnicas para la evaluación y los métodos que se aplican para evaluar a XALIX 2.0.

3.2 Evaluación de arquitecturas de software

La arquitectura de *software* posee gran impacto sobre la calidad de un sistema, por lo que se hace necesario evaluarla para determinar su potencial, para alcanzar los atributos de calidad requeridos. Es importante destacar que la evaluación no define si una arquitectura es buena o no, simplemente expresa donde se encuentran los riesgos y fortalezas de la misma. El primer paso para evaluar una arquitectura es conocer qué es lo que se quiere evaluar, de esta forma es posible establecer la base para la evaluación, otra decisión importante es determinar cuándo se realizará la evaluación. Para esto, aunque es posible evaluar la arquitectura en cualquier fase del desarrollo, existen dos variantes definidas (CLEMENTS, PAUL *et al.*, 2002):

- **Evaluación temprana:** es aquella que no tiene que esperar a que la arquitectura esté totalmente especificada. Esta puede ser realizada desde fases tempranas y a lo largo del proceso de desarrollo, para examinar las decisiones arquitectónicas ya tomadas y decidir entre las opciones que están pendientes.
- **Evaluación tardía:** consiste en realizar la evaluación de la arquitectura cuando se encuentra establecida y la implementación se ha completado. Este es el caso general que se presenta en el momento de la adquisición de un sistema ya desarrollado. Se considera muy útil la evaluación del sistema en este punto, puesto que puede observarse el cumplimiento de los atributos de calidad asociados al sistema, y cómo será su comportamiento general.

La evaluación de una arquitectura de *software* pretende medir propiedades del sistema en base a especificaciones abstractas, como, por ejemplo, los diseños arquitectónicos. Las mediciones que se realizan sobre una arquitectura de *software* pueden tener distintos objetivos. Algunos de estos son (BOSCH, 2000):

- **La medición cualitativa:** se aplica para la comparación entre arquitecturas candidatas y tiene relación con la intención de saber la opción que se adapta mejor a cierto atributo de calidad. Este tipo de medición brinda respuestas afirmativas o negativas, sin mayor nivel de detalle.
- **La medición cuantitativa:** busca la obtención de valores que permitan tomar decisiones en cuanto a los atributos de calidad de una arquitectura de *software*. El esquema general es la comparación con márgenes establecidos, como lo es el caso de los requerimientos de desempeño, para establecer el grado de cumplimiento de una arquitectura candidata, o tomar decisiones sobre ella. Este enfoque permite establecer comparaciones, pero se ve limitado en tanto no se conozcan los valores teóricos máximos y mínimos de las mediciones con las que se realiza la comparación.
- **La medición de máximo y mínimo teórico:** contempla los valores teóricos para efectos de la comparación de la medición con los atributos de calidad especificados. El conocimiento de los valores máximos o mínimos permite el establecimiento claro del grado de cumplimiento de los atributos de calidad.

3.3 Atributos de calidad

La calidad del producto *software* se puede interpretar como el grado en que dicho producto satisface los requisitos de sus usuarios y se encuentran representados en el modelo de calidad definido por la ISO/IEC 25010. Este modelo se encuentra compuesto por las ocho características de calidad que se muestran a continuación (ISO25000, 2011):

Ilustración 7: Clasificación de los atributos de calidad

Clasificación de los atributos de calidad

La arquitectura de un sistema de *software* es determinante para que éste posea una combinación deseada o requerida de atributos de calidad. La arquitectura es la que exhibe o inhibe los atributos de calidad de un sistema, pues una arquitectura representa las decisiones más tempranas del diseño, que impactarán de manera decisiva sobre las etapas de desarrollo posteriores. De esta manera, es posible hacer predicciones sobre cuáles atributos de calidad, y cuáles no, poseerá el sistema de *software* mediante la evaluación de su arquitectura (BERTONI y VILLANUEVA, 2010).
En términos generales, en el trabajo de (CLEMENTS, PAUL C, 2002) se establece una clasificación de los atributos de calidad en dos categorías:

- **Observables vía ejecución:** aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución. La descripción de algunos de estos atributos se presenta en la *Tabla 3*.

