Universidad de las Ciencias Informáticas
Facultad 4

Título: MODELACIÓN DE ARQUITECTURA PARA APLICACIONES EMPRESARIALES EN PHP.

Trabajo de Diploma para optar por el título de

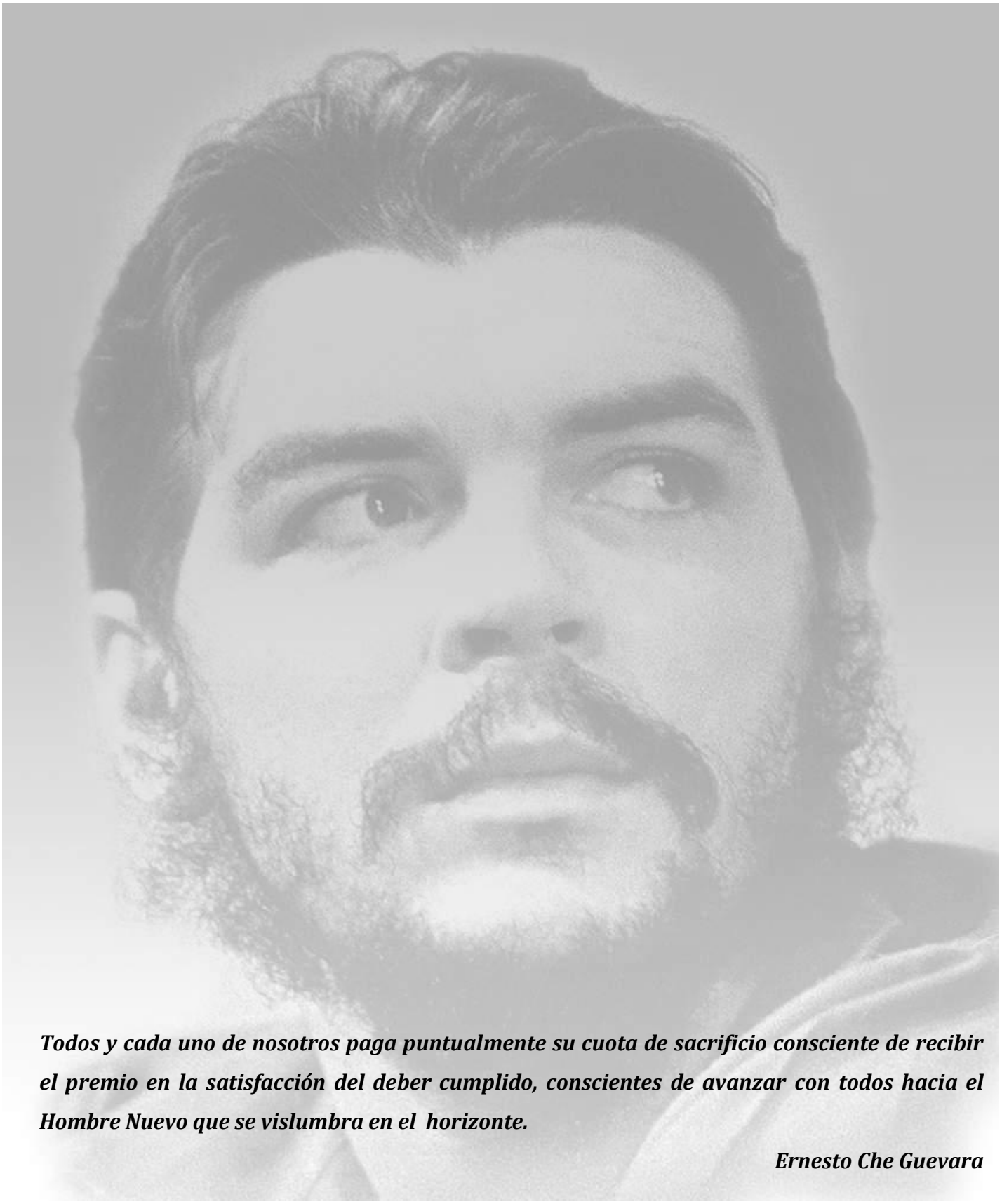
Ingeniero Informático

Autor(es): Alberto Hidalgo Reyes
Josué Vazquez Sorí

Tutor(es): Ing. Alain Eduardo Rodríguez Arias

Ciudad de La Habana, Cuba

Junio del 2008



Todos y cada uno de nosotros paga puntualmente su cuota de sacrificio consciente de recibir el premio en la satisfacción del deber cumplido, conscientes de avanzar con todos hacia el Hombre Nuevo que se vislumbra en el horizonte.

Ernesto Che Guevara

DECLARACIÓN DE AUTORÍA

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

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

Alberto Hidalgo Reyes

_____
Firma del Autor

Josué Vazquez Sorí

_____
Firma del Autor

Ing. Alain Eduardo Rodríguez Arias

_____
Firma del tutor

AGRADECIMIENTOS

A mi madre por todo el sacrificio que ha realizado toda su vida para darme la mejor educación, por todos sus consejos en los buenos y malos momentos, por estar a mi lado, por confiar en mí, por ser una gran madre.

A mi padre por brindarme todo su apoyo, por siempre confiar en mí, por todo el esfuerzo realizado para darme lo mejor.

A mi hermano Javier por compartir los momentos malos y buenos que hemos tenido que enfrentar en la vida , por apoyarme muchas veces , por ser mi hermano.

A mi tía Mercedes y a mi tío Gerardito pues sin su ayuda no hubiera sido posible terminar, por todos los consejos que me han dado, por apoyarme, por cuidarme todo este tiempo.

A mis primos Milton y Maikel por estar a mi lado y por ser mis hermanos.

A mi abuela, a mi madrina y a mi abuelo por estar pendientes de mí, por haberme querido toda la vida, por ayudarme.

A mi novia por brindarme su apoyo y su cariño y por no fallarme nunca.

Josué

A Fidel y a su espíritu UCI, por haber soñado con un futuro del cual ya puedo formar parte.

A mis padres por ser mi guía, mi inspiración, mi orgullo; por haberme enseñado a andar por la vida con confianza y seguridad, por el amor que han depositado en mí, por ser mi luz en los momentos más oscuros, por ser la fuente de mis fuerzas.

A Gretel por su cariño, confianza y apoyo inquebrantables en cada minuto, simplemente por quererme como lo hace.

A mi familia por estar siempre al tanto de cada acontecimiento de mi vida, por su apoyo en todo momento.

A mis abuelos, por haberme malcriado tanto.

A mi tía Ari por defenderme siempre, por la seguridad que siempre me dio en estos cinco años, por su confianza en mí.

A mi tío Felipe por estar siempre al tanto de mis estudios.

A mis suegros por estar siempre pendientes de mí, por su apoyo.

A mis amigos y compañeros de toda la vida, por haber sido los hermanos que nunca tuve, por ayudarme siempre.

A Adrián, Despaigne, del Toro y su familia por haberme brindado siempre su ayuda.

A mi viejo amigo Eduardo, por su apoyo y confianza incondicionales desde el primer momento, por haberme aceptado como hijo.

Alberto

Agradecimientos

De ambos:

A nuestro tutor por su paciencia y apoyo en todo momento.

A todos los que de una forma o de otra nos dieron sus ideas, experiencias y tiempo para realizar este trabajo.

Sin ustedes no hubiésemos llegado hasta aquí.

DEDICATORIA

A la Revolución Cubana, a Fidel y a la UCI.

A mis padres.

A ustedes va dedicado todo mi esfuerzo y trabajo.

Alberto.

A mis padres y mi hermano por todo su esfuerzo y confianza en mí.

A mis tíos y primos por su ayuda incondicional.

A todos mis compañeros, por apoyarme siempre.

Josué

.

RESUMEN

El presente trabajo de diploma contiene un estudio de los elementos fundamentales que constituyen la Arquitectura de Software, durante su desarrollo se realiza un análisis de estos elementos con el objetivo de lograr una correcta propuesta de arquitectura para el proyecto Residencia.

Como propuesta de solución a este proyecto, en este trabajo se elabora una arquitectura de software de dominio específico para desarrollar aplicaciones empresariales en PHP que interactúe con servicios web y utilice Propel como ORM. Como parte de la misma se desarrolló una arquitectura como línea de referencia del desarrollo de software, se trabajó con un framework, que responde a los requerimientos específicos de esas aplicaciones, se definió un flujo de trabajo para guiar la construcción del software, se definió la interacción con servicios web y se elaboró una propuesta de ambiente de desarrollo para la definición de la misma.

TABLA DE CONTENIDOS

INTRODUCCIÓN

El desarrollo de aplicaciones en el entorno empresarial requiere cada vez más alcanzar un conjunto de requisitos comunes tales como integración, seguridad, confiabilidad, escalabilidad y disponibilidad [IBM]. Sin el cumplimiento de estos no se puede hablar de calidad del producto desarrollado. Todos estos requisitos se deciden en el proceso de diseño de la arquitectura de la aplicación y los servicios Web y tienen gran impacto en la implementación de estos requisitos. (1).

La arquitectura del Software tiene que ver con la planificación y mantenimiento en la estructura del sistema de los conceptos iniciales a través del desarrollo y operación. Una buena arquitectura es una estructura del sistema estable que puede acomodarse a los cambios de requisitos y de tecnologías. (1).

Además se ha observado la evolución de las técnicas de abstracción de la arquitectura, como por ejemplo los modelos y los patrones; sin embargo, aún se debe desarrollar mucho más con respecto a sus valores apreciables.

El proyecto Residencia, tiene como principal objetivo la informatización de los procesos propios de la residencia de la Universidad de las Ciencias Informáticas, el cual estará basado en una arquitectura orientada a servicios, utilizando para ello herramientas de software libre, como por ejemplo PHP, el cual utilizaremos debido a que es de fácil uso y que la similitud con los lenguajes más comunes de la programación estructurada, permiten a la mayoría de los programadores experimentados crear aplicaciones complejas con una curva de aprendizaje muy suave, entre sus principales ventajas podemos mencionar que tiene un lenguaje multiplataforma, es libre por lo tanto es de fácil acceso para todos, tiene capacidad de conexión con la mayoría de los manejadores de base de datos que se utilizan en la actualidad entre muchas otras. Entonces como **situación problémica** surge la necesidad de definir una arquitectura para desarrollar aplicaciones empresariales en PHP que se adapte y sea flexible a las condiciones específicas de una entidad o empresa determinada y sea capaz de interactuar con servicios web, porque las arquitecturas de software no se pueden copiar ni reutilizar en su totalidad.

Por lo anterior planteado el **problema científico** podemos describirlo de la siguiente manera: ¿Como definir y elaborar una arquitectura para el desarrollo de aplicaciones empresariales en PHP?

Este trabajo tiene como **objeto de estudio** la definición y diseño de una arquitectura para el desarrollo de aplicaciones empresariales en PHP.

Encontrándose dentro de los Objetivos de la Investigación:

Generales:

- ✓ Diseñar una arquitectura para desarrollar aplicaciones empresariales en PHP en su primera versión, para ser utilizada en el proyecto de informatización de la Residencia

Específicos:

- ✓ Analizar los estilos arquitectónicos y las propuestas de arquitecturas existentes en el mundo para obtener sus experiencias y ventajas, así como sus desventajas para prever problemas en la propuesta planteada.
- ✓ Identificar qué características de estas propuestas tienen la calidad que facilite y garantice confeccionar aplicaciones empresariales en PHP.
- ✓ Analizar una serie de Patrones estándares a la hora de programar una aplicación Web.
- ✓ Realizar el diseño de la arquitectura propuesta.
- ✓ Validar la propuesta de arquitectura obtenida en el proyecto de Informatización de Residencia si las condiciones del proyecto lo permiten.

El Campo de acción es muy amplio y del objeto de estudio analizado, podemos definir que el **campo de acción** es la definición y diseño de una arquitectura en la creación de un software automatizado para la informatización del proyecto Residencia en la Universidad de las Ciencias Informáticas.

Ahora bien, como **Hipótesis** podemos plantear que si se define una arquitectura para desarrollar aplicaciones empresariales en PHP se logrará desarrollar sistemas de una forma más eficiente, segura y rápida utilizando una misma línea común y facilitando el trabajo para los analistas y diseñadores.

Dentro de las Tareas de la investigación encontramos:

- ✓ Selección de las herramientas para llevar a cabo el proyecto y la elección de la plataforma en la que se desarrollará la arquitectura fundamentando su elección.
- ✓ Selección de la metodología de análisis y diseño de sistemas informáticos, que faciliten y garanticen la creación con calidad de la misma.
- ✓ Investigar las propuestas de arquitecturas para aplicaciones Web que utilicen servicios web en el mundo para la obtención de experiencias y mejoras en una nueva versión.
- ✓ Analizar los documentos rectores de la arquitectura que están incluidos en el expediente de proyecto UCI para identificar cómo incluir la propuesta dentro de esos documentos.

1. CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA.

1.1. Introducción:

En este capítulo se realiza un estudio del marco teórico sobre las arquitecturas para desarrollar aplicaciones empresariales en PHP, analizando los principales conceptos para lograr su mejor entendimiento. Se caracterizarán las herramientas y tecnologías que se analizaron para ser utilizadas y que hoy en día juegan un rol importante a la hora de diseñar una buena arquitectura de software.

1.2. Arquitectura de Software.

En los inicios de la informática, (la programación) se consideraba un arte, debido a la dificultad que entrañaba para la mayoría de los mortales, pero con el tiempo se han ido desarrollando metodologías para conseguir esos propósitos. (3) Pero no es hasta 1968, cuando Edsger Dijkstra, de la Universidad Tecnológica de Eindhoven en Holanda propuso que se estableciera una estructuración correcta de los sistemas de software antes de lanzarse a programar, escribiendo código de cualquier manera [Dij68a], que, aunque no utiliza el término arquitectura para describir el diseño conceptual del software, sus conceptos sientan las bases para lo que luego evolucionaría como concepto de lo que hoy se conoce como *Arquitectura del Software.*

Una definición reconocida es la de Clements [Cle96a]: La AS es, a grandes rasgos, una vista del sistema que incluye los principales componentes del mismo, la conducta de esos componentes según se percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones.

Garlan and Shaw en 1994 y 1996 la conceptualizaron como una colección de componentes y conectores unidos con una descripción de la interacción entre componentes y conectores. (17).

La definición oficial de Arquitectura del Software es la IEEE Std 1471-2000 que dice así: “La Arquitectura del Software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución”.

Como podemos ver todas estas definiciones se basan o se enfocan sobre los componentes y conectores (cómo interactúan).

Componente: Es una organización no-trivial, casi independiente, y parte reemplazable de un sistema que cumple una clara función en el contexto de una arquitectura bien definida. Un componente se ajusta y proporciona la realización física de un conjunto de interfaces (Philippe Krutchen) (17).

Conectores: Definen la interacción entre los componentes y describen las reglas que gobiernan esas interacciones. (17).

Interfaces: Un componente define una interfaz a través de un conector que une vínculos con otro componente. Un componente puede tener múltiples interfaces. (17).

Desde Dijkstra hasta la actualidad existen muchas definiciones de arquitectura del software, de hecho, existen grandes compilaciones de definiciones alternativas o contrapuestas, como la colección que se encuentra en el SEI (*http://www.sei.cmu.edu/architecture/definitions.html),*y no parece que ninguna de ellas haya sido totalmente aceptada.

En un sentido amplio podríamos estar de acuerdo en que la Arquitectura del Software es el diseño de más alto nivel de la estructura de un sistema, programa o aplicación y tiene la responsabilidad de:

- Definir los componentes (módulos y subsistemas) principales.
- Definir las responsabilidades que tendrá cada de ellos.
- Definir la interacción que existirá entre dichos componentes.
- Control y flujo de datos
- Secuenciación de la información
- Protocolos de interacción y comunicación
- Ubicación en el hardware

Independientemente de las diversas definiciones, *¿Qué nos proporciona?* La arquitectura de software establece los fundamentos para que analistas, diseñadores, programadores, etc. trabajen en una línea común que permita alcanzar los objetivos y necesidades del sistema de información, permitiendo la organización en todo el desarrollo, define la organización fundamental de un sistema, encarnada en sus componentes, las relaciones entre ellos y con su entorno, y los principios que gobiernan su diseño y evolución.

La arquitectura de software está afectada no solo por la estructura y el comportamiento sino también por el uso, la funcionalidad, el rendimiento, la flexibilidad, la reutilización, la facilidad de comprensión, la restricciones y compromisos económicos y tecnológicos y la estética. (4)

1.3. Estilos Arquitectónicos

Las soluciones de diseño arquitectónicas que son comunes y reusables a lo largo de años de experiencia se han ido agrupando en lo que más tarde se les llamó *estilos*. El éxito del diseño de la arquitectura de software depende de los estilos que se decidan utilizar para el desarrollo de la misma.
Los estilos expresan la arquitectura en el sentido más formal y teórico, describen entonces una clase de arquitectura, o piezas identificables de las arquitecturas empíricamente dadas. Esas piezas se encuentran repetidamente en la práctica, trasuntando la existencia de decisiones estructurales coherentes. Una vez que se han identificado los estilos, es lógico y natural pensar en re-utilizarlos en situaciones semejantes que se presenten en el futuro. (17).
Los estilos arquitectónicos son arquitecturas comunes, marcos de referencia arquitectónica prototípica, formas comunes, clases de sistemas. (Rumbaugh & A.l 1991).
Otra definición de estilos arquitectónicos es la planteada por Perry & Wolf, en 1992

- Componentes (ahora: Elementos)
- Conectores
- Configuraciones
- Restricciones (Constraints)

Un estilo arquitectónico define tanto un vocabulario de tipos de componentes y conectores como un conjunto de restricciones sobre cómo combinar esos componentes y conectores. (2), que sirven para sintetizar estructuras de soluciones que luego serán refinadas a través del diseño, definir patrones posibles de las aplicaciones, evitando errores arquitectónicos, permiten evaluar arquitecturas alternativas con ventajas y desventajas conocidas ante diferentes conjuntos de requerimientos.
Existe una gran variedad de estilos arquitectónicos las cuales tienen diversas clasificaciones entre las que se encuentran:

- Estilos de flujo de datos
- Estilos centrados en datos
- Estilos de llamada y retorno
- Estilos de código móvil
- Estilos heterogéneos

A diferencia de los patrones de diseños, que son centenares, los estilos se ordenan en seis o siete clases fundamentales y unos veinte ejemplares, como máximo. Las arquitecturas complejas o compuestas resultan del agregado o la composición de estilos más básicos. Existen pocos estilos que encapsulan una enorme variedad de configuraciones.

1.3.1. Arquitectura Cliente-Servidor

Esta se encuentra dentro del estilo de llamada y retorno y consiste básicamente en que un programa -el Cliente informático- realiza peticiones a otro programa -el servidor- que le da respuesta. En esta arquitectura la capacidad de proceso está repartida entre los clientes y los servidores, aunque son más importantes las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema.

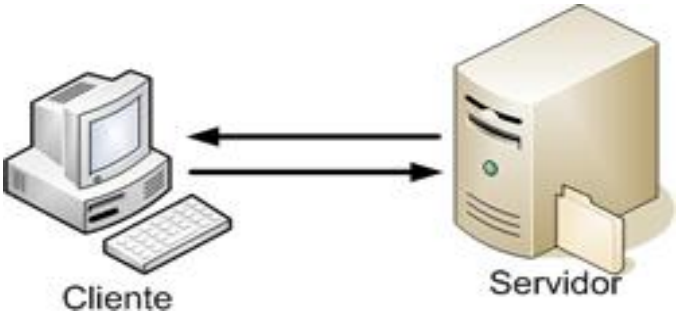


Figura 1.1 Cliente-Servidor

La separación entre cliente y servidor es una separación de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un sólo programa.

1.3.2. Arquitectura en capas

Estilo arquitectónico organizado jerárquicamente y dividido en capas en el que cada capa presta servicio a la capa inmediata superior y sirve de cliente a la capa inmediata inferior (Garlan y Shaw).
Este estilo proporciona un diseño basado en elevados niveles de abstracción, permitiéndoles así a los programadores separar un problema complejo en una secuencia incremental de pasos, permite la optimización ya que los cambios solo afectan a las capas vecinas, permite la reutilización ya que cada capa tiene una responsabilidad en particular: como los tipos de datos abstractos, se pueden hacer diferentes implementaciones de la misma capa respetando las interfaces con las capas adyacentes. Esto lleva a la posibilidad de definir interfaces de capas estándares para cada tipo de implementación.
Una especialización de este estilo es la arquitectura de tres capas con un reparto claro de funciones: una capa para la presentación (interfaz de usuario), otra para el cálculo (donde se encuentra modelado el negocio) y otra para el almacenamiento (persistencia). Una capa solamente tiene relación con la siguiente a través de la interface que expone.

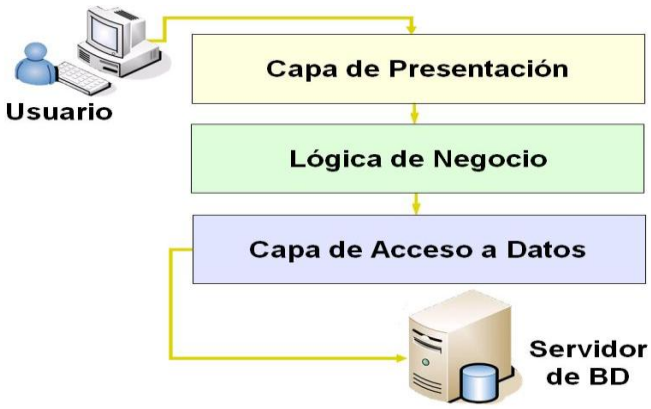


Figura 1.2 Arquitectura en capas

1.4. PATRONES ARQUITECTÓNICOS

Son patrones de alto nivel que fijan la arquitectura global de una aplicación.

"….los patrones se refieren más bien a prácticas de re-utilización y los estilos conciernen a teorías sobre la estructuras de los sistemas a veces más formales que concretas…" (15)

Buschmann et al. (1996) define patrón como una regla que consta de tres partes, la cual expresa una relación entre un contexto, un problema y una solución.

Estas son:

Contexto. Es una situación de diseño en la que aparece un problema de diseño.

Problema. Es un conjunto de fuerzas que aparecen repetidamente en el contexto.

Solución. Es una configuración que equilibra estas fuerzas.

Para Buschmann (1996) son plantillas para arquitecturas de software concretas, que especifican las propiedades estructurales de una aplicación y tienen un impacto en la arquitectura de subsistemas.

La selección de un patrón arquitectónico es, por tanto, una decisión fundamental de diseño en el desarrollo de un sistema de software.

Patrón Arquitectónico	***Descripción***
Layers(Capas)	Consiste en estructurar aplicaciones que pueden ser descompuestas en grupos de subtareas, las cuales se clasifican de acuerdo a un nivel particular de

	abstracción.
Pipes and Filters (Tuberías y Filtros)	Provee una estructura para los sistemas que procesan un flujo de datos. Cada paso de procesamiento está encapsulado en un componente filtro (filter). El dato pasa a través de conexiones (pipes), entre filtros adyacentes.
Blackboard	Aplica para problemas cuya solución utiliza estrategias no determinísticas. Varios subsistemas ensamblan su conocimiento para construir una posible solución parcial ó aproximada.
Broker	Puede ser usado para estructurar sistemas de software distribuido con componentes desacoplados que interactúan por invocaciones a servicios remotos. Un componente broker es responsable de coordinar la comunicación, como el reenvío de solicitudes, así como también la transmisión de resultados y excepciones.
Model-View-Controler	Divide una aplicación interactiva en tres componentes. El modelo (model) contiene la información central y los datos. Las vistas (view) despliegan información al usuario. Los controladores (controlers) capturan la entrada del usuario. Las vistas y los controladores constituyen la interfaz del usuario.
Presentation-Abstraction-Control	Define una estructura para sistemas de software interactivos de agentes de cooperación organizados de forma

	jerárquica. Cada agente es responsable de un aspecto específico de la funcionalidad de la aplicación y consiste de tres componentes: presentación, abstracción y control.
Microkernel	Aplica para sistemas de software que deben estar en capacidad de adaptar los requerimientos de cambio del sistema. Separa un núcleo funcional mínimo del resto de la funcionalidad y de partes específicas pertenecientes al cliente.
Reflection	Provee un mecanismo para sistemas cuya estructura y comportamiento cambia dinámicamente. Soporta la modificación de aspectos fundamentales como estructuras tipo y mecanismos de llamadas a funciones.

1.4.1. ESTILOS Y PATRONES ARQUITECTÓNICOS

Estiman que los estilos se pueden comprender como clases de patrones, o tal vez más adecuadamente como lenguajes de patrones. Un estilo proporciona de este modo un lenguaje de diseño con un vocabulario y un framework a partir de los cuales los arquitectos pueden construir patrones de diseño para resolver problemas específicos.

Algunos patrones coinciden con los estilos hasta en el nombre con que se los designa. (5)

Estilo Arquitectónico	***Patrón Arquitectónico***
Sólo describe el esqueleto estructural y general para aplicaciones	Existen en varios rangos de escala comenzando con patrones que definen la estructura básica de una aplicación
Son independientes del contexto al que puedan ser aplicados.	Partiendo de la definición de patrón, requieren de la especificación de un

	contexto del problema
Cada estilo es independiente de los otros	Depende de patrones más pequeños que contiene, patrones con los que interactúa, o de patrones que lo contengan
Expresan técnicas de diseño desde una perspectiva que es independiente de la situación actual de diseño.	Expresa un problema recurrente de diseño muy específico, y presenta una solución para él, desde el punto de vista del contexto en el que se presenta
Son una categorización de sistemas.	Son soluciones generales a problemas comunes

Como hemos podido ver, la diferencia entre los estilos y patrones arquitectónicos está en el nivel de abstracción en el que nos movemos. Los estilos se aplican a un nivel muy alto de abstracción, en el cual no interesa saber cuál es la semántica de los elementos que estamos utilizando, sólo hablamos de filtros y tubos u objetos sin interesarnos por lo que están representando. En el caso de los patrones, tenemos en cuenta el significado de los filtros, tubos u objetos, tal como hemos visto en los patrones de descomposición en capas –en el que se habla de presentación, dominio de aplicación y de datos- o en el de Módulo Tabla, que diferencia entre un objeto que representa un elemento -una línea de una tabla o registro- u otro objeto que representa la totalidad de la tabla. (2)

Podría decirse que mientras los estilos han enfatizado descriptivamente las configuraciones de una arquitectura, desarrollando incluso lenguajes y notaciones capaces de expresarlas formalmente, los patrones, aún los que se han caracterizado como arquitectónicos, se encuentran más ligados al uso y más cerca del plano físico, sin disponer todavía de un lenguaje de especificación. Los patrones se refieren más bien a prácticas de re-utilización y los estilos conciernen a teorías sobre la estructuras de los sistemas a veces más formales que concretas (15).

De forma que, aunque en ocasiones para referirnos a ellos mezclemos patrones con estilos y viceversa, debemos pensar que los estilos son una forma mucho más amplia o global de definir una arquitectura, desde un nivel de abstracción más alto, sin muchos detalles y los patrones van más ligados a la implementación de estos estilos, un poco más abajo en cuanto a la abstracción, directamente con el código o el lenguaje de programación que se utiliza.

