Universidad de las Ciencias Informáticas

Facultad 8

"Arquitectura de la versión multiplataforma de la colección de software educativo El Navegante"

Trabajo de diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores:

Anays Jorge Díaz

Ediel Cardona Marrero

Tutores:

Lic. Hector Matías González

Ing. Edier García Gutiérrez

Ciudad de La Habana

Junio 2010

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 2010.

_____
Anays Jorge Díaz
Firma del Autor

_____
Ediel Cardona Marrero
Firma del Autor

_____
Lic. Héctor Matías González
Firma del Tutor

_____
Ing. Edier García Gutiérrez
Firma del Tutor

Agradecimientos

A mis padres, con la realización de esta tesis no solo se cumple un sueño, este es el resultado de años de sacrificio, del esfuerzo de mis padres; quienes me han dado de sí, lo mejor. A ellos les debo quien soy, gracias por confiar en mí.

A mi hermana quien ha sido mi fuente de inspiración, por estar siempre ahí; en los buenos y malos momentos.

A mima y pipo (Cigilia y Toribio) quienes han sido los abuelos que no conocí, por haberme dado de sí lo mejor.

A toda mi familia en general, que siempre ha estado atenta a mis logros y preocupada por mis fiascos.

A mis amigas incondicionales, Yurima, Yolanda, Yoliernis quienes me han dado lo mejor de sí, me han aceptado como soy y donde siempre he encontrado un hombro en situaciones difíciles.

A mi compañero de tesis Ediel por soportar mis majaderías y tardanzas.

A mis amigos de la Universidad a quienes jamás olvidaré: Alexey y Elvis, quienes vienen conmigo desde el inicio, demasiadas fechorías para lograr olvidarlas.

A mis amigas que avanzada mi carrera se unieron a mi viaje Lianne y Martha.

A todos mis compañeros de aula y del apartamento.

A los tutores Hector y Edier, también a Liana que sin ser tutora asumió la responsabilidad de guiarnos en el trabajo.

A todos los que de un modo u otro han influido en mi vida durante el tránsito por la Universidad y siempre han estado al tanto de mí.
En fin a todas aquellas personas que quiero y me quieren les agradezco.

Anays.

A mis padres que se han sacrificado para que yo haya podido alcanzar este logro.
A mi novia por apoyarme estos 5 años y por estar ahí para mí siempre que la he necesitado.
A toda mi familia que de una forma u otra siempre me ha ayudado.
A los tutores Hector y Edier, a Liana que nos dio una gran ayuda y apoyo.
A todas mis amistades que han compartido conmigo estos años.
Muchas gracias a todos.

Ediel

Dedicatoria

A mis padres Jesús y Aracelys, por haberme apoyado más que en mi carrera en todos los años vividos, a mi hermana por siempre tener un consejo y a mi sobrino que amo.

Anays.

A mis padres que han estado apoyándome durante toda mi vida brindándome su amor, a mi hermano que espero ser un ejemplo para él, a mi novia que ha estado conmigo en todos los momentos dándome su cariño, a mi suegra que me ha acogido como un hijo.

Ediel

Resumen

El desarrollo de software educativo en Cuba se ha visto incrementado en los últimos años, como parte del auge tomado también a nivel mundial. Varias son las colecciones de software desarrolladas por el Ministerio de Educación, entre ellas hay una llamada “El Navegante”, que está dirigida a la enseñanza secundaria.

Propiciado por un acuerdo que persigue la utilización de la colección “El Navegante” en la República Bolivariana de Venezuela, se hace necesario el desarrollo de una nueva versión multiplataforma de la Colección para la web que sea flexible, eficiente, segura y organizada para la futura continuidad del desarrollo. Es preciso entonces la elaboración de una arquitectura adecuada que responda a las exigencias de la nueva versión.

El presente trabajo define la arquitectura de software que soportará la versión multiplataforma de la colección “El Navegante”, utilizando para ello, los flujos propuestos en la metodología RUP. La definición comprende los estilos, patrones y herramientas a utilizar, así como la estructura básica que comparten los productos de la Colección.

La arquitectura es validada mediante la técnica de evaluación basada en escenarios, aplicando el método ATAM y usando como instrumento el Árbol de Utilidades, permitiendo identificar tanto los puntos de riesgos como las fortalezas y debilidades de la misma. Con la realización de la evaluación se puede constatar la eficiencia del trabajo realizado y la obtención de una arquitectura óptima, que resuelve los problemas de la actual versión instalada en las escuelas secundarias del país y que imposibilitan su uso en la República Bolivariana de Venezuela.

Palabras claves: colección “El Navegante'', Arquitectura de Software, software educativo

Índice

Introducción

Las Tecnologías de la Información y las Comunicaciones (TIC) se han convertido en el vehículo común en el tránsito de la vida social de todos. La esfera de la educación no está excluida de todas las trasformaciones que estas nuevas tecnologías traen consigo, puesto que ha quedado evidenciada la importancia de vincular las mismas al proceso de enseñanza, entre otros aspectos, porque posibilita nuevos procesos de aprendizaje y transmisión del conocimiento a través de las redes telemáticas y brinda nuevos instrumentos y métodos para los procesos educativos.

Todo esto hace que un objetivo priorizado de la política nacional informática en Cuba lo constituya la utilización de la computación en la enseñanza, las investigaciones científicas y la gestión docente.

Desde el triunfo de la Revolución Cubana en 1959, el país se planteó un camino de desarrollo en el que se resolvieran los problemas materiales y espirituales de su población. Dentro de los innumerables cambios que tuvieron lugar, específicamente en la esfera de la educación; se trazó la meta de vincular las TIC a la educación en todos los niveles de enseñanza.

A partir de la asignación de un fondo financiero significativo en 1984, por parte del Gobierno, fue posible la adquisición de un gran número de microcomputadores que conllevaron a un amplio y acelerado proceso del uso de las TIC en los diferentes niveles educacionales. Para que dichos cambios tuvieran el resultado esperado era necesario la aplicación de medidas que hicieran posible el correcto uso de las nuevas tecnologías, por lo que se capacitaron a los profesores de la enseñanza primaria para que fueran capaces de impartir computación a sus alumnos; y de este modo iniciar la enseñanza de la computación desde la Educación Primaria. A su vez, paralelo al proceso de incorporar las computadoras a los diferentes niveles de enseñanza, comienzan a desarrollarse múltiples sistemas de diversas características que apoyan el proceso docente-educativo, estos son los que se conocen como software educativos (SWE). Conscientes del marcado paso de avance y el papel que estos desempeñan en el sistema educativo, centros docente de la producción y los servicios, entre los que se pueden mencionar a la Universidad de La Habana, el Instituto Superior Politécnico José Antonio Echeverría, el CEDISAC, el CENSAI y el caso propio de la Universidad de las Ciencias Informáticas (UCI), dieron una nueva dirección a su trabajo con vistas a promover el desarrollo de los mismos. En la UCI, actualmente se desarrollan estos software a los que se hace referencia, con el fin de

satisfacer tanto la demanda del país como los contratos que son firmados en el ámbito internacional.

En diciembre del año 2009 como resultado de la X Convención Mixta Cuba-Venezuela, surge el proyecto “Colecciones de Software Educativo Multisaber y El Navegante para Planteles de los niveles primaria y secundaria del Sistema Educativo”, que consiste en el desarrollo de 24 productos basados en las colecciones educativas, “Multisaber” con 14 y “El Navegante” con 10, ambas colecciones usadas actualmente en las escuelas cubanas.

La colección “El Navegante” está destinada a la enseñanza secundaria, los software educativos presentes en ésta abordan varias asignaturas. Los nombres de los software o productos, como también se les conocen, así como las asignaturas que tratan, se muestran en la figura 1.

Título del software	Asignaturas	7mo	8vo	9no
Elementos matemáticos	Matemática	X	X	X
El fabuloso mundo de las palabras	Literatura y Español	X	X	X
La naturaleza y el hombre	Geografía, Biología, Física, Química	X	X	X
Rainbow	Inglés	X	X	X
Informática Básica	Informática Básica	X	X	
Encuentro con el pasado	Historia Antigua y Medieval	X		
EduArte	Educación Artística	X		
GeoClío	Historia Moderna y Contemporánea		X	
Por los senderos de mi patria	Historia de Cuba y Arte cubano			X
Aprende construyendo	Educación Laboral y Dibujo Básico			X

Fig.1 Listado de productos de la colección “El Navegante”

Los productos que integran la colección “El Navegante” están estructurados didácticamente en 6 módulos básicos según la concepción ideada por especialistas del Ministerio de Educación de Cuba, y son una mezcla de diferentes elementos que representan las diversas tipologías de software educativo, a este tipo de software educativo se le denomina hiperentornos de aprendizaje o hiperentornos educativo.

La utilización de los productos de la Colección en Venezuela implica la corrección de aspectos que constituyen barreras de uso para los venezolanos y mejoras a los productos actuales, los cuales son:

- Los contenidos, diseño gráfico y funcionamiento no están adecuados al entorno venezolano.

- No existe homogeneidad en todos los productos en cuanto a navegación, diseño y funcionamiento.
- Los recursos multimedia utilizados en los productos, refiérase, imágenes, videos, textos, y sonidos, no poseen los permisos legales necesarios para su utilización en el software.
- Las tecnologías usadas en el desarrollo de los productos no se ajustan al Decreto Ley 3.390 de la República Bolivariana de Venezuela, el cual establece el uso prioritario de software libre.
- Los software no funcionan sobre el sistema operativo Linux.
- Los contenidos están incrustados dentro de los software por lo que se hace muy difícil la gestión y actualización de los mismos.

La necesidad de corregir los problemas mencionados y la inexistencia de una aplicación flexible, de alto rendimiento y permisible a desarrollos futuros, conllevan al siguiente **problema investigativo:** ¿Cómo garantizar la estructura de la versión multiplataforma, contextualizada al entorno venezolano, de la colección de software educativo "El Navegante"?

Para resolver el problema planteado el **objetivo general** que se persigue con el presente trabajo de diploma es definir la arquitectura de la versión multiplataforma, contextualizada al entorno venezolano, de la colección de software educativo "El Navegante".

El **objeto de estudio** de la investigación se centra en la Arquitectura de Software. De aquí se deriva que el **campo de acción** sea específicamente la arquitectura de la colección "El Navegante".

Según el objetivo general los **objetivos específicos** serían:

1. Realizar un estudio valorativo de las tendencias y tecnologías actuales acerca de los modelos arquitectónicos para el desarrollo.
2. Definir la línea base de la arquitectura.
3. Evaluar la arquitectura definida.

Para dar cumplimiento a los objetivos trazados se definieron las siguientes **tareas de la investigación**:

- Investigar los diferentes estilos y patrones arquitectónicos y de diseño existentes.
- Seleccionar los estilos y patrones a utilizar en el desarrollo de la Colección.

- Seleccionar las herramientas y tecnologías adecuadas que cumplan los requisitos para ser utilizadas en el desarrollo.
- Seleccionar los requerimientos críticos.
- Definir la arquitectura candidata según la metodología y las herramientas utilizadas.
- Implementar las bases estructurales de los patrones seleccionados.
- Implementar casos de estudio que demuestren la factibilidad de integración de las librerías y herramientas seleccionadas.
- Elaborar las vistas de la arquitectura.
- Evaluar la arquitectura definida.

El **resultado** que se espera obtener con la puesta en práctica de este trabajo es la definición de la arquitectura de la versión multiplataforma, contextualizada al entorno venezolano, de la colección de software educativo “El Navegante”.

El contenido del trabajo está estructurado de la siguiente manera:
Capítulo 1: Fundamentación teórica: En este capítulo se aborda en detalle lo relacionado con la fundamentación teórica que sustenta el presente trabajo. Descripción de los conceptos básicos de la Arquitectura de Software. Selección de los patrones y estilos arquitectónicos a utilizar en el desarrollo de la arquitectura, artefactos, metodología y herramientas que se emplearán para el desarrollo de la nueva versión de “El Navegante”.

Capítulo 2: Descripción de la arquitectura: Se explican las características del sistema propuesto. Se identifican los componentes arquitectónicos significativos y se representan las características del sistema en las vistas arquitectónicas.

Capítulo 3: Evaluación de la arquitectura: Se seleccionan los métodos de evaluación y atributos de calidad más adecuados. Se especifican las ventajas, riesgos de los diseños y se muestran los resultados de la evaluación luego de emplear tales métodos.

Capítulo 1: Fundamentación Teórica

1.1 Introducción

En el presente capítulo se realiza el estudio de todo el marco teórico referente al tema de la Arquitectura de Software para el desarrollo de una aplicación web, sus inicios, tendencias, así como una breve descripción de los principales conceptos asociados a ésta. Se analizan y seleccionan las metodologías, herramientas y tecnologías a utilizar en el desarrollo de las tareas definidas para dar cumplimiento a los objetivos específicos.

1.2 Análisis de soluciones existentes

Durante la investigación realizada se ha encontrado bibliografía referida al desarrollo de hiperentornos de aprendizaje o hiperentornos educativos en el ámbito internacional, no así referencias que hagan alusión a la existencia de alguna arquitectura de este tipo de software. En Cuba "El Navegante" es uno de los productos pioneros con estas características, y en la investigación llevada a cabo en busca de otros productos que presentaran una arquitectura similar que pudiera servir de guía o referencia para el desarrollo de la arquitectura de “El Navegante”, fueron consultados los proyectos existentes en la UCI, Alfaomega y Multisaber. En el momento de la entrevista no se obtuvo ningún resultado satisfactorio pues estos proyectos están en proceso de desarrollo y no tenían una arquitectura definida aún.

1.3 Historia y Surgimiento de la Arquitectura de Software

Los antecedentes de la Arquitectura de Software se remontan a la década de 1960, su historia no ha sido tan continua como la del campo más amplio en el que se inscribe, la Ingeniería del Software. Entre los primeros científicos en hacer planteamientos que se acercaran a lo que hoy se conoce como Arquitectura de Software se destaca Edsger Dijkstra de la Universidad Tecnológica de Holanda (1968), quien propuso que se hiciera una estructuración correcta de los sistemas de software antes de lanzarse a programar. Más tarde Fred Brooks Jr y Ken Iverson, en 1969, llamaban arquitectura a la estructura conceptual de un sistema en la perspectiva del programador. En 1975 Brooks utilizaba el concepto de arquitectura del sistema para designar “la especificación completa y detallada de la interfaz de usuario” y consideraba que el arquitecto es un agente del usuario, igual que lo es quien diseña su casa (1), empleando una nomenclatura que ya nadie aplica de ese modo. Contemporáneo a éstos, otro precursor importante, David Parnas, demostró que los criterios seleccionados en la descomposición de un sistema impactan en la

estructura de los programas y propuso diversos principios de diseño que debían seguirse a fin de obtener una estructura adecuada.

En la década de 1980 fueron perfeccionadas las técnicas descriptivas, las notaciones formales y para la caracterización de lo que sucedería en la siguiente década, ellos formulan esta otra frase que ha quedado inscrita en la historia de la especialidad: "La década de 1990, creemos, será la década de la Arquitectura de Software…" (2). La Arquitectura de Software quedó en estado de vida latente durante unos cuantos años, hasta comenzar su expansión explosiva con los manifiestos de Dewayne Perry de AT&TBell Laboratorios de New Jersey y Alexander Wolf de la Universidad de Colorado. Puede decirse que Perry y Wolf fundaron la disciplina, puesto que el primer estudio en que aparece la expresión Arquitectura de Software como se conoce hoy fue realizado por ellos en 1992.

La Arquitectura de Software se encuentra en una etapa de formación constante y están surgiendo nuevos aportes que desarrollan y amplían la disciplina.

1.4 Definición de Arquitectura de Software

Existen varios conceptos acerca de la Arquitectura de Software desde el punto de vista de sus autores, ya que cada uno de éstos aporta ideas que van enriqueciendo su definición; pero aun cuando se pueden hallar cientos de enunciados sobre qué es la Arquitectura de Software las ideas están centradas en las tres variantes siguientes:

- Arquitectura de Software como un proceso dentro del ciclo de vida de un sistema.
- Arquitectura de Software como la forma de articular los diferentes estilos y componentes dentro de una solución, es decir, como la Topología del Sistema.
- Arquitectura de Software como una disciplina profesional y académica.

Es muy habitual encontrar varias definiciones referentes a este tema, sin embargo, ya desde el año 2000 la IEEE publicó en su documento IEEE 1471 la definición oficial para ésta:

"La arquitectura de software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orienten su diseño y evolución" (3).

Esta definición deja claro que la Arquitectura de Software no se inscribe en ninguna metodología de desarrollo, sino que es en sí una disciplina que evoluciona durante todo el ciclo de desarrollo de software.

1.5 Importancia de la Arquitectura de Software

La Arquitectura de Software es de vital importancia en el desarrollo de un software, ya que abarca todo el proceso para dar distribución y orden a los elementos de un sistema informático, desde los requerimientos que debe satisfacer el mismo, las restricciones a las que está sujeto, hasta las propiedades no funcionales y las decisiones de diseño que gobiernan esta estructura.

Su importancia se evidencia en los siguientes aspectos:

Comunicación mutua: Los desarrolladores de un software pueden utilizar la Arquitectura de Software como base para crear un entendimiento mutuo y comunicarse entre sí. Esto es muy importante para tomar futuras decisiones y formar un consenso común respecto a éstas.

Decisiones tempranas de diseño: Mediante la Arquitectura de Software se toman decisiones tempranas del diseño sobre un sistema, lo cual tiene un peso importantísimo para la mitigación de riesgos potenciales, evitando que ocurran futuros desastres a gran escala.

Representaciones constructivas: Una descripción arquitectónica proporciona diagramas que brindan una representación del sistema, indicando los componentes y las dependencias entre ellos, los cuales constituyen una guía para el desarrollo.

Reutilización: La Arquitectura de Software promueve la reutilización a gran escala de una cantidad importante de componentes y frameworks. Esto reduce los costos de diseño y la cantidad de código se simplifica.

Evolución: La Arquitectura de Software estima los posibles cambios y los costos de las modificaciones a las que se puede someter un sistema, permitiendo comprender el grado de mejoramiento que éste puede alcanzar.

1.6 Estilos arquitectónicos

1.6.1 Definición

Un estilo arquitectónico define una familia de sistemas en términos de un patrón de organización estructural. En particular, según los autores, un estilo arquitectónico define tanto un vocabulario de tipos de componentes y conectores (como en el caso de filtros y tubos), como un conjunto de restricciones sobre cómo combinar esos componentes y conectores:

- Sirven para sintetizar estructuras de soluciones.

- Pocos estilos abstractos encapsulan una enorme variedad de configuraciones concretas.
- Definen los patrones posibles de las aplicaciones.
- Permiten evaluar arquitecturas alternativas con ventajas y desventajas conocidas ante diferentes conjuntos de requerimientos no funcionales.

1.6.2 Clases de Estilos arquitectónicos

Los principales estilos arquitectónicos que se usan en la actualidad están divididos por Clases de Estilos que engloban una serie de estilos arquitectónicos específicos:

Estilos de Flujo de Datos

- Tubería y filtros

Estilos Centrados en Datos

- Arquitectura de Pizarra o Repositorio

Estilos de Llamada y Retorno

- Modelo Vista Controlador (MVC)

Arquitecturas en Capas

- Arquitecturas Orientadas a Objetos
- Arquitecturas Basadas en Componentes

Estilos de Código Móvil

- Arquitectura de Máquinas Virtuales

Estilos Heterogéneos

- Sistemas Control de Procesos
- Arquitecturas Basadas en Atributos

Estilos Peer-to-Peer

- Arquitecturas Basadas en Eventos
- Arquitecturas Orientadas a Servicios
- Arquitecturas Basadas en Recursos

1.6.3 Estilos de Llamada y Retorno

Esta familia de estilos enfatiza la modificabilidad y la escalabilidad porque son más generalizadores en sistemas de gran escala. Dentro de los miembros de la familia se encuentran las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objetos y los sistemas jerárquicos en capas.

- **Modelo Vista Controladora (MVC)**

Se utiliza principalmente cuando es necesario modularizar la interfaz de usuario, las reglas de negocio y el control de eventos. Separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes:

Modelo: Administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado. Mantiene el conocimiento del sistema. No depende de ninguna vista y de ningún controlador.

Vista: Maneja la visualización de la información.

Controlador: Interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado. Tiene tres variantes principales: Activa, Pasiva y Documento-Vista.

- **Arquitecturas en Capas**

Este estilo se define como una organización jerárquica, tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. Al dividir un sistema en capas, cada capa puede tratarse de forma independiente, sin tener que conocer los detalles de las demás. La división de un sistema en capas facilita el diseño modular, en la que cada capa encapsula un aspecto concreto del sistema y permite además la construcción de sistemas débilmente acoplados, lo que significa que si se minimiza las dependencias entre capas, resulta más fácil sustituir la implementación de una capa sin afectar al resto del sistema. (2)

- **Arquitecturas Orientadas a Objetos**

Los componentes de este estilo son los objetos, o más bien instancias de los tipos de datos abstractos. Los objetos representan una clase de componentes denominada manager (administradores), debido a que son responsables de preservar la integridad de su propia representación. Un rasgo importante de este aspecto es que la representación interna de un objeto no es accesible desde otros objetos. (4)

- **Arquitecturas Basadas en Componentes**

Los sistemas de software basados en componentes se basan en principios definidos por una ingeniería de software específica, los componentes son las unidades de modelado, diseño e implementación. Las interfaces están separadas de las implementaciones y

conjuntamente sus interacciones son el centro de incumbencias en el diseño arquitectónico. Las funcionalidades y propiedades de los componentes pueden ser descubiertas y utilizadas en tiempo de ejecución. (4)

1.7 Patrones

Los patrones ayudan a construir la experiencia colectiva de Ingeniería de Software. Son una abstracción de "problema-solución" y se ocupan de problemas recurrentes. Identifican y especifican abstracciones de niveles más altos de componentes o clases individuales y también proporcionan un vocabulario y entendimiento común.