- **No observables vía ejecución:** aquellos atributos que se establecen durante el desarrollo del sistema. La descripción de algunos de estos atributos se presenta en la *Tabla 4*.

Tabla 3: Descripción de atributos de calidad observables vía ejecución.

Atributo de Calidad	Descripción
Disponibilidad	Es la medida de disponibilidad del sistema para el uso.
Confidencialidad	Es la ausencia de acceso no autorizado a la información.
Funcionalidad	Habilidad del sistema para realizar el trabajo para el cual fue concebido.
Desempeño	Es el grado en el cual un sistema o componente cumple con sus funciones designadas, dentro de ciertas restricciones dadas, como velocidad, exactitud o uso de memoria.
Confiabilidad	Es la medida de la habilidad de un sistema a mantenerse operativo a lo largo del tiempo.

Seguridad externa	Ausencia de consecuencias catastróficas en el ambiente. Es la medida de ausencia de errores que generan pérdidas de información.
Seguridad interna	Es la medida de la habilidad del sistema para resistir a intentos de uso no autorizados y negación del servicio, mientras se sirve a usuarios legítimos.

Tabla 4: Descripción de atributos de calidad no observables vía ejecución.

Atributo de Calidad	Descripción
Configurabilidad	Posibilidad que se otorga a un usuario experto a realizar ciertos cambios al sistema.
Integrabilidad	Es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados separadamente al ser integrados.
Integridad	Es la ausencia de alteraciones inapropiadas de la información.
Interoperabilidad	Es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema.
Modificabilidad	Es la habilidad de realizar cambios futuros al sistema.
Mantenibilidad	Es la capacidad de someter a un sistema a reparaciones y evolución. Capacidad de modificar el sistema de manera rápida y a bajo costo.

Portabilidad	Es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, software o una combinación de los dos.
Reusabilidad	Es la capacidad de diseñar un sistema de forma tal que su estructura o parte de sus componentes puedan ser reutilizados en futuras aplicaciones.
Escalabilidad	Es el grado con el que se pueden ampliar el diseño arquitectónico, de datos o procedimental.
Capacidad de Prueba	Es la medida de la facilidad con la que el software, al ser sometido a una serie de pruebas, puede demostrar sus fallas. Es la probabilidad de que, asumiendo que tiene al menos una falla, el *software* fallará en su próxima ejecución de prueba.

3.4 Técnicas de evaluación de arquitectura

Las técnicas utilizadas para la evaluación de atributos de calidad requieren grandes esfuerzos por parte del ingeniero de *software* para crear especificaciones y predicciones. Estas técnicas requieren información del sistema a desarrollar que no está disponible durante el diseño arquitectónico, sino al principio del diseño detallado del sistema (BOSCH, 2000).

En vista de que el interés es tomar decisiones de tipo arquitectónico en las fases tempranas del desarrollo, son necesarias técnicas que requieran poca información detallada y puedan conducir a resultados relativamente precisos. Las técnicas existentes en la actualidad para evaluar arquitecturas permiten hacer una evaluación cuantitativa sobre los atributos de calidad a nivel arquitectónico, pero se tienen pocos medios para predecir el máximo (o mínimo) teórico para las arquitecturas de *software*. Sin embargo, debido al costo de realizar este tipo de evaluación, en muchos casos los

arquitectos de *software* evalúan cualitativamente, para decidir entre las alternativas de diseño (CLEMENTS, PAUL *et al.*, 2002).

Evaluación basada en escenario

Un escenario es una breve descripción de la interacción de alguno de los involucrados en el desarrollo de un sistema con este. Consta de tres partes: el estímulo, el contexto y la respuesta. El estímulo es la parte del escenario que explica o describe lo que el involucrado en el desarrollo hace para iniciar la interacción con el sistema. Puede incluir la ejecución de tareas, cambios en el sistema, ejecución de pruebas, entre otros. El contexto describe qué sucede en el sistema al momento del estímulo. La respuesta describe, a través de la arquitectura, cómo debería responder el sistema ante el estímulo. Este último elemento es el que permite establecer cuál es el atributo de calidad asociado (REVIEWER-HERZOG, 2015).