1.5. Arquitectura Modelo Vista Controlador (MVC)

Patrón clásico del diseño Web conocido como arquitectura MVC, que está formado por tres niveles:

- *El modelo* representa la información con la que trabaja la aplicación, es decir, su lógica de negocio.
- *La vista* transforma el modelo en una página Web que permite al usuario interactuar con ella.
- *El controlador* se encarga de procesar las interacciones del usuario y realiza los cambios apropiados en el modelo o en la vista.

La arquitectura MVC separa la lógica de negocio (el modelo) y la presentación (la vista) por lo que se consigue un mantenimiento más sencillo de las aplicaciones. Por ejemplo, si una misma aplicación debe ejecutarse tanto en un navegador estándar como en un navegador de un dispositivo móvil, solamente es necesario crear una vista nueva para cada dispositivo; manteniendo el controlador y el modelo original. El controlador se encarga de aislar al modelo y la vista de los detalles del protocolo utilizado para las peticiones (HTTP, consola de comandos, email, etc.). El modelo se encarga de la abstracción de la lógica relacionada con los datos, haciendo que la vista y las acciones sean independientes de, por ejemplo, el tipo de gestor de bases de datos utilizado por la aplicación.

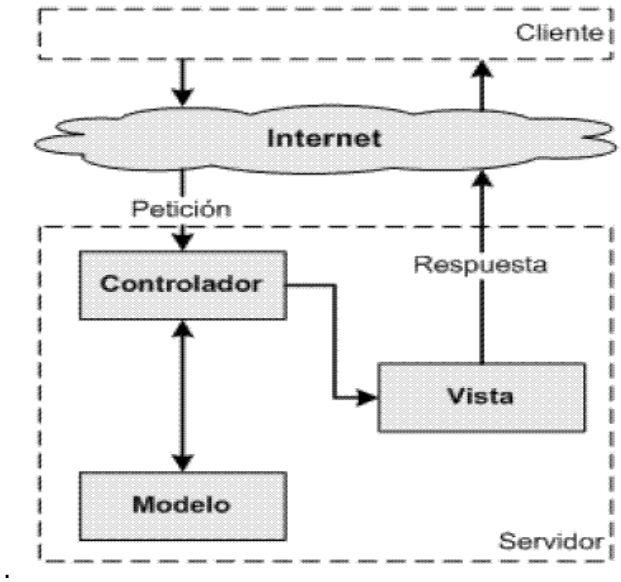


Figura 1.3 El patrón MVC

Este modelo de arquitectura presenta varias ventajas:

- Hay una clara separación entre los componentes de un programa; lo cual nos permite implementarlos por separado

- Hay un API muy bien definido; cualquiera que use el API, podrá reemplazar el Modelo, la Vista o el Controlador, sin aparente dificultad.
- La conexión entre el Modelo y sus Vistas es dinámica; se produce en tiempo de ejecución, no en tiempo de compilación.
- Al incorporar el modelo de arquitectura MVC a un diseño, las piezas de un programa se pueden construir por separado y luego unirlas en tiempo de ejecución. Posteriormente, si uno de los Componentes, se observa que funciona mal puede reemplazarse sin que las otras piezas se vean afectadas.

La porción del programa que transforma los datos dentro del Modelo en una presentación gráfica es la Vista. La Vista incorpora la visión del Modelo a la escena; es la representación gráfica de la escena desde un punto de vista determinado, bajo condiciones de iluminación determinadas.

El Controlador sabe que puede hacer el Modelo e implementa la interfaz de usuario que permite iniciar la acción.

1.6. TECNOLOGÍAS ACTUALES

1.6.1. Servicio

Una función sin estado, auto-contenida, que acepta una(s) llamada(s) y devuelve una(s) respuesta(s) mediante una interfaz bien definida. Los servicios pueden también ejecutar unidades discretas de trabajo: cómo sería editar y procesar una transacción. Los servicios no dependen del estado de otras funciones o procesos. La tecnología concreta utilizada para prestar el servicio no es parte de esta definición, lo que implica que en una SOA pueden coexistir aplicaciones escritas en distintas tecnologías y lenguajes de programación. (3)

Los objetivos principales que persiguen los servicios web son la interoperabilidad y la integración, esta última permite obtener la información solicitada en tiempo real, agilizando el proceso de toma de decisiones. (4)

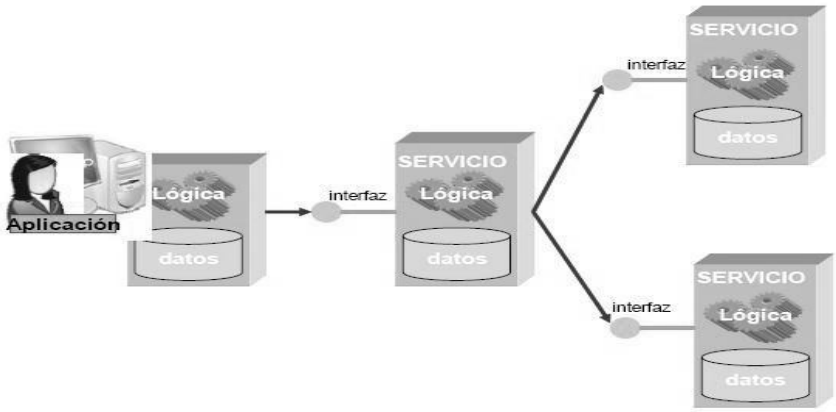


Figura 1.4 ***Sistema basado en servicios***

Sistema basado en servicios.

En la figura se representa un sistema sencillo basado en servicios. Una aplicación, además de implementar sus propios componentes de negocio y datos, también puede reutilizar la funcionalidad de servicios existentes en la red Empresarial. (4)

1.6.2. XML

XML son las siglas en inglés de Extensible Markup Language (lenguaje de marcado ampliable o extensible) desarrollado por el World Wide Web Consortium (W3C). No es más que un conjunto de reglas para definir etiquetas semánticas que nos organizan un documento en diferentes partes. XML es un metalenguaje que define la sintaxis utilizada para definir otros lenguajes de etiquetas estructurados. XML tiene otras aplicaciones entre las que destaca su uso como estándar para el intercambio de datos entre diversas aplicaciones o software con lenguajes privados como en el caso del SOAP. (3)

1.6.3. WSDL

WSDL son las siglas de Web Services Description Language, un formato XML que se utiliza para describir servicios Web. La versión 1.1 está en estado de "propuesta de recomendación" por parte del W3C. WSDL describe la interfaz pública a los servicios Web. Está basado en XML y describe la forma de comunicación, es decir, los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo. Las operaciones y mensajes que soporta se describen en abstracto y se ligan después al protocolo concreto de red y al formato del mensaje. (3)

1.6.4. UDDI

UDDI son las siglas del catálogo de negocios de Internet denominado Universal Description, Discovery, and Integration. El registro en el catálogo se hace en XML. UDDI es una iniciativa industrial abierta (sufragada por la OASIS), en el contexto de los servicios Web. El registro de un negocio en UDDI tiene tres partes:

Páginas blancas - dirección, contacto y otros identificadores conocidos

Páginas amarillas - categorización industrial basada en taxonomías

Páginas verdes- información técnica sobre los servicios que aportan las propias empresas.

UDDI es uno de los estándares básicos de los servicios Web cuyo objetivo es ser accedido por los usuarios y clientes y dar paso a documentos WSDL, en los que se describen los requisitos del protocolo y los formatos del mensaje solicitado para interactuar con los servicios Web del catálogo de registros. (3)

1.6.5. Esquema XSD (XML Schema)

Un esquema XSD es la descripción de la estructura de la información contenida en un archivo XML y de sus reglas, por ejemplo, longitud, tipo, obligatoriedad, etc. Se requiere para que los sistemas que transmiten o reciben archivos XML puedan validar la conformación de estos archivos, a reglas definidas por los autores. (3)

1.6.6. SOAP

SOAP (siglas de Simple Object Access Protocol) es un protocolo estándar creado por Microsoft, IBM y otros, está actualmente bajo el auspicio de la W3C que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. SOAP es uno de los protocolos utilizados en los servicios Web. (3)

Existen varios modelos de mensajes SOAP, pero el más común es el RPC (remote procedure calls), donde un nodo de le red (el cliente) envía un mensaje de solicitud a otro nodo (el servidor) y este último responde al mensaje del cliente. Estos mensajes son independientes del sistema operativo y pueden transportarse en protocolos como SMTP, MIME y HTTP (4)

1.6.7. REST

REST son las siglas en inglés de Representational State Transfer. Es un estilo de arquitectura y no un estándar definido. REST se basa en las URLs para identificar recursos, los cuales se retornan en XML puro. Debe aclarase que REST es una manera simplificada de ejecutar métodos remotos, las características principales y una comparación con SOAP se resumen a continuación: (3)

1.6.8. Comparación entre SOAP y REST

SOAP:

1. Los mensajes se representan en una estructura XML definida y estandarizada ("envelope").
2. Puede utilizarse sobre protocolos como: HTTP, SMTP, etc.
3. El acceso y la manipulación de datos es específica a la aplicación.
4. La seguridad no se define a nivel de SOAP y debe ser implementada por el desarrollador.
5. Esquemas XML (XSD) son utilizados para definir "contratos" o interfaces entre un cliente y el servicio.

REST:

1. Los mensajes son representados en XML plano.
2. Se utiliza HTTP como protocolo de transporte.
3. Se utiliza comandos de HTTP para acceder a los datos (GET, POST, DELETE, PUT).
4. Se utilizan URLs para identificar recursos de manera única.
5. Se utiliza HTTPS para brindar seguridad.
6. No hay un método formal para expresar los métodos brindados ("contrato").

1.6.9. ARQUITECTURA ORIENTADA A SERVICIOS (SOA)

SOA es un estilo de arquitectura de IT que soporta la orientación a servicios. SOA se basa en la independencia de plataformas de hardware, de sistemas operativos y de lenguajes de programación. SOA fortalece la reutilización de los sistemas actuales que se construyeron y se utilizaron durante años y crea un ambiente en el que los negocios y la tecnología de la información pueden interactuar entre sí.

SOA se fundamenta en:

- Ejecutar rápido, adaptarse al mercado, ganar ante la competencia.
- Reutilizar los componentes de los procesos de negocios.
- Medir los resultados y tomar acción sobre ellos.
- Garantizar resultados que sean repetibles y predecibles.
- Empezar donde sea necesario (área de negocios - área de tecnología).

La Arquitectura Orientada a Servicios (SOA), es un concepto de arquitectura de software que define la utilización de servicios para dar soporte a los requerimientos de software del usuario. Al contrario de las arquitecturas orientadas a objetos, las SOAs están formadas por servicios de aplicación débilmente acoplados y altamente interoperables. Los Servicios Web se han convertido en el estandarte de SOA,

ya que esta tecnología posee un conjunto de características que permiten cubrir todos los principios de la orientación a servicios. La definición de la interfaz encapsula las particularidades de una implementación, lo que la hace independiente del fabricante, del lenguaje de programación o de la tecnología de desarrollo. Con esta arquitectura, se pretende que los componentes software desarrollados sean muy reusables, ya que la interfaz se define siguiendo un estándar. En la figura, presentamos la estructura básica de funcionamiento de un SOA tradicional. (4)

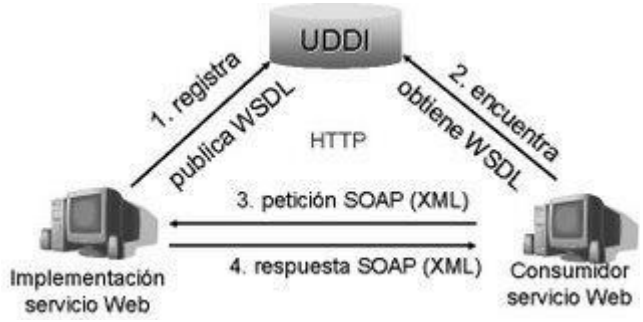


Figura 1.5 Funcionamiento de un SOA tradicional

En el gráfico anterior, se puede observar la existencia de tres roles claramente diferenciados:

- Cliente del servicio: Es el que solicita la ejecución del servicio web, y por lo tanto el que lo consume.
- Proveedor del servicio: Es el encargado de implementar el servicio web y ofrecerlo a los clientes.
- Registro del servicio: Es un repositorio donde se almacenan las descripciones de los servicios, para que así los clientes puedan buscar el servicio web que mejor se adapte a sus necesidades.

La secuencia de ejecución es la siguiente: 1. El proveedor del servicio da de alta el servicio web en el registro. Para realizar esto, el proveedor almacena en el registro el documento de descripción de éste. 2. El solicitante del servicio busca en el registro un servicio web que pueda adaptarse a sus necesidades. 3. Una vez seleccionado el servicio, el solicitante lo invoca mediante el envío de un mensaje SOAP, en el cual se indica la acción a realizar y los datos de entrada. 4. El servicio web recibe la petición y ejecuta la funcionalidad. Para finalizar envía un mensaje SOAP al solicitante con los resultados obtenidos. [4]

Como se aprecia, el intercambio entre cliente y servicio utilizando REST es mucho más simple y corto, por tanto, esta variante presenta un mayor rendimiento sacrificando otras funcionalidades que SOAP

aporta. En una SOA pueden combinarse ambas variantes para lograr un rendimiento óptimo en sistemas que lo requieran, sin sacrificar funcionalidad. (3)

1.6.10. WS-Security

La especificación WS-Security, que ha sido recientemente ratificada como estándar por OASIS, describe la forma de asegurar los servicios Web en el nivel de los mensajes, en lugar de en el del protocolo de transferencia o en el de la conexión.

WS-Security define la forma de conseguir integridad, confidencialidad y autenticación en los mensajes SOAP. La autenticación se ocupa de identificar al llamador. WS-Security utiliza tokens de seguridad para mantener esta información mediante un encabezado de seguridad del mensaje SOAP. La integridad del mensaje se consigue mediante firmas digitales XML, que permiten garantizar que no se han alterado partes del mensaje desde que lo firmó el originador. La confidencialidad del mensaje se basa en la especificación XML Encryption y garantiza que sólo el destinatario o los destinatarios a quien va destinado el mensaje podrán comprender las partes correspondientes.

1.7. Arquitectura de Software de Dominio Específico

En la década de los años 80, DARPA2 afrontó un fuerte problema desarrollando un gran sistema de software para la defensa, con un acercamiento llamado Arquitectura de Software de Dominio Específico (DSSA3). La motivación y fundamento para este programa fue basado en las siguientes suposiciones (17).

El desarrollo sobre dominios específicos puede ser optimizado. La reutilización de módulos es más probable si estos se obtienen de otras aplicaciones dentro del mismo dominio.

El objetivo principal fue reutilizar tantos objetos como fuera posible; además de minimizar la intención de resolver diferentes tipos de problemas antes de reinventar una solución para cada problema nuevo.

1.7.1. ¿Dominio?

Un dominio es un área de interés, usualmente representando un espacio del problema que es un subconjunto de una o más disciplinas. Por ejemplo, la investigación del ácido desoxirribonucleico (ADN) es un dominio. (17).

---

2 DARPA: Agencia de Investigación de Proyectos Avanzados de Defensa (DARPA por sus siglas en inglés) es una agencia del Departamento de Defensa de los Estados Unidos responsable del desarrollo de nuevas tecnologías para uso militar.

3 DSSA: siglas en inglés de Arquitectura de Software de Dominio Específico (Domain-Specific Software Architecture).
DARPA definió 4 pasos claves para las DSSA:

Por tanto, otro dominio puede ser las aplicaciones empresariales en PHP que utilicen Symfony y Propel. Cuando se construye una aplicación, se está desarrollando software para un espacio del problema, es decir, un conjunto de problemas a ser resueltos dentro del dominio.

En fin, un dominio es definido por un conjunto de situaciones o funciones comunes que las aplicaciones pueden resolver o hacer. La idea básica de las DSSA es hacer práctica la reutilización de software, enfocándose en los problemas específicos de dominios de software y desarrollando soluciones basadas en componentes para estos dominios de problemas. Estos componentes son genéricos y necesitan ser configurados en el momento que la aplicación es implementada.

Un modelo del dominio es desarrollado para establecer una terminología estándar y una semántica común para el dominio del problema.

Un conjunto de requerimientos de referencia para todas las aplicaciones dentro del dominio del problema que es desarrollado.

Una arquitectura de referencia es definida para una solución genérica, estandarizada y basada en componentes del sistema para anticipar los requerimientos del cliente.

Nuevas aplicaciones son desarrolladas para determinar requerimientos específicos de la aplicación, seleccionar o generar componentes particulares y ensamblarlos acorde a la arquitectura de referencia.

La arquitectura de referencia puede incluir componentes concretos que deberían ser embebidos en todas las aplicaciones que derivan de ella. Además, es adaptada para resolver los requerimientos particulares de los clientes.

La actividad principal es ajustar a la medida la arquitectura de referencia usando los requerimientos de la aplicación para producir la arquitectura específica de la aplicación.

1.7.2. Elementos de la DSSA

Una DSSA es un proceso e infraestructura que soporta el desarrollo de un modelo del dominio, requerimientos de referencia y una arquitectura de referencia para una familia de aplicaciones dentro de un dominio del problema. (17).

1.7.3. Modelo del dominio

El modelo del dominio es una representación de qué ocurre en el dominio, por ejemplo: operaciones de negocio en el dominio. Específicamente, esto describe las funciones a ser ejecutadas, las entidades que las ejecutan y los datos que fluyen entre ellas y si son usados o producidos por las funciones. El objetivo del modelo del dominio es estandarizar la terminología y la semántica del espacio del

problema, por ejemplo, el conjunto de problemas dentro del dominio que un conjunto de aplicaciones soportarán. (17).

1.7.4. Requerimientos de referencia

Los requerimientos de referencia son requerimientos estándares que surgen de diferentes áreas funcionales para todas las aplicaciones que pueden ser construidas en un dominio. Estos definen la estructura funcional general del espacio del problema, por ejemplo, los requerimientos funcionales. Además, especifican las restricciones sobre el diseño y la aplicación resultante, por ejemplo, los requerimientos no funcionales. Los requerimientos de las aplicaciones son realizados desde requerimientos particulares del cliente para cada aplicación específica. A veces, estos requerimientos son refinados a partir de requerimientos de referencia genéricos. (17).

1.7.5. Arquitectura de referencia

Una arquitectura de referencia es una arquitectura genérica, estándar, que describe a todos los sistemas dentro de un dominio determinado. Esto pudiera especificar tanto la plataforma de hardware o software, y los componentes. (17).

Los elementos de una arquitectura de referencia son los siguientes (17).

- Un modelo de la arquitectura de referencia que describe una configuración basada en componentes de hardware y software sobre un estilo arquitectónico.
- Un esquema arquitectónico que describe la estructura de alto nivel y restricciones de cada uno de los componentes, incluyendo las principales entidades de datos.
- Un diagrama de dependencias de componentes describiendo las interacciones a través de los principales componentes.
- Un conjunto de descripciones de interfaces de componentes describiendo el API de cada uno.
- Un conjunto de restricciones organizadas por parámetros, rango de valores, valores por defecto, y consecuencia sobre la arquitectura.

Según la anterior definición para la arquitectura de referencia según Kaiser y nuestro criterio redefiniríamos los elementos para la misma, acorde al dominio que estamos tratando, para utilizar los mismos como línea base en la organización y estructuración del desarrollo del software desde su inicio. Los nuevos elementos de la arquitectura de referencia son los siguientes:

- Definición del estilo/s y patrones arquitectónicos a utilizar.
- Convenciones o estándares de código (paquetes y archivos de código) y recursos (xml, archivos de propiedades, etcétera.).

- Estructura y organización del código.

Definición del estilo/s y patrones arquitectónicos a utilizar

La definición de estos elementos de arquitectura requiere de un análisis de las diferentes variantes disponibles y de los requisitos no funcionales de la aplicación. Las variantes a utilizar son:

- Cliente servidor
- Arquitectura de capas (tres capas).
- MVC

Convenciones o estándares de código

Define las normas y estructuras en las que se debe: programar el código fuente, nombrar las clases y recursos según sus responsabilidades y funciones; logrando con esto una uniformidad, organización y un mayor entendimiento en la forma de codificar y nombrar las clases, componentes y recursos de la aplicación. (6)

Estructura y organización del código

Define las normas a utilizar en cuanto a la separación del código de lado del cliente y el código de lado del servidor, ya sea mediante la utilización de motores de plantilla o no

1.7.6. Análisis y ambientes del dominio

El modelo del dominio es el resultado de un proceso de análisis del dominio, el cual es un aspecto esencial para diseñar los sistemas de software de alta calidad. El análisis del dominio es el proceso de analizar y entender el entorno del usuario final, por ejemplo, qué operaciones de negocio los usuarios ejecutan y en qué contexto. Dentro de la ingeniería de software, el análisis del dominio conlleva a estudiar el conocimiento base necesario para solucionar el problema del diseño del software. (17). El análisis del dominio involucra (17) :

- Identificar, capturar, organizar, y entender los objetos y las operaciones dentro del dominio.
- Una descripción de esos objetos y operaciones usando un vocabulario estandarizado.
- Identificar sistemas y problemas similares dentro del dominio.
- Determinar qué es reutilizable a través de los sistemas similares.

Según Stephen H. Kaiser en su libro “Software Paradigms”, existen tres ambientes que en el análisis del dominio se debería tener presente para definir y desarrollar una aplicación.

4 Paquete: se refiere a la agrupación de código fuente usado en java, que pudiera estar compuesto por uno o varios componentes.

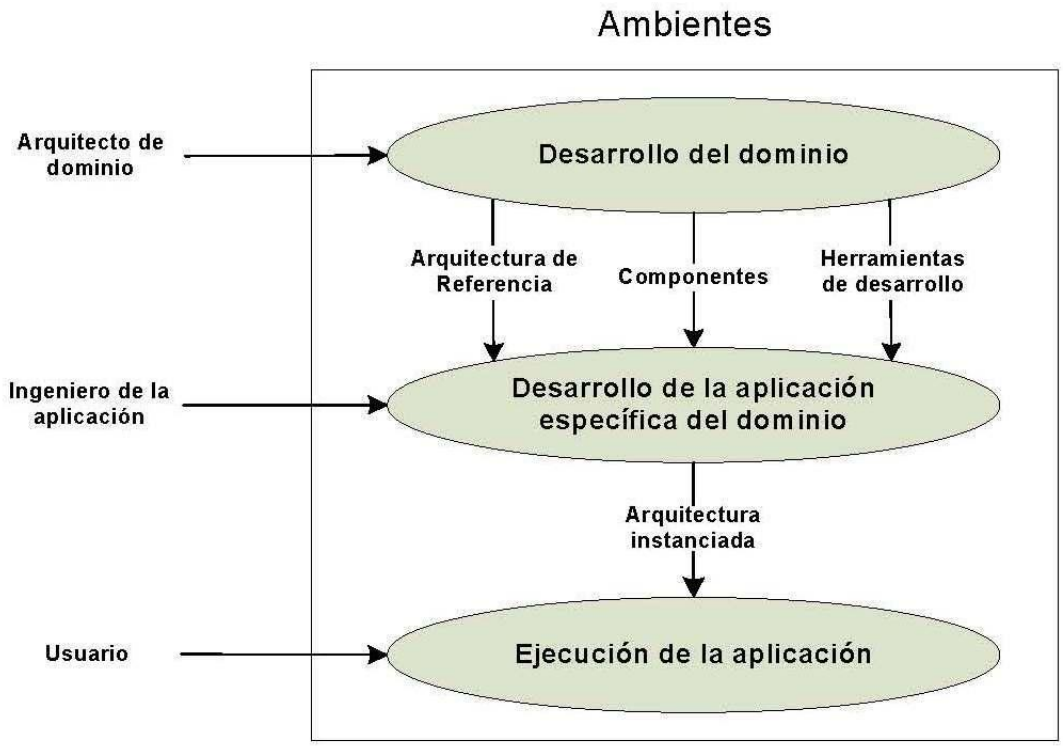


Figura 1.6 Ambientes de una DSSA

El primer ambiente es el mismo dominio, donde el análisis de los requerimientos y la definición de la arquitectura toman lugar. La arquitectura de referencia proporciona el esqueleto para las aplicaciones individuales desarrolladas para el dominio. El segundo ambiente es el entorno de desarrollo de la aplicación en el cual los requerimientos de la aplicación son producidos como respuesta y usados para desarrollar aplicaciones individuales. Finalmente, el ambiente de ejecución de la aplicación es donde la aplicación es usada para resolver los problemas. (17).

Para el desarrollo y definición de este trabajo se realizaron algunas modificaciones a este criterio planteado por Kaiser, ver figura 1.7

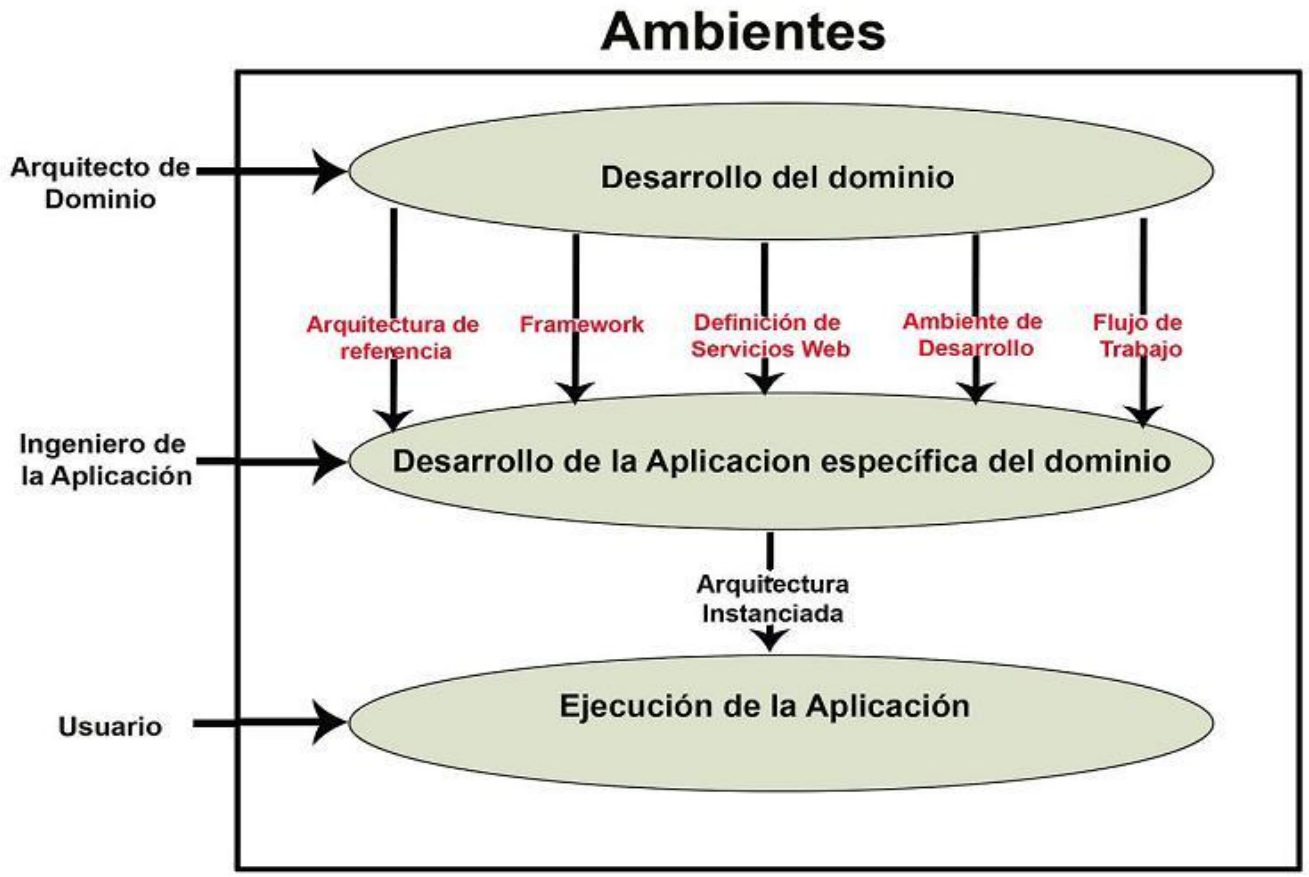


Figura 1.7 Modificaciones al concepto DSSA

Estas modificaciones consisten en reemplazar los términos: "componentes" por "framework" y "herramientas de desarrollo" por "ambiente de desarrollo". Además de agregar nuevos elementos: flujo de trabajo y definición de servicios Web.