Cada patrón describe un problema que ocurre una y otra vez en el ambiente, y luego describe el núcleo de la solución de este problema, de tal manera que puede usar esa solución un millón de veces más, sin hacer jamás la misma cosa dos veces.

1.7.1 Clasificación de los Patrones

Los patrones pueden clasificarse o dividirse en:

Patrones de Arquitectura: Relacionados a la interacción de objetos dentro o entre niveles arquitectónicos, problemas arquitectónicos, adaptabilidad a requerimientos cambiantes, performance, modularidad, acoplamiento. (3)

Patrones de Diseño: Conceptos de ciencia de computación en general, independiente de aplicación. Fueron creados dados los problemas de diseño, multiplicación de clases, adaptabilidad a requerimientos cambiantes proponiendo como solución: Comportamiento de factoría, Clase-Responsabilidad-Contrato (CRC). En esta clasificación se encuentran los patrones GRAPS (*General Responsability Asignment Software Patterns*) que describen los principios fundamentales de la asignación de responsabilidades a objetos y los GoF (*Gans of Four*) que enmarcan los llamados patrones de diseño Estructurales, Creacionales y de Comportamiento. (3)

Patrones de Análisis: Usualmente específicos de aplicación o industria. Resuelve problemas de modelado del dominio, completitud, integración, equilibrio de objetivos múltiples y de planeamiento para capacidades adicionales comunes. (3)

Patrones de Proceso o de Organización: Tratan todo lo relacionado con el desarrollo o los procesos de administración de proyectos, o técnicas, o estructuras de organización, resolviendo problemas de productividad, comunicación, efectividad y eficiencia. (3)

Patrones de Idioma: Son estándares de codificación y proyecto, creados para resolver las operaciones comunes bien conocidas en un nuevo ambiente o a través de un grupo, la legibilidad y la predictibilidad. (3)

Los elementos de un patrón son:

Nombre:

- Define un vocabulario de diseño.
- Facilita la abstracción.

Problema:

- Describe cuándo aplicar el patrón.
- Conjunto de fuerzas: objetivas y restricciones.
- Prerrequisitos.

Solución:

- Elementos que constituyen el diseño (plantilla).
- Forma canónica para resolver fuerzas.

Consecuencias:

- Resultados.
- Extensiones.
- Consensos.

1.7.2 Patrón Modelo Vista Controlador (MVC)

El patrón arquitectónico Modelo-Vista-Controlador (MVC) divide una aplicación interactiva en tres componentes. El modelo contiene la manipulación de datos, la vista muestra la información al usuario y el controlador maneja las peticiones de usuarios. La vista y el controlador en conjunto comprenden la interfaz de usuario, un mecanismo de cambio-propagación garantiza la coherencia entre ésta y el modelo.

Contexto: Aplicaciones interactivas con una interfaz humano-computadora flexible.

Problema: Las interfaces de usuarios son especialmente propensas a pedidos de cambios. Al extender la funcionalidad de una aplicación, se debe modificar los menús para acceder a estas nuevas funciones. Un cliente puede exigir una adaptación de interfaz específica, o un sistema puede necesitar ser transferido a otra plataforma. Incluso la actualización a una nueva versión del sistema puede implicar cambios en el código. Debido a esto las interfaces de usuario de los sistemas de larga duración pueden cambiar en muchas ocasiones.

Los diferentes usuarios establecen requisitos contradictorios sobre la interfaz de usuario. En consecuencia, el soporte a varios paradigmas de interfaz de usuario debe ser fácilmente incorporado.

La construcción de un sistema con la flexibilidad necesaria es costosa y propensa a errores si la interfaz de usuario está estrechamente entrelazada con el núcleo funcional. Esto puede resultar en la necesidad de desarrollar y mantener varios sistemas de software sustancialmente diferentes, uno para cada implementación de interfaz de usuario. La solución debe satisfacer los siguientes aspectos:

- La misma información se presenta de manera diferente en distintas ventanas, por ejemplo, en una gráfica circular o de barra.
- El comportamiento y la vista de la aplicación deben reflejar las manipulaciones de los datos inmediatamente.
- Deben ser fáciles y posibles los cambios en la interfaz de usuario, incluso en tiempo de ejecución.
- Portar la interfaz de usuario no debe afectar al código en el núcleo de la aplicación.

Solución: MVC se introdujo por primera vez en el medio ambiente de programación Smalltalk-80. Divide una aplicación interactiva en tres áreas: procesamiento, salida y entrada.

El modelo encapsula la manipulación de los datos y es independiente de las representaciones específicas o del comportamiento de entrada.

La vista obtiene los datos del modelo y muestra la información al usuario. Puede haber múltiples vistas del mismo modelo.

Cada vista tiene un controlador asociado, los controladores reciben las entradas, por lo general como eventos del movimiento del ratón y la activación de los botones del ratón o del teclado. El usuario interactúa con el sistema únicamente a través de los controladores.

La separación del modelo, de la vista y el controlador permite múltiples vistas del mismo modelo. Si el usuario cambia el modelo a través del controlador y una vista, todas las otras vistas que dependen de estos datos deben reflejar los cambios. El modelo por lo tanto notifica a todas las vistas cada vez que sus datos cambian, éstas recuperan los nuevos datos del modelo y actualizan la información que se muestra.

1.8 Estilos y Patrones Arquitectónicos

Se estima 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 marco de trabajo a partir de los cuales los arquitectos pueden construir patrones de diseño para resolver problemas específicos. Algunos de los patrones coinciden con los estilos hasta en el nombre con que se les designa. La diferencia entre los estilos y los patrones arquitectónicos está en el número y el nivel de abstracción en el que cada uno se mueve. 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 se están utilizando, solo se habla de filtros, tubos u objetos sin interesarnos lo que están representando. En el caso de los patrones, se tienen en cuenta el significado de estos elementos a los que se hacía referencia anteriormente.

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 reutilización y los estilos conciernen a teorías sobre las estructuras de los sistemas a veces más formales que concretas. (5)

Por lo que se puede concluir 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. 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.

1.9 Entorno de desarrollo

Los entornos de desarrollo de software son herramientas y tecnologías que ayudan a los programadores a desarrollar software sobre entornos más amigables. Es decir, aquellos en los que el programador puede acceder con el menor esfuerzo a diferentes recursos como editores, compiladores, herramientas de análisis, etcétera. (6)

Es de gran importancia la selección de un ambiente adecuado para poder explotar todos los beneficios de éste y agilizar el desarrollo del software. A continuación se abordan las herramientas y tecnologías a utilizar, teniendo en cuenta que sean libres, exponiendo sus características y ventajas para poder tener un mayor conocimiento de sus prestaciones y valorar su utilidad.

1.9.1 Metodología de desarrollo de software

Durante el desarrollo de un software, en un momento determinado, se debe definir una metodología. Todo desarrollo de software es riesgoso y difícil de controlar, y si no se utiliza una metodología, lo que se obtiene son clientes insatisfechos con el resultado y desarrolladores aún más insatisfechos.

En el mundo existen varias metodologías con diferentes métodos y técnicas para guiar el desarrollo de software. Escoger la que más se ajuste es muy importante para organizar el proceso, por lo que es necesario investigar sobre algunas de éstas en aras de seleccionar la que resulte más factible para el desarrollo de la arquitectura de la versión multiplataforma de la colección de software educativo "El Navegante". Lo más importante antes de elegir la metodología que se usará para la implementación del software, es establecer el alcance que tendrá y luego determinar la que más se adapta a la aplicación.

1.9.1.1 Programación Extrema (XP)

La Programación Extrema es una metodología ligera de desarrollo de software que se basa en la simplicidad, la comunicación y la realimentación o reutilización del código desarrollado. La metodología consiste en una programación rápida o extrema y es utilizada para proyectos de corto plazo. Esta metodología es basada en pruebas unitarias, la refabricación y la programación en pares.

Propuestas de la Programación Extrema:

- Empieza en pequeño y añade funcionalidad con retroalimentación continua.
- El manejo del cambio se convierte en parte sustantiva del proceso.
- El costo del cambio no depende de la fase o etapa.
- No introduce funcionalidades antes que sean necesarias.
- El cliente final se convierte en miembro del equipo.

Algunos de los aspectos polémicos que presenta XP por los que se determinó no usarlo son:

- El problema que más se menciona con los proyectos de XP es que es difícil predecir costo y tiempo de desarrollo.
- Otro problema de XP es que, si se utilizan diagramas UML, éstos tienden a estar poco actualizados, debido a la constante refactorización.

- La Escalabilidad: Históricamente, XP trabaja solamente en los equipos de doce o pocas personas. Una forma para evitar esta limitación es dividir el desarrollo en proyectos más pequeños y el equipo en grupos más pequeños.
- Otro aspecto que imposibilita su uso es la distancia existente entre el cliente y los desarrolladores del producto ya que su particularidad es tener, como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto.

1.9.1.2 Proceso Unificado de Desarrollo (RUP)

El Proceso Unificado de Desarrollo es una metodología para el desarrollo de software orientado a objetos. Es un proceso de desarrollo de software, definido como un conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema de software. No obstante, el proceso unificado es más que un proceso de trabajo, es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software, para diferentes áreas de aplicación, diferentes tipos de organizaciones y diferentes niveles de aptitud. Está constituido por 9 flujos de trabajo (los 6 primeros son conocidos como flujos de ingeniería y los 3 últimos de apoyo): modelamiento del negocio, requisitos, análisis y diseño, implementación, prueba, instalación, administración de configuración y cambios, administración de proyectos, ambiente, los cuales tienen lugar sobre 4 etapas o fases: inicio, elaboración, construcción y transición. Esta metodología es adaptable para proyectos a largo plazo y establece refinamientos sucesivos de una arquitectura ejecutable.

RUP es un proceso de desarrollo de software que junto a UML constituyen la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. (7)

Características específicas de RUP

Dirigido por casos de uso: Significa que el proceso de desarrollo sigue una trayectoria que avanza a través de los flujos de trabajo generados por los casos de uso. Los casos de uso se especifican y diseñan al principio de cada iteración y son la fuente a partir de la cual los ingenieros de prueba construyen sus casos de prueba, éstos describen la funcionalidad total del sistema.

- Centrado en la arquitectura: Los casos de uso guían a la arquitectura del sistema que influye en la selección de los casos de uso. La arquitectura involucra los

elementos más significativos del sistema y está influenciada, entre otros, por las plataformas de software, sistemas operativos, sistemas de gestión de bases de datos, además de otros como sistemas heredados y requerimientos no funcionales. Se presenta mediante varias vistas que se centran en aspectos concretos.

- Iterativo e incremental: RUP divide el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y éstas se definen según el nivel de madurez que alcanzan los productos que se van obteniendo con cada actividad ejecutada. La terminación de cada fase ocurre en el hito correspondiente a cada una, donde se evalúa que se hayan cumplido los objetivos de la fase en cuestión.

Modelo 4 +1 Vistas

Dentro de esta metodología el Modelo 4+1 Vistas describe la Arquitectura de Software usando cinco vistas concurrentes, cada una de las cuales trata una serie de aspectos. Este modelo permite que diversas partes involucradas puedan encontrar lo que necesitan en la Arquitectura de Software. Los ingenieros de sistemas pueden abordar en primer lugar la vista física y a continuación, ver el proceso; los usuarios finales, los clientes y especialistas, pueden aproximarse a los datos de la vista lógica, y los directores de proyectos y miembros del equipo de configuración del software, pueden abordar desde la visión de desarrollo. Estas cuatro vistas están guiadas por la vista de casos de uso que describe las funcionalidades del sistema que más inciden sobre su arquitectura.

- Vista de Casos de Uso: Esta vista representa un subconjunto del artefacto Modelo de casos de uso y lista los casos de uso o escenarios más significativos, con las funcionalidades centrales del sistema. Si el sistema se hace extenso entonces se debería organizar en paquete, lo cual facilitaría la comprensión de la vista de casos de uso.
- Vista Lógica: Esta vista representa un subconjunto del artefacto Modelo de diseño, en él, se representan los elementos de diseño más significativos para la arquitectura del sistema. Describe las clases más importantes, su organización en paquetes y subsistemas, y éstos a su vez en capas. También describe las realizaciones de casos de uso más importantes, como las que describen aspectos dinámicos del sistema.
- Vista de Implementación: En esta sección se describe la estructura general del modelo de implementación, en correspondencia con la vista lógica, se debe

representar la descomposición del software en capas y paquetes importantes para la arquitectura.

- Vista de Despliegue: Esta vista suministra una base para la comprensión de la distribución física de un sistema a través de nodos, es decir, modela la configuración en funcionamiento del sistema (software y hardware) y las relaciones entre sus componentes. Suele utilizarse cuando el sistema está distribuido.
- Vista Concurrente o de Procesos: Describe el diseño de concurrencia y aspectos de sincronización. Especifica las líneas de mando que ejecutan cada operación en cada una de las clases señaladas en la vista lógica. Los diseñadores realizan esta vista en varios niveles de abstracción, además de dividir el software en conjuntos independientes de tareas, es decir, se empaqueta en pequeños programas o librerías del subsistema. La notación usada es una expansión de la notación original de Booch para tareas de Ada y está enfocada en los elementos significativos de la arquitectura. Los estilos arquitectónicos más usados son los de tuberías y filtros o el de cliente/servidor.

1.9.2 Lenguajes de descripción de la arquitectura (ADL)

Los lenguajes de descripción de la arquitectura se utilizan para satisfacer requerimientos descriptivos de alto nivel de abstracción. Constituyen lenguajes descriptivos de modelado que se centran en la estructura de un sistema. Éste proporciona un modelo explícito de componentes, conectores y sus respectivas configuraciones. (8)

Existen gran variedad de ADLs, los principales de éstos se mencionan a continuación:

Acme: Lenguaje de intercambio de ADL.

Aesop: ADL de propósito general, énfasis en estilos.

Darwin: ADL con énfasis en dinámica.

Jacal: Notación para la descripción y prototipado.

MetaH: ADL específico de dominio.

Rapide: ADL para simulación.

UniCon: ADL con énfasis en estilos.

xADL: Basado en XML.

UML: El Lenguaje Unificado de Modelado (UML) es un lenguaje semi-formal de modelado. Aunque éste no es un ADL en el sentido usual de la expresión, constituye una herramienta de uso habitual en el modelado, aunque ya se piensa en él como un metalenguaje. (8)

Se propone la utilización de UML para el diseño de la arquitectura pues permite especificar, visualizar, construir y documentar artefactos del sistema como las vistas arquitectónicas. Además, constituye un lenguaje que la mayoría de los desarrolladores conocen. A continuación se describen sus principales características.

1.9.2.1 El Lenguaje Unificado de Modelado (UML)

UML ofrece soporte para clases, clases abstractas, relaciones, comportamiento por iteración, empaquetamiento, entre otros. Estos elementos se pueden representar mediante nueve tipos de diagrama, que son: de clases, de objetos, de casos de uso, de secuencia, de colaboración, de estados, de actividades, de componentes y de desarrollo. (9)

UML presenta características generales y razones por las que resulta interesante su aplicación para efectos de la representación de una Arquitectura de Software. Permite el soporte para algunos de los conceptos asociados a las arquitecturas de software, como los componentes, los paquetes, las librerías y la colaboración. Además, admite la descripción de componentes en la Arquitectura de Software en dos niveles; se puede especificar solo el nombre del componente o especificar las clases o interfaces que implementan éstos. (6).

1.9.3 Lenguajes de desarrollo

Un lenguaje de programación es una serie de comandos que permiten codificar instrucciones de manera que sean entendidas y ejecutadas por una computadora. (10)

A medida que pasó el tiempo, las tecnologías fueron desarrollándose y surgieron nuevos problemas a resolver, que dieron lugar al desarrollo de lenguajes de programación para la web, que permitieran interactuar con los usuarios y utilizaran sistemas de bases de datos.

1.9.3.1 Lenguajes del lado del cliente

- **HTML (Lenguaje de Marcas de Hipertexto)**

HTML es el lenguaje con el que se definen las páginas web. Básicamente se trata de un conjunto de etiquetas que sirven para definir el texto y otros elementos que compondrán una página web. (11)

Este lenguaje es sencillo y permite describir hipertexto. El texto se presenta de forma estructurada y agradable, no necesita de grandes conocimientos cuando se cuenta con un

editor de páginas web o WYSIWYG, los archivos son pequeños y es admitido por todos los navegadores.

- **CSS (Hojas de Estilo en Cascada)**

CSS, es una tecnología que permite crear páginas web de una manera más exacta. Gracias a las CSS se tiene un mayor control sobre los resultados finales de la página, pudiendo hacer muchas cosas que no se podía hacer utilizando solamente HTML, como incluir márgenes, tipos de letra, fondos, colores. (12)

Con CSS se puede separar el contenido del diseño, siendo muy útil cuando se quiere cambiar un aspecto del diseño de un sitio web. Permite definir aspectos concretos de un documento facilitando su diseño. Del mismo modo, se pueden definir los estilos de tal manera que será diferente la visualización en una PDA que en una PC, o también hojas de estilo especiales para impresión.

- **Javascript**

Javascript es un lenguaje de script multiplataforma (cross-platform) orientado a objetos. Es pequeño y ligero, no es útil como un lenguaje independiente, más bien está diseñado para una fácil incrustación en otros productos y aplicaciones, tales como los navegadores Web. Dentro de un entorno anfitrión, Javascript puede ser conectado a los objetos de su entorno para proveer un control programable sobre éstos. (13).

La característica principal de Javascript, es la de ser un lenguaje de scripting pero, sobre todo, la de ser el lenguaje de scripting por excelencia soportado por todos los navegadores, sin lugar a dudas, el más usado. Éste es seguro y fiable ya que los scripts tienen capacidades limitadas por razones de seguridad, es ejecutado en el cliente por lo que el servidor no es utilizado más de lo debido y como desventaja el código del script debe descargarse completamente para poder ejecutarlo.

1.9.3.2 Lenguaje del lado del servidor

- **Ruby**

Un lenguaje de programación dinámico y de código abierto enfocado en la simplicidad y productividad. Su elegante sintaxis se siente natural al leerla y fácil al escribirla. (14)

Ruby tiene un conjunto de funcionalidades entre las que se encuentran las siguientes: manejo de excepciones, como Java y Python, para facilitar el manejo de errores. Escribir extensiones en C para Ruby es más fácil que hacer lo mismo para Perl o Python, con una

API muy elegante para utilizar Ruby desde C. Múltiples expresiones por líneas, separadas por punto y coma ";". Ruby puede cargar librerías de extensiones dinámicamente si el Sistema Operativo lo admite. Permite desarrollar soluciones a bajo costo y es multiplataforma.

- **Python**

Es un lenguaje de programación creado en el año 1990 por Guido van Rossum, es el sucesor del lenguaje de programación ABC. Python es comparado habitualmente con Perl. Los usuarios lo consideran como un lenguaje más limpio para programar; permite la creación de todo tipo de programas incluyendo los sitios web. Suele ser considerablemente más corto que su equivalente en lenguajes como C. El entorno de ejecución de Python detecta muchos de los errores de programación que escapan al control de los compiladores proporcionando información muy valiosa para detectarlos y corregirlos. Puede usarse como lenguaje imperativo procedimental o como lenguaje orientado a objetos. Posee un rico juego de estructuras de datos que se pueden manipular de modo sencillo. Tiene gran cantidad de funciones y librerías, pero tiene la desventaja de ser lento.

- **PHP**

Es un lenguaje de programación utilizado para la creación de sitio web. PHP es un acrónimo recursivo que significa "Hypertext Pre-processor". Es un lenguaje de script incrustado dentro del HTML. La mayor parte de su sintaxis ha sido tomada de C, Java y Perl con algunas características específicas de sí mismo. La meta del lenguaje es permitir rápidamente a los desarrolladores la generación dinámica de páginas. (15)

PHP no necesita ser compilado para ejecutarse y es muy fácil de aprender. Se caracteriza por ser un lenguaje muy rápido. Soporta en cierta medida la orientación a objeto, clases y herencia. Es un lenguaje multiplataforma, posee capacidad de conexión con la mayoría de los manejadores de base de datos: MysSQL, PostgreSQL, Oracle, MS SQL Server, entre otras. Se puede expandir su potencial utilizando módulos. Tiene gran documentación en su página oficial la cual incluye descripción y ejemplos de cada una de sus funciones. Incluye gran cantidad de funciones. No requiere definición de tipos de variables ni manejo detallado del bajo nivel.