Evaluación basada en la experiencia

En muchas ocasiones los arquitectos e ingenieros de *software* otorgan valiosas ideas que resultan de utilidad para la evasión de decisiones erradas de diseño, estas ideas se basan en factores subjetivos (como la experiencia) y están respaldadas por una línea lógica de razonamiento, que se puede adquirir por el trabajo realizado en proyectos similares. Por tanto, el principal instrumento de evaluación con que cuenta esta técnica es precisamente la intuición y experiencia con que cuentan los arquitectos y demás miembros del equipo de desarrollo. Existen dos tipos de evaluación basada en experiencia: la evaluación informal, que es realizada por los arquitectos de *software* durante el proceso de diseño, y la realizada por equipos externos de evaluación de arquitecturas (BOSCH, 2000).

Evaluación basada en prototipo

Esta técnica consiste en implementar una parte de la arquitectura de *software* y ejecutarla en el contexto del sistema. Para su utilización se necesita mayor información sobre el desarrollo, disponibilidad de *hardware* y los elementos que constituyen el contexto del sistema de *software*. Mediante esta técnica se obtiene un resultado de evaluación con mayor exactitud (BOSCH, 2000).

3.5 Métodos para la evaluación de una arquitectura.

Los métodos de evaluación arquitectónica, evalúan el potencial del diseño arquitectónico para alcanzar los niveles deseados en cuanto a los requisitos de calidad, estos métodos se ven asistidos de las técnicas de evaluación arquitectónica analizadas anteriormente. Actualmente, existen diversos

métodos para realizar pruebas a la arquitectura de *software*, cada uno con características específicas, escoger un método de evaluación requiere tener bien definidos los atributos que se desean evaluar. Algunos de los métodos de evaluación son los siguientes (CLEMENTS, PAUL *et al.*, 2002):

- Método de Análisis de Arquitectura de *Software* (*Software Architecture Analysis Method*, SAAM).
- Método de Revisión Intermedio de Diseño *(Active Reviews for Intermediate Designs*, ARID).
- Método de Análisis de Acuerdos de Arquitectura de *Software* (*Architecture Trade-off Analysis Method*, ATAM).

SAAM

El SAAM fue el primer método en ser ampliamente difundido y documentado. Se creó originalmente para el análisis de la modificabilidad de una arquitectura, pero en la práctica ha demostrado ser muy útil para evaluar de forma rápida atributos de calidad, tales como la portabilidad, escalabilidad e integrabilidad.

ARID

El ARID es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. Es un híbrido entre *Active Design Review* (ADR) y ATAM. Las preguntas giran en torno a la calidad y completitud de la documentación y la suficiencia, ajuste y conveniencia de los servicios que provee el diseño propuesto.

ATAM

El ATAM está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM. El método se concentra en la identificación de los estilos arquitectónicos o enfoques arquitectónicos utilizados y ayuda a los involucrados en el proyecto a entender las consecuencias de las decisiones arquitectónicas tomadas con respecto a los atributos de calidad.

En la *Tabla 5* se realiza una comparación entre los tres métodos mencionados con anterioridad para tener a modo de resumen las principales diferencias entre estos.

Tabla 5: Tabla comparativa de los métodos de evaluación de arquitecturas

	ATAM	SAAM	ARID
Atributos de calidad contemplados	Modificabilidad Seguridad Confiabilidad Desempeño	Modificabilidad Funcionabilidad	Conveniencia del diseño evaluado.
Objetos analizados	Estilos Arquitectónicos Documentación Flujo de Datos Vistas Arquitectónicas.	Documentación Vistas arquitectónicas.	Especificación de los componentes.
Etapas del proyecto en las que se aplica	Luego que el diseño de la arquitectura ha sido establecido.	Luego que la arquitectura cuenta con funcionalidad ubicada en módulos.	A lo largo del diseño de la arquitectura.
Enfoques utilizados	Árbol de utilidad y lluvia de ideas para articular los requerimientos de calidad. Análisis arquitectónico que detecta puntos sensibles, puntos de balance y riesgos.	Lluvia de ideas para escenarios y articular los requerimientos de calidad. Análisis de los escenarios para verificar funcionalidad o estimar el costo de los cambios.	Revisiones de diseño, lluvia de ideas para obtener escenarios.