El **flujo de trabajo** permite orientar el desarrollo de la aplicación como si fuera una guía con pasos a seguir de qué se debe hacer primero y qué después. Logrando de esta forma acortar el tiempo de desarrollo al máximo aprovechando todos los beneficios de la arquitectura de referencia definida y del framework a utilizar.

La **Definición de servicios Web** consiste en determinar cualquier componente de Software externo que brinde servicios, específicamente los Servicios Web. En base a las características y ventajas mencionadas para los Servicios Web se define si se utilizará algún servicio Web de terceras partes o si

se abordará el desarrollo de servicios Web como parte de las componentes de la aplicación y cómo van a interactuar con el framework utilizado.

1.8. Ambiente de desarrollo

El ambiente de desarrollo (Development Environment) es algo imprescindible en la producción de software. Es donde se definen el conjunto de herramientas y tecnologías (frameworks), versiones a usar y su integración, que intervienen en un proceso de desarrollo de software. (6)
A continuación se presentan un conjunto de herramientas utilizadas en el desarrollo de este trabajo como son: un ambiente de desarrollo integrado (IDE), un servidor web, un servidor para control de versiones, un sistema gestor de base de datos, herramientas de modelado, el framework y sus plugins, herramientas de diseño web. Se exponen sus características y ventajas para poder tener un mayor conocimiento de sus prestaciones y valorar su utilidad.

1.8.1. Ambiente de desarrollo integrado (IDE)

Un entorno de desarrollo integrado o Integrated Development Environment (IDE), en inglés, es un programa compuesto por un conjunto de herramientas para un programador. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien, poder utilizarse para varios.
Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI. Los IDEs pueden ser aplicaciones por si solas o pueden ser parte de aplicaciones existentes. El lenguaje Visual Basic por ejemplo puede ser usado dentro de las aplicaciones de Microsoft Office, lo que hace posible escribir sentencias Visual Basic en forma de macros para Microsoft Word.
Los IDEs proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, Java, C#, Delphi, Visual Basic, Object Pascal, etc. En algunos lenguajes, un IDE puede funcionar como un sistema en tiempo de ejecución, donde se permite utilizar el lenguaje de programación en forma interactiva, sin necesidad de trabajo orientado a archivos de texto, como es el caso de Smalltalk u Objective-C.
Es posible que un mismo IDE pueda funcionar con varios lenguajes de programación. Este es el caso de Eclipse, que mediante pluggins se le puede añadir soporte de lenguajes adicionales.

Zend Studio

Editor web orientado a la programación de páginas PHP, con ayudas en la gestión de proyectos y depuración de código. El programa, además de servir de editor de texto para páginas PHP,

proporciona una serie de ayudas que pasan desde la creación y gestión de proyectos hasta la depuración de código.

Consta de dos partes en las que se dividen las funcionalidades de parte del cliente y del servidor. Las dos partes se instalan por separado, la del cliente contiene la interfaz de edición y la ayuda. Permite además hacer depuraciones simples de scripts, aunque para disfrutar de toda la potencia de la herramienta de depuración habrá que disponer de la parte del servidor, que instala Apache y el módulo PHP o, en caso de que estén instalados, los configura para trabajar juntos en depuración.(10)

Cubre todas las típicas características de un IDE más un conjunto de características especiales para el entorno PHP con el objetivo de ayudar a la productividad del desarrollador de PHP. Esta lista muestra alguna de las características especiales de Zend Studio:

- SOAP: generación del WSDL a partir de la definición de clases
- PHP (incluyendo la evaluación de comentarios PHPDoc)
- SOAP: auto completado de código para clientes SOAP basado en un archivo WSDL (local o remoto)
- SCM: Soporte integrado para Subversión y CVS
- DB: Visor de la estructura de base de datos y de contenidos, cliente SQL para diferentes tipos de bases de datos
- Editor: Gestión de archivos vía FTP o SFTP para una edición remota rápida.

Zend Studio for Eclipse

Zend Studio para Eclipse IDE es de la próxima generación en la familia Zend Studio. Es la última versión del popular entorno de programación integrado (IDE). Diseñado para desarrolladores profesionales de PHP, esta nueva versión combina un IDE versátil y potente con las capacidades de expansión del ecosistema del proyecto Eclipse.

Dispone de un entorno mucho más flexible y profesional para controlar todo el ciclo de vida de un desarrollo.

Entre sus funcionalidades, destacaría las capacidades de refactorización del código fuente, funcionalidad que permite adecuar el comportamiento externo de una función/clase sin cambiar el funcionamiento interno, que junto a los nuevos wizards y capacidades de generación de código facilitarán el trabajo a los desarrolladores.

Desde el punto de vista de un IDE completo, disponer de un buen debugger local con la conexión a los servidores de desarrollo, junto a una política de trabajo en equipo y un sistema de control de versiones es posible manejar sin problemas proyectos complejos en PHP.

Características nuevas que incluye:

- Código refactoring.
- Generación de código y magos Archivo.
- Código de Cobertura.
- PHP Unit pruebas de apoyo.
- El acceso al ecosistema de plugins de Eclipse.
- Apoyar el desarrollo de múltiples idiomas.
- Mejora el Editor de PHP con el formato avanzado, para listas de tareas y problemas de vista.
- Mejora del soporte de JavaScript.
- Mejora de apoyo, incluyendo HTML, HTML WYSIWYG, Código plegables, arrastrar y soltar los componentes y más.
- Mejora de depuración y Perfiles con Path Mapping.
- Mejora de Zend Framework con el apoyo del Proyecto Framework, plantillas y código MVC.

PDT (PHP Development Tool)

El proyecto PDT PHP ofrece un marco de desarrollo para la plataforma Eclipse. Este proyecto de desarrollo abarca todos los componentes necesarios para desarrollar PHP y facilitar la extensibilidad. Se aprovecha de la actual Web Tools Project en la prestación de los desarrolladores de PHP con capacidades (13).

Entre las características principales se encuentran:

- Editor sensible al contexto, el cual provee de resaltamiento de código, asistente de código y autocompletado de código.
- Integración con el modelo del proyecto Eclipse, que permite para inspeccionar el uso de las vistas del contorno del fichero y del proyecto, así como la nueva vista PHP Explorer.
- Soporte para el debug incremental del código de PHP.
- Extensos frameworks y APIs que permiten a los desarrolladores e ISVs (vendedores de software independientes) fácilmente extender PDT para crear nuevas e interesantes herramientas orientadas al desarrollo de PHP.
- Perfecta integración con el proyecto Web Tools.
- La adhesión a las normas de Eclipse.
- Extensibilidad.

Aqua Data Studio

Es una herramienta multiplataforma de búsqueda y administración para bases de datos, escrita en Lenguaje de programación Java. Da la habilidad de crear, editar y ejecutar escrituras de SQL, y también examinar y modificar estructuras de bases de datos visualmente. Proporciona un ambiente integrado con todos los mayores Base de datos relacional como Microsoft SQL Server, MySQL, PostgreSQL, Oracle, DB2, Sybase e Informix, más conexiones con los genéricos ODBC y JDBC.

1.9. Framework

En el desarrollo de software, un framework es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje de scripting entre otros softwares para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Un framework representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo, la cual extiende o utiliza las aplicaciones del dominio.

Los frameworks son diseñados con el intento de facilitar el desarrollo de software, permitiendo a los diseñadores y programadores pasar más tiempo identificando requerimientos de software que tratando con los tediosos detalles de bajo nivel de proveer un sistema funcional. Sin embargo, hay quejas comunes acerca de que el uso de frameworks añade código innecesario y que la preponderancia de frameworks competitivos y complementarios significa que el tiempo que se pasaba programando y diseñando ahora se gasta en aprender a usar frameworks.

Fuera de las aplicaciones en la informática, un framework puede ser considerado como el conjunto de procesos y tecnologías usados para resolver un problema complejo. Es el esqueleto sobre el cual varios objetos son integrados para una solución dada.

También podemos definirlo en el contexto de la programación como un set de funciones o código genérico que realiza tareas comunes y frecuentes en todo tipo de aplicaciones (creación de objetos, conexión a base de datos, limpieza del código, etc.). Esto brinda una base sólida sobre la cual desarrollar aplicaciones concretas y permite obviar los componentes más triviales y genéricos del desarrollo.

En general, los frameworks son construidos en base a lenguajes orientados a objetos. Esto permite una mejor modularización de los componentes y óptima reutilización de código. Además, en la mayoría de los casos un framework implementará uno o más patrones de diseño de software que aseguren la escalabilidad del producto. Un patrón de diseño es un set de metodologías probadas para resolver

problemas comunes en el diseño de aplicaciones. Una convención, si se quiere, que facilita la comprensión de la arquitectura de la aplicación. (8)

No hay una definición común pero la mayoría tiene un tema común: la reutilización. Una definición ampliamente aceptada es dada por R. E. Johnson, and B. Foote en 1988 en su publicación “Designing Reusable Classes (R. E. JOHNSON, B. FOOTE 1988):

"Un framework es un conjunto de clases que personifican un diseño abstracto para soluciones de una familia de problemas relacionados..."

Otro punto de vista, según R. E. Johnson y V. F. Russo en “Reusing Object-Oriented Designs” incluye (R. E. JOHNSON, V. F. RUSSO)

“Una clase abstracta es un diseño para un objeto simple. Un framework es el diseño de un conjunto de objetos que colaboran para llevar a cabo un conjunto de responsabilidades. De esta manera los frameworks son diseños a escalas más grandes que las clases abstractas. Los frameworks son una forma de reutilizar el diseño a un nivel superior.” (6)

Los framework son una clase de arquitectura de dominio específico. La diferencia principal es que un framework es un diseño orientado a objeto mientras que una arquitectura de un dominio específico puede no serlo (Johnson, 1997).

1.10. Propel

Propel es un servicio de objeto persistente y de consulta -- lo que significa que Propel provee un sistema para almacenar objetos en una base de datos y un sistema para búsqueda y restauración de objetos desde una base de datos. Propel le permite realizar consultas complejas y manipulación de bases de datos sin escribir una sola cláusula SQL. Propel hace más fácil la escritura de aplicaciones, más fácil de desplegar, y mucho más fácil para migrar si alguna vez la situación lo amerita.

Propel puede ser descrito como un mapeado objeto-relacional, una capa DAO, o una capa objeto persistente. Propel es un puerto de Apache torque. Basado en acercamientos probados, desarrollados por el proyecto Torque y optimizados para PHP, Propel espera proporcionar un inteligente y comprensivo servicio de manejo de datos con un mínimo costo de realización para su aplicación en PHP. (9)

Para esos familiares con patrones O/R, Propel inicialmente implementa el patrón entrada de datos en fila, como lo describe Martin Fowler, para representar la base de datos. Por citar a Fowler:

Una entrada de datos de fila le da objetos que lucen exactamente como el registro en su estructura de registros pero puede ser accedido con los mecanismos regulares de su lenguaje de programación habitual. Todos los detalles de acceso de fuentes de datos están ocultos detrás de esta interfaz.

Sin embargo, Propel también genera las clases para cada tabla que exhibe algunas de las propiedades de la tabla del patrón datos de entrada:

Una tabla de entrada de datos almacena todo el SQL para acceder a una sola tabla o vista: selecciones, inserciones, actualizaciones, y eliminaciones. Otro código llama los métodos para todas las interacciones con la base de datos.

En Propel las clases de tabla de entrada de datos es llamada clases Peer, mientras las clases de filas de entrada de datos son llamadas entidad o clases objeto.

Como una aplicación, Propel tiene dos componentes principales (y ahora formalmente separados):

Un motor generador para construir sus clases y archivos SQL (generador-propel)

Un ambiente de ejecución que proporciona herramientas para construir consultas SQL, ejecutando consultas compiladas, y herramientas para el manejo de conexiones para múltiples bases de datos simultáneamente (propel)

El ambiente de ejecución proporciona una capa de abstracciones y encapsulación de bases de datos reglas lógicas de negocios. Las clases Propel representan la capa modelo del tradicional MVC, diseñado para encapsular cualquier nivel de validación de dato necesitado por su aplicación.

1.11. Symfony

Symfony es un completo framework diseñado para optimizar, gracias a sus características, el desarrollo de las aplicaciones Web. Para empezar, separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación Web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación Web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. El resultado de todas estas ventajas es que no se debe reinventar la rueda cada vez que se crea una nueva aplicación Web.

Symfony está desarrollado completamente con PHP 5. Ha sido probado en numerosos proyectos reales y se utiliza en sitios web de comercio electrónico de primer nivel. Symfony es compatible con la mayoría de gestores de bases de datos, como MySQL, PostgreSQL, Oracle y SQL Server de Microsoft. Se puede ejecutar tanto en plataformas *nix (Unix, Linux, etc.) como en plataformas Windows.

Características de Symfony:

Symfony se diseñó para que se ajustara a los siguientes requisitos:

- Fácil de instalar y configurar en la mayoría de plataformas (y con la garantía de que funciona correctamente en los sistemas Windows y *nix estándares)

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

Symfony automatiza la mayoría de elementos comunes de los proyectos web, como por ejemplo:

- La capa de internacionalización que incluye Symfony permite la traducción de los datos y de la interfaz, así como la adaptación local de los contenidos.
- La capa de presentación utiliza plantillas y layouts que pueden ser creados por diseñadores HTML sin ningún tipo de conocimiento del framework. Los helpers incluidos permiten minimizar el código utilizado en la presentación, ya que encapsulan grandes bloques de código en llamadas simples a funciones.
- Los formularios incluyen validación automatizada y relleno automático de datos ("repopulation"), lo que asegura la obtención de datos correctos y mejora la experiencia de usuario.
- Los datos incluyen mecanismos de escapes que permiten una mejor protección contra los ataques producidos por datos corruptos.
- La gestión de la caché reduce el ancho de banda utilizado y la carga del servidor.
- La autenticación y la gestión de credenciales simplifican la creación de secciones restringidas y la gestión de la seguridad de usuario.
- El sistema de enrutamiento y las URL limpias permiten considerar a las direcciones de las páginas como parte de la interfaz, además de estar optimizadas para los buscadores.
- El soporte de e-mail incluido y la gestión de APIs permiten a las aplicaciones web interactuar más allá de los navegadores.

- Los listados son más fáciles de utilizar debido a la paginación automatizada, el filtrado y la ordenación de datos.
- Los plugins, las factorías (patrón de diseño “Factory”) y los “mixin” permiten realizar extensiones a medida de Symfony.
- Las interacciones con Ajax son muy fáciles de implementar mediante los helpers que permiten encapsular los efectos JavaScript compatibles con todos los navegadores en una única línea de código.
- Entorno de desarrollo y herramientas
- Symfony puede ser completamente personalizado para cumplir con los requisitos de las empresas que disponen de sus propias políticas y reglas para la gestión de proyectos y la programación de aplicaciones. Por defecto incorpora varios entornos de desarrollo diferentes e incluye varias herramientas que permiten automatizar las tareas más comunes de la ingeniería del software.
- Las herramientas que generan código automáticamente han sido diseñadas para hacer prototipos de aplicaciones y para crear fácilmente la parte de gestión de las aplicaciones.
- El framework de desarrollo de pruebas unitarias y funcionales proporciona las herramientas ideales para el desarrollo basado en pruebas (“test-driven development”).
- La barra de depuración web simplifica la depuración de las aplicaciones, ya que muestra toda la información que los programadores necesitan sobre la página en la que están trabajando.
- La interfaz de línea de comandos automatiza la instalación de las aplicaciones entre servidores.
- Es posible realizar cambios “en caliente” de la configuración (sin necesidad de reiniciar el servidor).
- El completo sistema de log permite a los administradores acceder hasta el último detalle de las actividades que realiza la aplicación. (10)

ExtJS

Es una librería Javascript que hace de puente a las librerías de Yahoo!, jQuery y Prototype+Scriptaculous para ofrecernos de forma sencilla componentes de interfaz grafica de usuarios en nuestras aplicaciones cliente. Entre los componentes que nos ofrece encontramos diálogos, menús, tablas, layouts, paneles, pestañas y mucho más.

1.12. Lenguajes de Modelado

El lenguaje de modelado de objetos es un conjunto estandarizado de símbolos y de modos de disponerlos para modelar (parte de) un diseño de software orientado a objetos.
Algunas organizaciones los usan extensivamente en combinación con una metodología de desarrollo de software para avanzar de una especificación inicial a un plan de implementación y para comunicar dicho plan a todo un equipo de desarrolladores. El uso de un lenguaje de modelado es más sencillo que la auténtica programación, pues existen menos medios para verificar efectivamente el funcionamiento adecuado del modelo. Esto puede suponer también que las interacciones entre partes del programa den lugar a sorpresas cuando el modelo ha sido convertido en un software funcionante.

1.12.1. Visual Paradigm

Visual Paradigm para UML (Lenguaje de Modelado) es una herramienta CASE que utiliza “UML” como lenguaje de modelaje, soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor costo. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación.
Visual Paradigm ofrece:

- Entorno de creación de diagramas para UML 2.0
- Diseño centrado en casos de uso y enfocado al negocio que genera un software de mayor calidad.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Capacidades de ingeniería directa (versión profesional) e inversa.
- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- Disponibilidad de múltiples versiones, para cada necesidad.
- Disponibilidad de integrarse en los principales IDEs.
- Disponibilidad en múltiples plataformas

1.13. Servidor Web

Un servidor web es un programa que implementa el *protocolo HTTP* (*hypertext transfer protocol*). Este protocolo está diseñado para transferir lo que llamamos hipertextos, páginas web o páginas HTML (hypertext markup language): textos complejos con enlaces, figuras, formularios, botones y objetos incrustados como animaciones o reproductores de música.

Un servidor web es un ordenador que usa el protocolo http para enviar páginas web al ordenador de un usuario cuando el usuario las solicita a través de un navegador. Los servidores web, servidores de correo y servidores de bases de datos son a lo que tiene acceso la mayoría de la gente al usar Internet.

1.13.1. Servidor Apache

El servidor HTTP Apache es un software (libre) servidor HTTP de código abierto para plataformas Unix (BSD, GNU/Linux, etc.), Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual. Cuando comenzó su desarrollo en 1995 se basó inicialmente en el código del popular NCSA HTTPd 1.3, pero más tarde fue reescrito por completo.

Apache presenta entre otras características mensajes de error altamente configurables, bases de datos de autenticación y negociado de contenido.

Ventajas:

- Modular
- Open source
- Multi-plataforma
- Extensible
- Popular (fácil conseguir ayuda/soporte)
- Gratuito

1.14. SGBD

Un Sistema Gestión de Bases de Datos –SGBD- (Data Base Management System DBMS) consiste en una colección de datos interrelacionados y un conjunto de programas para acceder a esos datos. El objetivo primordial de un SGBD es proporcionar un entorno que sea a la vez conveniente y eficiente para ser utilizado al extraer y almacenar información de la base de datos.

“El sistema de gestión de la base de datos (SGBD) es una aplicación que permite a los usuarios definir, crear y mantener la base de datos, y proporciona acceso controlado a la misma. (12)

1.14.1. PostgreSQL

PostgreSQL es un Sistema de Gestión de Bases de Datos Objeto-Relacionales (ORDBMS), de código abierto desde su inicio y con una activa y dedicada comunidad de desarrolladores. Ofrece una potencia adicional sustancial al incorporar los siguientes cuatro conceptos adicionales básicos en una vía en la que los usuarios pueden extender fácilmente el sistema: clases, herencia, tipos, funciones. Otras

características aportan potencia y flexibilidad adicional: Restricciones (Constraints), Disparadores (triggers), Reglas (rules), Integridad transaccional. Debido a eso se considera la base de datos de código abierto más avanzada hoy día disponible, posibilitando además control de concurrencia multi-versión, soportando casi toda la sintaxis SQL y contando también con un amplio conjunto de enlaces con lenguajes de programación (incluyendo C, C++, PHP, Java, perl, tcl y python).

1.15. Control de versiones

Se llama control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Los sistemas de control de versiones facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas (por ejemplo, para algún cliente específico).

El control de versiones se realiza principalmente en la industria informática para controlar las distintas versiones del código fuente. Sin embargo, los mismos conceptos son aplicables a otros ámbitos como documentos, imágenes, sitios web, etcétera. Aunque un sistema de control de versiones puede realizarse de forma manual, es muy aconsejable disponer de herramientas que faciliten esta gestión (CVS, Subversion, SourceSafe, ClearCase, Darcs, Plastic SCM, Git, Mercurial, etc.

1.15.1. Subversion

Subversion es un sistema de control de versiones libre y de código fuente abierto, es decir, maneja ficheros y directorios a través del tiempo. Hay un árbol de ficheros en un repositorio central. El repositorio es como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios hechos a sus ficheros y directorios. Esto le permite recuperar versiones antiguas de sus datos, o examinar el historial de cambios de los mismos. En este aspecto, mucha gente piensa en los sistemas de versiones como en una especie de “máquina del tiempo”. Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores. A cierto nivel, la capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración. Se puede progresar más rápidamente sin un único conducto por el cual deban pasar todas las modificaciones. Y puesto que el trabajo se encuentra bajo el control de versiones, no hay razón para temer que la calidad del mismo vaya a verse afectada por la pérdida de ese conducto único—si se ha hecho un cambio incorrecto a los datos, simplemente deshaga ese cambio. (11)

Algunos sistemas de control de versiones son también sistemas de administración de configuración de software. Estos sistemas son diseñados específicamente para la administración de árboles de código

fuente, y tienen muchas características que son específicas del desarrollo de software— tales como el entendimiento nativo de lenguajes de programación, o el suministro de herramientas para la construcción de software. Sin embargo, Subversión no es uno de estos sistemas. Subversion es un sistema general que puede ser usado para administrar cualquier conjunto de ficheros. Para usted, esos ficheros pueden ser código fuente— para otros, cualquier cosa desde la lista de la compra de comestibles hasta combinaciones de vídeo digital y más allá. (11)

Rapid SVN y KDESVN

Clientes gráficos para el Subversion que permiten su integración con el sistema en los distintos Sistemas Operativos Linux.

TortoiseSVN

Es un cliente gratuito de código abierto para el sistema de control de versiones Subversion. Maneja ficheros y directorios a lo largo del tiempo. Los ficheros se almacenan en un repositorio central. El repositorio es prácticamente lo mismo que un servidor de ficheros ordinario, salvo que recuerda todos los cambios que se hayan hecho a sus ficheros y directorios. Esto permite que pueda recuperar versiones antiguas de sus ficheros y examinar la historia de cuándo y cómo cambiaron sus datos, y quién hizo el cambio. Es fácil de usar, permite ver el estado de los archivos desde en el explorador de Windows y permite también crear gráficos de todas las revisiones asignadas.

1.16. Conclusiones

Con el análisis desarrollado en este capítulo se puede llegar a la conclusión, de que para definir nuestra DSSA, es necesario, un ambiente de desarrollo, una arquitectura de referencia, un framework, la definición de servicios web y un flujo de trabajo para guiar el desarrollo de la aplicación, garantizando el uso de patrones de diseño y buenas prácticas. Se ha llegado a la definición más práctica de qué elementos se tienen que tener en cuenta a la hora de realizar una arquitectura de software de dominio específico, lo cual se pondrá en práctica en el desarrollo de los siguientes capítulos.

2. CAPÍTULO 2: DOMINIO Y ARQUITECTURA

2.1. Introducción:

En al capítulo anterior se ha abordaron los elementos que se deben tener en cuenta para desarrollar una DSSA. En este definiremos cual sería nuestro dominio específico y la arquitectura de referencia, utilizando las definiciones anteriormente analizadas. La DSSA que desarrollaremos permitirá estructurar y organizar el desarrollo de aplicaciones empresariales en PHP que la usen haciendo buenas prácticas y patrones.

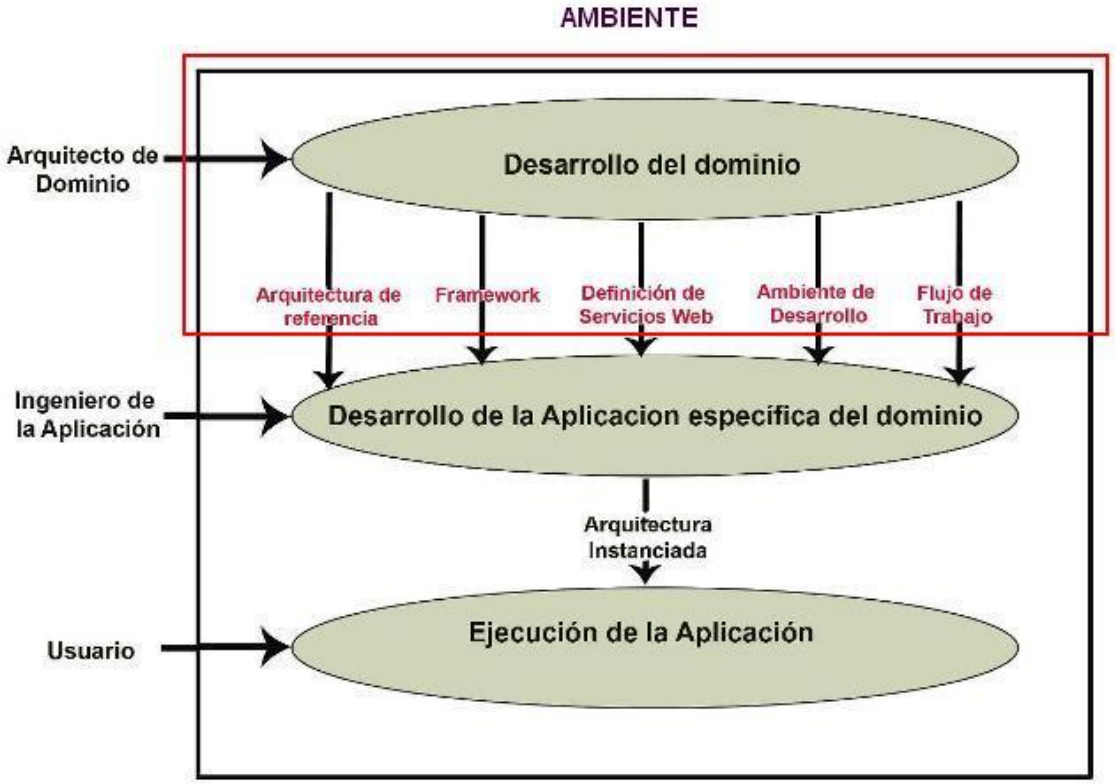


Fig. 2.1 Marco de Trabajo

2.2. Definición del dominio:

Para definir el dominio para las aplicaciones empresariales en PHP se analizarán e identificarán los problemas comunes entre las aplicaciones del mismo.