Este lenguaje se utilizará para la implementación del proyecto, ya que existe una amplia comunidad en la Universidad y en el mundo. Es el más utilizado para aplicaciones web,

hay gran documentación del mismo y es muy fácil de utilizar por lo que agiliza el trabajo. Los integrantes del equipo de desarrollo poseen conocimientos del mismo y no será necesario un tiempo de capacitación. También cuenta con una gran cantidad de módulos que brindan muchas funcionalidades como son: acceso a datos, manejo de la cache para mejorar la eficiencia, administrar la autenticación de usuarios y muchos otros.

1.9.4 Framework (Marco de trabajo)

En general, con el término framework, se está refiriendo a una estructura de software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. En otras palabras, un framework se puede considerar como una aplicación genérica incompleta y configurable a la que se puede añadir las últimas piezas para construir una aplicación concreta. (16)

Los objetivos principales que persigue un framework son acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de implementación como el uso de patrones. Permite a los programadores pasar más tiempo identificando requerimientos que tratando los tediosos detalles de bajo nivel. La mayoría de los frameworks implementarán uno o más patrones de diseño de software que aseguren la escalabilidad del producto.

1.9.4.1 Framework para el cliente

En su mayoría, los frameworks Javascript proveen componentes totalmente compatibles entre los distintos navegadores, esto aumenta la portabilidad de la aplicación. Poseen comunicación asíncrona (AJAX), capacidad de manipulación del DOM de manera dinámica, manejo de JSON, XML, CSS y eventos y otras características para aumentar la interactividad y experiencia del usuario. Ejemplos de éstos son:

- Dojo Toolkit: Está compuesto por widgets que son componentes de código en Javascript pre-empaquetados que pueden ser utilizados para enriquecer sitios web con varias características que trabajan a través de la mayoría de los navegadores, tales como: menús, tabs, tooltips y tablas ordenables.
- Ext: Incluye interoperabilidad con jQuery. Posee controles para campos de textos, incluyendo áreas de texto, controladores selectores de fecha, campos numéricos para radiobutton y checkbox. También componentes para crear y manipular datagrids donde goza de cierta ventaja sobre otros frameworks. Es posible crear

"ventanas" con barras de herramientas y menús con estilo de aplicaciones de escritorio, diálogos modales y eventos.

- jQuery: es muy rápida, ligera y simplifica el desarrollo de la parte del cliente de las aplicaciones web. Es un nuevo tipo de bibliotecas de Javascript que permite simplificar la manera de interactuar con los documentos HTML.

Es muy difícil decir cuál de éstos es el mejor, unos tienen ventajas sobre otros pero todos funcionan muy bien, sin embargo jQuery es un producto con una aceptación muy buena por parte de los programadores y un grado de penetración en el mercado muy amplio, lo que hace suponer que es una de las mejores opciones. Además, es un producto serio, estable, bien documentado y con un gran equipo de desarrolladores a cargo de la mejora y actualización del framework. Otro aspecto interesante es la dilatada comunidad de creadores de plugins o componentes, lo que hace fácil encontrar soluciones ya creadas en éste para implementar algunos elementos como interfaces de usuario, galerías, votaciones, efectos diversos, etcétera. Todo esto hace que jQuery sea la elección para el desarrollo, unido a que en la UCI se está realizando un proyecto al cual el producto se pudiera integrar y éste utiliza jQuery.

1.9.4.2 Framework para el servidor

En la mayoría de los frameworks existentes se puede encontrar una serie de características como son: la abstracción de URLs y sesiones, herramientas e interfaces necesarias para integrarse con gestores de acceso a datos. Implementan una serie de controladores para gestionar eventos, mecanismos para la autenticación y control de acceso, internacionalización, separación entre el diseño y contenido y cache propia del framework para mejorar el rendimiento. La mayoría de los frameworks web están basados en el patrón MVC.

Entre los frameworks desarrollados para PHP se encuentran:

- Zend Framework: es una implementación que usa código 100% orientado a objetos. La estructura de los componentes de Zend Framework es algo único; cada componente está construido con una baja dependencia de otros componentes. Esta arquitectura débilmente acoplada permite a los desarrolladores utilizar los componentes por separado. A menudo, se refiere a este tipo de diseño como "use-at-will" (uso a voluntad). Aunque se pueden utilizar de forma individual, los componentes de la biblioteca estándar de Zend Framework conforman un potente y extensible framework de aplicaciones web al combinarse. Zend

Framework ofrece un gran rendimiento, una robusta implementación MVC y una abstracción de base de datos fácil de usar.

- CakePHP: provee de una extensible arquitectura para el desarrollo, mantenimiento y el despliegue de aplicaciones web. Usando los patrones de diseño más comunes como MVC, CakePHP reduce los costos de desarrollo y ayuda a los desarrolladores a escribir menos código (17). Es compatible con las versiones 4 y 5 de PHP, posee una licencia flexible, integra CRUD para la interacción con bases de datos, posee generación de código. Brinda la posibilidad de crear rápidamente flexibles plantillas mediante los helpers para AJAX, Javascript y formularios HTML. Tiene componentes para el manejo de correo, cookies, seguridad y sesiones de usuario. Tiene un sistema de cache flexible y configurable.

- Symfony: Es un framework PHP que facilita el desarrollo de las aplicaciones web. Se encarga de todos los aspectos comunes y aburridos de las aplicaciones web, dejando que el programador se dedique a aportar valor desarrollando las características únicas de cada proyecto. Symfony es además el framework más documentado del mundo, ya que cuenta con miles de páginas de documentación distribuidas en varios libros gratuitos y decenas de tutoriales (18). Es fácil de instalar y configurar en sistemas Windows, Mac y Linux, compatible solamente con PHP 5 desde hace años para asegurar mayor rendimiento, está basado en la premisa “convenir en vez de configurar”, solo se debe configurar lo que no es convencional. Se puede adaptar con facilidad a las políticas y arquitecturas propias de cada empresa u organización, es flexible y extensible mediante un completo mecanismo de plugins. Presenta soporte a más de 40 idiomas y fácilmente traducible a cualquier otro idioma.

Symfony es el framework que se utilizará por tratarse del mejor framework documentado y es sumamente estable por la gran cantidad de pruebas funcionales y unitarias a la que es sometido, es infinitamente escalable si se disponen de los recursos necesarios, Yahoo utiliza Symfony para programar aplicaciones con 20 millones de usuarios. Éste sigue una política de tipo LTS (soporte a largo plazo). Las versiones estables se mantienen durante 3 años sin cambios pero con una corrección continua de los errores. Además, en la UCI también es el más utilizado y como se mencionaba anteriormente la posible futura integración, este proyecto al que se podrá integrar el producto también usa este framework.

Symfony considera un proyecto como *"un conjunto de servicios y operaciones disponibles bajo un determinado nombre de dominio y que comparten el mismo modelo de objetos"* (19). Dentro de un proyecto las operaciones se agrupan de forma lógica en aplicaciones, las cuales comparten el mismo modelo de datos y se ejecutan independientes unas de otras. Estas aplicaciones están compuestas por módulos los cuales representan una página web o un grupo de páginas con un propósito relacionado. Los módulos almacenan las acciones que representan a cada una de las posibles operaciones, trabajar con éstas es muy similar a trabajar con páginas de una aplicación web tradicional. A estas acciones se les puede hacer corresponder una vista específica o pueden retornar la respuesta en formato de texto.

Está diseñado con el patrón Controlador Frontal que consiste en la existencia de un único controlador en el sistema que soluciona el problema de la descentralización presente en el patrón MVC. Es el único punto de entrada a la aplicación, carga la configuración, comprueba las restricciones de seguridad, manejo de errores y determina la acción a ejecutarse.

También tiene integrado el ORM (Mapeo de objetos-relacional) Doctrine que se encarga del acceso a datos brindando un conjunto de clases mapeadas que permiten la interacción con la base de datos a través de objetos, evitando la incompatibilidad entre los distintos gestores de bases de datos y estos objetos pueden tener tipos de datos que los gestores de bases de datos no poseen, por ejemplo, si un profesor tiene muchos estudiantes, se tendrá un objeto profesor el cual tendrá un atributo que es una lista de todos sus estudiantes. Con éste viene integrado una herramienta de línea de comando que tiene un grupo de funcionalidades como generar la base de datos a partir de un archivo YML y viceversa, también generar el modelo a partir de este YML, crear el CRUD de los elementos del modelo y otras.

Una petición a Symfony tiene el siguiente flujo básico:

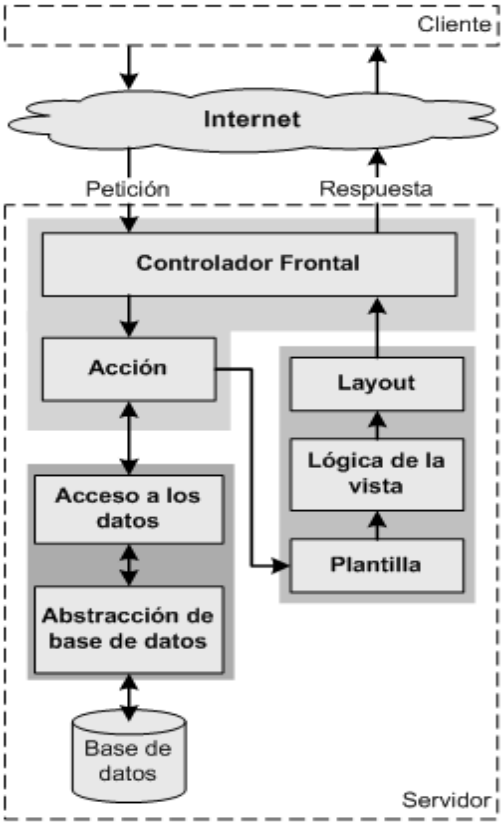


Fig. 4 Flujo básico de una petición a Symfony

1.9.5 Ambiente de desarrollo integrado (IDE)

Un entorno de desarrollo integrado o IDE, 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. Proporciona herramientas muy útiles para los programadores, tales como depuradores de código, ayuda en línea de uso del lenguaje y un constructor de interfaz gráfica.

Existen varios IDEs que permiten el trabajo con tecnologías web, pero los programadores del proyecto están familiarizados con el IDE NetBeans y tienen experiencia de trabajo con el mismo, además éste presenta soporte nativo a los frameworks Symfony y jQuery por lo en él se realizará la implementación de la Colección. NetBeans es un premiado entorno de desarrollo integrado disponible para Windows, Mac, Linux y Solaris. El proyecto NetBeans consiste en un IDE de código abierto y una plataforma de aplicaciones que permiten a los desarrolladores la creación rápida de diferentes tipos de aplicaciones utilizando la plataforma Java, así como JavaFX, PHP, Javascript y Ajax, Ruby y Ruby onRails, Groovy y Grails, y C / C + +. (20)

El proyecto NetBeans es apoyado por una vibrante comunidad de desarrolladores y ofrece una amplia documentación y recursos de capacitación, así como una variada selección de plugins de terceros. Permite la creación de aplicaciones web PHP con la nueva versión de PHP 5.3.

La integración con Symfony permite desarrollar aplicaciones de forma más sencilla y productiva. En primer lugar, es posible crear nuevos proyectos y aplicaciones directamente desde el IDE. También se pueden ejecutar todas las tareas de Symfony, incluso pasándole argumentos y opciones, visualizando el resultado sin necesidad de utilizar una consola de comandos externa. También brinda un buen soporte a jQuery y resulta muy cómodo para el desarrollo ya que se trabaja con un solo IDE.

1.9.6 Herramienta Case

Conjunto de métodos, utilidades y técnicas que facilitan la automatización del ciclo de vida del desarrollo de sistemas de información, completamente o en alguna de sus fases. Estas herramientas brindan ayuda a los analistas, ingenieros de software y desarrolladores, por ejemplo permiten el modelado de los sistemas mediante diferentes diagramas y generación de código a partir de éstos, y viceversa.

Visual Paradigm:

Es una herramienta profesional multiplataforma que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientado a objetos, construcción, pruebas y despliegue. Soporta todos los diagramas UML, diagramas de SysML y el diagrama entidad-relación ayudando a una rápida construcción de aplicaciones con calidad y a un menor coste. Pueden encontrarse múltiples usuarios trabajando sobre el mismo proyecto gracias a su integración con SVN. Produce documentación del sistema en formato PDF, HTML y MS Word. Genera código en una amplia gama de lenguajes, dentro de éstos PHP, tiene la capacidad de crear el esquema de clases a partir de una base de datos y crear la definición de base de datos a partir del esquema de clases. Posee una interfaz amigable y profesional, y se puede modelar en varios idiomas.

1.9.7 Sistema Gestor de Bases de Datos

Consiste en una colección de datos interrelacionados y una colección de programas para acceder a los datos. Proporciona un entorno conveniente y eficiente para los usuarios que lo usan para la recuperación y almacenamiento de la información. (21)

MySQL:

Es el SGBD más usado del mundo, con más de 100 millones de copias de su software descargado o distribuido en toda su historia. Con su velocidad, fiabilidad y facilidad de uso, elimina los problemas más importantes asociados con el tiempo de inactividad, mantenimiento y administración de aplicaciones en línea. Es multiplataforma, presenta API disponibles para C, C++, Eiffel, Java, Perl, PHP, Python, Ruby y TCL. Proporciona sistemas de almacenamientos transaccionales y no transaccionales, un sistema de reserva de memoria muy rápido basado en hilos, las funciones SQL están implementadas usando una librería altamente optimizada y deben ser tan rápidas como sea posible. Normalmente, no hay reserva de memoria tras toda la inicialización para consultas y está probado con un amplio rango de compiladores diferentes.

PostgreSQL:

Es un poderoso sistema de base de datos relacional. Cuenta con más de 15 años de desarrollo activo y una arquitectura probada que se ha ganado una sólida reputación de confiabilidad, integridad de datos y corrección. Funciona en todos los principales sistemas operativos, incluyendo Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), y Windows. Es totalmente compatible con ACID, tiene soporte completo para claves foráneas, uniones, vistas, disparadores y procedimientos almacenados (en varios idiomas). Se incluye la mayoría de tipos de datos de SQL: 2008, incluyendo INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL, y TIMESTAMP. También es compatible con el almacenamiento de objetos binarios, incluyendo imágenes, sonidos o vídeo. Tiene interfaces de programación nativa de C / C + +, Java, .NET, Perl, Python, Ruby, Tcl, ODBC, entre otros, y posee una documentación excepcional. (22)

Es una base de datos empresarial multiplataforma que cuenta con sofisticadas funciones de replicación asincrónica, backup en línea, permite consultas muy complejas y tiene un sofisticado optimizador para éstas, es compatible con un conjunto de caracteres internacionales, cuenta con vistas, integridad referencial, triggers y posee control de versionado concurrente (MVCC). Es un sistema altamente escalable, tanto en la enorme cantidad de datos que puede manejar y en el número de usuarios concurrentes que puede acomodar.

PostgreSQL es la elección del SGDB ser más eficiente cuando se desea almacenar un gran número de datos, siendo la eficiencia un requisito fundamental en el desarrollo de

aplicaciones web. Es fácil su integración con Symfony lo que brinda amplias prestaciones.

1.9.8 Servidor Web

Un servidor web es un programa encargado de atender y responder diferentes tipos de peticiones realizadas por los navegadores web usando protocolos HTTP (protocolo de traslado de hipertexto) y HTTPS (versión cifrada y autenticada del mismo). Básicamente el funcionamiento de éstos es estar a la espera de las peticiones, cuando recibe una petición busca el recurso solicitado y lo envía utilizando la misma conexión por donde recibió la petición.

Apache es el servidor web a utilizar ya que es el más usado a través de los años, por su excelencia, su configurabilidad, robustez y estabilidad. Es flexible, rápido, eficiente y continuamente actualizado y adaptado a los nuevos protocolos HTTP 1.1. Algunas de sus características son:

- Multiplataforma
- Es gratuito y de código abierto, brindando así transparencia del software y permitiendo que se puedan analizar los detalles de su funcionamiento.
- Incentiva la retroalimentación de los usuarios, obteniendo nuevas ideas, informes de fallos y parches para la solución de los mismos.
- Permite personalizar la respuesta ante los posibles errores que se puedan ocasionar. Es posible configurarlo para que ejecute un determinado script cuando ocurra un error.
- Es extensible por su característica de ser modular. Ejemplos de éstos son mod_auth_ldap para autentificar usuarios contra un servidor LDAP, mod_rewrite para reescritura de direcciones, mod_auth para autenticar usuarios usando archivos de texto y otros.

Conclusiones

Para definir una arquitectura es necesario aplicar estilos y patrones arquitectónicos, así como patrones de diseño para asegurar una estructura que cumpla con todos los requerimientos del proyecto. Con el estudio realizado hasta el momento fue posible la toma de las siguientes decisiones: se va a utilizar el estilo Modelo Vista Controlador empleando el framework de PHP Symfony 1.4. La descripción de la arquitectura se hará mediante las 4+1 vistas según propone la metodología RUP. En la vista se utilizará el

framework jQuery 1.4 para mejorar la interacción con el usuario y poder trabajar asincrónicamente y de este modo optimizar el rendimiento de la aplicación. El IDE seleccionado para el desarrollo será NetBeans 6.8 el cual brinda soporte a Symfony y jQuery, como SGDB PostgreSQL 8.4 y la Colección correrá en un servidor web Apache 2.

Capítulo 2: Características del Sistema

2.1 Introducción

Para realizar el diseño de un sistema informático es necesario aclarar todos los conceptos y sus relaciones utilizando un lenguaje perceptible por todos los involucrados. Es por ello que en el presente capítulo se realiza la descripción de la propuesta de solución para facilitar su comprensión. Puesto que fue definido que no es necesario un modelo completo del negocio se realiza un Modelo del Dominio que agrupa los principales conceptos con los que se trabajan. Además, se identifican los requisitos no funcionales y los casos de uso críticos que regirán el desarrollo del sistema. También se realizan los Diagramas de Casos de Uso con la intención de reflejar las relaciones entre los actores del sistema y las secuencias de acciones en las que están involucrados, se realizará el diseño de la aplicación y otros aspectos de importancia arquitectónica.

2.2 Modelo de Dominio

Una vez analizado el problema e identificados los principales conceptos presentes en el mismo, se hace posible el empleo de éstos en la elaboración del Modelo de Dominio donde se presenta un marco conceptual y la relación existente entre las definiciones.

2.2.1 Análisis de los conceptos del Dominio

Para lograr una mayor comprensión del diagrama de Modelo de Dominio que se expondrá a continuación, es necesario realizar previamente la definición de los conceptos que involucra dicho modelo.

- **Colección Navegante:** Colección de software educativo destinada a apoyar el proceso de enseñanza – aprendizaje de los estudiantes en las escuelas secundarias venezolanas.
- **Ejercicio:** Módulo que permite ejercitar y comprobar las habilidades adquiridas por los estudiantes.
- **Estudiante:** Persona que cursa estudios en las instituciones de nivel secundario.
- **Imagen:** Representación gráfica de una expresión.
- **Juegos:** Módulo que contiene juegos que ejercitan el contenido aprendido por el estudiante.
- **Liceo:** Institución educativa centrada en la formación de estudiantes, que tiene como objetivo fundamental organizar la enseñanza y el aprendizaje de contenidos

básicos y capacitar al estudiante para proseguir estudios superiores, o bien para incorporarse al mundo laboral.

- **Profesor:** Módulo que le permite al profesor configurar algunas opciones del producto en cuestión.
- **Media:** Se utiliza para referirse a los textos, imágenes, sonidos o videos.
- **Mediateca:** Módulo que agrupa las medias presentes en el producto.
- **Módulo:** Se le denomina al conjunto de elementos fundamentales que componen el producto.
- **Hiperentorno de aprendizaje:** Sistema informático basado en tecnología hipermedia, donde se mezclan diferentes elementos que representan las diversas tipologías de software educativo con el propósito de atender las necesidades didácticas.
- **Profesor:** Persona responsable de educar, guiar y supervisar a los estudiantes en el proceso de enseñanza - aprendizaje.
- **Resultados:** Es el módulo que visualiza la traza del estudiante durante la interacción de los mismos con los restantes módulos de la aplicación.
- **Sonido:** Media que brinda información sonora sobre un contenido.
- **Temas:** Es el módulo en el que se presentan los contenidos de las asignaturas o temas al cual corresponde el software.
- **Texto**: Es una composición de signos codificado en un sistema de escritura (como un alfabeto) que forma una unidad de sentido.
- **Video:** Su objetivo radica en brindarle al usuario una información audiovisual, para que perciba imágenes y sonidos.

2.2.2 Diagrama del Modelo de Dominio

El Modelo de Dominio o Modelo Conceptual, permite de manera visual mostrar al usuario los principales conceptos que se manejan en el dominio del sistema en desarrollo. Un Modelo de Dominio es una representación de las clases conceptuales del mundo real y no de componentes de software.