3.6 Evaluación de la arquitectura propuesta.

Después del análisis realizado sobre las diferentes técnicas y métodos de evaluación de arquitecturas, se seleccionan, para evaluar la arquitectura propuesta, la técnica de evaluación basada en escenarios y el método de evaluación ATAM.

Evaluación mediante ATAM

El método ATAM está compuesto por nueve pasos divididos en cuatro fases, los cuales se adaptarán de acuerdo a los requerimientos del proyecto. Después de la descripción de la solución propuesta y el análisis realizado en los capítulos 1 y 2, se procede a evaluar la arquitectura propuesta empleando este método. Para establecer la prioridad de los escenarios se analizó el riesgo de no contar con la característica que representa el escenario en la arquitectura, para esto formularon las siguientes interrogantes:

- ¿Qué ocurre si el escenario no se cumple?
- ¿Cuánto esfuerzo es necesario para cumplir el escenario?
- ¿Es posible compensar el no responder a este escenario?

Árbol de utilidad

Es un esquema en forma de árbol que presenta los atributos de calidad refinados hasta el establecimiento de escenarios que especifican a detalle el nivel de prioridad de cada uno. Identifica los atributos de calidad que destacan en una arquitectura, estos son planteados por los involucrados en el desarrollo. Tiene un nodo raíz que es la utilidad general de la arquitectura, el segundo nivel son los atributos de calidad que contiene una serie de escenarios, señalando la escala de importancia y dificultad asociada (CLEMENTS, PAUL *et al.*, 2005).

Tabla 6: Árbol de utilidad

Atributo	Subatributos	Escenario	Prioridad
Compatibilidad	Interoperabilidad	En la arquitectura deben coexistir múltiples bundles de terceros sin que uno invalide al otro.	Media
Seguridad	Confidencialidad	El sistema protege contra el acceso a datos e información de personas no autorizadas, ya sea accidental o deliberadamente.	Media
	Integridad	El sistema previene accesos o modificaciones no autorizados a sus datos.	Alta
Mantenibilidad	Modularidad	Al realizar un cambio en un componente del	Alta

		sistema debe tener un impacto mínimo en los demás.	
	Reusabilidad	Los elementos de la arquitectura deben ser reusables en otros desarrollos.	Alta
	Capacidad de ser modificado	El sistema puede ser modificado de forma efectiva y eficiente sin introducir defectos o degradar el desempeño.	Alta

Especificación de los escenarios

Las tablas que a continuación se presentan incluyen las descripciones de los escenarios que se especificaron en el árbol de utilidad.

Tabla 7: Descripción del escenario 1

Escenario #1: En la arquitectura deben coexistir múltiples bundles de terceros sin que uno invalide al otro.	
Atributo(s)	Compatibilidad - Interoperabilidad
Estímulo	Se integran los bundles *FOS_User* y *SonataAdminBundle*
Respuesta	La arquitectura soporta la coexistencia de estos bundles
Explicación	La arquitectura utiliza las facilidades ofrecidas por el *framework* Symfony para la integración de bundles de terceros a través del composer verificando la compatibilidad entre los bundles a integrar e instalando todas las dependencias necesarias por estos bundles.

Tabla 8: Descripción del escenario 2

Escenario #2: El sistema protege contra el acceso a datos e información de personas no autorizadas, ya sea accidental o deliberadamente.

Atributo(s)	Seguridad – Confidencialidad
Estímulo	Se intenta acceder y navegar en un demo funcional de la arquitectura
Respuesta	Se deniega el acceso pidiendo la autenticación.
Explicación	La arquitectura previene el acceso no autorizado a través del sistema de autenticación brindado por Symfony.