Nuestro dominio se enmarca en todas las aplicaciones web desarrolladas en PHP, específicamente las que utilicen Propel para la persistencia, que sean capaces utilizar o brindar servicios para dar soporte a una futura arquitectura SOA.

2.3. Requerimientos de referencia del dominio

Muchas aplicaciones se han desarrollado en el ámbito correspondiente al dominio definido anteriormente. En diferentes áreas de estas aplicaciones han surgido problemas similares. Estos convergen en requerimientos comunes para toda aplicación en este dominio y son expuestos a continuación:

- *Seguridad:* En la mayoría de las aplicaciones es necesario manejar los requerimientos de Autenticidad, Integridad, Confidencialidad de la información.
- *Auditoría:* Es necesario tener registrado qué pasa en la aplicación constantemente y quién es el responsable de cada acción realizada.
- *Almacenamiento de datos en Bases de Datos:* Se necesita guardar los datos de la aplicación en bases de datos. Es necesario que este almacenamiento sea óptimo por lo que se deben manipular de forma correcta las conexiones a la base de datos, sean persistentes o no.
- *Manejo de Transacciones:* Se necesitan las operaciones sean transaccionales y en muchos casos éstas se conforman por más de una petición del cliente.
- *Capa de presentación flexible y rica en estilos:* La capa de presentación debe cumplir esta característica para soportar las exigencias de presentación para los distintos tipos de clientes que puede presentarse en una misma aplicación y las mismas pueden sufrir cambios.
- *Alto rendimiento de procesamiento de las peticiones del cliente:* Es necesario que las peticiones del cliente se respondan en el menor tiempo posible por lo que es necesario que todas las capas de la aplicación interactúen de la manera más eficiente posible.
- *Interacción con aplicaciones externas mediante servicios web*: Es muy común la interacción con sistemas externos por lo que se hace necesario que existan estándares en el uso de las tecnologías necesarias para cumplir este requerimiento.
- *Administración de los aspectos configurables de la aplicación:* Toda aplicación necesita ser configurable tanto en la etapa de despliegue como en tiempo de desarrollo de la manera más simple posible.

2.4. Arquitectura de Referencia

La arquitectura de referencia permite orientar el diseño de las capas lógicas a través de una propuesta de diseño base; brinda una estructura física para soportar el código, creando así un esqueleto base,

definido por el framework a utilizar y sobre el cual se va a trabajar. Está basada fundamentalmente en los estilos arquitectónicos Cliente-servidor, Arquitectura en capas y el patrón MVC.

Diseño de las capas lógicas

Dado que optamos por utilizar el framework Symfony por las características y ventajas expuestas en el capítulo anterior, utilizaremos la estructura de diseño de las capas lógicas y organización del código que nos propone dicho framework para el desarrollo de las aplicaciones Web.

Estructura del diseño de las capas lógicas

Symfony toma lo mejor de la arquitectura MVC y la implementa de forma que el desarrollo de aplicaciones sea rápido y sencillo.

En primer lugar, el controlador frontal y el layout son comunes para todas las acciones de la aplicación. Se pueden tener varios controladores y varios layouts, pero solamente es obligatorio tener uno de cada uno. El controlador frontal es un componente que sólo tiene código relativo al MVC, por lo que no es necesario crear uno ya que Symfony lo genera de forma automática. Las clases de la capa del modelo también se generan automáticamente, en función de la estructura de datos de la aplicación. La librería Propel se encarga de esta generación automática, ya que crea el *esqueleto* o estructura básica de las clases y genera automáticamente el código necesario.

La abstracción de la base de datos es completamente invisible al programador, ya que la realiza otro componente específico llamado Creole. Así, si se cambia el sistema gestor de bases de datos en cualquier momento, no se debe reescribir ni una línea de código, ya que tan sólo es necesario modificar un parámetro en un archivo de configuración.

Por último, la lógica de la vista se puede transformar en un archivo de configuración sencillo, sin necesidad de programarla.

La estructura del diseño de las capas lógicas se representa en la figura 2.2

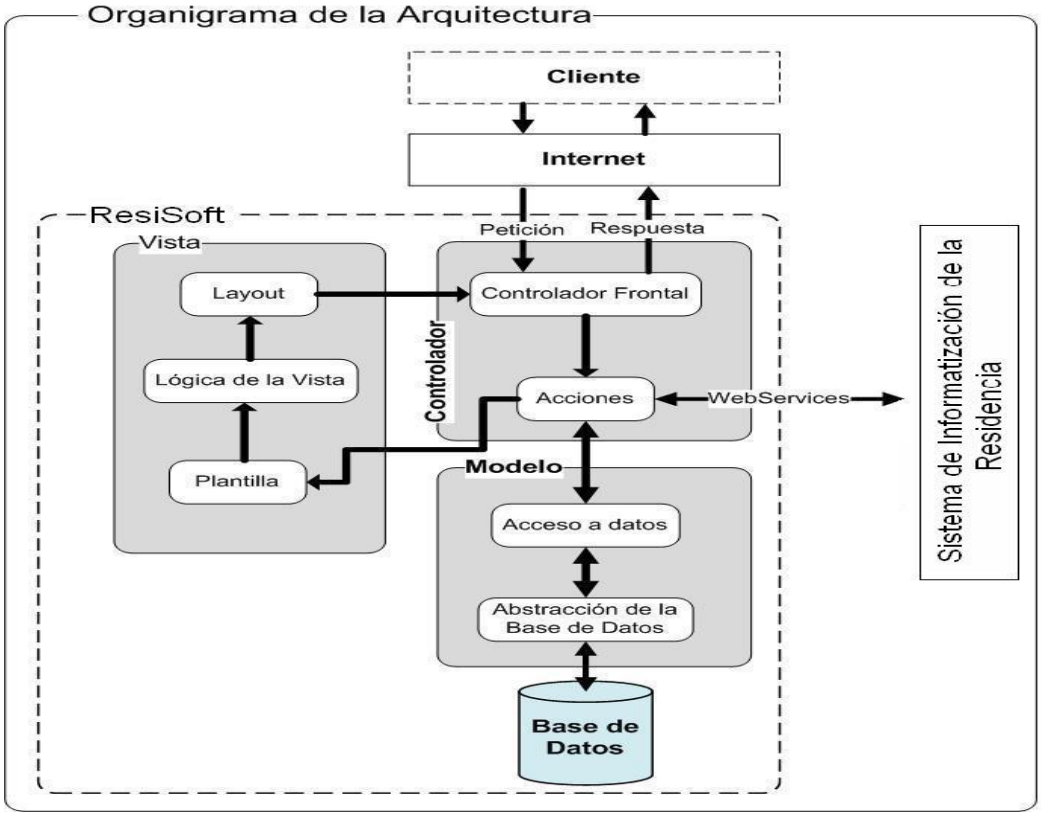


Figura 2.2 Estructura del diseño de las capas lógicas

2.4.1. Estructura Interna de un proyecto

Dentro de un proyecto, las operaciones se agrupan de forma lógica en aplicaciones. Normalmente, una aplicación se ejecuta de forma independiente respecto de otras aplicaciones del mismo proyecto. Lo habitual es que un proyecto contenga dos aplicaciones: una para la parte pública y otra para la parte de gestión, compartiendo ambas la misma base de datos. También es posible definir proyectos que estén formados por varios sitios Web pequeños, cada uno de ellos considerado como una aplicación.

Cada aplicación está formada por uno o más módulos. Un módulo normalmente representa a una página Web o a un grupo de páginas con un propósito relacionado.

Los módulos almacenan las acciones, que representan cada una de las operaciones que se pueden realizar en un módulo.

Normalmente las acciones se describen mediante verbos. Trabajar con acciones es muy similar a trabajar con las páginas de una aplicación Web tradicional, aunque en este caso dos acciones diferentes pueden acabar mostrando la misma página.

Estructura interna de un proyecto

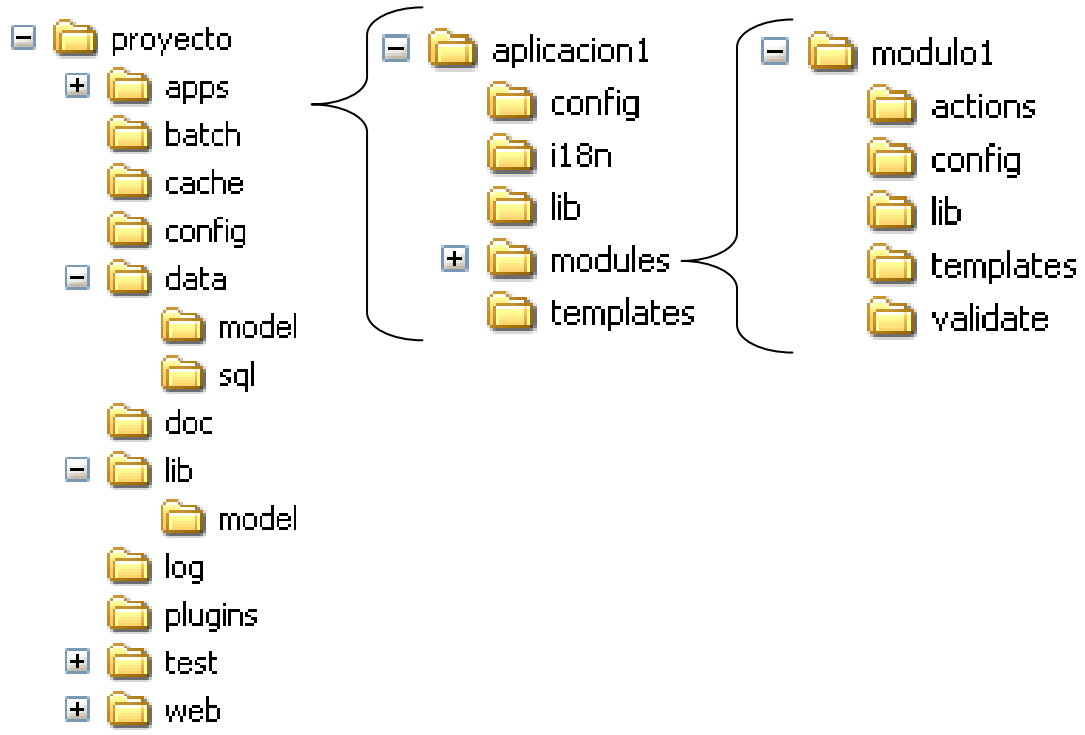


Figura 2.3 Estructura interna de un proyecto

Estructura de la raíz del proyecto

La raíz de cualquier proyecto Symfony contiene los siguientes directorios:

```
apps/
     frontend/
     backend/
batch/
cache/
config/
data/
     sql/
doc/
lib/
    model/
log/
```

```
plugins/
test/
    unit/
    functional/
web/
    css/
    images/
    js/
uploads/
```

Tabla 2-1. Directorios en la raíz de los proyectos Symfony

Directorio	Descripción
apps/	Contiene un directorio por cada aplicación del proyecto (normalmente, frontend (frontal) y backend (trasera) para la parte pública y la parte de gestión respectivamente)
batch/	Contiene los scripts de PHP que se ejecutan mediante la línea de comandos o mediante la programación de tareas para realizar procesos en lotes (*batch processes*)
cache/	Contiene la versión cacheada de la configuración y (si está activada) la versión cacheada de las acciones y plantillas del proyecto. El mecanismo de cache (que se explica en el Capítulo 12) utiliza los archivos de este directorio para acelerar la respuesta a las peticiones web. Cada aplicación contiene un subdirectorio que guarda todos los archivos PHP y HTML preprocesados.
config/	Almacena la configuración general del proyecto
data/	En este directorio se almacenan los archivos relacionados con los datos, como por ejemplo el esquema de una base de datos, el archivo que contiene las instrucciones SQL para crear las tablas e incluso un archivo de bases de datos de SQLite.
doc/	Contiene la documentación del proyecto, formada por tus propios documentos y por la documentación generada por PHPdoc
lib/	Almacena las clases y librerías externas. Se suele guardar todo el código común a todas las aplicaciones del proyecto. El subdirectorio model/ guarda el modelo de objetos del proyecto
log/	Guarda todos los archivos de log generados por Symfony, también se puede utilizar para guardar los logs del servidor web, de la base de datos o de cualquier otro componente

	del proyecto. Symfony crea un archivo de log por cada aplicación y por cada entorno
plugins/	Almacena los plugins instalados en la aplicación
test/	Contiene las pruebas unitarias y funcionales escritas en PHP y compatibles con el framework de pruebas de Symfony. Cuando se crea un proyecto, Symfony crea algunas pruebas básicas.
Web/	La raíz del servidor web. Los únicos archivos accesibles desde Internet son los que se encuentran en este directorio.

Estructura de cada aplicación

Todas las aplicaciones de Symfony tienen la misma estructura de archivos y directorios:

```
apps/
  [nombre aplicacion]/
     config/
     i18n/
     lib/
     modules/
     templates/
        layout.php
        error.php
        error.txt
```

La tabla 2-2 describe los subdirectorios de una aplicación.

Tabla 2-2. Subdirectorios de cada aplicación Symfony

Directorio	Descripción
config/	Contiene un montón de archivos de configuración creados con YAML. Aquí se almacena la mayor parte de la configuración de la aplicación, salvo los parámetros propios del framework. También es posible redefinir en este directorio los parámetros por defecto si es necesario.

i18n/	Contiene todos los archivos utilizados para la internacionalización de la aplicación, sobre todo los archivos que traducen la interfaz. La internacionalización también se puede realizar con una base de datos, en cuyo caso este directorio no se utilizaría
lib/	Contiene las clases y librerías utilizadas exclusivamente por la aplicación.
modules/	Almacena los módulos que definen las características de la aplicación
templates/	Contiene las plantillas globales de la aplicación, es decir, las que utilizan todos los módulos. Por defecto contiene un archivo llamado layout.php, que es el layout principal con el que se muestran las plantillas de los módulos

Estructura de cada módulo

Cada aplicación contiene uno o más módulos. Cada módulo tiene su propio subdirectorio dentro del directorio *modules* y el nombre del directorio es el que se elige durante la creación del módulo.

Esta es la estructura de directorios típica de un módulo:

```
apps/
  [nombre aplicacion]/
    modules/
      [nombre modulo]/
          actions/
            actions.class.php
          config/
          lib/
          templates/
            indexSuccess.php
          validate/
```

La tabla 2-3 describe los subdirectorios de un módulo.

Tabla 2-3. Subdirectorios de cada módulo

Directorio	Descripción
actions/	Normalmente contiene un único archivo llamado actions.class.php y que corresponde a la clase que almacena todas las acciones del módulo. También es posible crear un archivo diferente para cada acción del módulo
config/	Puede contener archivos de configuración adicionales con parámetros exclusivos del módulo.
lib/	Almacena las clases y librerías utilizadas exclusivamente por el módulo.
templates/	Contiene las plantillas correspondientes a las acciones del módulo. Cuando se crea un nuevo módulo, automáticamente se crea la plantilla llamada indexSuccess.php.
validate/	Contiene archivos de configuración relacionados con la validación de formularios.

Estructura externa del proyecto

Existen pocas restricciones sobre la estructura del directorio Web, que es el directorio que contiene los archivos que se pueden acceder de forma pública. Si se utilizan algunas convenciones básicas en los nombres de los subdirectorios, se pueden simplificar las plantillas.

La siguiente es una estructura típica del directorio Web:

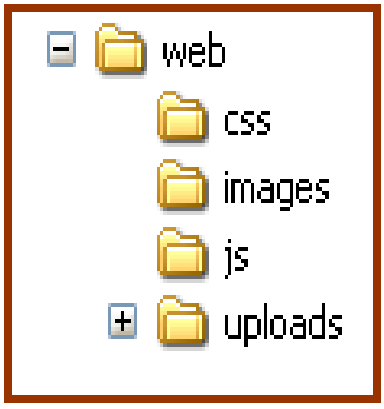


Figura 2.4 Estructura típica del directorio Web

Tabla 2.4. Subdirectorios habituales en la carpeta web

Directorio	Descripción
css/	Contiene los archivos de hojas de estilo creados con CSS (archivos con extensión .css.

images/	Contiene las imágenes del sitio con formato .jpg, .png o .gif
js/	Contiene los archivos de JavaScript con extensión .js
uploads/	Se almacenan los archivos subidos por los usuarios. Aunque normalmente este directorio contiene imágenes, no se debe confundir con el directorio que almacena las imágenes del sitio (images/). Esta distinción permite sincronizar los servidores de desarrollo y de producción sin afectar a las imágenes subidas por los usuarios.

2.5. El Controlador

La capa del controlador, que contiene el código que liga la lógica de negocio con la presentación, está dividida en varios componentes que se utilizan para diversos propósitos:

- El controlador frontal es el único punto de entrada a la aplicación. Carga la configuración y determina la acción a ejecutarse.
- Las acciones contienen la lógica de la aplicación. Verifican la integridad de las peticiones y preparan los datos requeridos por la capa de presentación.
- Los objetos request, response y session dan acceso a los parámetros de la petición, las cabeceras de las respuestas y a los datos persistentes del usuario. Se utilizan muy a menudo en la capa del controlador.
- Los filtros son trozos de código ejecutados para cada petición, antes o después de una acción. Por ejemplo, los filtros de seguridad y validación son comúnmente utilizados en aplicaciones web. Puedes extender el framework creando tus propios filtros.

2.5.1. El Controlador Frontal

Todas las peticiones web son manejadas por un solo *controlador frontal*, que es el punto de entrada único de toda la aplicación en un entorno determinado.

Cuando el controlador frontal recibe una petición, utiliza el sistema de enrutamiento para asociar el nombre de una acción y el nombre de un módulo con la URL escrita (o pinchada) por el usuario.

2.5.1.1 El Trabajo del Controlador Frontal en Detalle

El controlador frontal se encarga de despachar las peticiones, lo que implica algo más que detectar la acción que se ejecuta. De hecho, ejecuta el código común a todas las acciones, incluyendo:

1. Define las constantes del núcleo.

2. Localiza la librería de Symfony
3. Carga e inicializa las clases del núcleo del framework.
4. Carga la configuración.
5. Decodifica la URL de la petición para determinar la acción a ejecutar y los parámetros de la petición.
6. Si la acción no existe, redireccionará a la acción del error 404.
7. Activa los filtros (por ejemplo, si la petición necesita autenticación).
8. Ejecuta los filtros, primera pasada.
9. Ejecuta la acción y produce la vista.
10. Ejecuta los filtros, segunda pasada.
11. Muestra la respuesta.

2.6. Acciones

Las acciones son el corazón de la aplicación, puesto que contienen toda la lógica de la aplicación. Las acciones utilizan el modelo y definen variables para la vista. Cuando se realiza una petición web en una aplicación Symfony, la URL define una acción y los parámetros de la petición.

Una acción puede elegir la forma en la que se ejecuta su vista, devolviendo un valor correspondiente a una de las constantes de la clase sfView. Dentro de una acción, se pueden manipular los diferentes elementos del contexto, incluidos el objeto de la petición (sfRequest) y el objeto de la sesión del usuario actual (sfUser).

Para los casos en los que es necesario realizar un grupo de procedimientos antes o después de la llamada de todas las acciones, las clases *sfActions* y *sfAction* permiten el uso de los métodos *preExecute* y *postExecute* respectivamente. Esto ayuda a la reutilización del código que sea común para un gran grupo de acciones en el caso de las clases que hereden de *sfActions* y para el caso de las acciones que hereden de *sfAction* el código que se desee sea ejecutado justo antes o después de la llamada de la acción.

2.7. La Vista

La vista se encarga de producir las páginas que se muestran como resultado de las acciones. La vista en Symfony está compuesta por diversas partes, estando cada una de ellas especialmente preparada para que pueda ser fácilmente modificable por la persona que normalmente trabaja con cada aspecto del diseño de las aplicaciones.

- Los diseñadores web normalmente trabajan con las plantillas (que son la presentación de los datos de la acción que se está ejecutando) y con el layout (que contiene el código HTML común a todas las páginas). Estas partes están formadas por código HTML que contiene pequeños trozos de código PHP, que normalmente son llamadas a los diversos *helpers* disponibles.
- Para mejorar la reutilización de código, los programadores suelen extraer trozos de las plantillas y los transforman en componentes y elementos parciales. De esta forma, el layout se modifica para definir zonas en las que se insertan componentes externos. Los diseñadores web también pueden trabajar fácilmente con estos trozos de plantillas.
- Los programadores normalmente centran su trabajo relativo a la vista en los archivos de configuración YAML (que permiten establecer opciones para las propiedades de la respuesta y para otros elementos de la interfaz) y en el objeto respuesta. Cuando se trabaja con variables en las plantillas, deben considerarse los posibles riesgos de seguridad de XSS (*cross-site scripting*) por lo que es necesario conocer las técnicas de escape de los caracteres introducidos por los usuarios.
- Independientemente del tipo de trabajo, existen herramientas y utilidades para simplificar y acelerar el trabajo normalmente tedioso de presentar los resultados de las acciones.

2.7.1. Helpers

Los helpers son funciones de PHP que devuelven código HTML y que se utilizan en las plantillas.

2.7.2. Layout de las páginas

El layout es un archivo, que también se denomina *plantilla global*, almacena el código HTML que es común a todas las páginas de la aplicación, para no tener que repetirlo en cada página. El contenido de la plantilla se integra en el layout, o si se mira desde el otro punto de vista, el layout *decora* la plantilla.

La *plantilla global* puede ser adaptada completamente para cada aplicación. Se puede añadir todo el código HTML que sea necesario. Normalmente se utiliza el layout para mostrar la navegación, el logotipo del sitio, etc. Incluso es posible definir más de un layout y decidir en cada acción el layout a utilizar.

Figura 2.5 Plantilla decorada con un layout

2.8. Elementos parciales

Un elemento parcial es un trozo de código de plantilla que se puede reutilizar. Por ejemplo, en una aplicación de publicación, el código de plantilla que se encarga de mostrar un artículo se utiliza en la página de detalle del artículo, en la página que lista los mejores artículos y en la página que muestra los últimos artículos. Se trata de un código perfecto para definirlo como elemento parcial.

Al igual que las plantillas, los elementos parciales son archivos que se encuentran en el directorio templates/, y que contienen código HTML y código PHP. El nombre del archivo de un elemento parcial siempre comienza con un guión bajo (_), lo que permite distinguir a los elementos parciales de las plantillas, ya que todos se encuentran en el mismo directorio templates/.

2.8.1. Componentes

Un componente es como una acción, solo que mucho más rápido. La lógica del componente se guarda en una clase que hereda de sfComponents y que se debe guardar en el archivo action/components.class.php. Su presentación se guarda en un elemento parcial.

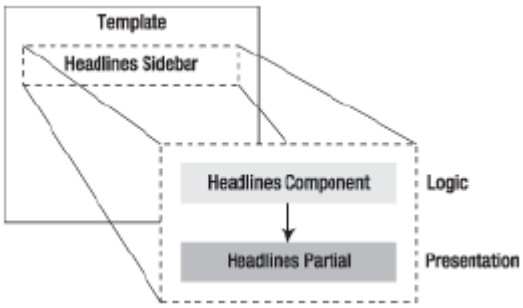


Figura 2.6 Uso de componentes en las plantillas

2.8.2. Slots

Un slot es una zona que se puede definir en cualquier elemento de la vista (layout, plantilla o elemento parcial). La forma de rellenar esa zona es similar a establecer el valor de una variable. El código de relleno se almacena de forma global en la respuesta, por lo que se puede definir en cualquier sitio

(layout, plantilla o elemento parcial). Se debe definir un slot antes de utilizarlo y también hay que tener en cuenta que el layout se ejecuta después de la plantilla (durante el proceso de *decoración*) y que los elementos parciales se ejecutan cuando los llama una plantilla.

2.9. Configuración de la vista

La vista está formada por dos partes:

- La presentación HTML del resultado de la acción (que se guarda en la plantilla, en el layout y en los fragmentos de plantilla)
- El resto, que incluye entre otros los siguientes elementos:
  - Declaraciones <meta>: palabras clave (keywords), descripción (description), duración de la cache, etc.
  - El título de la página: no solo es útil para los usuarios que tienen abiertas varias ventanas del navegador, sino que también es muy importante para que los buscadores indexen bien la página.
  - Inclusión de archivos: de JavaScript y de hojas de estilos.
  - Layout: algunas acciones necesitan un layout personalizado (ventanas emergentes, anuncios, etc.) o puede que no necesiten cargar ningún layout (por ejemplo en las acciones relacionadas con Ajax).

En la vista, todo lo que no es HTML se considera configuración de la propia vista y Symfony permite dos formas de manipular esa configuración. La forma habitual es mediante el archivo de configuración view.yml. Se utiliza cuando los valores de configuración no dependen del contexto o de alguna consulta a la base de datos. Cuando se trabaja con valores dinámicos que cambian con cada acción, se recurre al segundo método para establecer la configuración de la vista: añadir los atributos directamente en el objeto sfResponse durante la acción.

2.9.1. El archivo view.yml

Cada módulo contiene un archivo view.yml que define las opciones de su propia vista. De esta forma, es posible definir en un único archivo las opciones de la vista para todo el módulo entero y las opciones para cada vista. Las claves de primer nivel en el archivo view.yml son el nombre de cada módulo que se configura.

2.10. Slots de componentes

Si se combina el poder de los componentes que se han visto anteriormente y las opciones de configuración de la vista, se consigue un modelo de desarrollo de la vista completamente nuevo: el sistema de slots de componentes. Se trata de una alternativa a los *slots* que se centra en la reutilización y en la separación en capas. De esta forma, los slots de componentes están mucho más estructurados que los *slots*, pero son un poco más lentos de ejecutar.

Al igual que los *slots*, los slots de componentes son zonas que se pueden definir en los elementos de la vista. La principal diferencia reside en la forma en la que se decide qué código rellena esas zonas. En un *slot* normal, el código se establece en otro elemento de la vista; en un slot de componentes, el código es el resultado de la ejecución de un componente, y el nombre de ese componente se obtiene de la configuración de la vista.

Symfony - ExtJS

ExtJS nos permite desarrollar presentaciones basadas en Ajax y JSON de forma fácil e interactiva proporcionándole a nuestra aplicación un ambiente cómodo y agradable. Para lograr la interacción de los dos frameworks, utilizamos las ventajas anteriormente descritas de Symfony para configurar su presentación.

Para ello agregamos dos directorios a la parte pública del proyecto dentro de la carpeta */web/js/ExtJS y /web/css/ExtJS,* en los cuales se guardarán las librerías ExtJS, los fichero .js de la presentación de cada módulo y sus hojas de estilos CSS respectivamente.