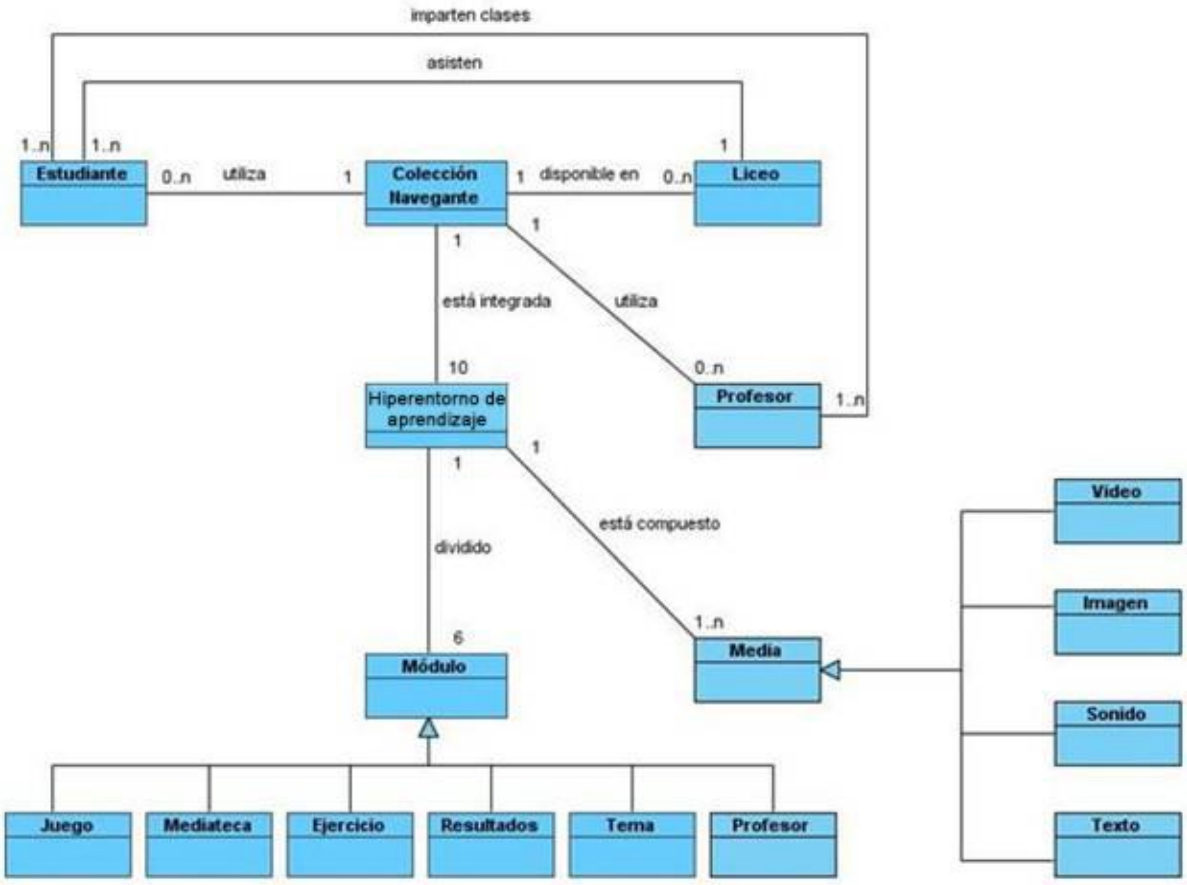


Fig.2 Modelo de Dominio

2.3 Descripción del Sistema Propuesto

Teniendo en cuenta el estudio realizado, se determinaron las principales funcionalidades y características que poseerá el sistema. Se pretende obtener una colección de 10 software educativos contextualizados al entorno venezolano, que estarán dirigidos al nivel de secundaria básica, cada uno de éstos será un hiperentorno de aprendizaje basado en tecnologías web, desarrollado con herramientas libres y capaz de ejecutarse en cualquier plataforma.

Cada hiperentorno será capaz de mostrar contenidos educativos, medias, contará con juegos, ejercicios, un glosario de términos, una Mediateca, un módulo llamado Resultados que dará la posibilidad de llevar el control de los resultados obtenidos por los estudiantes, así como consultar la navegación de éstos, permitiendo saber cuáles fueron los materiales que consultó el mismo, tendrá un módulo profesor donde se realizará la

gestión de usuarios, la configuración del sistema y además contará con artículos de capacitación para los profesores. También se hará una aplicación para la gestión de los contenidos educativos que estarán presentes en la Colección.

2.4 Requerimientos del Software

Un requisito de software es una condición que debe cumplir un sistema para satisfacer un contrato, estándar, especificación u otra documentación formalmente impuesta.

2.4.1 Requisitos No Funcionales (RNF)

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener.

RNF de hardware:

Todas las computadoras deben contar con:

1. Procesador Pentium 233 MHz (recomendado 500 MHz o mayor).
2. 64 MB de RAM (recomendado 128 MB y 256M en caso del servidor).
3. 20 GB de espacio en disco duro.
4. Lector de CD-ROM.
5. Dispositivos de audio.
6. Soporte de video que admita resolución de al menos 1024x768 y 24 bits.
7. Dispositivo de red.
8. El sistema interactuará con una impresora que permita imprimir los diferentes contenidos como respuesta a las funcionalidades del sistema.

RNF de apariencia e interfaz externa:

1. El diseño de las interfaces debe ser amigable y sencilla.
2. El sistema proporcionará claridad y correcta organización de la información, permitiendo la interpretación correcta e inequívoca de ésta.
3. El diseño de la interfaz gráfica deberá garantizar la distinción visual entre los elementos del sistema.

RNF Legales:

1. Cada una de las medias (imágenes, videos, sonidos, textos) que se utilicen en el producto debe tener el permiso legal de sus titulares y su aprobación para hacer uso de ellas.

RNF de soporte:

1. Se realizará la transferencia tecnológica de la Colección a los clientes.

2. Se impartirán clases a los profesores venezolanos para explicar el funcionamiento y utilidad del producto.

RNF de Portabilidad:

1. El sistema podrá ser utilizado bajo cualquier sistema operativo que tenga disponible uno de los navegadores que soporte la Colección.

RNF de Seguridad:

1. El sistema deberá tener el control de la información a la que accede cada usuario.
2. No se denegará el acceso a la aplicación a ningún usuario.

RNF de Usabilidad:

1. El sistema debe ser de fácil manejo para los usuarios que tengan niveles básicos sobre computación.
2. El sistema contará con una ayuda que servirá para guiar a los usuarios en el manejo del mismo.

RNF de software:

1. Computadora Personal con navegador Internet Explorer 8, Mozilla Firefox 3.5.x y Chrome 4.x.
2. El sistema será capaz de trabajar en cualquier sistema operativo.

RNF de diseño y la implementación:

1. Framework de Javascript para el cliente jQuery 1.4.
2. PHP 5.2.
3. Framework de PHP parar el servidor Symfony 1.4.
4. Servidor de aplicaciones web Apache 2.x.
5. Gestor se bases de datos PostgreSQL 8.4.

RNF de Rendimiento:

1. El sistema debe permitir un rápido acceso de búsqueda a la información en tiempos relativamente cortos, dependiendo de la conexión.
2. La aplicación debe ser concebida para el consumo mínimo de recursos.

2.5 Algunas características usadas de Symfony

Para el desarrollo del proyecto se crean dos aplicaciones Symfony, el *backend* en la cual se lleva a cabo la gestión de los contenidos educativos presentes en la Colección y la administración de usuarios de esta aplicación y el *frontend* para presentar estos contenidos y donde se gestionan los estudiantes y profesores. Estos últimos podrán ver el desempeño de sus estudiantes en el hiperentorno.

La vista en Symfony está basada en el patrón decorador, donde las plantillas (vistas) que se muestran con cada acción pueden ser decoradas con un layout que es común para toda la aplicación. Para el proyecto se crearán tres layouts, uno para cada aplicación y el tercero para elegir a qué producto de la Colección se navegará o se le hará la gestión de contenidos.

Fig.3 Plantilla decorada con un layout

Para el acceso a datos el ORM Doctrine crea tres clases por cada tabla de la base de datos, una base donde se definen los campos, relaciones y restricciones de la tabla, esta clase no se va a modificar pues cada vez que se genera el modelo se borra y se crea nuevamente. Otra que hereda de ésta y sus instancias corresponden a un registro en la base de datos. Un objeto de esta clase tiene los métodos set y get de todos sus atributos, los cuales corresponden a los campos que posee la tabla en la base de datos, un método para salvar y eliminar el objeto de la base de datos y si la tabla está relacionada con otra existe un atributo que representa dicha relación. En esta clase se podrán redefinir algunos métodos que presenta como el *toString* y se le agregarán métodos que solo afecten al registro correspondiente. La última clase contiene métodos que devuelven una lista de objetos de la clase anterior, esta clase se utilizará para crear métodos cuyo resultado sea una consulta a la base de datos que retorne una colección de registros de la tabla correspondiente.

2.6 Extensiones al framework jQuery

Existen varias formas de lograr la herencia en Javascript como es a través del encadenamiento del p*rototype* (propiedad de los objetos Javascript), otra variante es copiar todas las propiedades de un objeto a otro y varias formas más, cuál elegir depende del estilo, preferencias y proyecto a realizar. Mezclando estas diversas formas se crea una extensión de jQuery para declarar clases que puedan heredar entre ellas, de ser necesario, lo que permite que la programación en el cliente sea orientada a objetos. Esto se puede observar en el ejemplo del Anexo 2.

Una forma de mejorar el rendimiento de las páginas es tener todo el código Javascript en un solo fichero para disminuir el número de peticiones, pero para el desarrollo y mantenimiento de las aplicaciones esto es un obstáculo, pues se tendrían archivos con demasiadas líneas de código. Es por ello que se creó una extensión de jQuery para la inclusión de ficheros CSS y Javascript, ésta se encarga de, dado una lista de archivos que se necesitan, el servidor crea un solo archivo uniendo todos los archivos pedidos y éste es el que se envía al cliente. El orden de ejecución de estos archivos vendrá dado por el orden de la lista.

Esta función recibe dos parámetros:

1.- tipo: string -> puede ser “css” o “script” según el tipo de archivo.

2.- archivos: array -> arreglo de archivos a incluir

Ejemplo:

$.incluir("css", ["lib.jquery-ui"]);

$.incluir(“script”, [“geMediadorEjemplo” , ”com.meMediadorPrincipal”]);

Para poder organizar estos archivos en el servidor y que los programadores no tengan que especificar la ruta a cada uno de éstos, se crea un estándar de nombres, mediante el mismo solo es necesario el nombre del archivo y el servidor sabrá su localización. Este estándar consiste en poner un prefijo a los archivos según el módulo al que pertenecen. Si están en la carpeta componentes que pertenece a un módulo, o si son archivos generales como librerías y widgets, se les agrega otro prefijo seguido por “.”. La estructura de carpetas en que se organizan estos archivos será explicada más adelante. La Tabla 1 muestra estos prefijos, cómo se nombran los archivos y qué cadena se utiliza en el arreglo archivos para incluirlos.

Prefijo	**Archivo**	**Cómo se incluye**	**Descripción**
ge	geMediadorGeneral.js	geMediadorGeneral	Archivos del módulo general
re	reResultados.css	reResultados	Archivos del módulo Resultados
me	meVideos.js	meVideos	Archivos del módulo Mediateca

te	teTemas.css	teTemas	Archivos del módulo Temas
po	poEstudiantes.js	poEstudiantes	Archivos del módulo Profesor
ej	ejVoF.js	ejVoF	Archivos del módulo Ejercicios
ju	juLudo.js	juLudo.	Archivos del módulo Juegos
lib.	jquery-ui.js	lib.jquery-ui	Archivos generales a todo el proyecto
wid.	jquery-tira.js	wid.jquery-tira	Archivos de las widgets a utilizar
com.	meMediadorPrincipal.js	com.meMediadorPrincipal	Archivos que se encuentran en la carpeta componentes de los módulos

Tabla 1 Prefijos de los archivos CSS y Javascript para su inclusión en las páginas.

2.7 Restricciones para el desarrollo

Para mejorar el rendimiento de las aplicaciones se siguen un conjunto de reglas que se muestran a continuación.

1.- Se cargan los CSS y después los scripts, permitiendo la visualización de la página antes que carguen los scripts, de esta forma, se le va mostrando al usuario la página mientras se carga el script.

2.- Las imágenes tendrán formato PNG8, para esto se puede utilizar *pngcrush* para reducir el tamaño de la imagen y *pngquant* para convertir la imagen a PNG8, ambas son aplicaciones de línea de comando de Linux.

3.- Las imágenes tendrán la resolución con que se mostrarán en la página, evitando escalarlas, es decir, si la imagen a mostrar va a tener una resolución de 100 x 100 y la traída del servidor es de 200 x 200 ésta tendría un tamaño mayor que la que realmente se necesitaría.

4.- Usar la técnica de *CSS Sprites* que consiste en combinar varias imágenes en una sola y mostrarlas por la propiedad background-image de los elementos html, esto permitirá reducir el número de peticiones http.

5.- Los scripts y CSS serán minimizados y comprimidos para enviarlos al cliente, se cargarán en demanda y si en un momento se requieren varios archivos, éstos se combinarán en uno solo reduciendo el número de peticiones http.

6.- Se llevará el control de los CSS y scripts cargados, de tal forma que se verifique en el cliente si ya los archivos están incluidos y si no se han incluido aún, entonces se hace la petición.

2.8 Vista de Casos de Uso

2.8.1 Casos de uso arquitectónicamente significativos

La arquitectura del sistema está condicionada por los casos de uso, los cuales representan la agrupación de los requisitos funcionales a la hora de implementar la solución de un software, éstos proporcionan un medio para que los desarrolladores, usuarios finales y trabajadores lleguen a una comprensión común del sistema propuesto. Son empleados además para la validación de la arquitectura a lo largo del desarrollo.

Los casos de uso, para facilitar el desarrollo, fueron agrupados en paquetes según los módulos de la aplicación. Los resúmenes de la descripción correspondiente a estos casos de uso se pueden encontrar en el Anexo 1.