Tabla 9: Descripción del escenario 3

Escenario #3: El sistema previene accesos o modificaciones no autorizados a sus datos.	
Atributo(s)	Seguridad – Integridad
Estímulo	Se intenta modificar un nomenclador en el demo funcional sin autorización
Respuesta	Se deniega el acceso a la acción por falta de permisos para ejecutarla.
Explicación	La arquitectura previene la modificación no autorizada a través del sistema de seguridad y control de roles y permisos brindado por Symfony.

Tabla 10: Descripción del escenario 4

Escenario #4: Al realizar un cambio en un componente del sistema debe tener un impacto mínimo en los demás.	
Atributo(s)	Mantenibilidad – Modularidad
Estímulo	Se desactiva el bundle *SonataAdmin*
Respuesta	La arquitectura debe mostrar un error informando la necesidad de activar el bundle de Sonata
Explicación	La desactivación de este bundle provoca que las visualizaciones de las vistas contengan error ya que son implementadas a partir de la estructura generada por Sonata. Esto significa que si se desea desactivar este bundle por completo sea

	necesario modificar todas las vistas que ya se hayan generado. Una vez modificadas se comprueba la correcta funcionalidad de las vistas.

Tabla 11: Descripción del escenario 5

Escenario #5: Los elementos de la arquitectura deben ser reusables en otros desarrollos.	
Atributo(s)	Mantenibilidad – Reusabilidad
Estímulo	Se trata de implementar diferentes vistas de administración para una misma entidad
Respuesta	La arquitectura debe permitir la generación de CRUD diferentes para una misma entidad sin generar conflictos entre estos
Explicación	A partir de la integración del *SonataAdminBundle* es posible generar diferentes vistas de gestión sobre una misma entidad definiendo formularios, rutas y vistas diferentes y permitiendo la coexistencia entre ellos sin afectarse uno a otro.

Tabla 12: Descripción del escenario 6

Escenario #6: El sistema puede ser modificado de forma efectiva y eficiente sin introducir defectos o degradar el desempeño.	
Atributo(s)	Mantenibilidad – Capacidad de ser modificado
Estímulo	Se trata de agregar un nuevo bundle a la arquitectura a través del composer
Respuesta	La arquitectura permite la integración del nuevo bundle sin generar conflictos con los existentes
Explicación	Mediante el gestor de depencias de PHP, *Composer*, se instala el bundle ***friendsofsymfony/rest-bundle*** que se utiliza para extender la creación de servicios web de Symfony. Para lograr la correcta integración se siguen los pasos detallados con anterioridad en el Capítulo 2.

Resultados de la evaluación con ATAM

A partir de la información obtenida tras la aplicación del ATAM, se identificó un riesgo en la arquitectura propuesta, este riesgo está asociado al escenario 4.

Tabla 13: Riesgo detectado en la evaluación

Escenario	Riesgo
4	La desactivación del *SonataAdminBundle* genera la necesidad de modificar todas las vistas generadas en esta arquitectura que en dependencia de la cantidad y complejidad de estas vistas puede no ser factible modificarlas.

Luego del análisis del riesgo detectado se determinó, por parte del colectivo de desarrolladores del centro FORTES que este no afecta considerablemente el desempeño de la arquitectura propuesta y la probabilidad de ocurrencia del riesgo asociado al escenario 4 es muy baja. Además, partiendo de la idea de que la evaluación solo proporciona el potencial de arquitectura para alcanzar los atributos de calidad requeridos, se realizó una prueba de la arquitectura con los desarrolladores del centro FORTES. Como objetivo principal de esta prueba se consideró:

- Probar la funcionalidad de la arquitectura propuesta.
- Encontrar errores y funcionamientos no adecuados en la arquitectura.
- Dar a conocer al equipo de desarrollo la arquitectura candidata a utilizar para la creación de aplicaciones web.