Figura 2.7 *Symfony - ExtJS*

Luego, para que las mismas sean cargadas automáticamente en todas las páginas de la presentación, configuramos el archivo *viw.yml.*

En la aplicación:

```
  stylesheets:    [ext/resources/css/ext-all] #Librería css de ExtJS

  javascripts:    [ext/ext-base, ext/ext-all] #Librerías base de ExtJS
```

En cada módulo

```
indisciplinaSuccess:                          #Plantilla
  javascripts:    [Trab_Educ_Indisciplina] #Presentación ExtJS
                                           #asociada a la plantilla
```

De esta forma cada página de la aplicación cargará la librería ExtJS con sus CSS en el layout y los archivos .js con la presentación de cada módulo en la plantilla.

2.11. El Modelo

ORM

Las bases de datos siguen una estructura relacional. PHP 5 y Symfony, por el contrario, son orientados a objetos. Por este motivo, para acceder a la base de datos como si fuera orientada a objetos, es necesaria una interfaz que traduzca la lógica de los objetos a la lógica relacional. Esta interfaz se denomina *"mapeo de objetos a bases de datos"* (ORM, de sus siglas en inglés *"object-relational mapping"*). Un ORM consiste en una serie de objetos que permiten acceder a los datos y que contienen en su interior cierta lógica de negocio. Una de las ventajas de utilizar estas capas de abstracción de objetos/relacional es que evita utilizar una sintaxis específica de un sistema de bases de datos concreto. Esta capa transforma automáticamente las llamadas a los objetos en consultas SQL optimizadas para el sistema gestor de bases de datos que se está utilizando en cada momento.

La lógica de negocio de las aplicaciones web depende casi siempre en su modelo de datos. El componente que se encarga por defecto de gestionar el modelo en Symfony es una capa de tipo ORM realizada mediante el proyecto Propel. En las aplicaciones Symfony, el acceso y la modificación de los datos almacenados en la base de datos se realiza mediante objetos; de esta forma nunca se accede

de forma explícita a la base de datos. Este comportamiento permite un alto nivel de abstracción y permite una fácil portabilidad.

La capa del modelo es la más compleja del framework Symfony. Una de las razones de esta complejidad es que la manipulación de datos es una tarea bastante intrincada. Las consideraciones de seguridad relacionadas con el modelo son cruciales para un sitio web y no deberían ignorarse. Otra de las razones es que Symfony se ajusta mejor a las aplicaciones medianas y grandes en un entorno empresarial. En ese tipo de aplicaciones, las tareas automáticas proporcionadas por el modelo de Symfony suponen un gran ahorro de tiempo, por lo que merece la pena el tiempo dedicado a aprender su funcionamiento interno.

2.11.1. Las clases del modelo

El esquema se utiliza para construir las clases del modelo que necesita la capa del ORM. Para reducir el tiempo de ejecución de la aplicación, estas clases se generan mediante una tarea de línea de comandos llamada propel-build-model.

> symfony propel-build-model

Al ejecutar ese comando, se analiza el esquema y se generan las clases base del modelo, que se almacenan en el directorio *lib/model/om/* del proyecto, con el siguiente esquema de nombre: Base[Nombre de la tabla BD].php, Base[Nombre de la tabla BD]Peer.php, y las clases de mapeo de las tablas de la BD [Nombre de la tabla BD]MapBuildes.php que se almacenan en *lib/model/map/* , todo esto por cada tabla de la BD.

Ejemplo: (tablas de la BD proyecto Residencia)

- BaseSancionado.php
- BaseSancionadoPeer.php
- BaseIndisciplina.php
- BaseIndisciplinaPeer.php
- SancionadoMapBuilder.php
- IndisciplinaMapBuilder.php

Además, se crean las verdaderas clases del modelo de datos en el directorio *lib/model/,* de la siguiente forma: [Nombre de la tabla BD].php y [Nombre de la tabla BD] Peer.php.

Ejemplo:

- Sancionado.php
- SancionadoPeer.php
- Indiscilplina.php
- IndisciplinaPeer.php

2.11.2. Clases Objetos y Clases Peer

Sancionado e Indisciplina son clases objeto que representan un registro de la base de datos. Permiten acceder a las columnas de un registro y a los registros relacionados.

SancionadoPeer e *IndisciplinaPeer* son clases de tipo "peer"; es decir, clases que tienen métodos estáticos para trabajar con las tablas de la base de datos. Proporcionan los medios necesarios para obtener los registros de las tablas. Sus métodos devuelven normalmente un objeto o una colección de objetos de la clase objeto relacionado.

2.12. Conexiones con la base de datos

Aunque el modelo de datos es independiente de la base de datos utilizada, es necesario utilizar una base de datos concreta. La información mínima que necesita Symfony para realizar peticiones a la base de datos es su nombre, los datos de acceso y el tipo de base de datos. Esta información se indica en el archivo databases.yml que se encuentra en el directorio *config/.*

Las opciones de la conexión se establecen para cada entorno. Se pueden definir diferentes opciones para los entornos prod, dev y test, o para cualquier otro entorno definido en la aplicación. También es posible redefinir esta configuración en cada aplicación, estableciendo diferentes valores para las opciones en un archivo específico de la aplicación.

De esta forma es posible disponer de políticas de seguridad diferentes para las aplicaciones públicas y las aplicaciones de administración del proyecto, y definir distintos usuarios de bases de datos con privilegios diferentes. Cada entorno puede definir varias conexiones y cada conexión hace referencia a un esquema con el mismo nombre.

2.13. Patrones de diseño en Symfony

Patrones GRASP implementados

Creador

En la clase *nombremóduloActions* se encuentran las acciones definidas para cada módulo. En las acciones se crean los objetos de las clases que representan las entidades y de los objetos de otras

clases que intervienen en la lógica del módulo, por ejemplo de nuestro caso, las clases fachadas de servicios web, evidenciando de este modo que la clase Actions es "creador" de dichos objetos.

Experto

Este es uno de los más utilizados, partiendo que el framework en su implementación del patrón MVC divide el controlador en dos capas, el controlador frontal con la clase sfController se encarga de decodificar la petición y transferirla a la acción correspondiente, controlando la entrada y salida de la misma y el controlador de los módulos encargado de la lógica del módulo y el modelo con el Propel define una capa de abstracción de la BD que encapsula toda la lógica de los datos y la capa de acceso a datos que utiliza el Creole para manipular toda la conexión, la clase sfRequest almacena todos los elementos que forman la petición (parámetros, cookies, cabeceras, etc.) y sfResponse contiene las cabeceras de la respuesta y los contenidos. El contenido de este objeto se transforma en la respuesta HTML que se envía al usuario.

Alta Cohesión

La clase nombreActions tiene la responsabilidad de definir las acciones para las plantillas y colabora con otras para realizar diferentes operaciones, instanciar objetos y acceder a las properties, es decir, está formada por diferentes funcionalidades que se encuentran estrechamente relacionadas proporcionando que el software sea flexible frente a grandes cambios. Symfony agrupa las clases por funcionalidades que son fácilmente reutilizables, bien por su uso directo o por herencia.

Front-Controlle (Controlador)

Todas las peticiones Web son manejadas por un solo controlador frontal, que es el punto de entrada único de toda la aplicación en un entorno determinado. Cuando el controlador frontal recibe una petición, utiliza el sistema de enrutamiento para enviar la acción al controlador responsable de la solución.

Bajo Acoplamiento

Symfony asigna a cada clase una responsabilidad para mantener pocas dependencias entre las mismas.

Tipos de patrones GOF implementados

Singleton (Instancia única)

El controlador frontal proporciona un punto de acceso global a la aplicación. En una acción, el método getContext() devuelve el mismo *singleton*. Se trata de un objeto muy útil que guarda una referencia a todos los objetos del núcleo de Symfony relacionados con una petición dada, y ofrece un método accesor para cada uno de ellos:

- sfController: El objeto controlador (->getController())
- sfRequest: El objeto de la petición (->getRequest())
- sfResponse: El objeto de la respuesta (->getResponse())
- sfUser: El objeto de la sesión del usuario (->getUser())
- sfDatabaseConnection: La conexión a la base de datos (->getDatabaseConnection())
- sfLogger: El objeto para los logs (->getLogger())
- sfI18N: El objeto de internacionalización (->getI18N())

Se puede llamar al singleton sfContext::getInstance() desde cualquier parte del código.

Abstract Factory (Fábrica abstracta)

Permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando. Por ejemplo, cuando el framework necesita crear un nuevo objeto para una petición busca en la definición de la factoría el nombre de la clase que se debe utilizar para esta tarea. Como la definición por defecto de la factoría para las peticiones es sfWebRequest, Symfony crea un objeto de esta clase para tratar con las peticiones.

En la categoría Estructurales:

Decorator *(Envoltorio)*

Añade funcionalidad a una clase, dinámicamente. El archivo layont.php, que también se denomina plantilla global, almacena el código HTML que es común a todas las páginas de la aplicación, para no tener que repetirlo en cada página. El contenido de la plantilla se integra en el layout, o si se mira desde el otro punto de vista, el layout decora la plantilla.

Composite *(Objeto compuesto)*

Permite tratar objetos compuestos como si de uno simple se tratase. Sirve para construir objetos complejos a partir de otros más simples y similares entre sí, gracias a la composición recursiva y a una estructura en forma de árbol. Esto simplifica el tratamiento de los objetos creados, ya que al poseer todos ellos una interfaz común, se tratan todos de la misma manera.

Proxy

Los filtros de seguridad de Symfony actúan como proxy entre las peticiones al servidor y las respuestas y lo mismo sucede con las validaciones. En Symfony Se pueden entender los filtros como una forma de empaquetar cierto código de forma similar a preExecute() y postExecute(), pero a un nivel superior (para toda una aplicación en lugar de para todo un módulo).

Fachada

Symfony, mediante el sistema de configuración de archivos yml, implementa este patrón permitiendo acceder a diferentes configuraciones del framework desde un único lugar. Además de que el acceso a la aplicación es a través del controlador frontal que implementa este patrón.

Adapter

Symfony da la posibilidad de cambiar a otro sistema de base de datos completamente diferente a mitad de desarrollo, solo basta con configurar un archivo yml. La capa de abstracción utilizada encapsula toda la lógica de los datos. El resto de la aplicación no tiene que preocuparse por las consultas SQL y el código SQL que se encarga del acceso a la base de datos es fácil de encontrar.

Registry

En una acción, el método getContext(), un objeto muy útil que guarda una referencia a todos los objetos del núcleo de Symfony.

Template Method

Symfony tiene una estructura específica a la hora de implementar un método, esto depende de la clase en que se implemente la funcionalidad.

2.14. Interacción con Servicios Web

La aplicación debe ser capaz de interactuar o intercambiar datos con otros sistemas a través de servicios web, dando la posibilidad no solo de invocar un servicio determinado sino de ofrecer funcionalidades encapsuladas dentro de un servicio web. Posibilitando dar soporte a una futura arquitectura SOA.

En el caso de la invocación del o los servicios que se vayan a utilizar para resolver un problema determinado de la lógica de la aplicación, se hará a través de un módulo de acceso dichos servicios, que servirá de fachada al exterior de la aplicación y frente a los servicios web o una UDDI determinada, haciendo uso de buenas prácticas y las ventajas que nos proporciona el patrón *Fachada y Singleton.* Los cuales se tendrán en cuenta dentro de la aplicación como módulos externos.

Para ello se decidió implementar una clase *sServicio.php* con la responsabilidad de gestionar la comunicación con el exterior, así como la cantidad de instancias de un servicio dado en la aplicación.

La misma va a estar almacenada dentro de la carpeta /lib del framework, para que el mismo la reconozca, la cargue automáticamente y pueda ser invocada dentro de cualquier módulo en toda la aplicación, según lo propone la estructura de directorios del mismo.

Figura 2.8 Principales funciones de sServicio.php

En el caso de que la aplicación tenga la necesidad de exportar alguna funcionalidad al exterior a través de un servicio web, se propone que use una carpeta llamada *servicio/publico,* para los que son de dominio público y *servicio/gestión* para los de gestión interna*;* donde se almacene los WSLD de los mismo así como su implementación, los cuales deben estar en la carpeta */web* del framework, que es la parte pública del mismo, evitando así agujeros en la seguridad. Las URLs de los mismos seguirán los esquemas definidos en los lineamientos arquitectónicos de la UCI. (Ver http://uddi.uci.cu).

Figura 2.9 Servicios Web

PHP5 incluye soporte nativo para la creación de servicios web con la *php_soap.dll* la cual debe estar habilitada en el *php.ini,* y el WSLD será generado con el Eclipse PDT o el Zend Studio for Eclipse.

2.15. Convenciones o estándares de códigos

Las convenciones y estándares de código proporcionan un lenguaje común en el equipo de desarrollo y entre aplicaciones. Con el objetivo de lograrlo y atendiendo al campo de acción de nuestra propuesta proponemos que se usen establecidas en los lineamientos de arquitectura propuestos por la Universidad de Ciencia Informáticas. (Ver http://uddi.uci.cu).

El nombre de los archivos que se creen para la capa de presentación ya sean .js o .css será el nombre del modulo al que pertenece unido al nombre de la acción que cumplen y si son globales tendrán el nombre de la aplicación.

TrabEduc_Indisciplina.js (TrabEduc -> Módulo, Indisciplina -> Acción)

2.16. Seguridad

Garantizar la seguridad de aplicaciones web es algo primordial para todo equipo de desarrollo y en la cual se emplea mucho tiempo. Symfony, como framework de desarrollo, incluye una serie de funcionalidades, estructura y ficheros de configuración, los cuales nos permiten desarrollar aplicaciones web seguras y auditables.

Seguridad del Sistema de Directorios

El sistema de directorios que ofrece la arquitectura de Symfony permite separar los scripts donde se encuentran recogidos todos los segmentos de código de la aplicación del lado del servidor, dígase archivos PHP, YML y XML(de configuración), de los archivos asequibles por el cliente mediante su navegador. Esto se debe a que todos estos archivos no se encuentran en una carpeta compartida por el Servidor Web, por lo que no se puede acceder a ella mediante ninguna vía.

Todos los archivos que son cargados por los navegadores (CSS, JS, Imágenes y los Controladores Frontales) se encuentran en una carpeta que es la carpeta donde el usuario navegará. Se hace necesario configurar el Servidor Web Apache para las aplicaciones que se hospeden en un Host compartido con otras aplicaciones de otra índole como el que se muestra a continuación:

```
<VirtualHost *:80>
ServerName miaplicacion.dominio.cu
DocumentRoot "/var/htdocs/miproyecto/web"
DirectoryIndex index.php
<Directory "/var/htdocs/miproyecto/web">
AllowOverride All
Allow from All
</Directory>
</VirtualHost>
```

Además de esta configuración Symfony permite guardar los directorios que contienen las librerías del Framework y de la aplicación en cualquier lugar del servidor. Para que el controlador frontal reconozca donde se guarda este directorio se carga en un grupo de constantes que se define al comienzo de cada controlador. Esto ayuda además en la utilización de varios controladores frontales para la misma aplicación, ayudando a manejar las dos partes de una aplicación como la parte de gestión y la de administración.

Métodos de Validación y Manejo de Errores

Validación de Formularios

Las validaciones de los datos enviados por el cliente en un formulario se realizan mediante archivos YML, esto permite que el código de validación sea menos repetitivo en los métodos de validación y la ejecución de los mismos se realice de forma más segura. Estas validaciones se realizan en el archivo *acción.yml* que se encuentra en el directorio *modules/modulo/acción/valídate/* y se refiere a el nombre de la acción de dicho módulo. La ejecución de las validaciones se realizan antes de la llamada a la función *validateMyAccion*, así sea positivo o negativo su resultado. Después del resultado de la validación de un formulario es posible mostrar los datos incorrectos al usuario en los mismos campos del mismo junto con mensajes de error que muestran donde están los datos incorrectos.

Manejo de Errores

La clase *Action* presenta métodos mágicos para el manejo de errores y la validación de los datos para cada una de las acciones que posee o para todas en general. Estas funciones se nombran *validateMiAccion()* y *handleErrorMiAccion()* para el caso de una acción en específico y en caso de no encontrar *handleErrorMiAccion()* se remite a la función de la clase *handleError()*. Este método es genérico para manejar todos los errores de la clase. En caso contrario Symfony retorna *sfView:ERROR* que lanza la plantilla *miAccionError.php*.

El flujo de trabajo en la validación de los datos y el manejo de los errores se muestra a continuación.

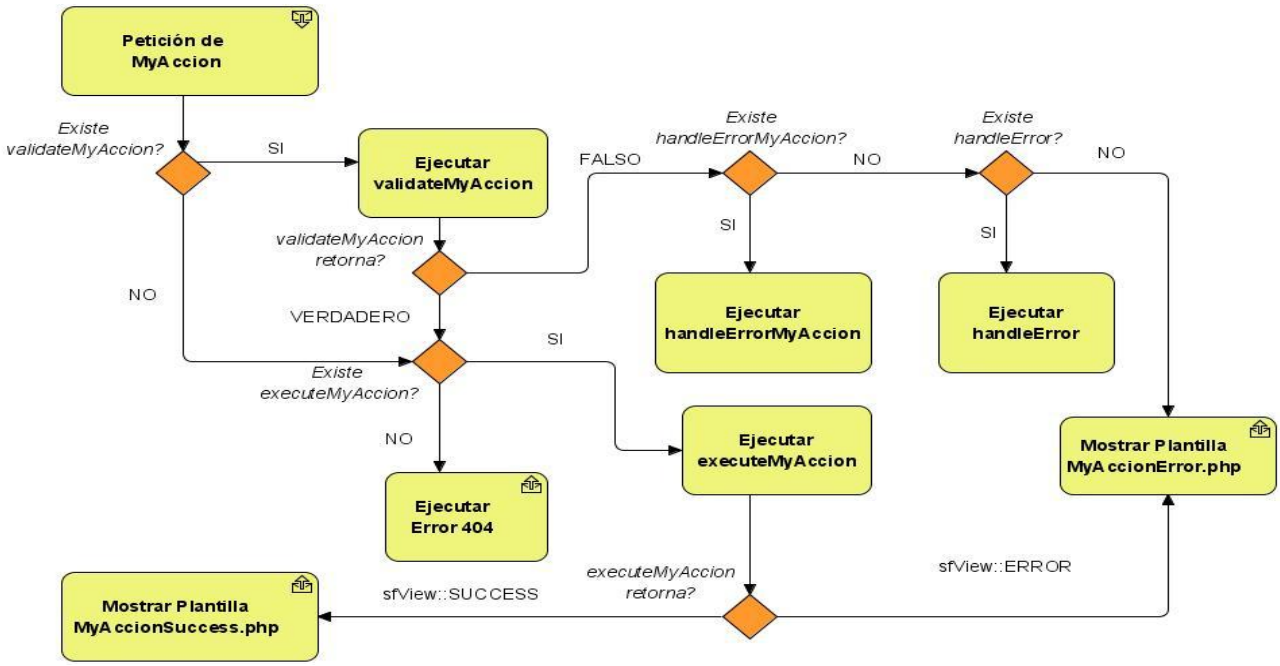


Figura 2.10 Flujo de Trabajo del proceso de Validación y Manejo de Errores para una acción.

Filtros

Symfony cuenta con un sistema de validación muy efectivo basándose en filtros. Estas validaciones se encargan de los procesos de seguridad de las acciones entre otras tareas. De hecho, Symfony procesa cada petición como una cadena de filtros ejecutados de forma sucesiva. Cuando el framework recibe una petición, se ejecuta el primer filtro. En algún punto, llama al siguiente filtro en la cadena, luego el siguiente, y así sucesivamente. Cuando se ejecuta el último filtro, los filtros anteriores pueden finalizar, y así hasta el filtro de sfRenderingFilter que siempre es el primero por definición del núcleo de Symfony.

Los filtros declarados por los desarrolladores son ubicados en la cadena de filtros que ejecuta Symfony, por lo que cada filtro se divide en tres partes, el código que se ejecuta antes de la llamada al método *$filterChain->execute()* y el código que se ejecuta después de la llamada a la misma función.

Los filtros no son utilizados solo con fines de seguridad para las aplicaciones. También tienen otras utilidades como la reutilización de código, ya que son pedazos de código que se ejecutan siempre antes de la llamada a las acciones de un módulo o una aplicación. (Ver figura 2.11)

Mecanismo de escape

Para evitar XSS (cross-site scripting) en los datos ingresados en las plantillas de forma dinámica por usuarios maliciosos es necesario tomar ciertas medidas en las aplicaciones, una de estas técnicas son los mecanismos de escape. Symfony provee de ciertos mecanismos para evitar que este proceso sea un método repetitivo, mediante la utilización de los parámetros *escaping_strategy* y *escaping_method* en el archivo de configuración *miaplicacion/config/settings.yml*.

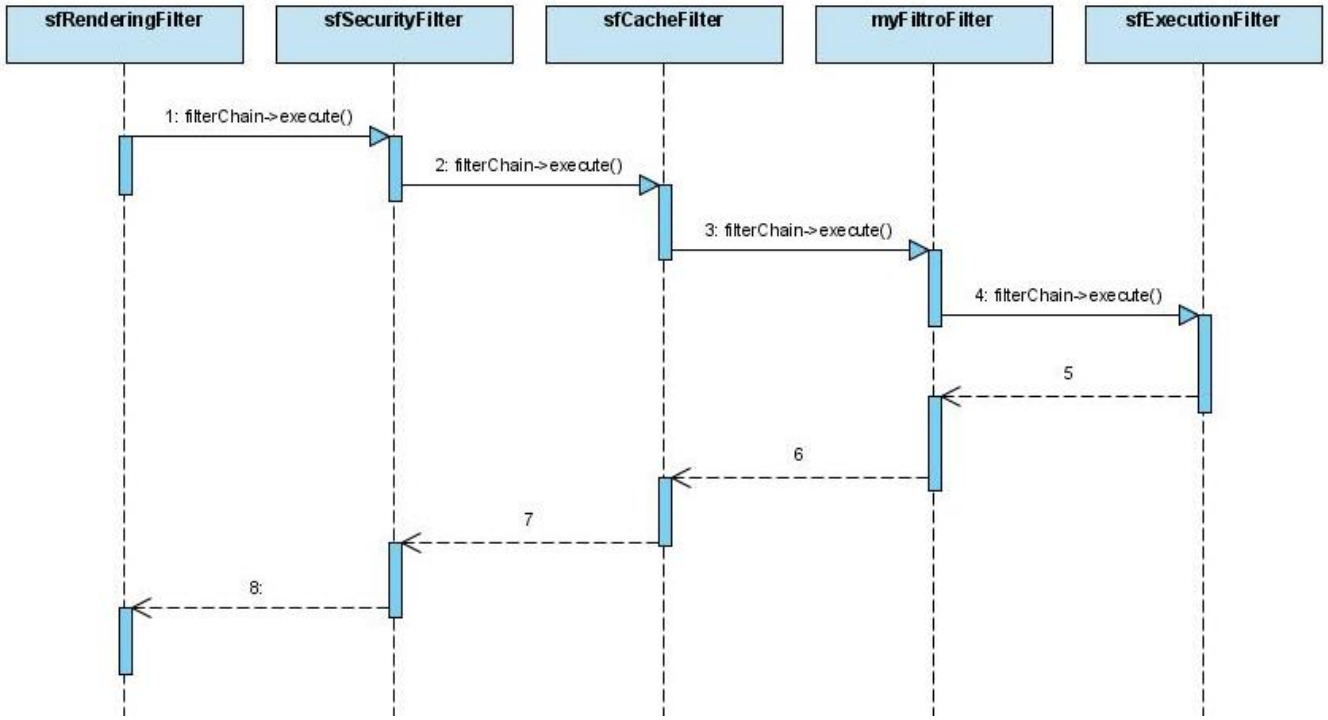


Figura 2.11 Secuencia de ejecución de los Filtros en Symfony

La utilización de los mecanismos de escape en Symfony utiliza la función *htmlentities ()* de PHP en todas las variables mostradas en la plantilla y el Layout. Los mecanismos de escape incluyen además las salidas de los arreglos y objetos ya que todos los valores de estos tipos de datos utilizados en los elementos de la vista son decorados con el tipo de dato *sfOutputEscaperArrayDecorator* y *sfOutputEscaperObjetctDecorator* respectivamente para facilitar su manipulación en los mecanismos de escape. Los métodos de estas clases contienen un parámetro adicional que permite modificar el valor de *escaping_method* de forma dinámica para una salida en específico.

SQL Injection. Uso de Criteria

Otros de los problemas de seguridad más clásicos son los ataques de Inyección SQL. Symfony también promueve un mecanismo para escapar de esta amenaza mediante el uso de las clases *Criteria* que están disponibles en el modelo.

A pesar de que el modelo en Symfony permite utilizar consultas SQL de manera pura se debe construir las consultas utilizando la clase *Criteria*. Ya que las utilidades de esta clase permite crear las consultas tan complejas como se necesite y todos los valores pasados son escapados para evitar el ataque de Inyección SQL. El uso de Criteria trae otras ventajas como la optimización del código SQL para el gestor de Base de Datos que se esté utilizando y la obtención de los resultados en un arreglo de Objetos del tipo seleccionado.

Sistema de Enrutamiento

Los sistemas de Enrutamiento utilizados por Symfony, brindan ciertas garantías de seguridad en cuanto a ocultar el sistema de directorios de la aplicación. Symfony maneja un sistemas de URL "limpias", es decir, que estas están asociadas a la acción que ejecutan. La utilización de este sistema tiene varias ventajas desde el punto de vista de la seguridad como las que se muestran a continuación:

- La seguridad se implementa a nivel de acciones, no a nivel de páginas. Es decir, se evita la necesidad de incluir al comienzo de cada página un script que valide si el usuario tiene permisos para acceder a la página, ya que esto puede ser violado. La única manera de acceder a los scripts es mediante el controlador frontal, y este a su vez se encarga de validar si la acción solicitada por el cliente es accesible por el mismo mediante la utilización de filtros. Al no existir otras vías de entrada a la aplicación, se garantiza que el eslabón más débil de la aplicación tenga toda la seguridad orientada hacia él.
- Al utilizar el sistema de enrutamiento de Symfony se evita mostrar la estructura interna del directorio de la aplicación. Otro aspecto importante en la seguridad de una aplicación web.
- Los parámetros pasados por GET como parte de la dirección de una petición no contienen información trascendental para la integridad de la aplicación como los nombres de las variables que se están enviando, ya que solamente se muestra el resultado de las mismas, gracias a la configuración del archivo *routing.yml* que contiene el formato de las variables que se pasarán a la petición. Esto permite hacer las URL más seguras para la aplicación y personalizables para el usuario.
- El uso de este tipo de enrutamiento permite que cualquier URL no reconocida se redirige a una página especificada por el programador y los usuarios no pueden navegar por el directorio raíz

del servidor mediante la prueba de diferentes URL. La razón es que no se visualiza el nombre del script utilizado o el de sus parámetros.

Internamente el framework interpreta las URIs normalmente, pero estas son interpretadas para el usuario como las URL limpias, una manera de realizar convertir las URI a URL es mediante la utilización del HELPER *url_for()*. Para imprimir links de manera dinámica en las aplicaciones se utiliza este HELPER que trasforma las URI a URL para los usuarios.

Sesiones de Usuario

Symfony maneja automáticamente las sesiones del usuario y es capaz de almacenar datos de forma persistente entre peticiones. Utiliza el mecanismo de manejo de sesiones incluido en PHP y lo mejora para hacerlo más configurable y más fácil de usar.