2.8.2 Diagramas de Casos de uso del Sistema

Paquete Temas

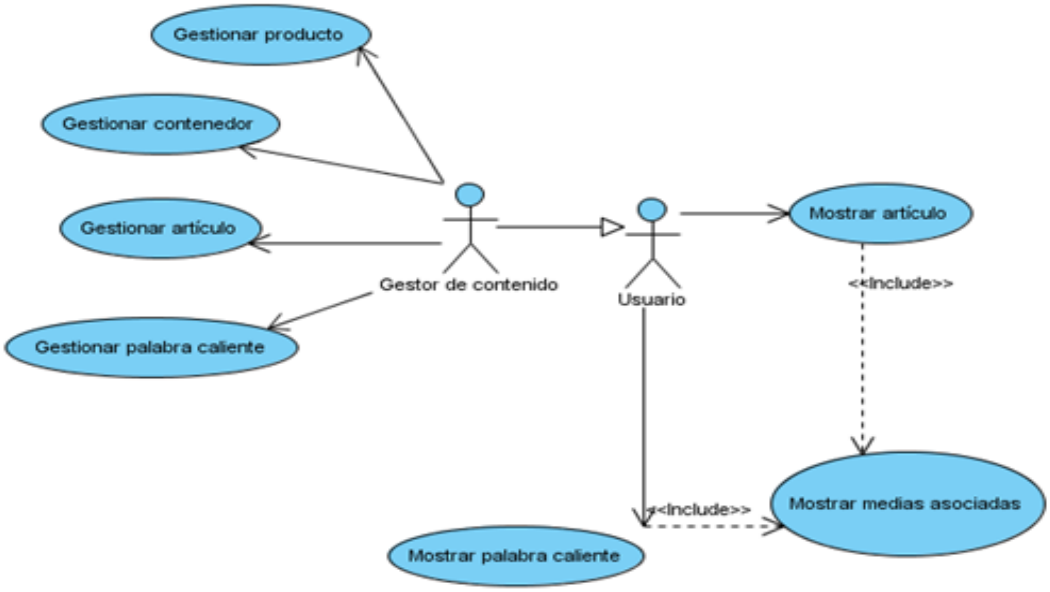


Fig.5 Diagrama de casos de uso del paquete Temas

Paquete Mediateca

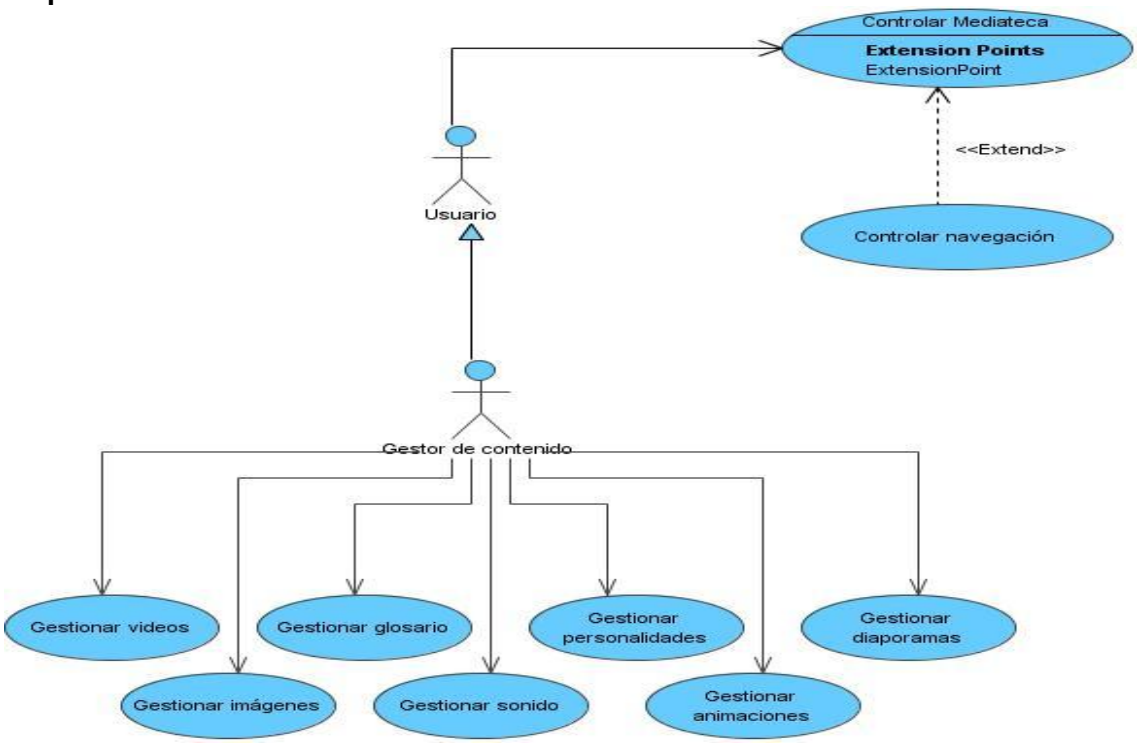


Fig.6 Diagrama de casos de uso del paquete Mediateca

Paquete Ejercicios

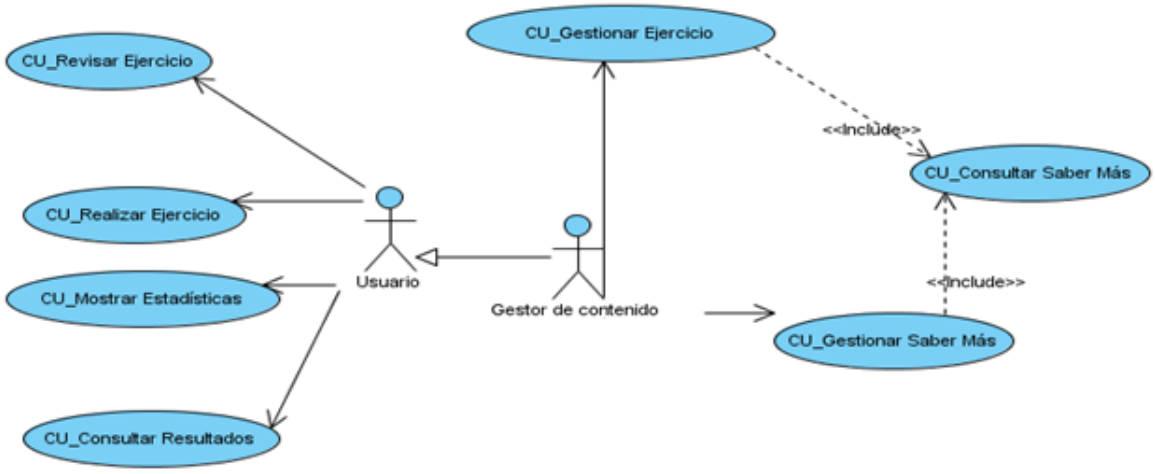


Fig.7 Diagrama de casos de uso del paquete Ejercicios

Paquete Juegos

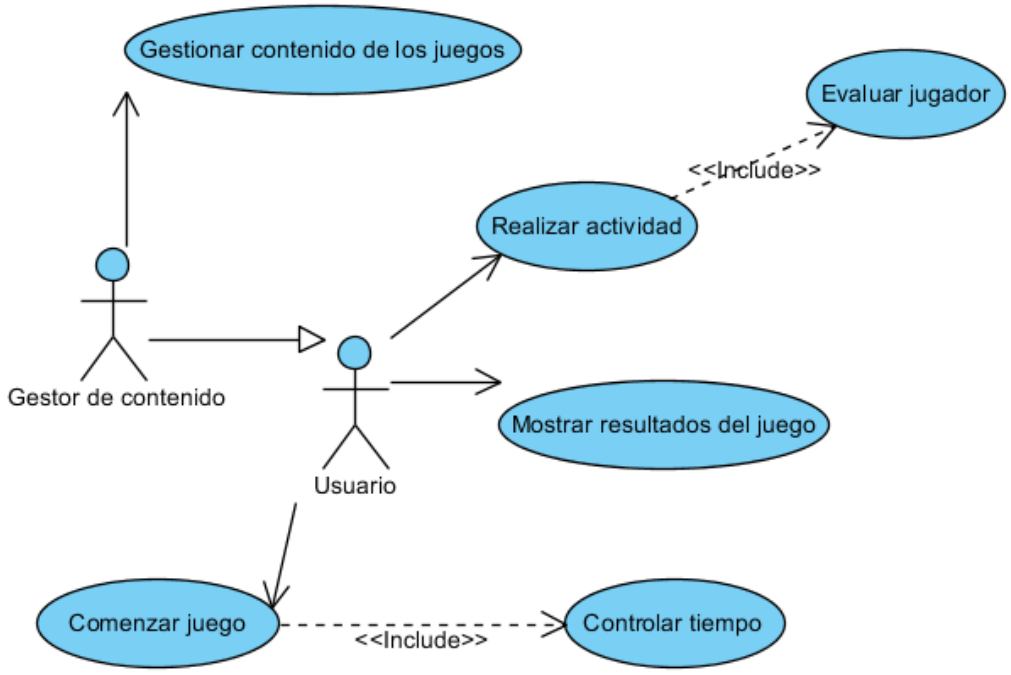


Fig.8 Diagrama de casos de uso del paquete Juegos

Paquete Profesor

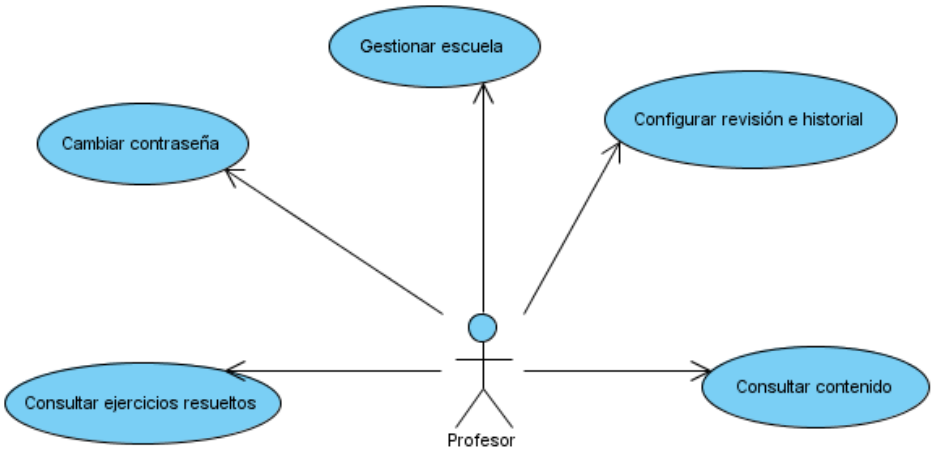


Fig.9 Diagrama de casos de uso del paquete Profesor

Paquete Resultados

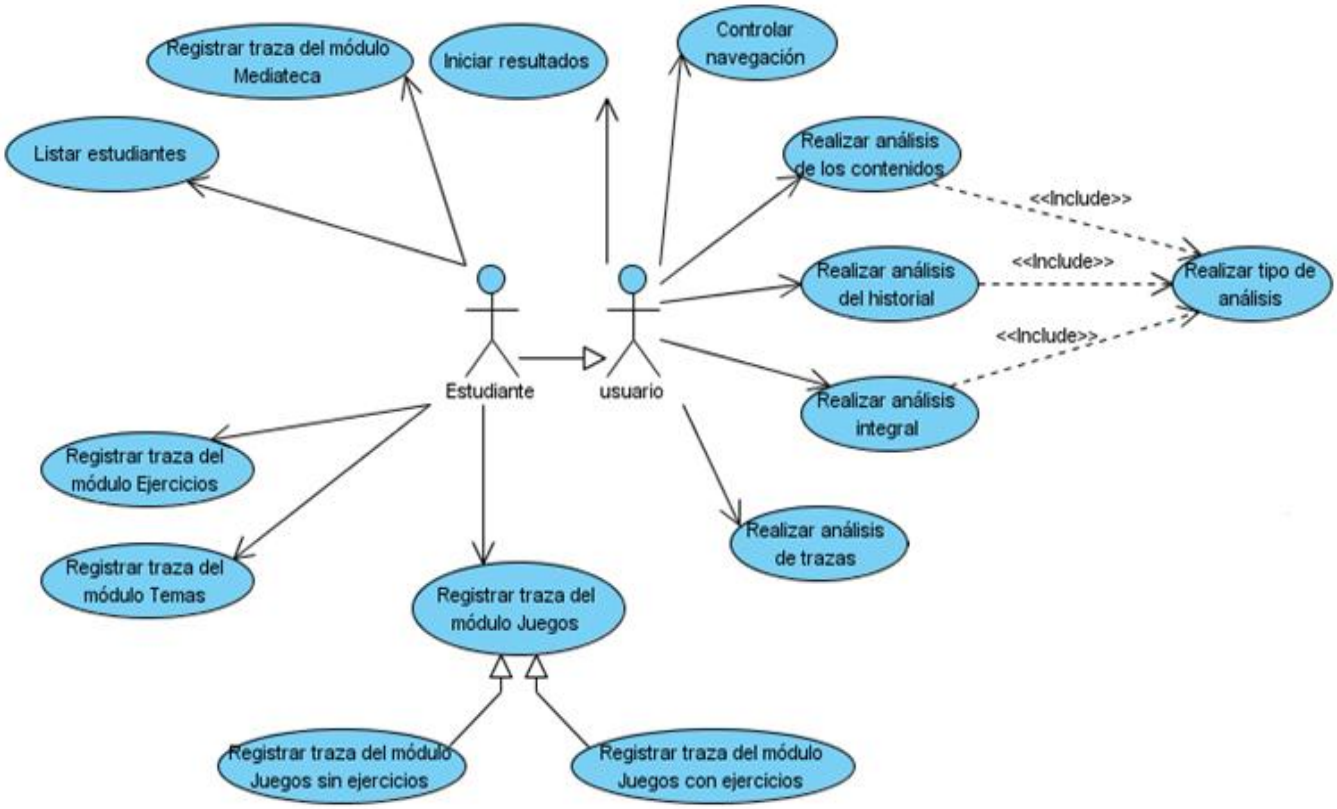


Fig.10 Diagrama de casos de uso del paquete Resultados

Paquete General

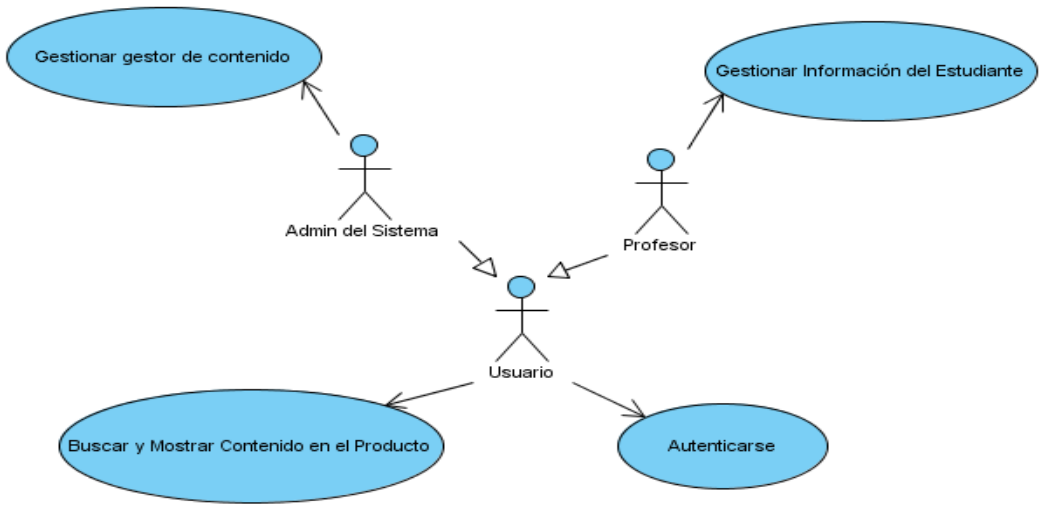


Fig.11 Diagrama de casos de uso del paquete General

2.9 Vista Lógica

La vista lógica comprende los elementos del diseño significativos para la arquitectura, el proyecto se divide en paquetes de diseño para facilitar el desarrollo de los mismos, estos paquetes son independientes de forma que si cambian los requerimientos de un paquete los otros no se verán afectados.

El diseño es agrupado en dos paquetes generales correspondientes a las aplicaciones a desarrollar, donde el único punto de contacto entre ellos es el modelo de datos con que trabajan.

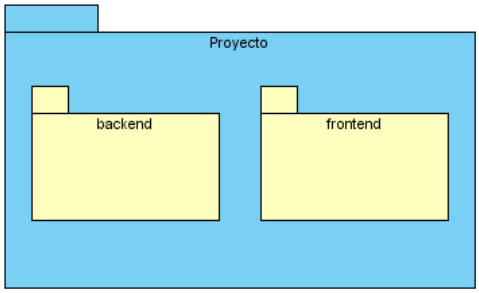


Fig.12 Paquete general del proyecto

El backend está dividido en 6 subsistemas de diseño, 5 de ellos para gestionar los contenidos educativos y otro para la administración de usuarios.

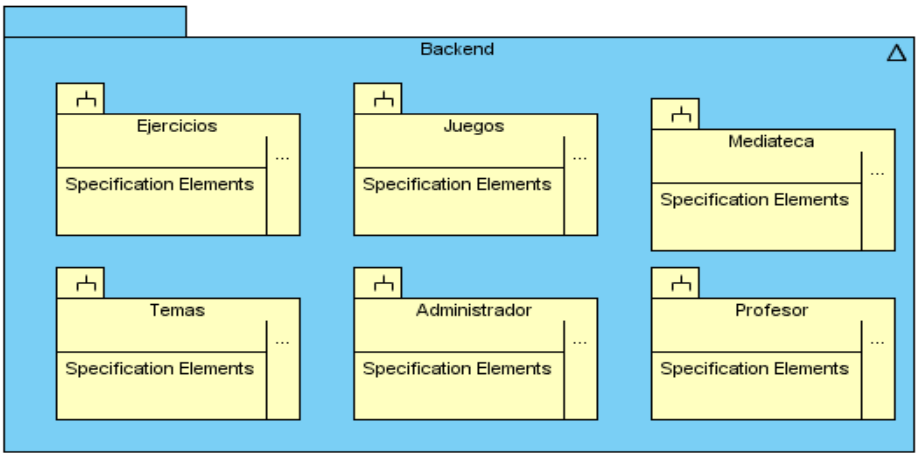


Fig.13 Subsistema de la aplicación backend

Descripción de los subsistemas

Ejercicios: gestión de todos los ejercicios a mostrar en la Colección.

Juegos: gestión de los contenidos presentes en algunos juegos como son los crucigramas y las sopas de letras.

Mediateca: gestión de todas las medias, el glosario de términos y las personalidades.

Temas: gestión de los temas de la Colección.

Administrador: administración de los usuarios que harán la gestión de los contenidos educativos.

Profesor: gestión de los artículos del profesor.

Todos los subsistemas anteriores están estructurados de la misma forma por lo que solamente se hace la representación a uno de ellos.

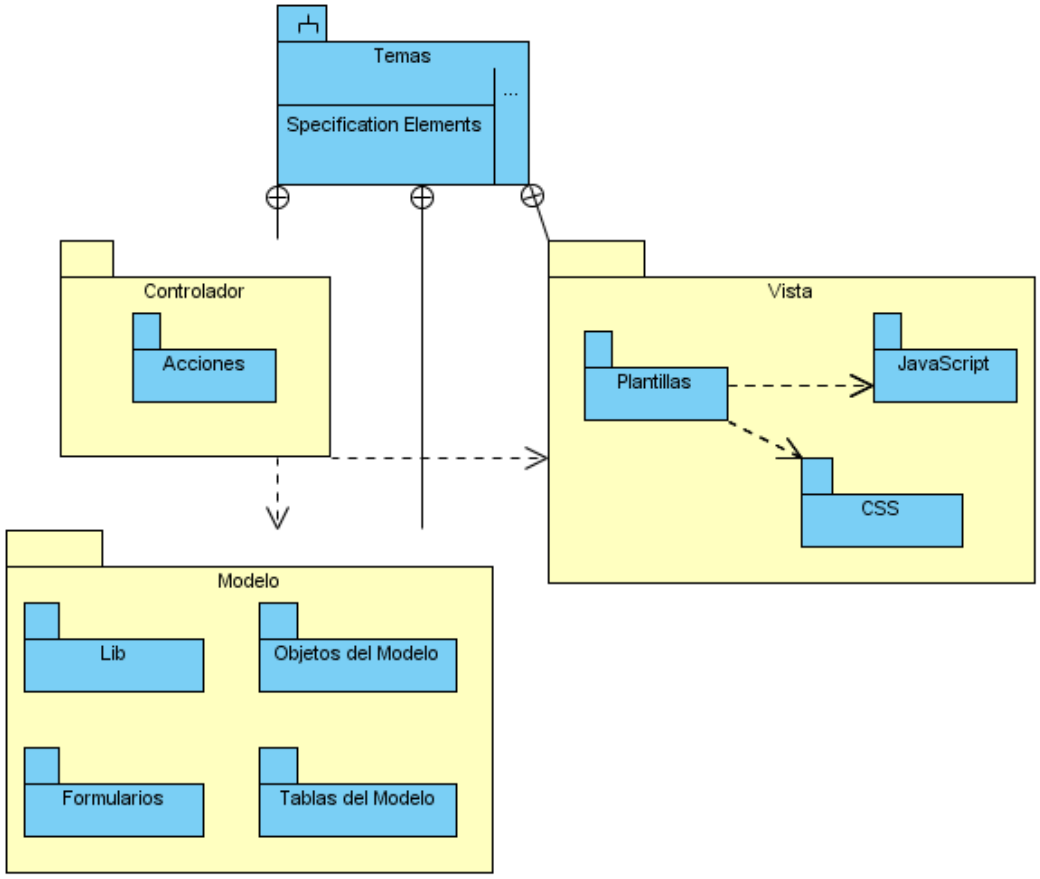


Fig.14 Paquetes de subsistema Temas

Descripción de los paquetes del subsistema Temas

Plantillas: páginas para mostrar información al usuario.

Javascript: conjunto de scripts necesarios para las plantillas.

CSS: conjunto de archivos CSS necesarios para las plantillas.

Acciones: contiene las clases de las acciones responsables de la lógica del negocio, verifican la integridad de las peticiones y preparan los datos requeridos por la vista para la presentación, en cada clase se define una sola acción para un mejor entendimiento de la aplicación.

Formularios: clases que definen formularios para la entrada de datos a la aplicación.

Lib: clases que brindan algunas funcionalidades auxiliares al proyecto.

Objetos del Modelo: clases que representan una tupla de una tabla del modelo.

Tablas del Modelo: clases cuyos métodos generalmente devuelven una colección de objetos, de las clases del paquete anterior.

A continuación se presenta un ejemplo de un diagrama de clases de este subsistema para mostrar el diseño de clases contenido en los paquetes anteriores. Se tiene el controlador frontal index.php que recibe todas las peticiones y éste le indica a Symfony el módulo y la acción a ejecutar. Las acciones consultan el modelo de datos, realizan la lógica de negocio y preparan los datos; y Symfony muestra las vistas correspondientes a la acción. En la vista, las páginas servidoras construyen las páginas clientes enviadas a los usuarios. Las páginas clientes hacen link al controlador frontal, enviando como parámetro la acción a ejecutar, al igual que los formularios hacen submit a este controlador frontal.

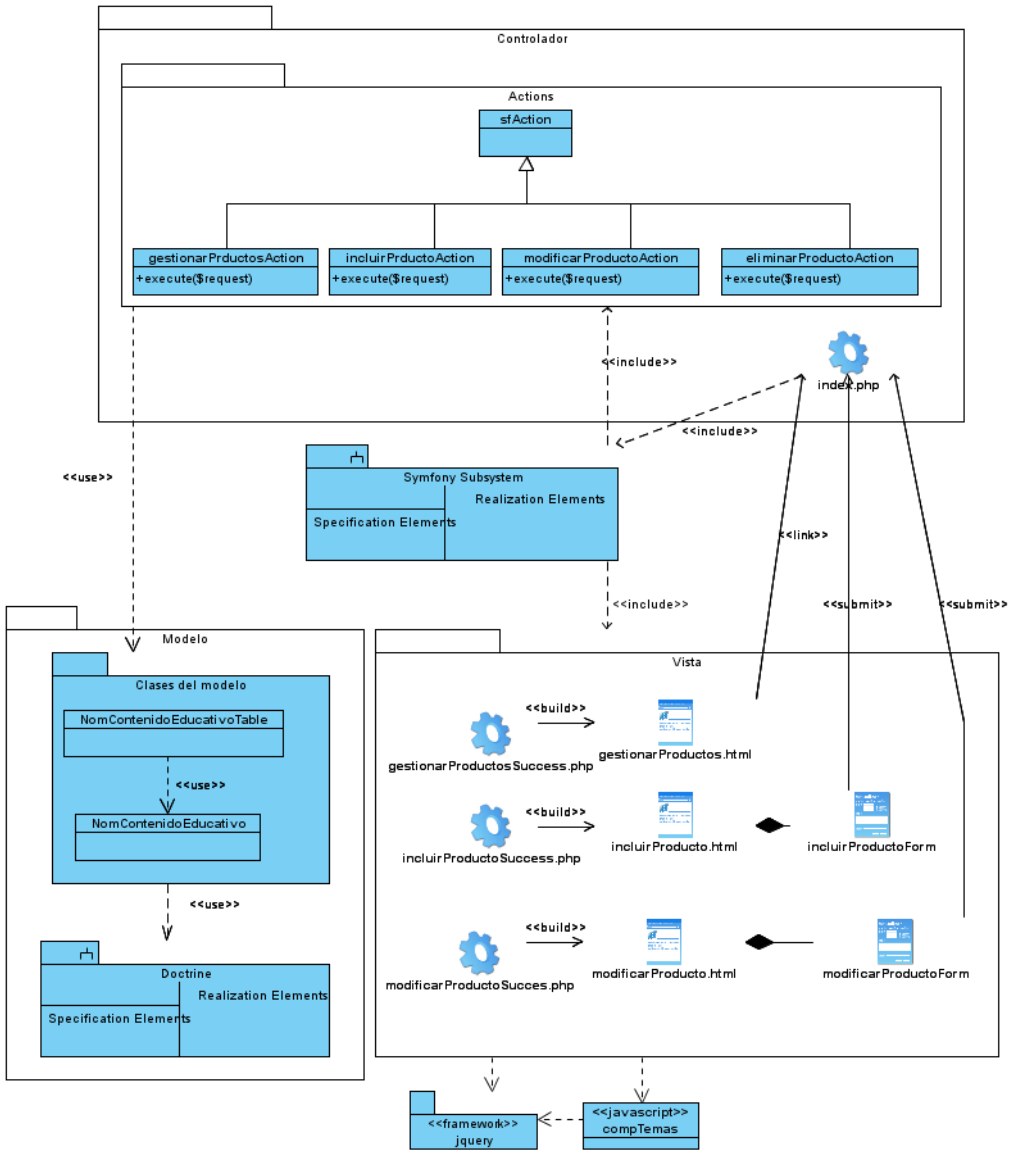


Fig.15 Ejemplo de diagrama de clases

La interfaz de la aplicación frontend tiene elementos como animaciones y efectos visuales para mejorar la interactividad con los usuarios. Para que esto sea posible la vista de la aplicación está en la PC cliente, y ésta pide los datos al servidor asincrónicamente para mostrarlos posteriormente, estos datos tienen formato de codificación de texto JSON.

La vista está compuesta de componentes visuales, que son una determinada parte de la interfaz que se va a mostrar en un área de la página. Un componente puede estar compuesto de widgets de jQuery que son elementos visuales como un menú, un árbol, un botón, etcétera y puede contener otros componentes. La comunicación entre los componentes se hace a través del patrón mediador, por lo que cada componente posee

una clase mediador que controla toda la funcionalidad del mismo y lleva a cabo la comunicación con otros componentes.

El mediador es un objeto que encapsula la interacción entre un conjunto de objetos, promoviendo el bajo acoplamiento, un objeto ya no tiene que conocer en detalles cómo interactuar con otros objetos, sino que lo hace a través del mediador.

El patrón se pone en práctica porque cada componente contiene otros componentes y un grupo de widgets por lo que sería muy engorroso si todos estos elementos se conocieran entre sí para poder intercambiar mensajes, por lo que el mediador es el que coordina la interacción entre éstos. Un mediador solo conoce al mediador que lo creó, a todos los widgets y mediadores que él crea.