Las pruebas realizadas arrojaron la presencia de errores en el componente de las traducciones, los cuales fueron solventados oportunamente mediante la configuración correcta de los archivos ***parameters.yml*** y ***config.yml*** de Symfony. También se constató que la familiarización con la arquitectura y su estructura general es sencilla y cómoda. Se demostró también que la arquitectura es completamente funcional y capaz de adaptarse, aunque se aprecia la posibilidad de mejorar en aspectos que se detallan en las recomendaciones de este trabajo.

3.7 Conclusiones del capítulo

El análisis de los atributos de calidad y sus clasificaciones, así como las técnicas de evaluación de la arquitectura, la clasificación de los principales métodos para su evaluación y la selección del método ATAM, permitió la confección del árbol de utilidad y la especificación de los escenarios, lo que arrojó un riesgo. Además, se realizaron pruebas con los desarrolladores, los que ofrecieron una excelente retroalimentación acerca de la posibilidad de mejorar la arquitectura.

Conclusiones generales

Al culminar la investigación se arriban a las conclusiones siguientes:

- Se logró desarrollar la arquitectura XALIX 2.0, dando cumplimiento al objetivo de la investigación. En esta nueva arquitectura se actualiza el núcleo de Symfony a la versión 3.4, garantizando soporte a largo plazo, modernización de los lenguajes utilizados, entre otras mejoras. Además, XALIX 2.0 viene a suplir una de las grandes limitantes de su predecesora, la falta de documentación y la mantenibilidad.
- La utilización del enfoque de vistas arquitectónicas seleccionado en el presente trabajo brindó la guía para describir la arquitectura de manera organizada y estructurada.
- Asociado a la evaluación de la propuesta del presente trabajo, se seleccionó el método ATAM, el cual permitió la correcta medición de atributos de calidad que incrementan el valor final de la arquitectura.

Recomendaciones

A la arquitectura propuesta, se le pueden aplicar modificaciones que mejoren su desempeño, por lo que se recomienda evaluar la posibilidad de incorporar un bundle para la gestión del control de acceso basado en roles que extienda las opciones de seguridad ofrecidas por Symfony. Se recomienda, además, valorar la factibilidad de emplear la arquitectura propuesta para los nuevos proyectos web del centro FORTES.

Referencias Bibliográficas

(W3C), T. W. W. W. C. *HTML & CSS* [Consultado el: 22 de Octubre de 2014]. Disponible en: http://www.w3.org/standards/webdesign/htmlcss#whatcss.

ACHOUR, M.; BETZ, F.*, et al. Manual de PHP* [Consultado el: 17 de Octubre de 2014]. Disponible en: http://docs.php.net/manual/es.

ALVAREZ, S. *Sistemas gestores de bases de datos* Última actualización: 31 de Julio de 2007. [Consultado el: 17 de Octubre de 2014]. Disponible en: http://www.desarrolloweb.com/articulos/sistemas-gestores-bases-datos.html.

BERTONI, F. A. y VILLANUEVA, S. Identificación de Atributos de Calidad en Requerimientos. 2010, nº

BOSCH, J. *Design and use of software architectures: adopting and evolving a product-line approach.* Pearson Education, 2000. ISBN 0201674947.

CLEMENTS, P.; KAZMAN, R.*, et al.* Evaluating software architectures: methods and case studies. *Addison-Wesley Professional*, 2005, nº ISSN 978-0201704822.

---. *Evaluating software architectures: methods and case studies.* Addison-Wesley, 2002, ISBN 9780201704822.

CLEMENTS, P. C. Software architecture in practice. *Diss. Software Engineering Institute*, 2002, nº

CORDERO, R. N. L. y SALAVERT, I. R. *Las vistas arquitectónicas de software y sus correspondencias mediante la gestión de modelos.* Tesis Doctoral, Universidad Politécnica de Valencia, 2010.

DUNGLAS, K. *Persistence in PHP with the Doctrine ORM.* Packt Publishing Ltd, 2013. ISBN 1782164111.