El objeto sesión del usuario actual se accede en la acción con el método getUser(), que es una instancia de la clase sfUser. Esta clase dispone de un contenedor de parámetros que permite guardar cualquier atributo del usuario. Esta información estará disponible en otras peticiones hasta terminar la sesión del usuario. Los atributos de usuarios pueden guardar cualquier tipo de información (*cadenas de texto, arrays y arrays asociativos*). Se pueden utilizar para cualquier usuario, incluso si ese usuario no se ha identificado.

El manejo de sesiones de Symfony se encarga de gestionar automáticamente el almacenamiento de los IDs de sesión tanto en el cliente como en el servidor. En el cliente se guarda en las *cookies* y en el servidor en un fichero o en una base de datos, todo esto altamente configurable en el fichero *factories.yml.*

```
all:
  storage:
    class: sfMySQLSessionStorage / sfSessionStorage       #Servidor o (/) cliente
  param:

    database: DATABASE_CONNECTION    # Nombre de la conexión a la base de datos que
                    se utiliza.
  session_name: mi_nombre_cookie # Cliente
```

Figura 2.12 ***factories.yml***

Seguridad de la Acción

La posibilidad de ejecutar una acción puede ser restringida a usuarios con ciertos privilegios. Las herramientas proporcionadas por Symfony para este propósito permiten la creación de aplicaciones seguras, en las que los usuarios necesitan estar autenticados antes de acceder a alguna característica o a partes de la aplicación. Añadir esta seguridad a una aplicación requiere dos pasos: declarar los requerimientos de seguridad para cada acción y autenticar a los usuarios con privilegios para que puedan acceder estas acciones seguras.

Antes de ser ejecutada, cada acción pasa por un filtro especial que verifica si el usuario actual tiene privilegios de acceder a la acción requerida. En Symfony, los privilegios están compuestos por dos partes:

- Las acciones seguras requieren que los usuarios estén autenticados.
- Las credenciales son privilegios de seguridad agrupados bajo un nombre y que permiten organizar la seguridad en grupos o roles.

Las páginas login y secure son bastante simples y fáciles de personalizar. Se puede configurar que acciones se ejecutan en caso de no disponer de suficientes privilegios en el archivo *settings.yml* de la aplicación cambiando el valor de las propiedades mostradas en el listado.

```
login_module: default   # Modulo por defecto a ser llamado cuando
login_action: login       # un usuario no autenticado accede a un
                          # recurso seguro (Seguridad en nuestro caso)
#
secure_module: default  # Modulo por defecto a ser llamado cuando
secure_action: secure     # un usuario no tiene las credenciales
                          # correctas para una acción (Seguridad)
```

Figura 2.13 settings.yml

Para restringir el acceso a una acción se crea y se edita un archivo de configuración YAML llamado *security.yml* en el directorio *config/* del módulo. En este archivo se pueden especificar los requerimientos de seguridad que los usuarios deberán satisfacer para cada acción o para todas (all) las acciones.

La sintaxis YAML utilizada en el archivo *security.yml* permite restringir el acceso a usuarios que tienen una combinación de credenciales, usando asociaciones de tipo AND y OR. Con estas combinaciones se pueden definir flujos de trabajo y sistemas de manejo de privilegios muy complejos.

```
ver:
  is_secure:    off    # Todos los usuarios pueden ejecutar la acción "ver"
modificar:
  is_secure:    on     # La acción "modificar" es sólo para usuarios autenticados
borrar:
  is_secure:    on     # Sólo para usuarios autenticados
  credentials:  admin  # Con credencial "admin"
all:
  is_secure:    off    # off es el valor por defecto
```

Figura 2.14 *security*.yml

De esta forma podemos desarrollar un *modelo de seguridad basado en roles*, el cual nos permite asignar roles a los usuarios según la responsabilidad que tenga en un módulo determinado o en la aplicación. Debido a esto las políticas de acceso basadas en roles regulan el acceso de los usuarios a la información en término de las acciones a realizar dentro de su trabajo (rol). Entonces, por la integración entre los roles y las responsabilidades de los usuarios, damos cumplimiento a los principios de mínimo privilegio y separación de responsabilidades, limitando a los usuarios solo a los privilegios necesarios para hacer su trabajo.

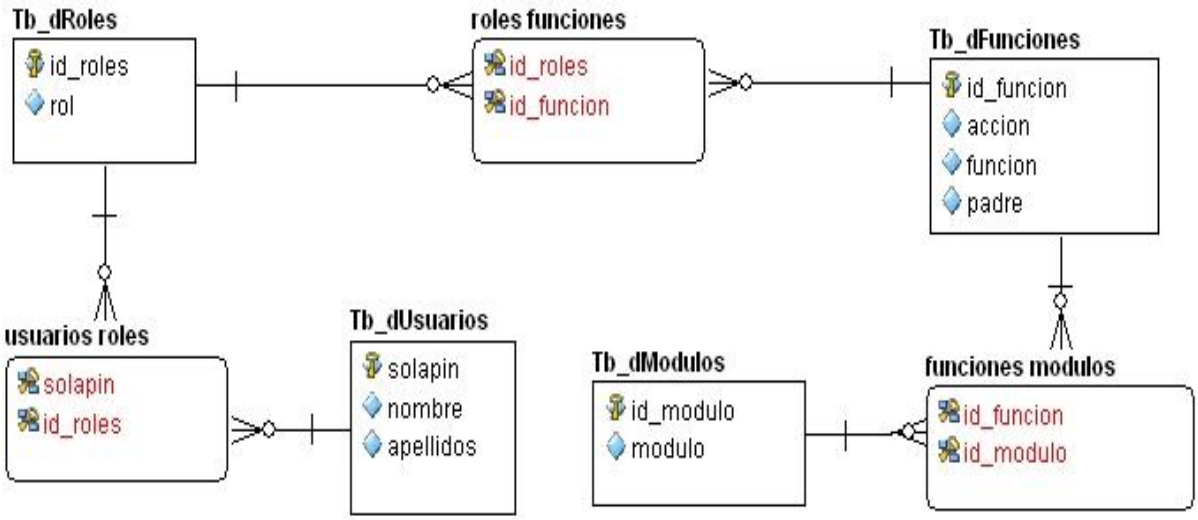


Figura 2.15 Diagrama de BD para la definición de roles.

Para ello, en el proceso de autenticación construimos dinámicamente un objeto JSON con las acciones a la cual el usuario tiene acceso según su rol y el módulo en que va a trabajar, con el cual se genera un menú en la plantilla de la vista.

Módulo: Seguridad

Tanto en la aplicación *frontend* como en la *backend* se crea un módulo de seguridad (*seguridad*) con la responsabilidad de gestionar todo el proceso de autenticación, control de usuarios y la generación del objeto JSON, en el cual estarán las plantillas de login y logout. Al mismo será redireccionadas todas las peticiones de autenticación o verificación de credenciales.

Si según las características del negocio que se vaya a implementar se busca que el usuario siempre se autentique antes de entrar a la aplicación, entonces modificamos los siguientes parámetros del archivo *settings.yml*.

```
default_module: seguridad
default_action: index
```

Módulo: Gestión de Administración

Symfony es capaz de generar módulos más avanzados para la parte de gestión o administración de las aplicaciones, también basados en las definiciones de las clases del modelo del archivo *schema.yml*. Se puede crear toda la parte de administración de la aplicación mediante módulos generados automáticamente. Los ejemplos de esta sección describen los módulos de administración creados para una aplicación llamada backend, que es una aplicación en la que solo trabajan los administradores, separada de aplicación a la cual acceden los usuarios finales.

Los módulos de administración interpretan el modelo con la ayuda de un archivo de configuración especial llamado *generator.yml*, que se puede modificar para extender los componentes generados automáticamente y para controlar el aspecto visual de los módulos fácilmente.

Proceso de Autenticación

Para autenticarse al sistema el usuario debe entrar los datos que se requieran para verificar que es quien dice ser (ver figura 2.16). El mecanismo verifica que todos los datos sean válidos mediante el siguiente procedimiento:

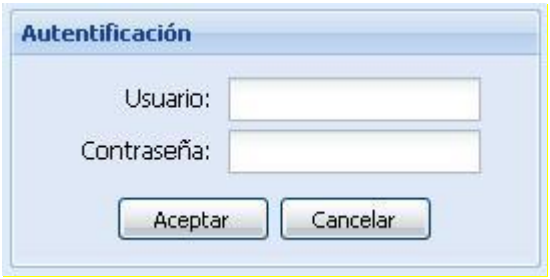


Figura 2.16 Interfaz de Autenticación.

1. El usuario entra los datos en la página de autenticación (*login*), a la cual puede llegar tanto por una página principal o por tratar de acceder a un recurso que está asegurado y aún no se ha autenticado.
2. Al enviar sus datos (credenciales) los mismos son verificados con un servidor LDAP, haciendo uso de la *php_ldap.dll* presente en la API del PHP5 y activada en el *php.ini* o con un servicio web de que me permita conocer la autenticidad del usuario de los cuales se toma su número de solapín (*Id_Expediente en bases de datos uci*).
3. Se verifica si es usuario de la aplicación, consultando la tabla Tb_dUsuario de la base de datos mediante el número de solapín antes obtenido.
4. En caso de ser válidas las credenciales se procede a:
   - Iniciar la sesión del usuario y se le otorga el estado de autenticado, haciendo uso de las funcionalidades del objeto sfUser del Symfony.
   - Crear el objeto JSON.
   - Redireccionar a la página de inicio de la aplicación o en caso de él haber pedido una URL se enviará a la misma.
5. En el caso en que no sean válidas las credenciales se le enviará a la misma página del login con el parámetro para que verifique sus datos.

2.17. Auditoría

Tanto los desarrolladores como los administradores de aplicaciones, necesitan de la auditoría para saber quien efectuó una acción determinada en el sistema y cuándo, para identificar acciones que arrojen fallos en el sistema así como sus posibles causas, para detectar reiterados intentos fallidos al sistema o en horarios no establecido, para obtener información de la relación entre los objetos del sistema antes una acción dada, consultas a la base de datos, entre otros.

Normalmente, esta información se obtiene mediante los archivos de log y las herramientas de depuración o *debug*, donde se almacena una traza del proceso que se ejecuta y tanto PHP como Symfony guardan mucha información en este tipo en archivos.

PHP dispone de una directiva llamada *error_reporting*, que se define en el archivo de configuración *php.ini*, y que especifica los eventos de PHP que se guardan en el archivo de log. Symfony permite redefinir el valor de esta opción, tanto a nivel de aplicación como de entorno, en el archivo *settings.yml.*

Además de los archivos de log creados por PHP, Symfony también guarda mucha información de sus propios eventos en otros archivos de log. Los archivos de log creados por Symfony se encuentran en el directorio *miproyecto/log/.* Symfony crea un archivo por cada aplicación y cada entorno, con el siguiente formato: *nombreaplicacion_entorno.log ej. (miaplicacion_dev.log).* Los mismos pueden configurarse en el fichero *loggin.yml* de la aplicación.

La sintaxis de los archivos de log generados es muy sencilla. Cada evento resulta en una nueva línea en el archivo de log de la aplicación. Cada línea incluye la fecha y hora a la que se ha producido, el tipo de evento, el objeto que ha sido procesado y otros detalles relevantes que dependen de cada tipo de evento y/o objeto procesado así como consultas SQL enviadas a la base de datos, las plantillas que se han procesado, las llamadas realizadas entre objetos, etc.

```
Nov 15 16:30:25 symfony [info ] {sfAction} call "barActions->executemessages()"
Nov 15 16:30:25 symfony [debug] SELECT bd_message.ID, bd_message.SENDER_ID, bd_...
Nov 15 16:30:25 symfony [info ] {sfCreole} executeQuery(): SELECT bd_message.ID...
Nov 15 16:30:25 symfony [info ] {sfView} set slot "leftbar" (bar/index)
Nov 15 16:30:25 symfony [info ] {sfView} set slot "messageblock" (bar/mes...
Nov 15 16:30:25 symfony [info ] {sfView} execute view for template "messa...
Nov 15 16:30:25 symfony [info ] {sfView} render "/home/production/miproyecto/...
Nov 15 16:30:25 symfony [info ] {sfView} render to client
```

Figura 2.17 ***log/miaplicacion_dev.php***

Además de los mensajes generados por Symfony, también es posible añadir mensajes propios en el archivo de log desde el código de la aplicación, utilizando alguna de las técnicas mostradas a continuación:

```
// Desde la acción
$this->logMessage($mensaje, $nivel);

// Desde una plantilla
<?php use_helper('Debug') ?>
<?php log_message($mensaje, $nivel) ?>
```

Los cuales se emplearán para personalizar la entrada y salida de un usuario al sistema luego de la autenticación.

2.18. Flujo de Trabajo

El flujo de trabajo define un grupo de paso necesarios, que deben seguir los desarrolladores para elaborar los componentes de una aplicación. En la DSSA en que estamos trabajando se utiliza las ventajas de la generación del código del framework propuesto, por lo que nuestro flujo estaría enmarcado en las funcionalidades específicas de las capas de presentación y negocio dado que la capa de acceso a datos es generada a línea de comando, además de un flujo de interacción con servicios web. Si estos pasos están bien definidos optimizamos la interacción entre los desarrolladores. Se define un rol correspondiente a cada capa lógica de la aplicación: programador de interfaz de usuario (programador de IU), programador de lógica de negocio (programador de LN) que dada las características del framework realizarán alguna lógica de acceso a datos y los programadores de servicios web (programador WS) que implementarán todo lo concerniente a la interacción con servicios web.

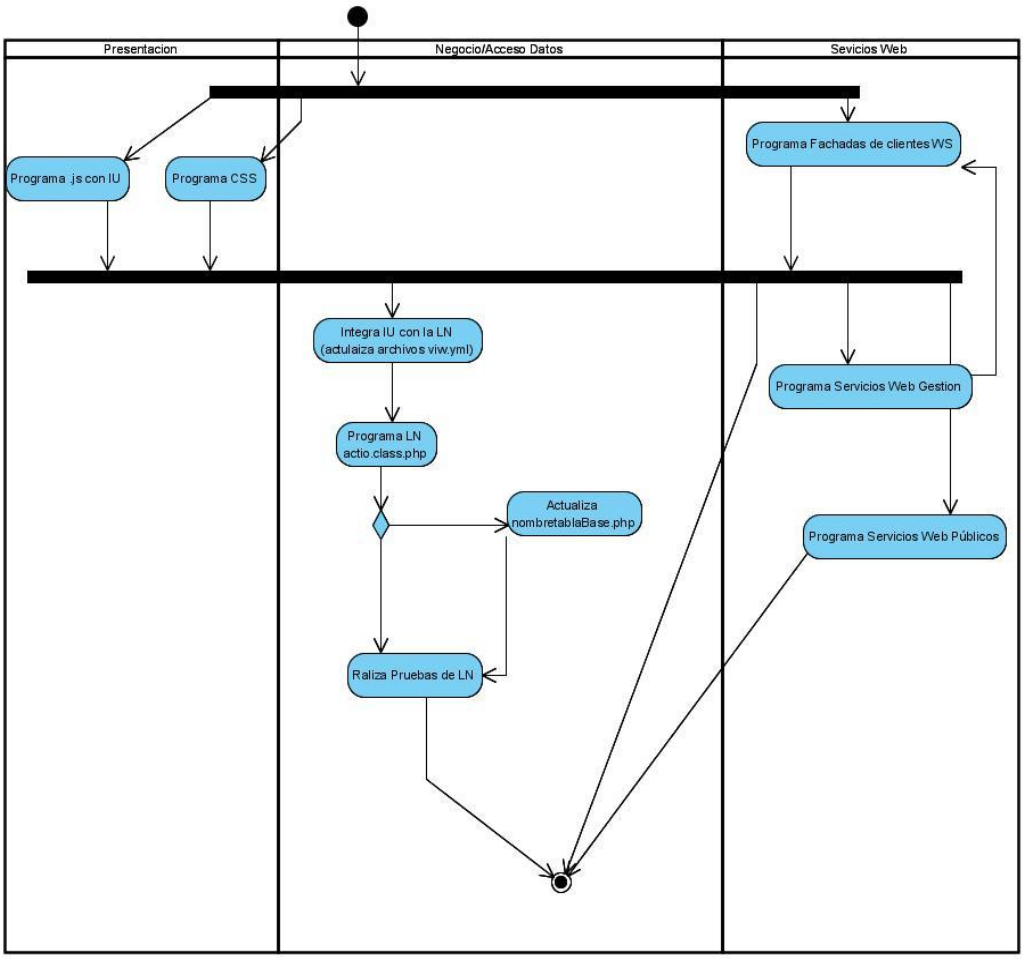


Figura 2.18. Flujo de trabajo.

Interfaz de Usuario

1. Implementar los componentes de las interfaces en javascript, utilizando los componentes las librerías ExtJS.
2. Implementar las CCS para cada componente en caso de que se requieran nuevos estilos.
3. Mantener y diseñar el layout.php de la aplicación e integrarlo.

Lógica de Negocio/Acceso a Datos

1. Implementar la LN en los archivos *action.class.php.*
2. Integrar la IU con la LN, actualizando lo fichero *vew.yml* para cada módulo donde trabaje
3. Actualiza el fichero *setting.yml* para cada módulo donde trabaje.
4. En caso de que el negocio lo requiera implementa consultas a la base de datos en las clases del modelo ya generadas.
5. Realiza las pruebas de la LN.
    a. Crea los archivos con las pruebas unitarias de la LN.

Servicio Web

1. Implementa las fachadas a clientes de servicios web.
2. Si el negocio los requiere implementa servicios web públicos o de gestión.

2.19. Propuesta de Ambiente de desarrollo

Teniendo en cuenta el estudio realizado en el "capítulo 1" de las características de las tecnologías y herramientas que proponemos, nuestro ambiente de desarrollo quedaría dividido en herramientas de desarrollo y frameworks. Este ambiente de desarrollo, dada las condiciones que exige el cliente, será enfocado totalmente al software libre y multiplataforma, principalmente sobre Linux.

2.19.1. Herramientas de Desarrollo

- ***Ambiente de desarrollo integrado*:** se propone al Eclipse PDT como IDE para desarrollo de software. Incluyendo un conjunto de plugins que amplían sus funcionalidades como plataforma de desarrollo de aplicaciones web en PHP.

  Los plugins propuestos son los siguientes:

    - ***Subclipse*:** Para permitir de una forma mucho más ágil y cómoda el desarrollo colaborativo de software en un equipo de desarrolladores.

- ***Plugin de Aptana para Eclipse v1.0:*** Plugin utilizado por el Eclipse para permitir el auto-completamiento y corrección de código CSS, JS y HTML, incluye las facilidades del IDE Aptana. Es FREEWARE y puede ser integrado al Zend Studio para Eclipse.

- ***Servidor Web:*** El servidor web propuesto para el desarrollo y despliegue de la aplicación es el Apache.

- ***Control de versiones*:** Como servidor de control de versiones se recomienda el *Subversion.*

- ***Sistema gestor de base de datos:*** Para cualquier tipo de aplicaciones se recomienda el *PostgreSQL* por ser libre además de las características expuestas en el Capítulo 1.

- **Herramientas de modelado:**
  - *Visual Paradigm Suite:* Para modelar toda la aplicación, por ejemplo, diagramas de clases, de componentes, diagramas de casos de usos, diagramas de clases persistentes para la base de datos, etcétera.
  - *Aqua Data Studio:* Para la administración y modificación de bases de datos.
  - *PgAdmin3:* Para la administración y modificación de bases de datos.

2.19.2. Frameworks

Según los requerimientos y necesidades específicas de cada aplicación se recomiendan dos ORM a usar:

- *Symfony:* Diseñado para optimizar, gracias a sus características, el desarrollo de las aplicaciones Web en PHP. (*Versión 1.0.X*)
- *Propel:* ORM usado para la persistencia en las aplicaciones, el cuál está integrado a Symfony. (*Versión 1.2*)
- *Creole:* Capa de abstracción a la base de datos y el cual viene integrado a la versión 1.2 de Propel.
- *Doctrine:* ORM usado para la persistencia en las aplicaciones y basado en Hibernate, el cual está integrado a Symfony como plugin.
- *PDO:* Capa de abstracción a la base de datos y el cual viene integrado al Doctrine.

- *ExtJS:* Diseñado para personalizar, gracias a sus características, el desarrollo de la capa de presentación en aplicaciones Web. (*Versión 2.0*).

Plugins para Symfony:

- sfTCPDFPlugin: Para generar reportes en formato pdf.
- sfSuperCachePlugin: Crea versiones cacheadas de las páginas web en el directorio de la cache bajo el directorio web raíz del proyecto, de forma que el servidor web pueda servirlas lo más rápidamente posible.
- sfOptimizerPlugin: Optimiza el código fuente de la aplicación para que se ejecute más rápidamente en el entorno de producción.

2.20. Conclusiones

Con el desarrollo de este capítulo se ha llegado a la definición del dominio en que se enmarca una arquitectura para aplicaciones empresariales en PHP y los requerimientos de referencias identificados en este dominio. Además se expuso la propuesta de Arquitectura de Dominio Específico (DSSA), permitiendo de esta forma realizar el diseño de las capas lógicas guiados por esta arquitectura de dominio. Se ha explicado la propuesta de convenciones de nombres o estándares de codificación tanto para códigos fuentes y recursos como para las estructuras de código de las aplicaciones. Se presentó un flujo de trabajo que define y organiza las principales tareas del proceso de desarrollo. Se elaboró una propuesta de ambiente de desarrollo donde se expusieron las características de las herramientas y tecnologías propuestas.

3. CAPÍTULO 3: DESCRIPCIÓN DE LA ARQUITECTURA

3.1. Introducción

En este capítulo modelamos la descripción de nuestra propuesta de arquitectura, basándose en el uso del Modelo de "4+1" Vistas de la Arquitectura del Software. Para desarrollar dicha descripción utilizamos como referencia el Proyecto Residencia, que es parte importante de nuestro campo de acción, aunque el mismo puede tomarse como referencia para cualquier equipo de desarrollo que utilice nuestra propuesta.

3.2. Modelo de "4+1" Vistas de la Arquitectura del Software

El modelo 4+1 describe la arquitectura del software usando cinco vistas concurrentes. Cada vista se refiere a un conjunto de intereses de diferentes stakeholders del sistema y se diseñan mediante un proceso centrado en la arquitectura, motivado por escenarios y desarrollado iterativamente.

- *La vista lógica* describe el modelo de objetos del diseño cuando se usa un método de diseño orientado a objetos. Para diseñar una aplicación muy orientada a los datos, se puede usar un enfoque alternativo para desarrollar algún otro tipo de vista lógica, tales como diagramas de entidad-relación.
- *La vista de procesos* describe los aspectos de concurrencia y sincronización del diseño.
- *La vista física o de despliegue* describe el mapeo del software en el hardware y refleja los aspectos de distribución.
- *La vista de implementación (desarrollo)* describe la organización estática del software en su ambiente de desarrollo.

Los diseñadores de software pueden organizar la descripción de sus decisiones de arquitectura en estas cuatro vistas, y luego ilustrarlas con un conjunto reducido de casos de uso o escenarios, los cuales constituyen la quinta vista. La arquitectura evoluciona parcialmente a partir de estos escenarios. (16).

Dado que nuestra propuesta está basada en el uso de del Symfony como framework de PHP es necesario conocer cómo funcionan las aplicaciones desarrolladas en el mismo, descritas en el capítulo 2 y sobre las cuales vamos a describir las vistas arquitectónicas.

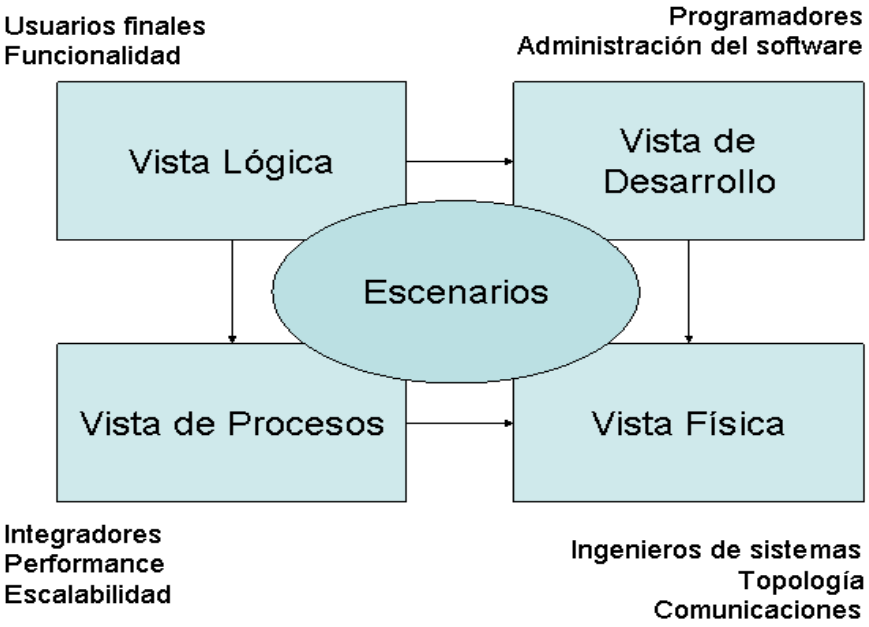


Figura 3.1 Vistas Arquitectónicas

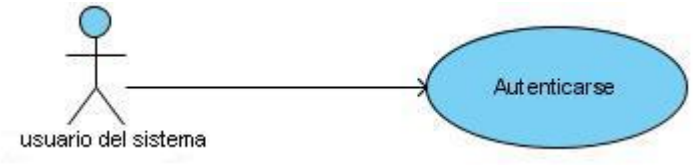
3.3. VISTA DE CASOS DE USO

Esta vista nos brinda información de escenarios (casos de usos) arquitectónicamente significativos los cuales contienen las funcionalidades críticas para el sistema. Se refiere a críticos cuando son: funciones que son las más importantes, la razón de existir del sistema, o que tienen la mayor frecuencia de uso, o que presentan cierto riesgo técnico que debe ser mitigado (Kruchten), o que sirven para validar la arquitectura propuesta por el uso de la mayoría de los elementos de la misma. Aunque debido a la capacidad de generación de código del Symfony todas las funcionalidades dentro de la gestión de administración no se van a tener en cuenta dentro de esta vista.

3.3.1. Módulo de Seguridad

Este módulo se responsabiliza de la gestión de autenticación de los usuarios. Contiene el caso de uso Autenticar Usuario, el cual es arquitectónicamente significativo.

- CU_Autenticar: El caso de uso comienza cuando el Usuario solicita entrar al Sistema, el mismo se identifica con su usuario y contraseña. Termina cuando el Usuario entra al Sistema o es denegado el acceso en caso de que no esté autorizado.



3.3.2. Módulo de Trabajo Educativo

Este módulo se encarga de todo lo concerniente al trabajo con los estudiantes en la residencia como la gestión de evaluaciones, gestión de responsables (edificio, apartamento y escaleras), gestión de indisciplinas y gestión de estudiantes.