Un ejemplo de cómo podría quedar una página:

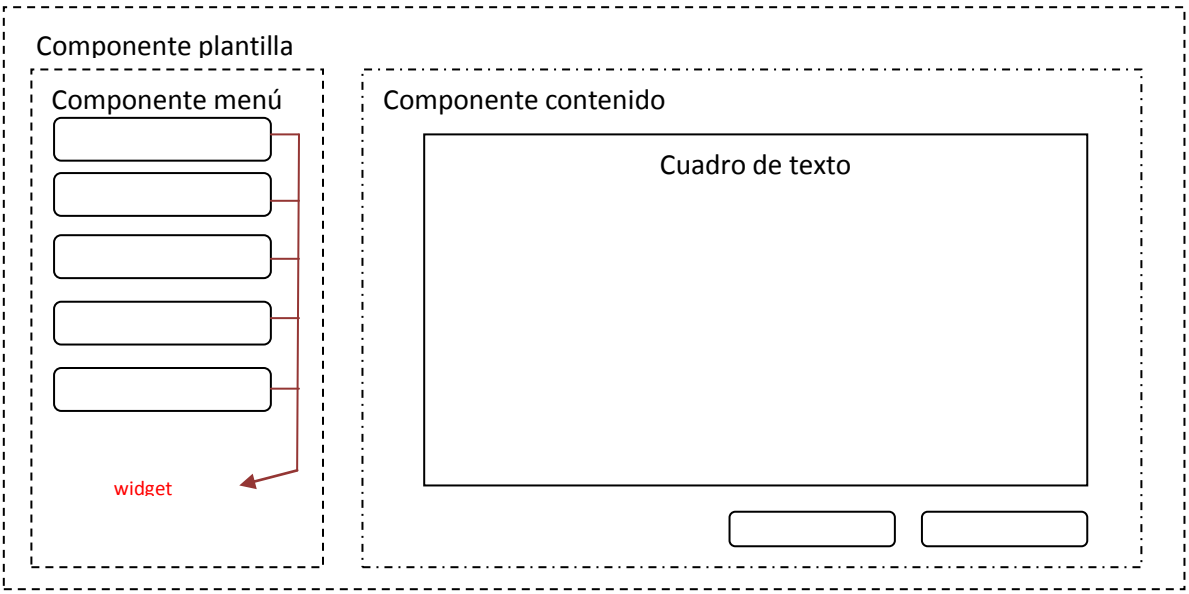


Fig.16 Estructura de una página cliente.

En esta aplicación la vista es movida completamente al cliente y el mediador será su controlador, comunicándose con el controlador del servidor para el intercambio de datos. El mediador será el encargado de construir las vistas dinámicamente utilizando plantillas html. La estructura quedaría de la siguiente forma:

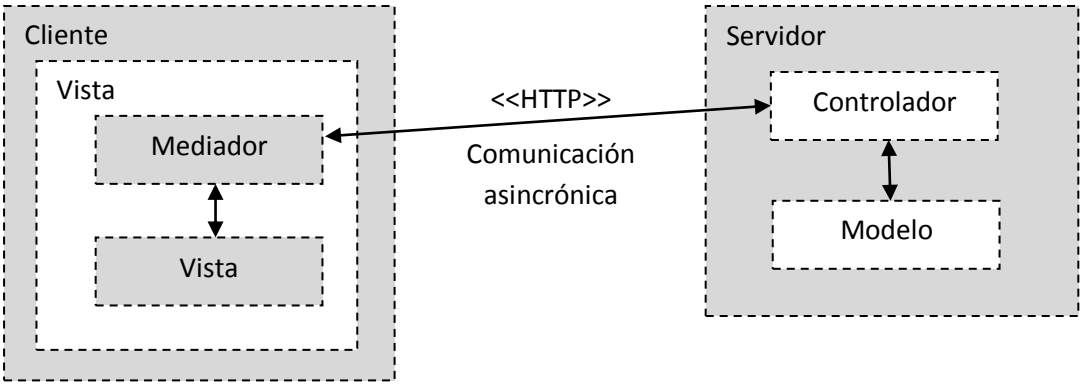


Fig.17 Estructura del patrón MVC de la aplicación frontend

El frontend está dividido en 7 subsistemas de diseño.

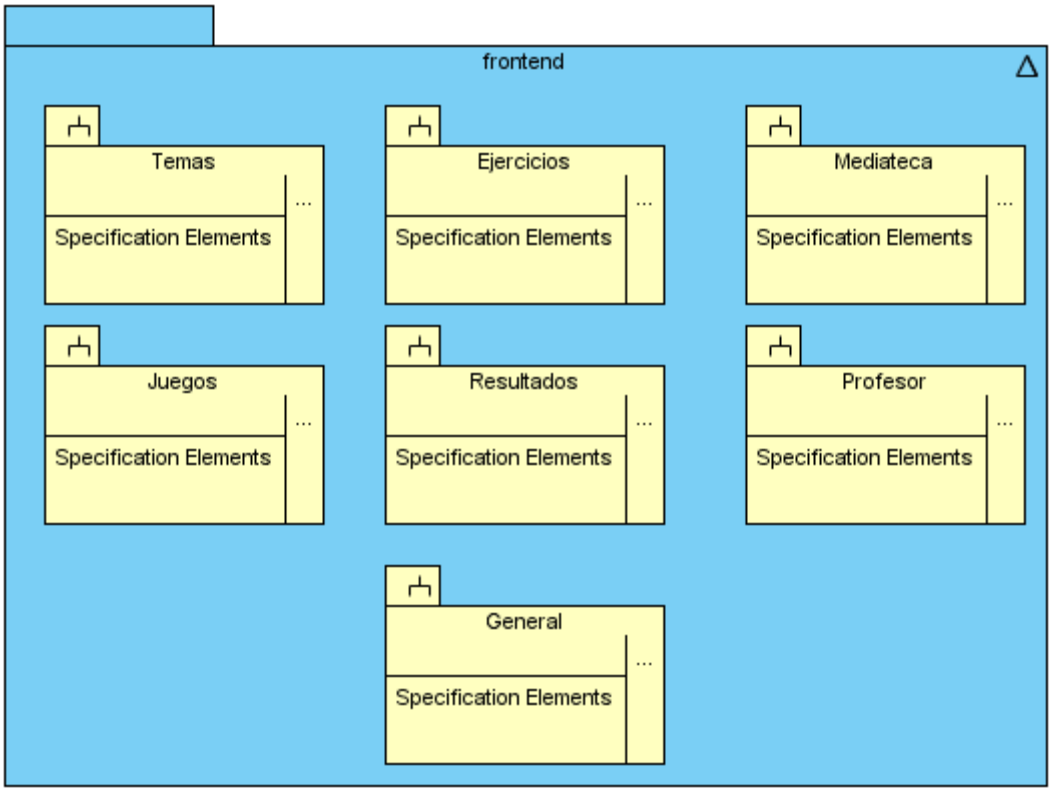


Fig.18 Subsistema de aplicación frontend

Todos los subsistemas anteriores están estructurados de la misma forma por lo que solamente se hará la representación de uno de ellos:

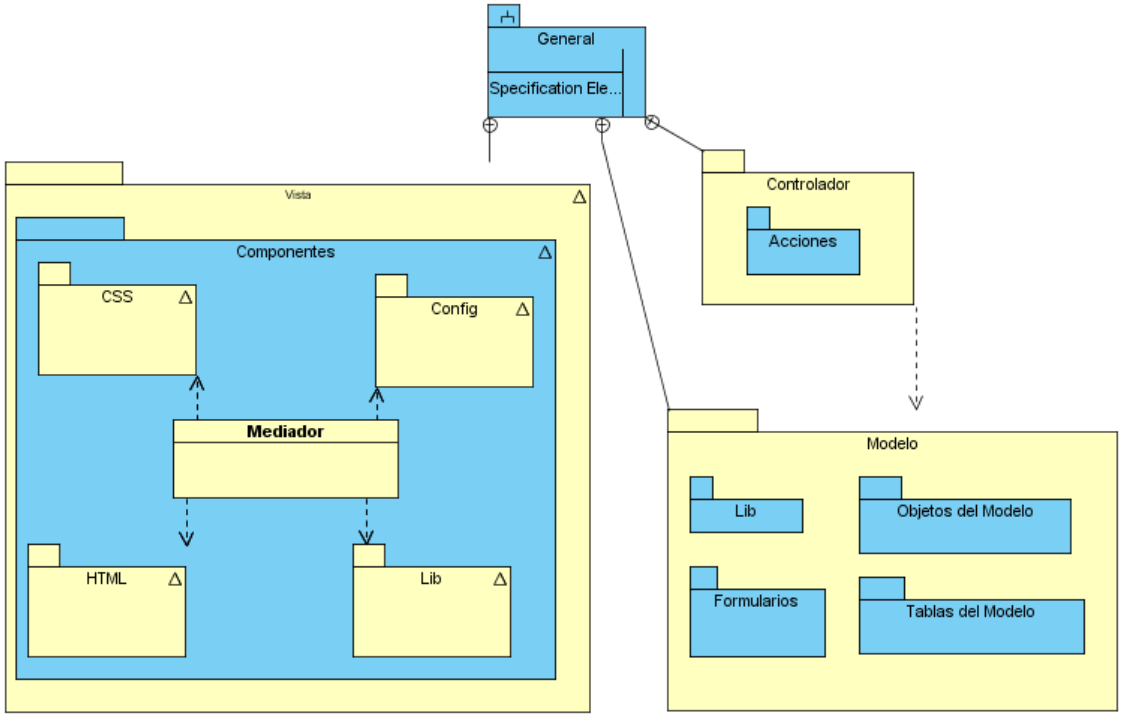


Fig.19 Paquetes que componen los subsistemas

La colección “El Navegante” cuenta con 10 productos, los cuales tienen diferentes diseños y contenidos educativos, pero la funcionalidad de uno a otro es la misma, aunque puede existir pequeños cambios o algunos presenten funcionalidades que otros no. Es por ello que la implementación de los componentes visuales será la misma para los 10 productos. Los archivos de configuración y las hojas de estilo sí cambiarán entre éstos, al permitir cambiar la interfaz visual de un producto a otro y también configurar algún comportamiento específico de los componentes visuales. Todo el estilo visual será dado por los CSS, éstos son los que especifican las imágenes, posicionamiento, tamaño, etcétera, que tendrán los elementos visuales, de forma que de un producto a otro pueda cambiar completamente la interfaz visual con solo modificar estos archivos.

Descripción de los paquetes del subsistema General

Componentes: todos los componentes visuales que conforman la vista de este subsistema.

CSS: hojas de estilo necesitadas por los componentes visuales.

Config: archivos cuyo formato es JSON que se utilizan para configurar algunos aspectos de los componentes visuales.

HTML: plantillas html que utilizan los componentes para visualizar la aplicación.

Lib: frameworks, clases y plugins Javascript utilizados en los componentes visuales.

Mediador: clase que controla el componente y lleva a cabo la comunicación con el servidor.

Acciones: contiene las clases de las acciones responsables de la lógica del negocio, verifican la integridad de las peticiones y preparan los datos requeridos por la vista para la presentación. En cada clase se define una sola acción para mejor entendimiento de la aplicación y estas acciones no tienen asociadas vistas, sino que devuelven los datos en formato JSON.

Formularios: clases que definen formularios para la entrada de datos a la aplicación.

Lib: clases que brindan algunas funcionalidades auxiliares al proyecto.

Objetos del Modelo: clases que representan una tupla de una tabla del modelo.

Tablas del Modelo: clases cuyos métodos generalmente devuelven una colección de objetos, de las clases del paquete anterior.

A continuación se tiene un ejemplo de un diagrama de clases de este subsistema para mostrar el diseño de clases contenido en los paquetes anteriores. En el cliente está la página de la aplicación, la cual incluye los componentes, clases, librerías y framework Javascript. La clase geMediadorGeneral es la encargada de controlar la página en general, toda la comunicación de ésta con el servidor se hace de forma asincrónica, ésta carga la plantilla geUsuario.html y muestra una interfaz para que los usuarios se autentiquen. Los datos entrados por el usuario son enviados al controlador frontal especificando el módulo así como la acción que deben procesar los mismos. Las acciones consultan el modelo si es necesario y elaboran una respuesta que es enviada en formato JSON.

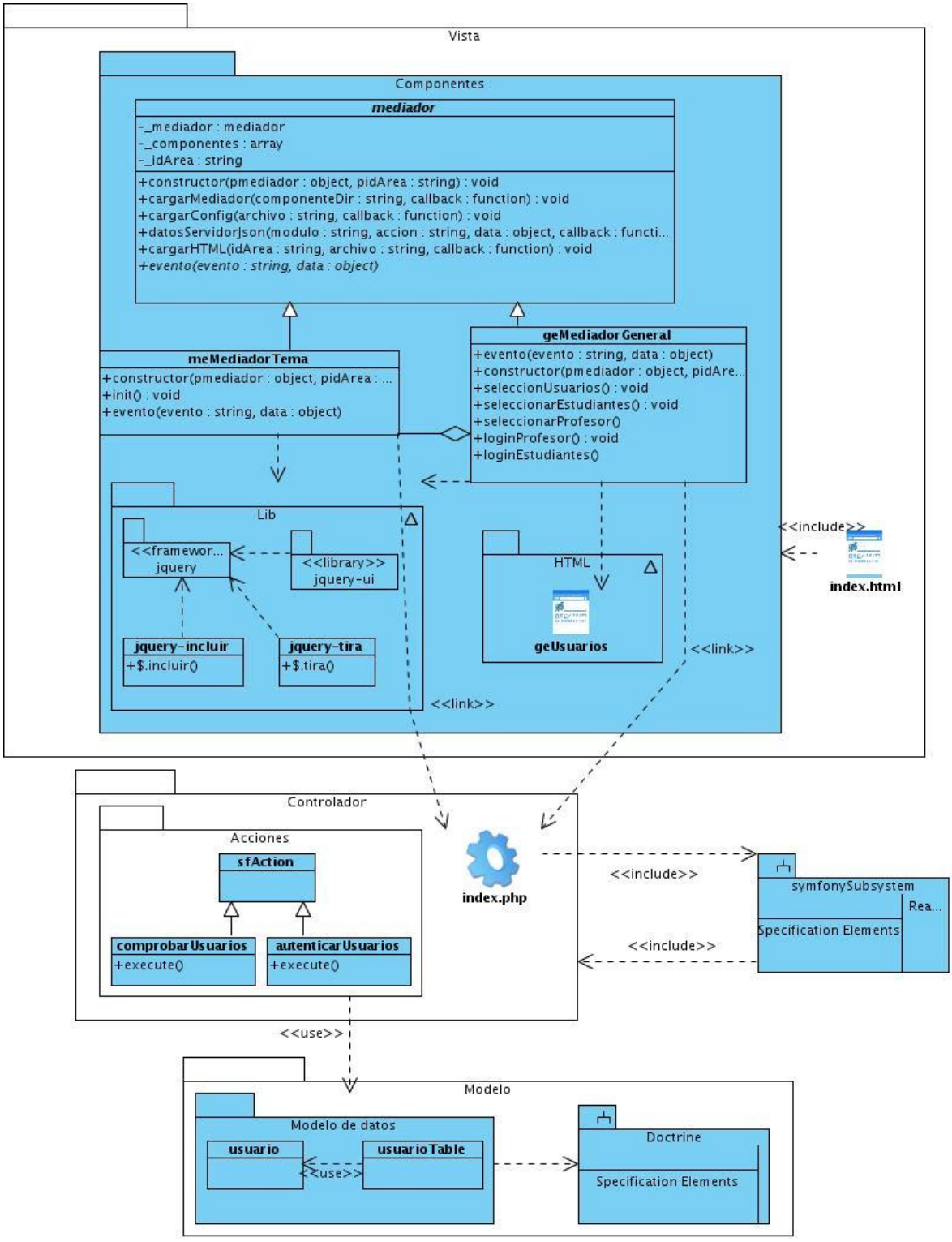


Fig.20 Ejemplo de diagrama de clases

2.10 Vista de Despliegue

El producto para su explotación puede estar desplegado en dos formas. A continuación se muestran cómo están distribuidos los nodos y dispositivos en estas dos configuraciones.

1.- Instalarlo localmente en una PC. De esta forma, todas las características del software tienen que estar en todas las PC de los estudiantes.

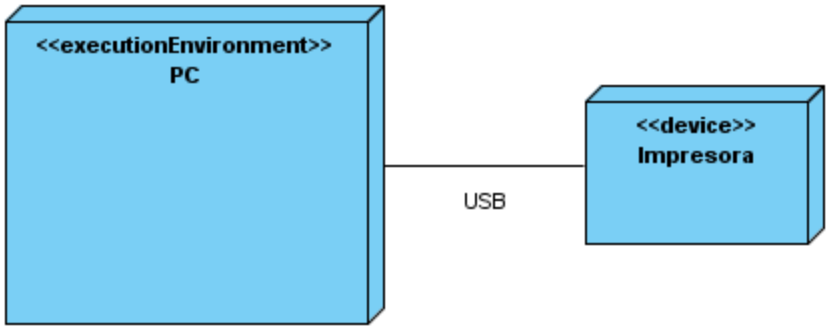


Fig.21 Diagrama de despliegue forma 1

2.- Tener una PC como servidor central donde resida el Servidor Web y el de Bases de Datos. Desde cualquier PC se puede acceder al producto mediante la web sin tenerlo instalado.

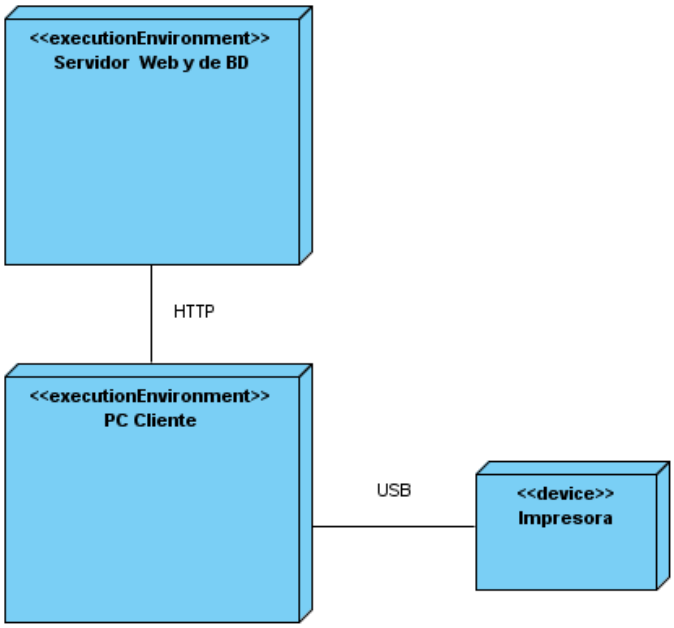


Fig.22 Diagrama de despliegue forma 2

Descripción de la capacidad que el dispositivo provee al sistema

El único dispositivo que utiliza el sistema es una impresora. El sistema tiene incluida la funcionalidad de impresión de los temas y el glosario, presentes en el software, por ello se necesita que la PC del usuario tenga conectada una impresora. No se necesita de una impresora en específico, solo requiere estar activa y lista para imprimir cuando el usuario esté trabajando con el software.

Descripción de la funcionalidad y capacidad del nodo

Forma 1:

Nodo PC: Ordenador donde se instala la Colección, es decir, se instala un Servidor Web, uno de BD y todos los archivos que necesita el software para su correcto funcionamiento.

Forma 2:

PC Cliente: Ordenador Cliente que se conecta a través de un navegador web al servidor central donde reside la Colección. No se necesita tener instalado el producto localmente.

PC Servidor: Servidor central, el cual tiene instalado un Servidor Web, uno de BD y hospeda todos los componentes necesarios para el funcionamiento del producto.

Descripción de elementos e interfaces de comunicación

<<HTTP>> Representa la conexión que se va a establecer entre una PC Cliente que se va a conectar con el servidor central donde reside el Servidor Web y BD, en otras palabras, significa la conexión entre el navegador de usuario y el servidor del sistema.

<<USB>> Conexión que existe entre la impresora y la PC Cliente del usuario. Se modela con USB, pero puede existir otro tipo, todo depende del tipo de impresora que esté usando el usuario y el tipo de conexión que utilice ésta.

2.11 Vista de Implementación

Los componentes constituyen una parte modular del sistema, desplegables y reemplazables que encapsulan implementación y un conjunto de interfaces y proporcionan la realización de los mismos. Éstos pueden ser implementados por uno o más artefactos como ficheros ejecutables, binarios, etcétera.

A continuación se presenta la distribución de los componentes de la aplicación backend, en el paquete vista se tienen los componentes correspondientes a cada módulo, que agrupan todas las plantillas que tendrá la aplicación. Este paquete utiliza los componentes: framework de Javascript jQuery, librería jQuery-UI, hojas de estilos CSS y

los archivos Javascript necesarios. El paquete controlador tiene agrupado los componentes que contienen todas las acciones disponibles en la aplicación. El componente formulario, que representa el conjunto de clases de formularios. En el modelo están presentes el ORM Doctrine para el acceso a datos y el componente modelo que contiene todas las clases del modelo de datos. Por último, el framework Symfony necesario para la ejecución de toda la aplicación.