EGUILUZ, J. *Desarrollo web ágil con Symfony2* [Consultado el: 10 de junio de 2018]. Disponible en: http://standars.optimeconsulting.net/wp-content/uploads/2016/09/desarrollo-agil-symfony-2.8.pdf.

---. *New in Symfony 3.3: Automatic Console logging* 2 de febrero de 2017, [Consultado el: 15 de febrero de 2018]. Disponible en: http://symfony.com/blog/new-in-symfony-3-3-automatic-console-logging.

---. *Nuevo en Symfony 2.7: mejoras para ser más productivo* de 2017]. Disponible en: http://symfony.es/noticias/2015/04/23/nuevo-en-symfony-27-mejoras-para-ser-mas-productivo/.

---. *Symfony publicará una nueva versión 2.8 para facilitar el paso a Symfony 3* de 2017]. Disponible en: http://symfony.es/noticias/2015/04/10/symfony-publicara-una-nueva-version-28-para-facilitar-el-paso-a-symfony-3/.

---. *symfony.es* 2017, [Consultado el: 16 de febrero de 2018]. Disponible en: http://symfony.es/noticias-de-symfony-en-2017.

GÓMEZ, O. *Documentando la arquitectura de software* [Consultado el: 10 de junio de 2018]. Disponible en: http://osgg.net/omarsite/resources/papers/doc_arch.pdf.

---. *Evaluando la Arquitectura de Software, métodos de evaluación.* 2007, 2007, vol. No.02, Disponible en: http://osgg.net/omarsite/resources/papers/doc_arch.pdf.

GUTIERREZ, D. *UML Diagramas de Paquetes* [Consultado el: 10 de junio de 2018]. Disponible en: http://www.codecompiling.net/files/slides/UML_clase_05_UML_paquetes.pdf.

GUTIERREZ, J. J. *¿Qué es un framework web?* [Consultado el: 20 de octubre de 2017]. Disponible en: http://www.lsi.us.es/~javierj/investigacion_ficheros/Framework.pdf.

HILLIARD, R. Ieee-std-1471-2000 recommended practice for architectural description of software-intensive systems. *IEEE, http://standards. ieee. org*, 2000, vol. 12, nº 16-20, p. 2000.

IBM. [Consultado el: 23 de octubre de 2017]. Disponible en: https://www.ibm.com/Search/?q=arquitectura&lnk=mhsrch&v=18&en=utf&lang=en&cc=us&ui=mhb#q=arquitectura&site=ibmcom.

IEEE. *IEEE 4201* [Consultado el: 1 de noviembre de 2017]. Disponible en: Iso-architecture.org.

INC, N. *Welcome to NGINX Wiki* [Consultado el: 12 de mayo de 2018]. Disponible en: https://www.nginx.com/resources/wiki/.

ISO25000. *ISO/IEC 25010* [Consultado el: 13 de mayo de 2018]. Disponible en: http://iso25000.com/index.php/normas-iso-25000/iso-25010.

ISO/IEC. *ISO/IEC 25010* [Consultado el: 1/3/ de 2018]. Disponible en: http://iso25000.com/index.php/normas-iso-25000/iso-25010.

JACOBSON, I.; BOOCH, G.*, et al. El Lenguaje Unificado de Modelado (UML)* [Consultado el: 29 de Octubre de 2014]. Disponible en: http://ingenieriasoftware2011.files.wordpress.com/2011/07/el-lenguaje-unificado-de-modelado-manual-de-referencia.pdf.

KROLL, P. y KRUCHTEN, P. *The rational unified process made easy: a practitioner's guide to the RUP.* Addison-Wesley Professional, 2003. ISBN 0321630009.

KRUCHTEN, P. Architectural blueprints—The “4+ 1” view model of software architecture. *Tutorial Proceedings of Tri-Ada*, 1995, vol. 95, nº p. 540-555.

LARMAN, C. *UML y Patrones.* 1999, 520 p. Disponible en: http://sunshine.prod.uci.cu/gridfs/sunshine/books/Craig_Larman_-_UML_y_Patrones.pdf.