Los casos de usos arquitectónicamente significativos correspondientes a este módulo son:

- CU_Gestionar_Evaluación_Estudiante: Este caso de uso se inicia cuando la instructora va a emitir la evaluación mensual del estudiante. Permite insertar, y modificar evaluaciones.
- CU_Gestionar_Indisciplina: Como su nombre lo indica se encarga de reportar las indisciplinas leves en al sistema y emitir la sanción.
- CU_Gestionar_Datos_Generales: Gestiona datos generales del estudiante.
- CU_Gestionar_Responsable: Se encarga de insertar o modificar los responsables de edificios, escaleras y apartamentos**.**

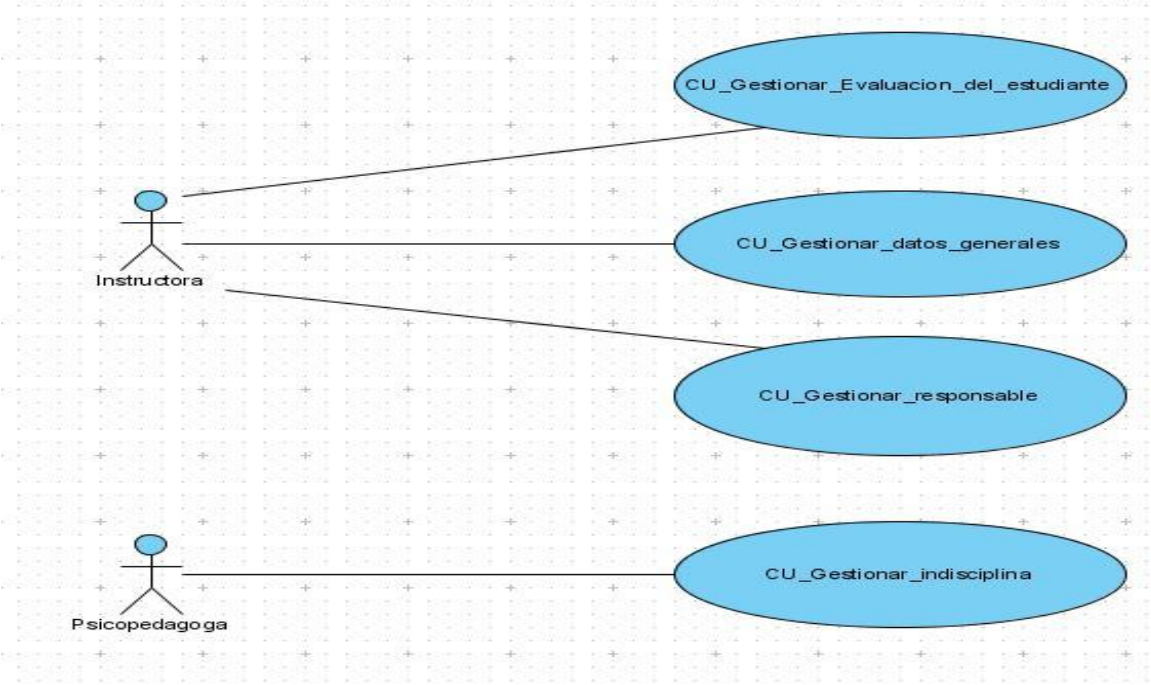


Figura 3.3 Diagrama de CU

3.3.3. Módulo de Avituallamiento y Lavandería:

Como su nombre lo indica, este módulo gestiona los procesos de control y entrega de avituallamiento y lavandería de estudiantes y profesores en la residencia. Tanto para estudiantes como para profesores se realizan los mismo procesos como la gestión de la boleta de entrega y recepción de avituallamiento, el cambio del mismo, emitir reportes con la cantidad de productos con que se dispone y su disponibilidad.

Los casos de usos arquitectónicamente significativos correspondientes a este módulo son:

- CU_Cambiar_Avituallamiento: Este caso de uso se inicia cuando un usuario se presenta a los locales de cambio de avituallamiento y solicita que se le cambie la ropa.
- CU_Gestionar_Boleta: El caso de uso comienza cuando un usuario se presenta en el departamento para realizar alguna gestión con su boleta de control de entrega y recepción de avituallamiento. Permite insertar, modificar o eliminar.

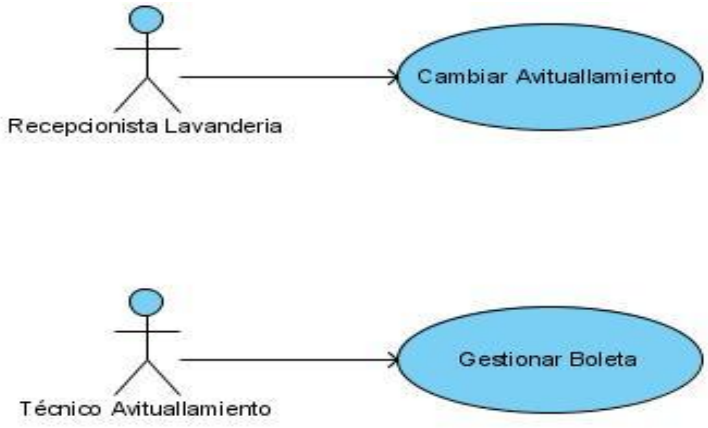


Figura 3.4 Diagrama CU.

3.4. VISTA LOGICA

Esta vista describe las clases más importantes que formarán parte del ciclo de desarrollo del sistema de software. Se describen también los paquetes o subsistemas que conforman el sistema y las relaciones de dependencia o de uso que existen entre ellos. Esta descomposición en clases y subsistemas se hace para potenciar el análisis funcional y sirve para identificar mecanismos y elementos de diseño comunes a diversas partes del sistema.

A continuación mostramos la división del sistema en módulos, con el objetivo de promover la reusabilidad y facilitar el mantenimiento del sistema ante cambios en los procesos de negocio, garantizando que solo se modifique el módulo al que pertenece la funcionalidad afectada por el cambio. También se facilita el trabajo del equipo de desarrollo ya que permite que se puedan desarrollar módulos paralelos, con sus dependencias.

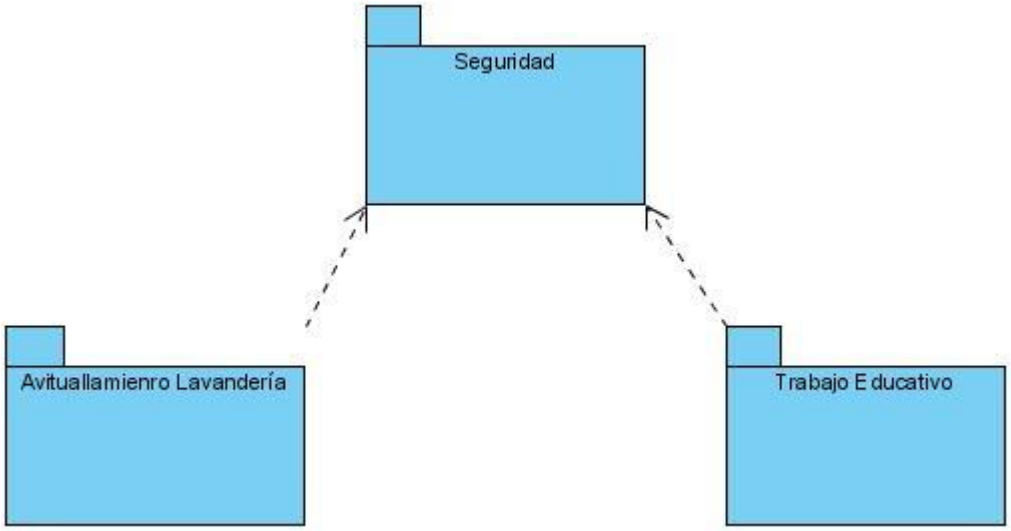


Figura 3.5 División del sistema en módulos

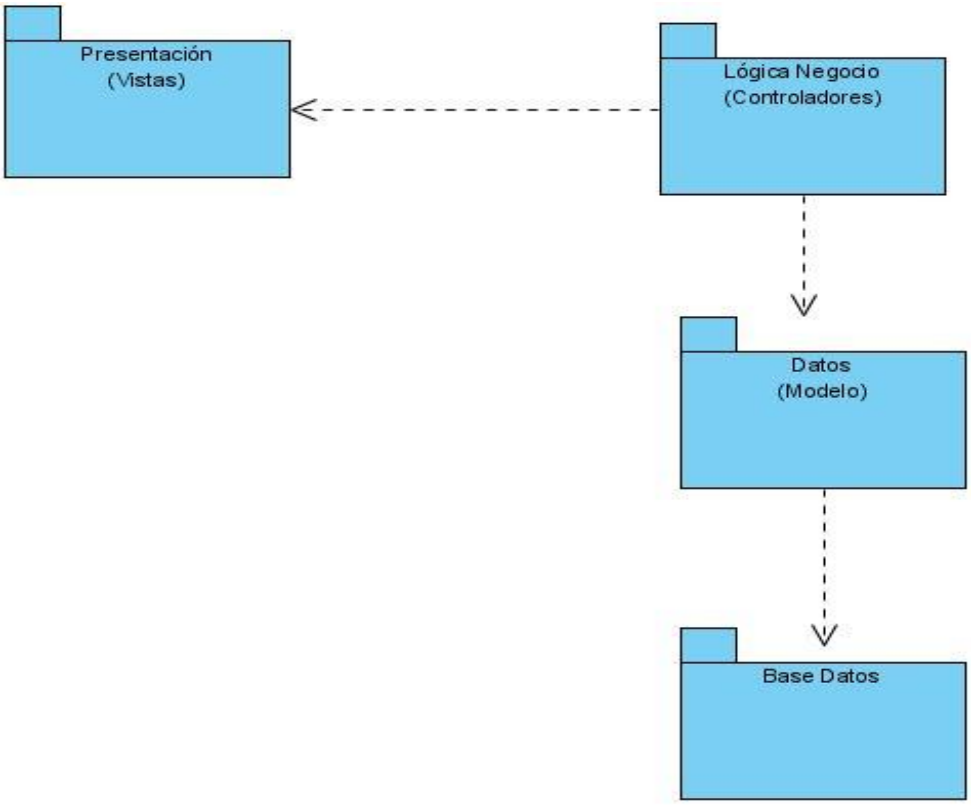


Figura 3.6 La distribución de la aplicación por capas haciendo uso del MVC

Descripción de los Paquetes		
Nombre	***Estereotipo***	***Descripción***
Presentación (Vistas)	Paquete	Conjunto los archivos que componen las páginas clientes del sistema.
Lógica del Negocio (Controladores)	Paquete	Conjunto de clases y archivos de configuración encargados de gestionar todos los procesos de la lógica de negocio.
Acceso a Datos (Modelo)	Paquete	Conjunto de clases que controlan el tratamiento de los datos.
Base de Datos	Paquete	Conjunto de tablas relacionadas que contienen los datos.

Capa de Presentación (Vista):

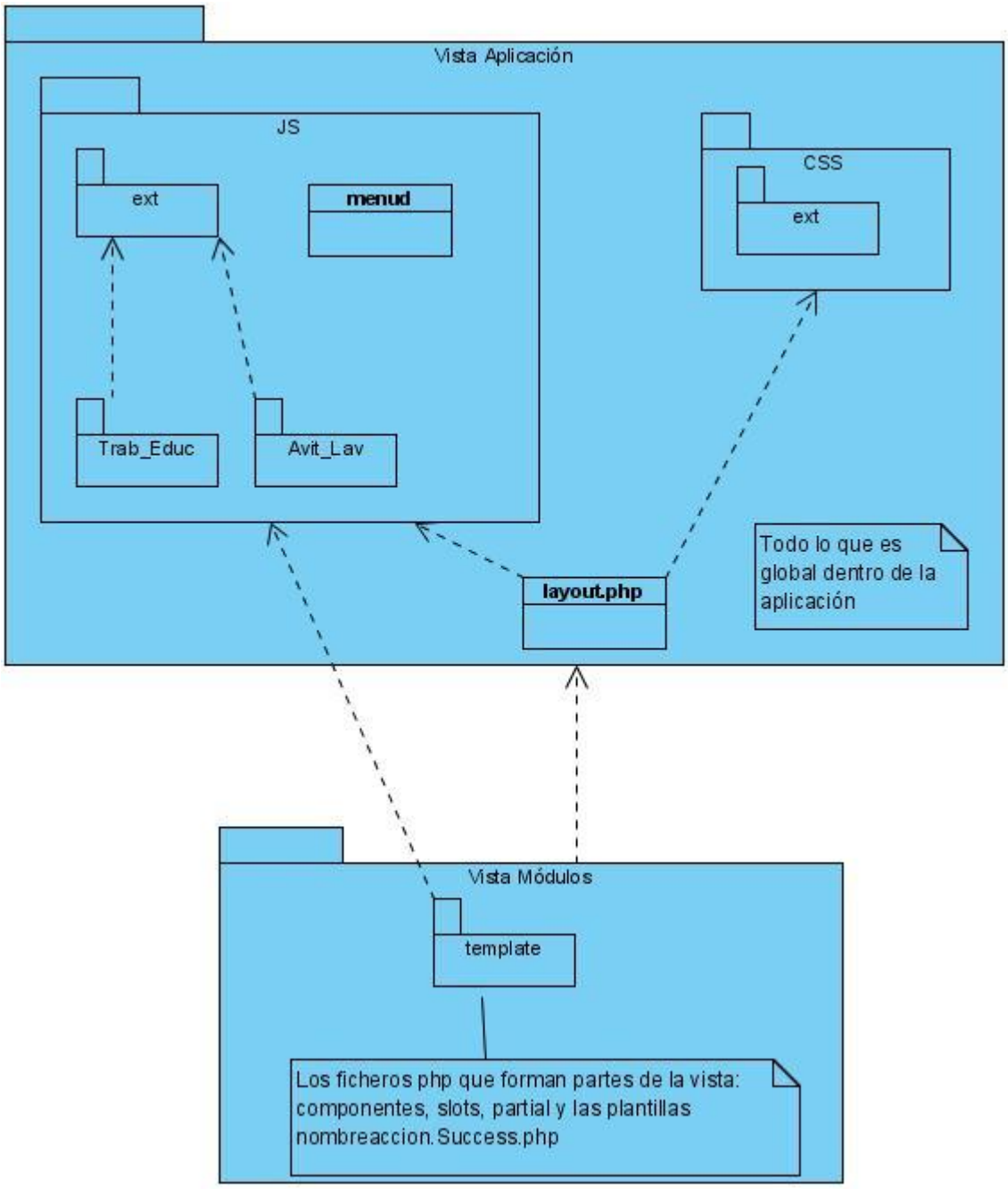


Figura 3.7 Capa de Presentación (Vista)

La presentación queda dividida en dos partes:

- La vista global que contiene todo lo que es común para toda la aplicación, como el layout (decora la plantilla del los módulos), las librerías ExtJS, los CSS y las librerías y CSS del menú, así como los archivos .js con la presentación ExtJS de cada modulo.

- La vista de los módulos que contienen todas las plantillas, donde se cargan los .js con la presentación ExtJS de cada módulo, así como los slot, partial y los componentes.

Descripción de la Vista Aplicación		
Nombre	***Estereotipo***	***Descripción***
JS	Paquete	Conjunto de archivos y clases Javascript común en toda la aplicación y el fichero _menu.xml del menú y los archivos .js con la presentación de cada módulo.
CSS	Paquete	Conjunto de las hojas de estilo común en toda la aplicación.
layout.php	Archivo	Contiene el código común a todas las páginas clientes.

Paquete JS de la Vista Aplicación		
Nombre	***Estereotipo***	***Descripción***
ext	Paquete	Librería y clases con las API del ExtJS. (adasdasd)
menud.js	Fichero	Contiene las funciones para cargar y generar el menú ExtJS
Avit_ Lav	Paquete	Conjunto de archivos .js con la presentación ExtJS del módulo de Avituallamiento y Lavandería
Trab_Educ	Paquete	Conjunto de archivos .js con la presentación ExtJS del módulo de Trabajo Educativo

Paquete CSS de la Vista Aplicación		
Nombre	***Estereotipo***	***Descripción***
ext	Paquete	Contiene las hojas de estilo del ExtJS. (ext-all.css)

Descripción de la Vista del Modulo		
Nombre	***Estereotipo***	***Descripción***
template	Paquete	Contiene las plantillas php en las que se van a cargar la presentación ExtJS del módulo.

Capa Lógica (Controlador):

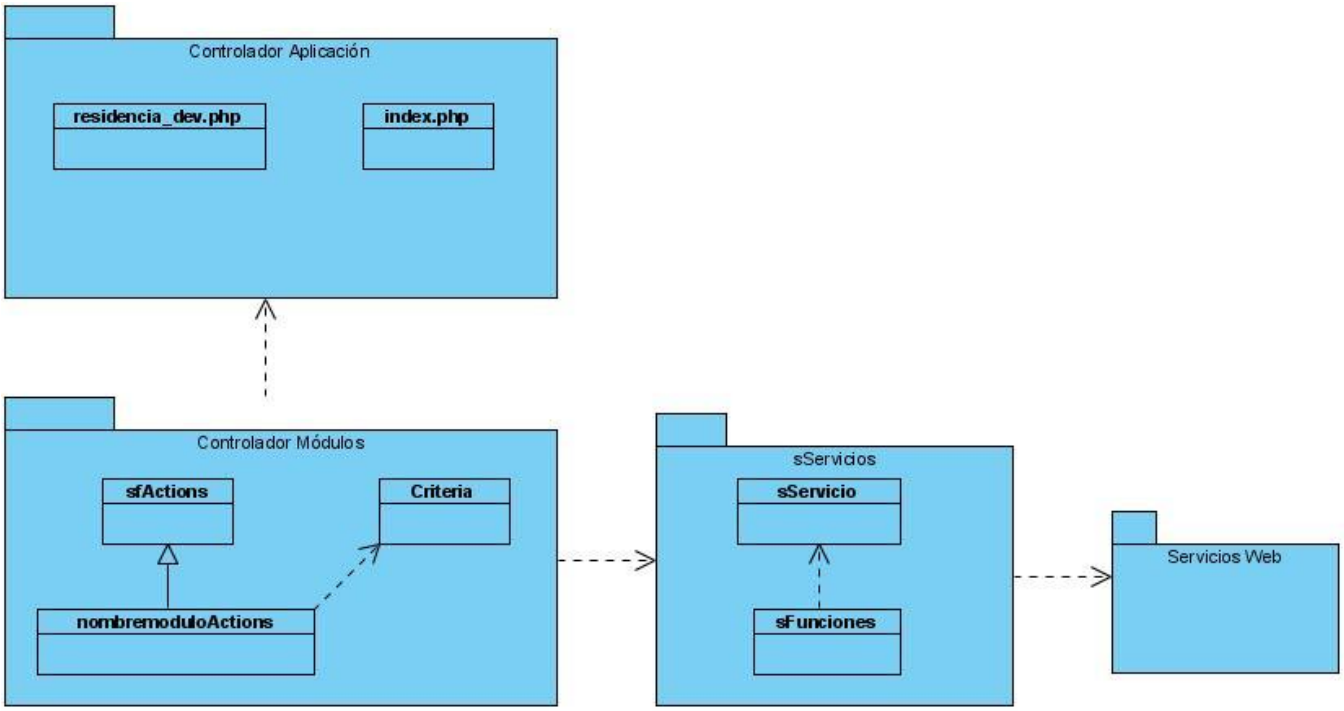


Figura 3.8 Capa Lógica (Controlador)

La lógica del negocio o Controlador se divide en dos partes:

- Controlador de la Aplicación que contiene los controladores frontales que son la vía de entrada y salida de la aplicación y son los encargados de cargar todas las configuraciones de la aplicación.
- Controladores de los módulos que son los que desarrollan toda la lógica del módulo.

Descripción del Controlador Aplicación		
Nombre	***Estereotipo***	***Descripción***
residencia_dev.php	Archivo	Controlador frontal para el entorno de desarrollo.
index.php	Archivo	Controlador frontal para el entorno de producción

Descripción del Controlador Módulo		
Nombre	***Estereotipo***	***Descripción***
sfActions	Clase	Clase del núcleo de Symfony, por la cual

		se accede a los objetos sfRequest, sfView, sfUser más usados.
nombremoduloActions	Clase	Clase responsable de la lógica de negocio del módulo.

Descripción del Paquete sServicio		
Nombre	***Estereotipo***	***Descripción***
sServicio	Clase	Clase fachada que gestiona y controla la interacción con servicios web.
sFuncion	Clase	Clase de funciones de los servicios web

Capa de Datos (modelo):

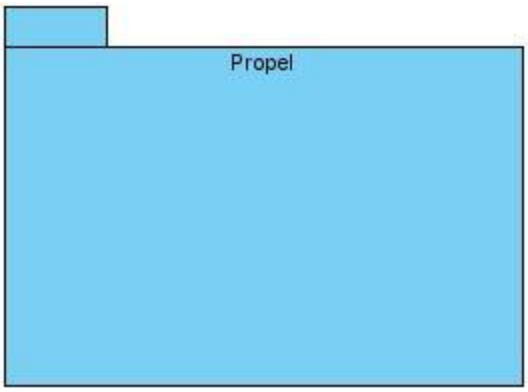


Figura 3.9 Capa de Datos (modelo)

En este paquete se encuentran todas las clases de mapeo de la base de datos generadas por el Propel, cuyo formato ya fue descrito en el capítulo anterior.

Los ***diagramas de clases*** son los más utilizados en el modelado de sistemas orientados a objetos. Un diagrama de clases muestra un conjunto de clases, interfaces y colaboraciones, así como sus relaciones. Los diagramas de clases se utilizan para modelar la vista de diseño estática de un sistema. Principalmente, esto incluye modelar el vocabulario del sistema, modelar las colaboraciones o modelar esquemas.

A continuación mostramos las principales clases de cada capa y sus interacciones.

La figura 3.10 muestra el diagrama de clases global de aplicación Symfony.

Descripción de Clases		
Nombre	***Estereotipo***	***Descripción***
Criteria	Clase	Clase del núcleo de Symfony, empleada

		para la elaboración de consultas SQL que interactúa con las clases del modelo de Propel.
view.yml	Fichero	Fichero de configuración de la vista de la aplicación y los módulos.
valídate.yml	Fichero	Fichero de configuración para la validación de formularios.
setting.yml	Fichero	Fichero de configuración global de la aplicación.
secutiry.yml	Fichero	Fichero de configuración de la seguridad de la aplicación y módulos.

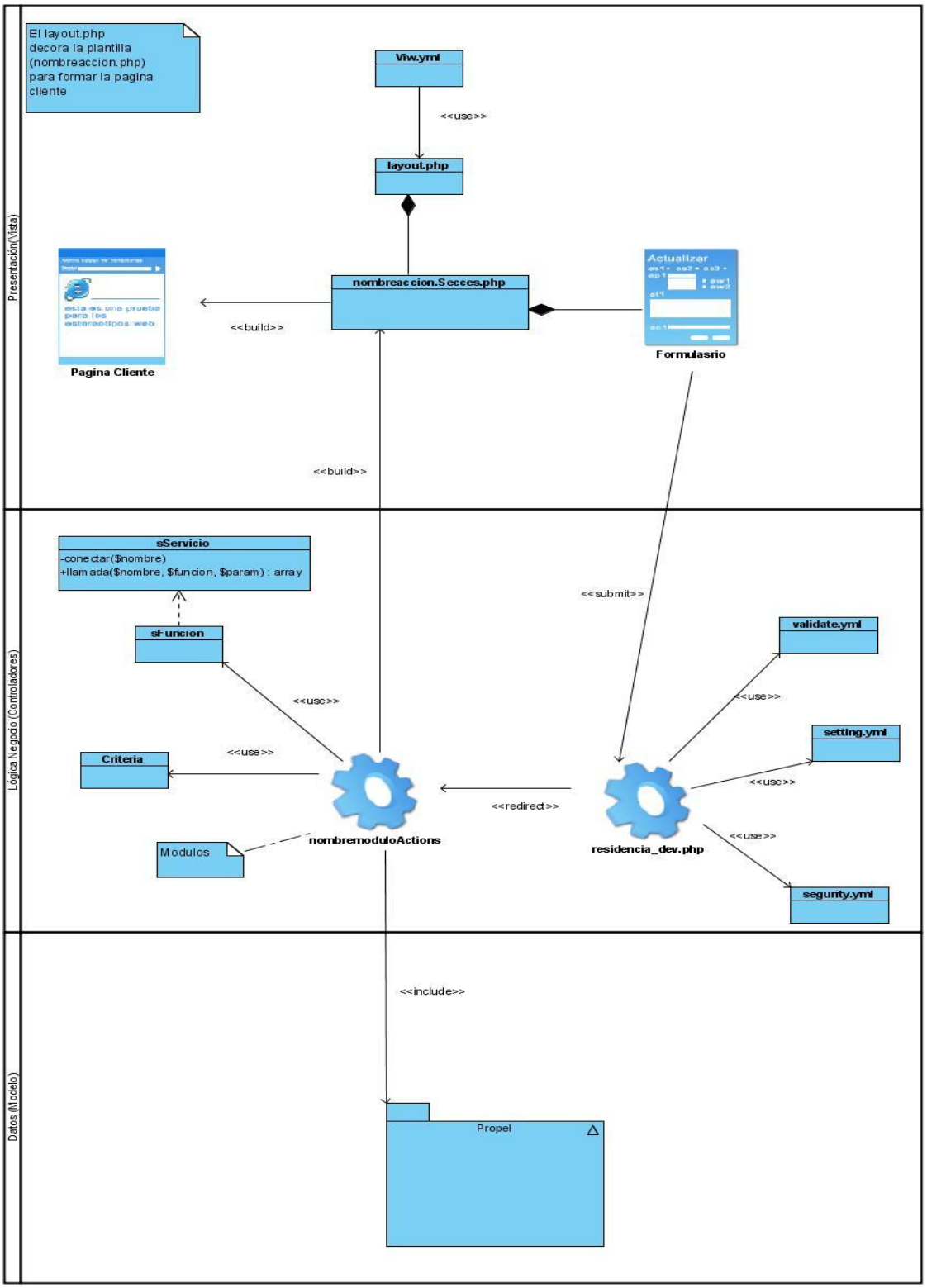


Figura 3.10 Diagrama de clases global de aplicación Symfony

La figura 3.11 muestra el Diagramas de clases del módulo de seguridad.

Descripción de Clases		
Nombre	***Estereotipo***	***Descripción***
Login	Clase	Clase fachada al servidor LDAP para la autenticación.
sfBaseSecurityUser	Clase	Clase del núcleo de Symfony para el manejo de sesiones de usuarios.
myUser	Clase	Clase generada por Symfony para manipular el proceso de autenticación, por módulos.