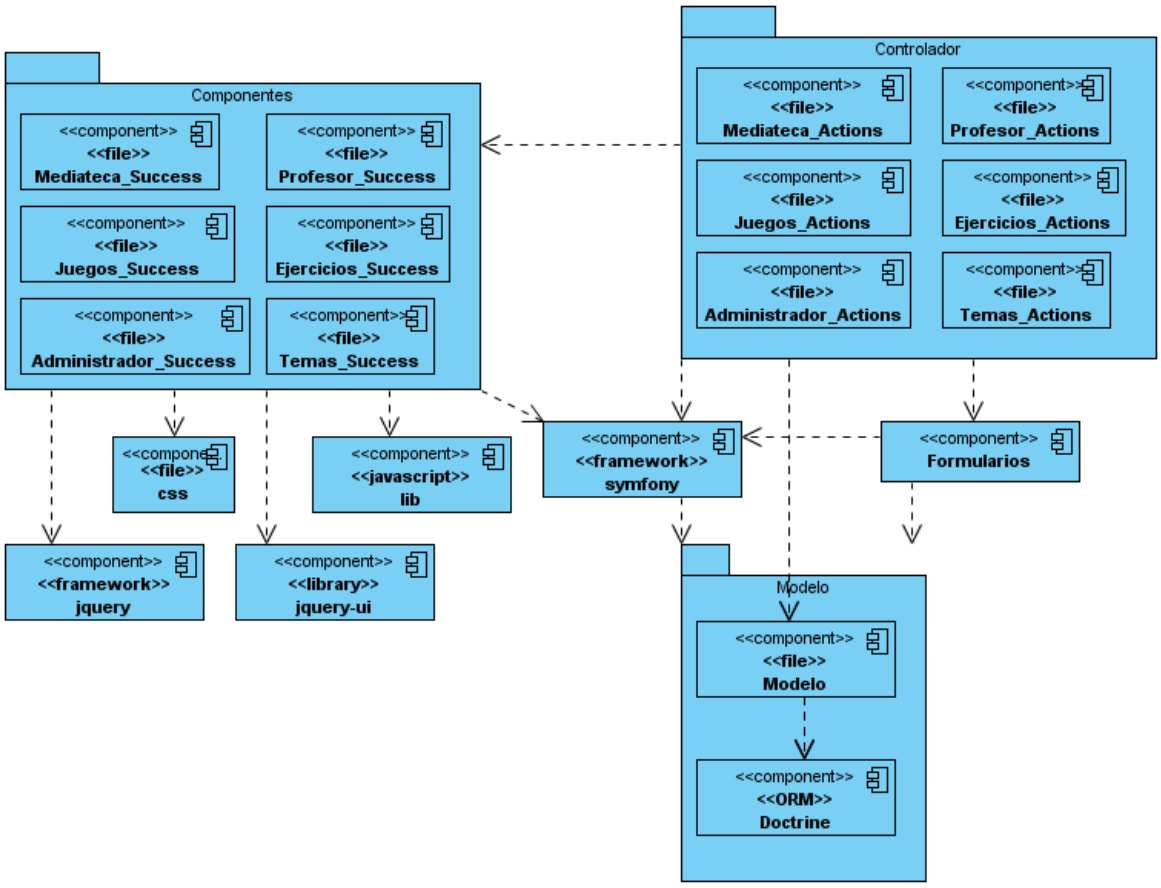


Fig.23 Diagrama de componentes de la aplicación backend

La aplicación frontend tiene en común con el backend los componentes Symfony, formularios, jQuery, jQuery-UI y el paquete modelo. El paquete controlador está compuesto por los componentes que agrupan todas las acciones. La vista está compuesta por todos los componentes que contienen las clases mediadoras de los componentes visuales. El componente lib agrupa las widgets, clases y plugins utilizados. En config están todos los archivos utilizados para configurar algunos aspectos del comportamiento de las vistas y en css las hojas de estilos necesarias.

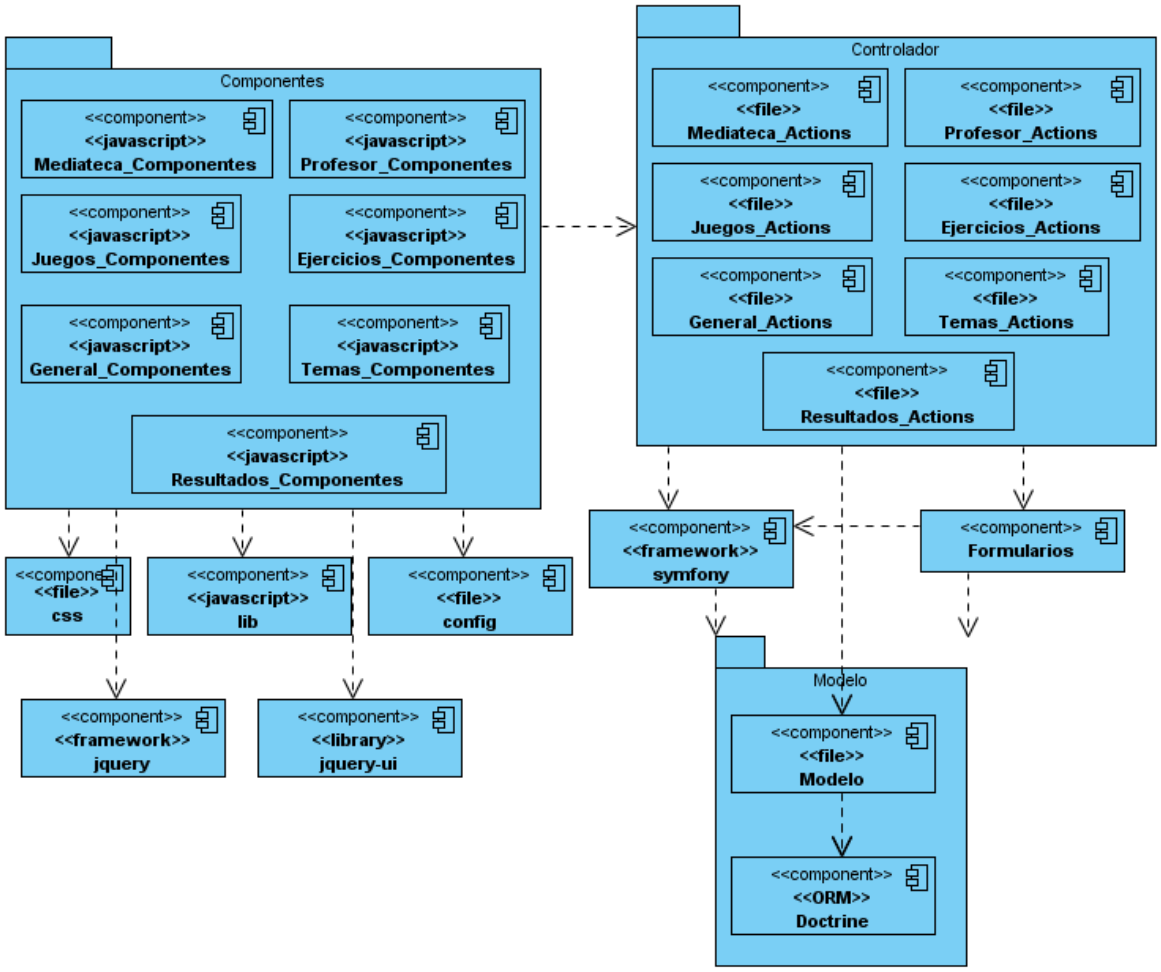


Fig.24 Diagrama de componentes de la aplicación frontend

Estructura de archivos

El proyecto tendrá la estructura de archivos propuesta por Symfony, dentro de la carpeta apps hay dos carpetas: frontend y backend, correspondientes a cada aplicación.

apps /
  frontend /
  backend /

La aplicación backend tiene un conjunto de módulos y cada uno de éstos tiene acciones y vistas. La carpeta config tiene algunas configuraciones específicas para el módulo, y la carpeta lib clases o funciones auxiliares. La estructura de todos los módulos es similar por lo que solo se muestra uno de ellos.

backend /
  modules /
    temas /

config /
lib /
actions /
templates /
ejercicios /
profesor /
general /
mediateca /
juegos /

La aplicación frontend tiene la misma estructura anterior, pero los módulos no tienen la carpeta templates pues todas las acciones devuelven los datos en formato JSON y no se requieren vistas.

La carpeta lib tiene las clases generadas por Symfony para trabajar con el modelo, dentro de éstas está la carpeta form, que tiene las clases de formularios correspondientes a cada tabla del modelo. Como los distintos módulos pueden utilizar los formularios de manera diferente, se creó la carpeta modules que tiene una carpeta para cada módulo, dentro de las cuales los usuarios crearán sus clases formularios que heredan de las generadas por Symfony.

lib /
form/
modules /
temas /
ejercicios /
profesor /
resultados /
mediateca /
juegos /

La carpeta web es la única que está pública en el servidor por problemas de seguridad. Dentro de ésta están los archivos utilizados para mostrar las vistas a los usuarios y los controladores frontales.

Dentro de web la carpeta config está compuesta por 10 carpetas, donde cada una pertenece a productos de la Colección y todas están divididas en módulos. Aquí se almacenan los archivos de configuración de los componentes visuales a mostrar, éstos tienen formato JSON.

web /
config /
EducArte /
temas /

ejercicios /
.
.
juegos /
general /

Las carpetas css e imágenes, son similares, pero además de las carpetas pertenecientes a los 10 productos hay una carpeta backend que corresponde a las hojas de estilo y a las imágenes de la aplicación backend. Las imágenes almacenadas aquí son las correspondientes a la interfaz visual, no a los contenidos educativos.

La carpeta html está compuesta por tres carpetas backend, frontend, éstas al igual que las anteriores divididas por módulos, la carpeta lib para aspectos generales y dentro de éstas la carpeta widget. Aquí se almacenan plantillas html utilizadas por los elementos de las vistas.

web /
  html /
    frontend /
    backend /
    lib /

La carpeta js tiene la misma estructura que la anterior, ésta no está dividida en productos como css, config e imágenes porque el código script y las plantillas html son las mismas para los 10 productos.

Por último, se tiene la carpeta mediateca, donde se guardan todas las medias correspondientes a los 10 productos, por lo que se crea una carpeta para cada producto y éstas se dividen en los diferentes tipos de medias.

mediateca /
  EducArte /
    sonido /
    video /
    imagenes/

Conclusiones

En este capítulo se realizó una descripción de la arquitectura del sistema utilizando el enfoque de las 4+1 vistas que propone RUP, teniendo en cuenta los objetivos y restricciones impuestas. Quedaron definidas las vistas de Casos de Uso, la Lógica, la de Despliegue y la de Implementación.

Capítulo 3: Evaluación de la arquitectura

3.1 Introducción

Definir la arquitectura es uno de los primeros pasos en el ciclo de vida del desarrollo de un software, e incorpora importantes decisiones de diseño. Estas decisiones son difíciles de cambiar una vez que el sistema se aplica, por lo se deben efectuar las pruebas necesarias una vez definida la arquitectura. El presente capítulo se refiere a la evaluación de la arquitectura, su necesidad e importancia; así como a los diferentes métodos, técnicas y atributos de calidad que se emplean para realizar la evaluación.

3.2 Evaluación de la arquitectura

La arquitectura es uno de los factores que determina el éxito o el fracaso de un software, por lo que es de vital importancia que haya sido bien diseñada. Para garantizar esto es preciso realizar la evaluación de la misma. El primer paso para la evaluación de una arquitectura es conocer qué es lo que se quiere evaluar. Por ello, la intención es más bien la evaluación del potencial de la arquitectura diseñada para alcanzar los atributos de calidad requeridos. (23)

El objetivo de la evaluación es saber si la arquitectura cumple con los requerimientos no funcionales y los atributos de calidad, para asegurar que el sistema satisfaga las necesidades demandadas por los clientes.

3.2.1 ¿Por qué evaluar la arquitectura?

Todos los diseños arquitectónicos implican desventajas en las cualidades del sistema, ya que éstas dependen en gran medida de las decisiones arquitectónicas, por lo que garantizar la calidad del producto final, está a menudo, estrechamente relacionado con avalar la calidad de la arquitectura y esto es posible si se realiza una evaluación de la misma. (24) Evaluar una Arquitectura de Software ayuda a prevenir todos los posibles desastres de un diseño que no cumple con los requerimientos de calidad del software.

Una evaluación de la arquitectura no dice si es buena o mala, solo identifica donde está el riesgo, las fortalezas y debilidades. Después se pueden tomar algunas decisiones definiendo si puede continuar el proyecto dada las debilidades identificadas, si hay que reforzar la arquitectura o si hay que comenzarla de nuevo. Es importante saber en qué etapa evaluar la arquitectura para poder determinar con mayor exactitud los posibles riesgos. (25)

Según Kazman (2001) la garantía de una arquitectura correcta cumple un papel fundamental en el éxito general del proceso de desarrollo, además del cumplimiento de los atributos de calidad del sistema. (24)

3.3 Atributos de Calidad

La calidad de software se define como el grado en el cual el software posee una combinación deseada de atributos. Tales atributos son requerimientos adicionales del sistema, que hacen referencia a características que éste debe satisfacer, diferentes a los requerimientos funcionales. (23)

Estas características o atributos son los que se conocen con el nombre de atributos de calidad, los cuales se definen como las propiedades de un servicio que presta el sistema a sus usuarios. (26)

Estos atributos de calidad pueden ser clasificados en dos categorías (24):

- Observables en tiempo de ejecución: aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución.
- No observables en tiempo de ejecución: aquellos atributos que se establecen durante el desarrollo del sistema.

A continuación se muestran los atributos de calidad (Fig. 25 y Fig. 26) y su descripción en las dos categorías mencionadas anteriormente.

Atributo de Calidad	Descripción
Disponibilidad	Es la medida de disponibilidad del sistema para el uso
Confidencialidad	Es la ausencia de acceso no autorizado a la información
Funcionalidad	Habilidad del sistema para realizar el trabajo para el cual fue concebido
Desempeño	Es el grado en el cual un sistema o componente cumple con sus funciones designadas, dentro de ciertas restricciones dadas, como velocidad, exactitud o uso de memoria.
Confiabilidad	Es la medida de la habilidad de un sistema a mantenerse operativo a lo largo del tiempo
Seguridad externa	Ausencia de consecuencias catastróficas en el ambiente. Es la medida de ausencia de errores que generan pérdidas de información
Seguridad interna	Es la medida de la habilidad del sistema para resistir a intentos de uso no autorizados y negación del servicio, mientras se sirve a usuarios legítimos

Fig.25 Descripción de atributos de calidad observables en tiempo de ejecución

Atributo de Calidad	Descripción
Configurabilidad	Posibilidad que se otorga a un usuario experto a realizar ciertos cambios al sistema
Integrabilidad	Es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados separadamente al ser integrados.
Integridad	Es la ausencia de alteraciones inapropiadas de la información
Interoperabilidad	Es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema. Es un tipo especial de *integrabilidad*
Modificabilidad	Es la habilidad de realizar cambios futuros al sistema.
Mantenibilidad	Es la capacidad de someter a un sistema a reparaciones y evolución
Portabilidad	Es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, software o una combinación de los dos
Reusabilidad	Es la capacidad de diseñar un sistema de forma tal que su estructura o parte de sus componentes puedan ser reutilizados en futuras aplicaciones
Escalabilidad	Es el grado con el que se pueden ampliar el diseño arquitectónico, de datos o procedimental
Capacidad de Prueba	Es la medida de la facilidad con la que el software, al ser sometido a una serie de pruebas, puede demostrar sus fallas. Es la probabilidad de que, asumiendo que tiene al menos una falla, el software fallará en su próxima ejecución de prueba

Fig.26 Descripción de atributos de calidad no observables en tiempo de ejecución

3.4 ¿Cuándo evaluar la arquitectura?

La evaluación clásica de la arquitectura se realiza cuando ésta se encuentra especificada totalmente y no se ha iniciado su implementación. La arquitectura puede ser evaluada en cualquier momento de desarrollo. Existen dos variaciones útiles para realizar esta evaluación, la evaluación temprana y la evaluación tardía. (27)

La evaluación temprana

No es necesario que la arquitectura esté completamente especificada para efectuar la evaluación, y esto abarca desde las fases tempranas de diseño y a lo largo del desarrollo (27).

La evaluación tardía

Se efectúa cuando la arquitectura se encuentra establecida y la implementación se ha completado. Éste es el caso general que se presenta en el momento de la adquisición de un sistema ya desarrollado.

Reglas de oro para determinar el momento de realizar evaluaciones

- ✓ Realizar una evaluación cuando el equipo de desarrollo comienza a tomar decisiones que afectan directamente a la arquitectura. (28)
- ✓ Cuando el costo de no tomar estas decisiones podría ser mayor que el costo de realizar una evaluación. (28)

3.5 Técnicas de Evaluación

Existen diversas técnicas que permiten realizar la evaluación de la arquitectura, éstas son ubicadas en dos grupos, las técnicas cualitativas y las cuantitativas. Con el uso de las técnicas de evaluación cualitativas se pueden utilizar escenarios, cuestionarios o listas de verificación, mientras que con el uso de las técnicas de evaluación cuantitativas se pueden emplear métricas, simulaciones, prototipos, experimentos o modelos matemáticos.

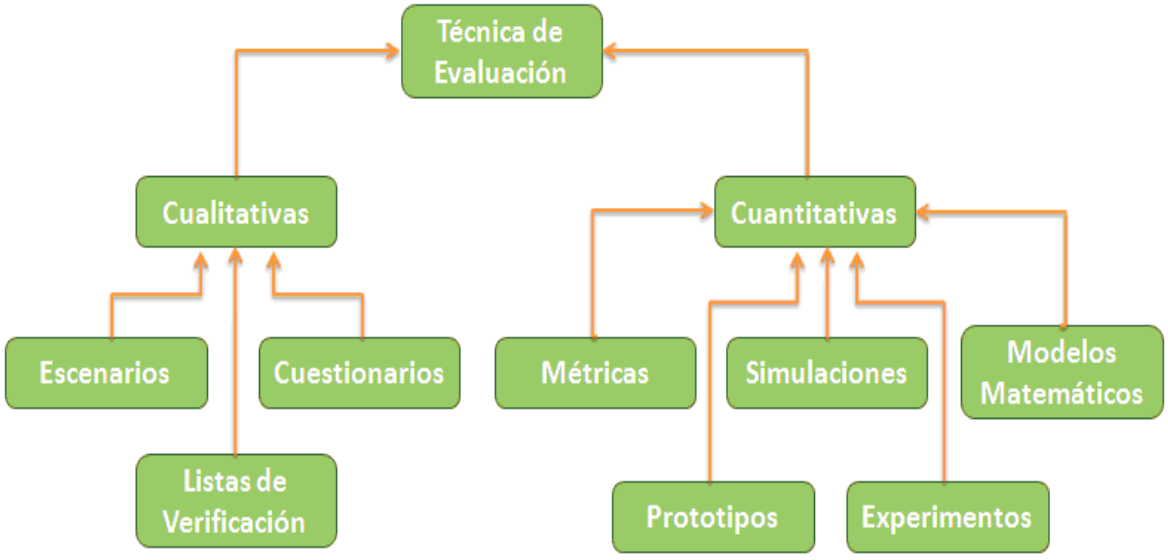


Fig.27 Clasificación de las Técnicas de Evaluación

3.5.1 Evaluación basada en simulación

La evaluación basada en simulación utiliza una implementación de alto nivel de la Arquitectura de Software. El enfoque básico consiste en la implementación de componentes de la arquitectura y la implementación a cierto nivel de abstracción del contexto del sistema donde se supone va a ejecutarse. La finalidad es evaluar el comportamiento de la arquitectura bajo diversas circunstancias. Una vez disponibles estas implementaciones, pueden usarse los perfiles respectivos para evaluar los atributos de calidad. (25)

3.5.2 Evaluación basada en modelos matemáticos

La evaluación basada en modelos matemáticos se utiliza para evaluar atributos de calidad operacionales. Permite una evaluación estática de los modelos de diseño arquitectónico y se presenta como alternativa a la simulación, dado que evalúan el mismo tipo de atributos. Ambos enfoques pueden ser combinados utilizando los resultados de uno como entrada para el otro (25). Entre las desventajas que presenta esta técnica se encuentra la inexistencia de modelos matemáticos apropiados para los atributos de calidad relevantes, y el hecho de que el desarrollo de un modelo de simulación completo puede requerir esfuerzos sustanciales.

3.5.3 Evaluación basada en experiencia

En muchas ocasiones los arquitectos e ingenieros de software otorgan valiosas ideas que resultan de utilidad para la evasión de decisiones erradas de diseño. Aunque todas estas experiencias se basan en factores subjetivos como la intuición y la práctica, la mayoría logran ser justificadas por una línea de razonamiento y pueden ser la base de otros enfoques de evaluación (25). La evaluación basada en experiencia puede dividirse en dos grupos: la evaluación informal, que es la realizada por los arquitectos de software durante el proceso de diseño y la realizada por equipos externos de evaluación de arquitecturas.

3.5.4 Evaluación basada en escenarios

De acuerdo con Kazman, un escenario es una breve descripción de la interacción de alguno de los involucrados en el desarrollo del sistema con éste. Un escenario consta de tres partes: el estímulo, el contexto y la respuesta.

El estímulo es la parte del escenario que explica o describe lo que el involucrado en el desarrollo hace para iniciar la interacción con el sistema. Puede incluir ejecución de

tareas, cambios en el sistema, ejecución de pruebas, configuración, entre otros. **El contexto** describe qué sucede en el sistema al momento del estímulo.