LAWSON, B. y SHARP, R. *Introducing HTML5.* 2011, 240 p. Disponible en: http://sunshine.prod.uci.cu/gridfs/sunshine/books/Introducing-HTML5---New-Riders.pdf. ISBN 978-0-321-68729-6.

LTD, V. P. I. *Visual Paradigm for UML* [Consultado el: 30 de Octubre de 2014]. Disponible en: http://www.visual-paradigm.com.

MARCA, H.-M. y QUISBERT, N.-S. *ANALISIS Y DISEÑO DE SISTEMAS II, “Diagrama de Despliegue”* [Consultado el: 10 de junio de 2018]. Disponible en: http://virtual.usalesiana.edu.bo/web/practica/archiv/despliegue.doc.

MARIÑO, C. V. *Programación en PHP5. Nivel Básico.* 2008, 180 p. Disponible en: https://n-1.cc/file/download/745373.

PÉREZ, J. E. *Introducción a CSS* [Consultado el: 17 de Octubre de 2014]. Disponible en: http://www.librosweb.es/css.

POSTGRESQL. *PostgreSQL Global Development Group* [Consultado el: 25 de Octubre de 2014]. Disponible en: http://www.postgresql.org/about/.

POTENCIER, F. *Symfony 3.1.0 released* [Consultado el: 10 de junio de 2018]. Disponible en: https://symfony.com/blog/symfony-3-1-0-released.

---. *Symfony 3.2.0 released* [Consultado el: 15 de febrero de 2018]. Disponible en: https://symfony.com/blog/symfony-3-2-0-released.

---. *symfony.es* de 2017]. Disponible en: http://symfony.es/documentacion/.

POTENCIER, F. y ZANINOTTO, F. Symfony, la guía definitiva. *Libros Web.[Online] www. librosweb. es*, 2008, nº

PUEBLA, Y. A. C.; GÓMEZ, E. C.*, et al. Procedimiento para la evaluación de arquitecturas de software basadas en componentes* [Consultado el: 10 de abril de 2018]. Disponible en: http://curiositysec.com/procedimiento-para-la-evaluacion-de-arquitecturas-de-software-basadas-en-componentes/.

REVIEWER-HERZOG, J. Software Architecture in Practice Third Edition Written by Len Bass, Paul Clements, Rick Kazman. *ACM SIGSOFT Software Engineering Notes*, 2015, vol. 40, nº 1, p. 51-52. ISSN 0163-5948.

REYNOSO, C. y KICILLOF, N. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* 2004,

SENSIOLABS. *Symfony Bundles Documentation* [Consultado el: 15 de mayo de 2018]. Disponible en: https://symfony.com/doc/bundles/.

SOMMERVILLE, I. *Ingeniería de software novena edición.* México: Pearson, 2011,

SONI, D.; NORD, R. L.*, et al.* Software architecture in industrial applications. En *Software Engineering, 1995. ICSE 1995. 17th International Conference on. 1995*. p. 196-196.

THAMES, J. P. B. *Ingeniería del software asistida por computadora (case)* [Consultado el: 19 de Junio de 2015]. Disponible en: http://es.slideshare.net/jpbthames/ingeniera-del-software-asistida-por-computadora-case.

TORRES, A. C.; PREVAL, K. S.*, et al. ARQUITECTURA DE REFERENCIA PARA PHP* Academia.edu, [Consultado el: 25 de mayo de 2018]. Disponible en: https://www.academia.edu/12057345/ARQUITECTURA_DE_REFERENCIA_PARA_PHP?ends_sutd_reg_path=true.

VILLA LÓPEZ, E. A. y RODRIGUEZ GUAMAN, E. J. *Inyección de Contenido en un VCMS (Video Content Management System) y la Automatización de Publicaciones en Medios Sociales (Facebook, Twitter).* Escuela Superior Politécnica de Chimborazo, 2015.

ZELDMAN, J. *Diseño con estándares web* [Consultado el: 20 de Octubre de 2014]. Disponible en: http://books.google.com.ar/books/about/Diseño_con_estándares_web.html?id=tMJ_AAAACAAJ.