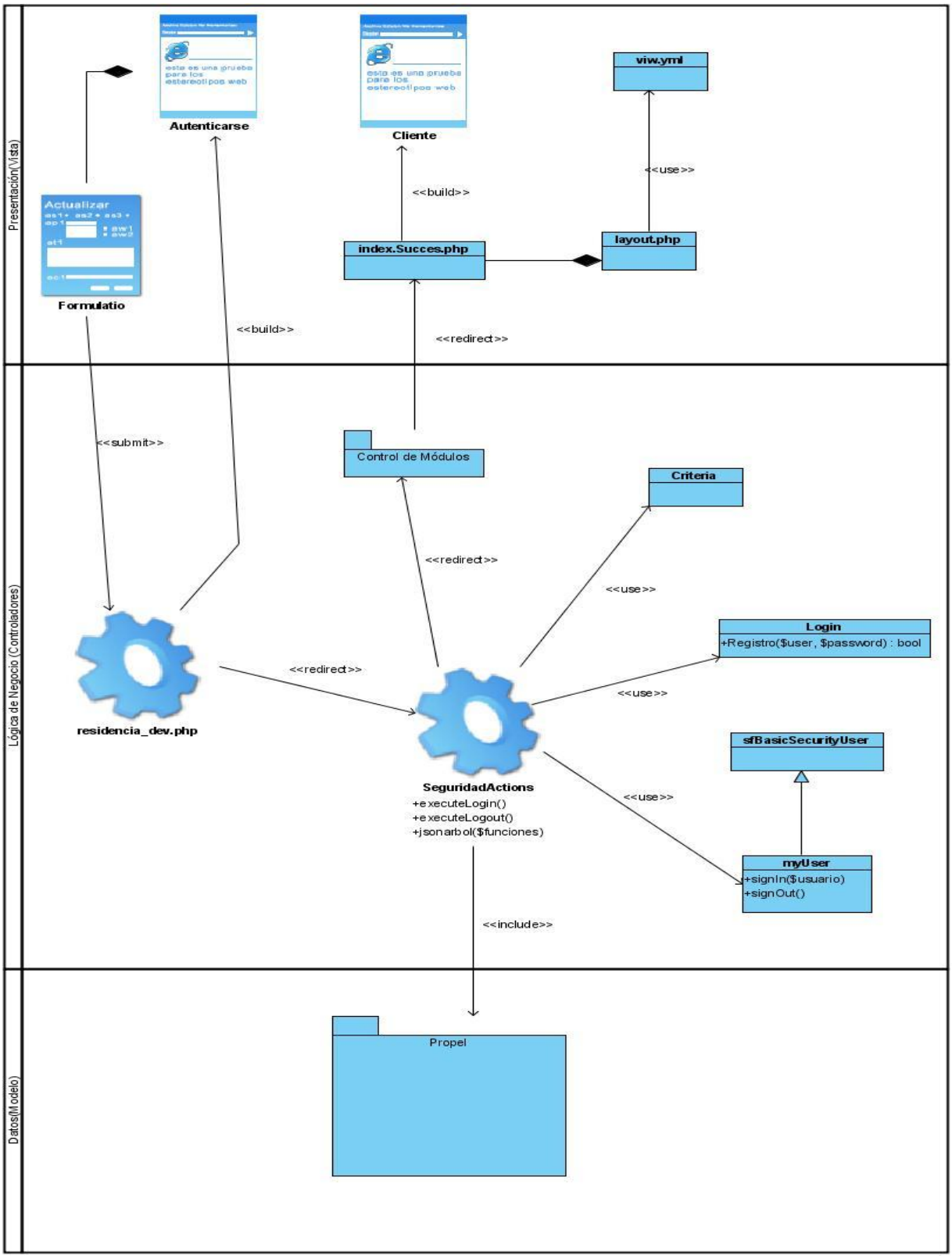


Figura 3.11 Diagramas de clases del módulo de seguridad

3.5. VISTA DE IMPLEMENTACIÓN

Esta vista describe la descomposición del software en capas y subsistemas de implementación, mostrando la colección de componentes de cada capa. También provee una vista de la trazabilidad de los elementos de diseño de la vista lógica ahora para la implementación.

Los componentes representan todos los tipos de elementos software que entran en la fabricación de aplicaciones informáticas. Pueden ser simples archivos, paquetes de Ada, bibliotecas cargadas dinámicamente, etc. Las relaciones de dependencia se utilizan en los diagramas de componentes para indicar que un componente utiliza los servicios ofrecidos por otro componente.

Descripción Componentes		
Nombre	***Estereotipo***	***Descripción***
ext-all.js, ext-base.js	Fichero	Ficheros con las librerías ExtJS
ext-all.css	Fichero	Fichero con los estilos ExtJS
layout.css	Fichero	Fichero con los estilos del layout de la aplicación.

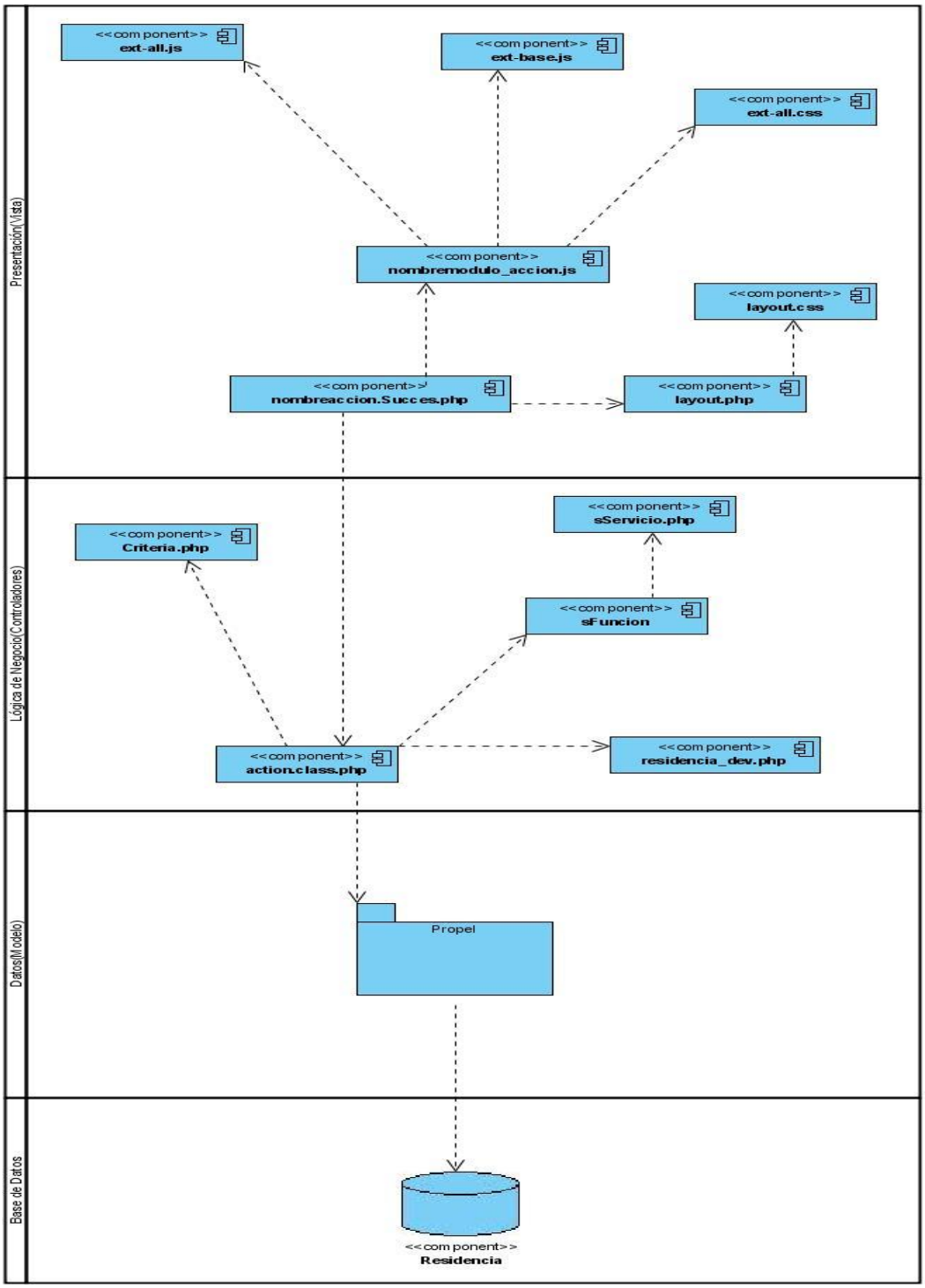


Figura 3.12 Diagramas de componentes global de la Aplicación

Módulo de Seguridad

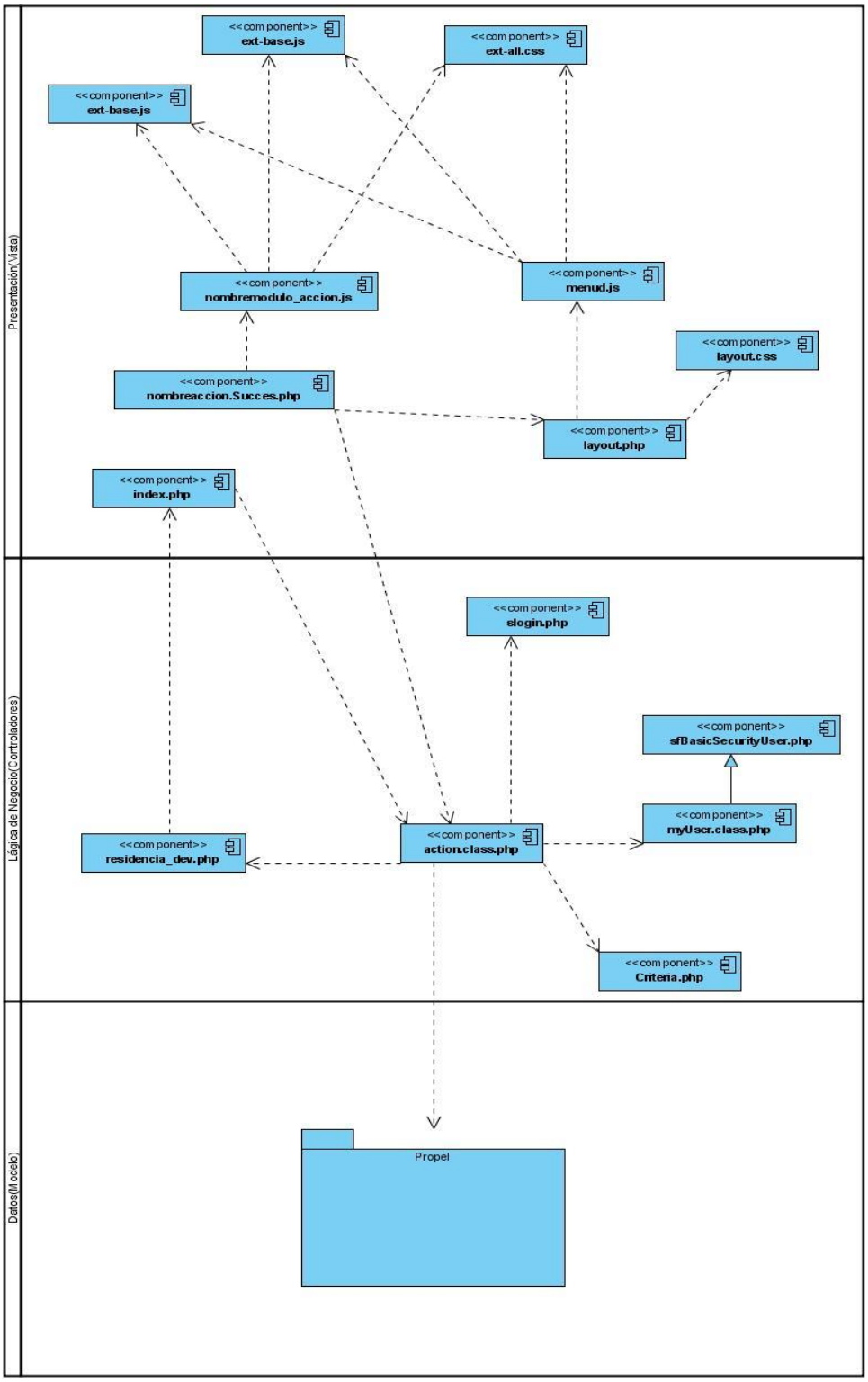


Figura 3.13 Diagramas de componentes del módulo de seguridad

3.6. VISTA DE DESPLIEGUE

La vista de despliegue suministra una base para la comprensión de la distribución física de un sistema a través de nodos, y propone cómo se satisfacen los requisitos no funcionales de software y hardware e influye en el rendimiento.

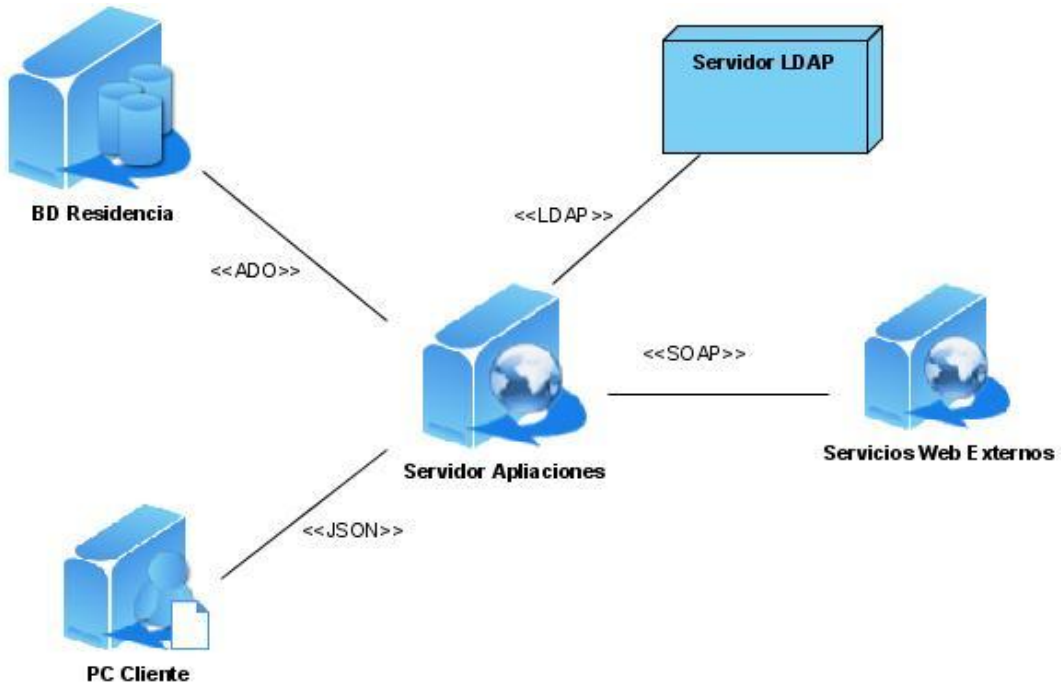


Figura 3.11 Diagramas de Despliegue

Nodo PC Cliente: Este ordenador corresponde con el puesto de trabajo de los clientes y del administrador del sistema.

Nodo BD Residencia: Servidor se base de datos en PostgreSQL con la base de datos de la aplicación.

Nodo Servidor de Aplicaciones: Nodo donde se encuentra la aplicación y donde se publican los servicios web propios de la aplicación ya sean de gestión o públicos.

Nodo Servicios Web Externos: Nodo donde se encuentran los servicios web externos con los que el sistema va a estar en constante intercambio de información.

Nodo Servidor LDAP: Como su nombre lo indica es el nodo al que la aplicación va acceder en el proceso de autenticación para consultar datos personales del usuario.

3.7. Conclusiones

En este capítulo describimos las vistas de la arquitectura, representando correctamente la estructura del sistema, mostrando sus partes fundamentales en forma de módulos y las principales relaciones que se establecen entre ellos. Las vistas documentadas describen los aspectos principales a tener en cuenta para el desarrollo del sistema. En este capítulo se cubren las necesidades de información de los interesados en el desarrollo del proyecto.

CONCLUSIONES

El diseño de una arquitectura de software debe considerarse una parte fundamental, crítica e imprescindible en el desarrollo de un sistema de software, ya que es precisamente en esta fase donde recae toda la creatividad, experiencia y creación de la propuesta de solución que más se adecue a las necesidades de nuestro cliente y le permita lograr sus objetivos.

Es importante que los sistemas de software estén respaldados por una arquitectura bien definida que responda a los requisitos funcionales y no funcionales establecidos para el sistema, que sea capaz de hacerlo evolucionar frente cambios tecnológicos, de fomentar la reutilización y de organizar el desarrollo.

Se realizó un estudio detallado de varios estilos y patrones arquitectónicos que existen en la actualidad para hacer uso de las mejores técnicas de diseño arquitectónico.

Se realizó un profundo estudio de arquitecturas de aplicaciones Web que utilizan servicios web para obtener la experiencia en el diseño de las mismas y adaptarlas a nuestra propuesta de solución.

Se investigaron las características de diferentes frameworks de PHP para el desarrollo de aplicaciones empresariales.

Se realizó un estudio de los diferentes componentes o frameworks de javascript, para manipular el ambiente de la presentación de aplicaciones web.

Se estudiaron las herramientas y metodologías y plataformas para darle soporte a nuestra propuesta de arquitectura durante el desarrollo de del sistema.

Se diseñó una arquitectura empresarial en PHP de tres capas que facilita el desarrollo paralelo del sistema, el mantenimiento y soporte de la aplicación al tratarse cada capa de forma individual. Se estructuró el sistema en módulos divididos por funcionalidades específicas, para facilitar el trabajo del equipo de desarrollo.

Se cumplieron todos los objetivos trazados en esta investigación y se dio respuesta a la pregunta planteada en el problema científico a partir de definir y elaborar una arquitectura empresarial en PHP.

RECOMENDACIONES

- Se recomienda aplicar los métodos de evaluación de la arquitectura para identificar las posibles debilidades.
- Se recomienda mantener un constante refinamiento de la arquitectura a lo largo del ciclo de desarrollo.
- Se recomienda que se continúe refinando los procedimientos del módulo se seguridad a lo largo del ciclo desarrollo.
- Se recomienda que se estudien otros frameworks javascript para el entorno de la presentación.
- Se recomienda investigar otros plugins o formas de integración de Symfony con herramientas para generar reportes.

Bibliografía

KAISER, S. H. Software Paradigms. 2005. p. 0471483478

Potencier, F. and F. Zaninotto. "*Symfony la guía definitiva"*, 2008.

Comunidad PHP (Universidad de las ciencias Informáticas), 2008.Disponible en: http://php.uci.cu/?q=node/158

Un poco de historia, Comunidad PHP (Universidad de las ciencias Informáticas), 2008. Disponible en: http://php.uci.cu/?q=node/9

Worsley, J. and Drake, J. PostgreSQL Práctico, 2008. Disponible en: http://www.sobl.org/traducciones/practical-postgres/node19.html

Rumbaugh, J. and Jacobson, I. and Booch, G. El lenguaje unificado de modelado, p 18.
Visual Paradigm for UML, 2008.Disponible en: http://www.visual-paradigm.com/product/vpuml/

ARQUITECTURA Modelo/Vista/Controlador, 2008. Disponible en: http://www.cica.es/formacion/JavaTut/Intro/tabla.html

Fowler, M. Enterprise Application Patterns, 2008. Disponible en: http://www.martinfowler.com/eaaCatalog

Zend Studio for Eclipse, 2008.Disponible en: http://www.zend.com/en/products/studio/

Visión general de las nuevas funcionalidades de Apache 2.0, 2008. Disponible en: http://httpd.apache.org/docs/2.0/es/new_features_2_0.html

Martín, M.A.G.Arquitectura Orientada a Servicios y los BPM,UCIENCIA, 2007.

J. Vlissides, J. Coplien, y N. Kerth, Some Patterns for Software Architecture, en Pattern Languages of Program Design, Vol. 2, pp. 255-269, 2008.

Referencias Bibliográficas:

(17).¿Arquitectura de Software?, 2008. Disponible en: http://siona.udea.edu.co/~aoviedo/Arquitectura%20de%20Software/Arquitectura%20de%20Software.htm#_Definiciones

(2)ALGUNOS TIPOS DE ARQUITECTURAS, 2008. Disponible en: http://homepage.mac.com/imaz/iblog/C612772037/E20050907222635/Media/Algunos%20Tipos%20de%20Arquitecturas.pdf

(3)Arquitectura de la IP: arquitectura, 2008. Disponible en: http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/arquitectura_soft.mspx

(4)Hidalgo .G, 2008. MODELACIÓN DEL SUBSISTEMA DE CONTROL DE LA INFORMACION GEOESPACIAL EN LINEA.

(5)Mendoza M. L. E, SISTEMAS DE INFORMACIÓN II TEORÍA, 2008, Disponible en: http://prof.usb.ve/lmendoza/Documentos/PS-6116/Teor%EDa%20PS6116%20Reingenier%EDa.pdf

(6)Pimentel, LA y Pérez. I, 2008, “ArBaWeb: ARQUITECTURA BASE SOBRE LA WEB”.Disponible en: *http://bibliodoc.uci.cu/TD/TD_0465_07.pdf*

(7)Symfony en pocas palabras, 2008. Disponible en: http://www.librosweb.es/symfony/capitulo1/symfony_en_pocas_palabras.htm

(8)El ataque de los frameworks, 2008.Disponible en: http://www.webtaller.com/maletin/articulos/el-ataque-de-los-frameworks.php

(9)Lellelid.H Propel –Guía de usuario, 2008, Disponible en: http://propel.phpdb.org/docs/es/user_guide/chapters/Introduction.html

(10)Zend Studio, 2008. Disponible en: http://www.desarrolloweb.com/articulos/1178.php

(11)Collins-Sussman, B. Fitzpatrick, W.B. Pilato, M.C, Control de versiones con Subversion, 2008. Disponible en: http://svnbook.red-bean.com/nightly/es/svn-book.pdf

(12)Sistema de gestión de base de datos, Disponible en: http://www.igac.gov.co:8080/igac_web/UserFiles/File/ciaf/TutorialSIG_2005_26_02/paginas/ctr_sistemasdegestiondebasededatos.htm

(13)PDT Project, 2008.Disponible en: http://www.eclipse.org/pdt/

(14)ExtJS 2.0, 2008.Disponible en: http://extjs.com/

(15) Estilos y Patrones en la Estrategia de Arquitectura de Microsoft (Versión 1.0 – Marzo de 2004) Carlos Reynoso – Nicolás Kiccillof, 2008.

(16) Kruchten P, Planos Arquitectónicos: El Modelo de “4+1” Vistas de la Arquitectura del Software, *2008.*

(17) KAISER, S. H. Software Paradigms. 2005. p. 0471483478.

GLOSARIO DE TÉRMINOS

AJAX: acrónimo de ***A****synchronous* ***J****avaScript* ***A****nd* ***X****ML* (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas o **RIA** (Rich Internet Applications). Éstas se ejecutan en el cliente, es decir, en el navegador de los usuarios y mantiene comunicación asíncrona con el servidor en segundo plano.

GUI: (Graphical User Interface): Interfaz Grafica de Usuario es un tipo de interfaz de usuario que utiliza un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Habitualmente las acciones se realizan mediante manipulación directa para facilitar la interacción del usuario con la computadora.

HTTP: (HyperText Transfer Protocol): protocolo de transferencia de hipertexto, es el protocolo usado en cada transacción de la Web (WWW). El hipertexto es el contenido de las páginas web, y el protocolo de transferencia es el sistema mediante el cual se envían las peticiones de acceso a una página y la respuesta con el contenido.

HTML: siglas de HyperText Markup Language (*Lenguaje de Marcado de Hipertexto*), es el lenguaje de marcado predominante para la construcción de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

Herramientas CASE: (***C****omputer* ***A****ided* ***S****oftware* ***E****ngineering*, Ingeniería de Software Asistida por Ordenador) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero

JavaScript: es un lenguaje de programación interpretado, es decir, que no requiere compilación, utilizado principalmente en páginas web.

jQuery: es un nuevo tipo de librerías de Javascript que permite simplificar la manera de interactuar con los documentos HTML, permite manejar eventos, desarrollar animaciones, y agregar interacción con la tecnología AJAX a nuestras páginas web.

JSON: acrónimo de "**J**ava**S**cript **O**bject **N**otation", es un formato ligero para el intercambio de datos. **JSON** es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML

LDAP: (*Lightweight Directory Access Protocol*) es un protocolo a nivel de aplicación que permite el acceso a un servicio de directorio ordenado y distribuido para buscar diversa información en un entorno de red. LDAP también es considerado una base de datos (aunque su sistema de almacenamiento puede ser diferente) a la que pueden realizarse consultas.

PHP: es un acrónimo recursivo que significa ***P****HP* ***H****ypertext* ***P****re-processor* (inicialmente PHP Tools, o, *P*ersonal *H*ome *P*age Tools). Es un lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas.

Prototype: es un *framework* escrito en JavaScript que se orienta al desarrollo sencillo y dinámico de aplicaciones web. Es una herramienta que implementa las técnicas AJAX y su potencial es aprovechado al máximo cuando se desarrolla con Ruby On Rails.

Plugin: Un plugin o *componente enchufable* (o plug-in -en inglés "enchufar"-, también conocido como addin, add-in, addon o add-on) es una aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica, generalmente muy específica, como por ejemplo servir como driver (controlador) en una aplicación, para hacer así funcionar un dispositivo en otro programa. Ésta aplicación adicional es ejecutada por la aplicación principal. Los plugins típicos tienen la función de reproducir determinados formatos de gráficos, reproducir datos multimedia, codificar/decodificar emails, filtrar imágenes de programas gráficos.

Servicio web: (en inglés *Web service*) es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet. La interoperabilidad se consigue mediante la adopción de estándares abiertos.

Servicio: Una función sin estado (Existen servicios asíncronos en los que una solicitud a un servicio crea, por ejemplo, un archivo, y en una segunda solicitud se obtiene ese archivo), auto-contenida, que acepta una(s) llamada(s) y devuelve una(s) respuesta(s) mediante una interfaz bien definida. Los servicios pueden también ejecutar unidades discretas de trabajo como serían editar y procesar una transacción. Los servicios no dependen del estado de otras funciones o procesos. *La tecnología concreta utilizada para prestar el servicio no es parte de esta definición*.

SMTP: (Simple Mail Transfer Protocol)**,** protocolo simple de transferencia de correo. Protocolo de red basado en texto utilizado para el intercambio de mensajes de correo electrónico entre computadoras o distintos dispositivos.

Unix: (registrado oficialmente como **UNIX**®) es un sistema operativo portable, multitarea y multiusuario; desarrollado, en principio, en 1969 por un grupo de empleados de los laboratorios Bell de AT&T, entre los que figuran Ken Thompson, Dennis Ritchie y Douglas McIlroy.

UDDI: son las siglas del catálogo de negocios de Internet denominado *Universal Description, Discovery and Integration*. El registro en el catálogo se hace en XML. UDDI es una iniciativa industrial abierta (sufragada por la OASIS) entroncada en el contexto de los servicios Web.

Web: (" World Wide Web ") o Red Global Mundial es un sistema de documentos de hipertexto y/o hipermedios enlazados y accesibles a través de Internet. Con un navegador Web, un usuario visualiza páginas web que pueden contener texto, imágenes, vídeos u otros contenidos multimedia, y navega a través de ellas usando hiperenlaces.

W3C: World Wide Web Consortium es un consorcio internacional que produce estándares para la World Wide Web. Está dirigida por Tim Berners-Lee, el creador original de URL (*Uniform Resource Locator*, Localizador Uniforme de Recursos), HTTP (*HyperText Transfer Protocol*, Protocolo de Transferencia de Hipertexto) y HTML (Lenguaje de Marcado de Hipertexto) que son las principales tecnologías sobre las que se basa la Web.

XML: sigla en inglés de ***Ex****tensible* ***M****arkup* ***L****anguage* («lenguaje de marcas extensible»), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

XML: Schema(**XSD**) es un lenguaje de esquema utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML de una forma muy precisa, más allá de las normas sintácticas impuestas por el propio lenguaje XML. Se consigue así, una percepción del tipo de documento con un nivel alto de abstracción. Fue desarrollado por el World Wide Web Consortium (W3C) y alcanzó el nivel de recomendación en mayo de 2001.