La respuesta describe, a través de la arquitectura, cómo debería responder el sistema ante el estímulo. Este último elemento es el que permite establecer cuál es el atributo de calidad asociado. (24)

En la actualidad para el empleo de las técnicas basadas en escenarios se usan dos instrumentos de evaluación: el Árbol de Utilidades (Utility Tree) propuesto por Kazman y los Perfiles (Profiles), propuesto por Bosch.

Árbol de Utilidades

Un Árbol de Utilidades es un esquema en forma de árbol que presenta los atributos de calidad de un sistema de software, refinados hasta el establecimiento de escenarios que especifican con suficiente detalle el nivel de prioridad de cada uno. La intención del uso del Árbol de Utilidades es la identificación de los atributos de calidad más importantes para un proyecto particular. No existe un conjunto preestablecido de atributos, sino que son definidos por los involucrados en el desarrollo del sistema al momento de la construcción del árbol.

El Árbol de Utilidades contiene como nodo raíz la utilidad general del sistema. Los atributos de calidad asociados al mismo conforman el segundo nivel del árbol, los cuales se refinan hasta la obtención de un escenario lo suficientemente concreto para ser analizado y otorgarle prioridad a cada atributo considerado. Cada atributo de calidad perteneciente al árbol contiene una serie de escenarios relacionados, y una escala de importancia y dificultad asociada, que será útil para efectos de la evaluación de la arquitectura. (24)

Perfiles

Se le denomina perfil, al conjunto de escenarios generalmente con alguna importancia relativa asociada a cada uno de ellos. El uso de perfiles permite hacer especificaciones más precisas del requerimiento para un atributo de calidad.

Los perfiles tienen asociados dos formas de especificación: perfiles completos y perfiles seleccionados. Los perfiles completos definen todos los escenarios relevantes como parte del perfil. Esto permite al Ingeniero de Software realizar un análisis de la arquitectura para el atributo de calidad estudiado de una manera completa, pues incluye todos los posibles

casos. Su uso se reduce a sistemas relativamente pequeños y solo es posible predecir conjuntos de escenarios completos para algunos atributos de calidad.

Los perfiles seleccionados se asemejan a la selección de muestras sobre una población en los experimentos estadísticos. Se toma un conjunto de escenarios de forma aleatoria, de acuerdo con algunos requerimientos. Si bien es informal, permite hacer proposiciones científicamente válidas.

3.6 Métodos de evaluación

Un método de evaluación sirve de guía a los involucrados en el desarrollo del sistema para la búsqueda de conflictos que puede presentar una arquitectura y sus soluciones (23). Seguidamente se abordan alguno de los métodos:

3.6.1 Método de Análisis de Arquitecturas de Software (SAAM)

El Método de Análisis de Arquitecturas de Software (Software Architecture Analysis Method, SAAM) es el primero que fue ampliamente promulgado y documentado. Fue originalmente creado para el análisis de la modificabilidad de una arquitectura, pero en la práctica ha demostrado ser muy útil para evaluar de forma rápida distintos atributos de calidad, tales como modificabilidad, portabilidad, escalabilidad e integrabilidad. (23)

El método de evaluación SAAM se enfoca en la enumeración de un conjunto de escenarios que representan los cambios probables a los que estará sometido el sistema en el futuro. Como entrada principal, es necesaria alguna forma de descripción de la arquitectura a ser evaluada. (23)

Con la aplicación de este método, si el objetivo de la evaluación es una sola arquitectura, se obtienen los lugares en los que la misma puede fallar, en términos de los requerimientos de modificabilidad.

Para el caso en el que se cuenta con varias arquitecturas candidatas, el método produce una escala relativa que permite observar que opción satisface mejor los requerimientos de calidad con la menor cantidad de modificaciones. (24)

3.6.2 Método de Análisis de Acuerdos de Arquitectura (ATAM)

El Método de Análisis de Acuerdos de Arquitectura (Architecture Trade-off Analysis Method, ATAM) está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM, explicado anteriormente. El nombre del método ATAM surge del hecho de que revela la forma en

que una arquitectura específica satisface ciertos atributos de calidad y provee una visión de cómo los atributos de calidad interactúan con otros, esto es, los tipos de acuerdos que se establecen entre ellos. (24)

El método se concentra en la identificación de los estilos arquitectónicos o enfoques arquitectónicos utilizados, estos elementos representan los medios empleados por la arquitectura para alcanzar los atributos de calidad, así como también permiten describir la forma en la que el sistema puede crecer, responder a cambios, e integrarse con otros sistemas, entre otros. (24)

Metodología ATAM:

El método de evaluación ATAM comprende 9 pasos, agrupados en 4 fases:

Fase 1: Presentación

1. Presentación del ATAM
2. Presentación de las metas del negocio
3. Presentación de la arquitectura

Fase 2: Investigación y análisis

4. Identificación de los enfoques arquitectónicos
5. Generación del Utility Tree
6. Análisis de los enfoques arquitectónicos

Fase 3: Pruebas

7. Lluvia de ideas y establecimiento de prioridad de escenarios
8. Análisis de los enfoques arquitectónicos

Fase 4: Reporte

9. Presentación de los resultados

3.6.3 Método de Análisis de Diseños Intermedios (ARID)

El método de Análisis de Diseños Intermedios (Active Reviews for Intermediate Designs, ARID) es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. En ocasiones, es necesario saber si un diseño propuesto es conveniente, desde el punto de vista de otras partes de la arquitectura. Según los autores, ARID es un híbrido entre Active Design Review (ADR) y Architecture Trade-off Method

(ATAM), descrito anteriormente. ADR es utilizado para la evaluación de diseños detallados de unidades del software como los componentes o módulos. Las preguntas giran en torno a la calidad y completitud de la documentación y la suficiencia, el ajuste y la conveniencia de los servicios que provee el diseño propuesto. (23)

Kazman propone que tanto ADR como ATAM proveen características útiles para el problema de la evaluación de diseños preliminares, dado que ninguno por sí solo es conveniente. En el caso de ADR, los involucrados reciben documentación detallada y completan cuestionarios, cada uno por separado. En el caso de ATAM, está orientado a la evaluación de toda una arquitectura (23). De la combinación de ambas filosofías surge ARID, para efecto de la evaluación temprana de los diseños de una Arquitectura de Software.

3.7 Resultados de la evaluación

Si las decisiones arquitectónicas determinan los atributos de calidad del sistema, entonces es posible evaluar dichas decisiones con respecto a su impacto sobre estos atributos. Muchos de los atributos no pueden ser medidos directamente, por lo que es preciso realizar el análisis de la arquitectura para determinar cuán satisfactoria es para el propósito del sistema.

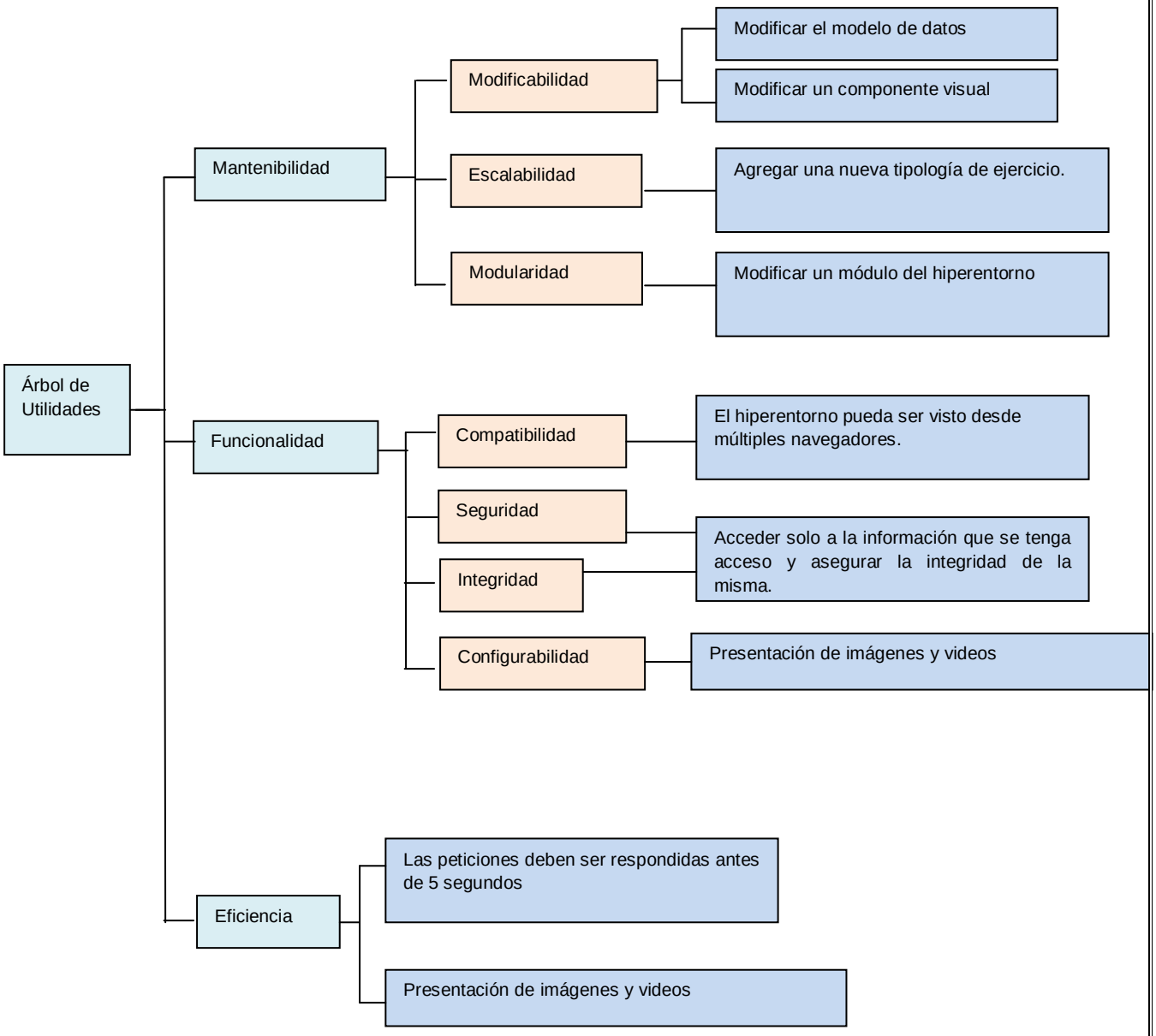
Para evaluar el diseño arquitectónico antes propuesto, luego de un análisis de algunas de las técnicas y métodos de evaluación, se decidió utilizar de las técnicas cualitativas, la evaluación basada en escenarios. Se usó el instrumento Árbol de Utilidades, para identificar cuáles son los atributos de calidad relevantes y el método ATAM para evaluar toda la arquitectura y comprobar si satisface los atributos de calidad.

Siguiendo los pasos del método seleccionado, se realizó la evaluación de la arquitectura definida en conjunto con los integrantes del equipo de desarrollo del proyecto. Según las fases del ATAM, el primer paso que se llevó a cabo fue realizar la presentación del método, donde se explicó su funcionamiento. Posteriormente se discutieron las metas del negocio donde fueron debatidos los principales conceptos presentes en el mismo, los objetivos que se persiguen, el alcance del proyecto, se dejó claro que se persiguen objetivos de tipo arquitectónicos. Para facilitar el entendimiento común de todos se describió la arquitectura para constatar que ésta cumple con dichos objetivos y contribuye a dar cumplimiento a los atributos de calidad que debe satisfacer el software. Luego se analizaron los atributos de calidad refinados hasta obtener los escenarios, se determinaron cuáles eran los más importantes para el sistema y quedó conformado el

Árbol de Utilidades que se muestra seguidamente, e identificados los escenarios, los que son considerados casos de pruebas que dan la posibilidad de confirmar todo el análisis desarrollado.

Árbol de Utilidades

El siguiente Árbol de Utilidades representa los atributos de calidad, refinados a escenarios, que son más importantes para el desarrollo, para percibir su comportamiento según el diseño arquitectónico.



Análisis de los escenarios.

Escenario	Modificar el modelo de datos				
Atributo	Modificabilidad				
Ambiente	El modelo de datos haya cambiado debido a modificaciones en los requerimientos				
Estímulo	Se cambia el diseño de la base de datos				
Respuesta	Se vuelve a mapear el modelo de datos a clases a través de Doctrine				
Decisiones de la Arquitectura		**Riesgo**	**Sensibilidad**	**Tradeoff**	**No Riesgo**
-- Utilizar el ORM Doctrine permite mapear la base de datos a objetos en lenguaje PHP. De manera que si cambia el modelo de datos cambiarán solamente las clases que corresponden a las tablas modificadas en la base de datos.				T1	NR1

Para este escenario hay un tradeoff, consiste en usar el ORM porque éste brinda una abstracción de la base de datos, la cual agiliza el desarrollo, permite la mantenibilidad y organiza el diseño del acceso a datos, concibiendo que disminuya la eficiencia a la hora de manipular los datos. Como no riesgo se tiene que modificar el modelo; no presentará problemas porque solo afecta a la parte cambiada. Doctrine presenta una clase base por cada tabla la cual es generada a partir de un esquema que describe la base de datos. Si se desea agregar funcionalidades a esa clase o cambiar alguna de las que presenta, estos cambios se realizan en otra clase que herede de ésta por lo que cuando se hagan cambios en el modelo se genera la clase base nuevamente y los cambios realizados permanecen en la clase hija.

Escenario	Agregar una nueva tipología de ejercicio
Atributo	Escalabilidad
Ambiente	Surge la necesidad de incorporar otra tipología de ejercicio a la Colección
Estímulo	Se necesita agregar una nueva tipología de ejercicio

Respuesta	Se agregan tablas al modelo de ser necesario y se crea un nuevo componente visual que consiste en el nuevo ejercicio			
Decisiones de la Arquitectura	**Riesgo**	**Sensibilidad**	**Tradeoff**	**No Riesgo**
--Utilizar PHP y Javascript para gestionar y mostrar el ejercicio		S1		
--La vista está compuesta por componentes visuales				

Para agregar una tipología de ejercicio, de ser preciso, se añaden tablas al modelo, pero éstas no afectan al diseño actual, solo se agregan a él. A la hora de mostrar el ejercicio se crea únicamente un nuevo componente visual y se controla cuándo mostrarlo. Sin embargo hay un punto de sensibilidad en que resulta muy difícil o imposible la representación y la gestión del ejercicio con las herramientas que se están utilizando.

Escenario	La Colección pueda ser vista desde múltiples navegadores			
Atributo	Compatibilidad			
Ambiente	Uso de la Colección			
Estímulo	Navegar por la Colección			
Respuesta	Se ve en el navegador la Colección con el mismo diseño para todos los navegadores			
Decisiones de la Arquitectura	**Riesgo**	**Sensibilidad**	**Tradeoff**	**No Riesgo**
--Utilizar jQuery como framework Javascript.				NR2
--Utilizar HTML, CSS y Javascript para la visualización	R1		T2	

Los diferentes navegadores web, disponibles actualmente para los sistemas operativos, no respetan o implementan la totalidad de los estándares y las formas en que dan soporte a Javascript y CSS no es la misma, por lo que consiste un riesgo en el desarrollo. Para lograr que se pueda navegar en la Colección con varios navegadores hay que tener en cuenta que cada funcionalidad que se implemente cumpla con estos requisitos. Como tradeoff, se tiene que algunos elementos de interfaz visual se pueden lograr a través de clases CSS y sin utilizar imágenes, mejorando el rendimiento, pero todos los navegadores

no soportan las mismas propiedades CSS, lo que llevaría a utilizar imágenes. Usar jQuery representa un no riesgo ya que este framework, además de sus grandes prestaciones para el trabajo con AJAX, CSS y manipulación del DOM, tiene todas sus funcionalidades soportadas por todos los navegadores, por lo que usarlo asegura la compatibilidad de la Colección con los navegadores.

Escenario	Las peticiones deben ser respondidas antes de 5 segundos			
Atributo	Eficiencia			
Ambiente	Uso de la Colección			
Estímulo	Navegar a una pantalla específica			
Respuesta	Se debe mostrar la pantalla antes de 5 segundos			
Decisiones de la Arquitectura	**Riesgo**	**Sensibilidad**	**Tradeoff**	**No Riesgo**
--Mover la vista completamente al cliente.		S2		
--Utilización de AJAX.				
--El servidor brinda los datos en formato JSON			T3	
--Usar el framework Symfony y el ORM Doctrine		S3	T4	

La utilización de AJAX mejora la interactividad con el usuario porque se pueden realizar cambios en las páginas sin necesidad de recargarlas. Las peticiones ocurren en segundo plano, así no interfieren en la visualización y se solicita solo la información a mostrar, esto hace que la cantidad de información pedida al servidor sea menor y se pueda cambiar la interfaz de manera más eficiente. Al mover la vista completamente al cliente hay un punto sensible, ya que se debe enviar una gran cantidad de archivos Javascript al cliente porque éstos son los encargados de visualizarlo todo, para mejorar la eficiencia estos archivos se cargarán en demanda, se enviarán varios en una misma petición y además estarán minimizados y comprimidos.

Otro punto sensible se manifiesta a través del framework Symfony que para dar respuesta a una petición, ésta pasa por un grupo de filtros y se cargan un gran número de clases. Todo esto requiere un mayor consumo de recursos y provoca que se prolongue el tiempo de respuesta. Como tradeoff se tiene que al pedir los datos en formato JSON al servidor se disminuye el procesamiento del lado del servidor y se pueden lograr respuestas más

rápidas, pero para esto debe crearse mayor cantidad de código Javascript para ser enviado al cliente. Otro tradeoff es que el uso del framework Symfony y el ORM Doctrine requiere un mayor número de recursos del servidor, pero brinda un conjunto de clases que agilizan el desarrollo y facilitan la mantenibilidad, modificabilidad y seguridad, así como la escalabilidad del sistema.

Escenario	Modificar un componente visual				
Atributo	Modificabilidad				
Ambiente	Se necesita cambiar la interfaz de un componente				
Estímulo	Cambiar la interfaz de un componente				
Respuesta	Debe ser posible cambiar la interfaz sin necesidad de modificar el controlador y el modelo				
Decisiones de la Arquitectura		**Riesgo**	**Sensibilidad**	**Tradeoff**	**No Riesgo**
--Uso del patrón MVC					NR3
--Mover la vista completamente al cliente					

Al usar el patrón MVC se separa la lógica de negocios de la forma en que se presentarán los datos, por lo que hacer uso de éste para el desarrollo constituye un no riesgo. Para modificar o crear un componente visual que realice la presentación de un conjunto de datos, no es obligatorio modificar el controlador o el modelo, solamente se modifica el componente, o se crea si es nuevo. El controlador se encarga de recibir peticiones de la vista y entregar los datos en formato JSON por lo que de no cambiar la lógica de negocio no será necesario modificar el controlador ni el modelo. Incluso hay varias vistas que pueden solicitar los mismos datos al controlador.

Escenario	Acceder solo a la información que se tenga autorización y asegurar la integridad de la misma
Atributo	Integridad, Seguridad
Ambiente	Uso de la Colección
Estímulo	Acceder o modificar información
Respuesta	Según los permisos del usuario éste puede acceder o gestionar determinada información

Decisiones de la Arquitectura	Riesgo	Sensibilidad	Tradeoff	No Riesgo
--Uso de Symfony				
--Uso de PostgreSQL como gestor de base de datos				NR4

El gestor de bases de datos PostgreSQL posee integridad transaccional lo que asegura la consistencia de los datos y las transacciones realizadas con los mismos, así se garantiza la integridad de la información que constituye un no riesgo. La seguridad se hace a través del framework Symfony el cual trabaja con las sesiones de usuarios. Para asegurar la aplicación en los puntos que se desee, Symfony tiene un sistema de credenciales que son requeridas para cada petición, los usuarios que no las posean no podrán acceder. También posee protección contra ataques de tipo CSRF (Cross Site Request Forgery) que se basan en explotar la confianza que los sitios web tienen con sus usuarios.

Escenario	Modificar un módulo de la Colección			
Atributo	Modularidad			
Ambiente	Necesidad de modificar un módulo de la Colección			
Estímulo	Modificar un módulo			
Respuesta	Los otros módulos de la Colección no deben ser afectados			
Decisiones de la Arquitectura	**Riesgo**	**Sensibilidad**	**Tradeoff**	**No Riesgo**
-Diseño de la aplicación en subsistemas		S4		

Al dividir el desarrollo en subsistemas, se estableció que unos fueran independientes de otros, lo que constituye un punto de sensibilidad debido a que algunos de estos subsistemas puedan depender necesariamente de las funcionalidades de otro, aunque sea en pocas ocasiones. Si sucediera esto hay que tener bien identificado los puntos de contacto para mantener los subsistemas funcionales. El diseño modular permite una gran mantenibilidad del sistema ya que los subsistemas están bien desacoplados.

Escenario	Presentación de imágenes y videos
Atributo	Eficiencia, Configurabilidad
Ambiente	Petición de imágenes y videos

Estímulo	Visualización de imágenes o videos			
Respuesta	Son mostrados las imágenes o videos			
Decisiones de la Arquitectura	**Riesgo**	**Sensibilidad**	**Tradeoff**	**No Riesgo**
-Utilizar imágenes en formato PNG8				
-Utilizar un servidor streaming de ser necesario			T5	

Al utilizar imágenes para el diseño en formato PNG8 y eliminando los metadatos de éstas, se hacen considerablemente más pequeñas y a la vista no pierden calidad visual, esto hace que el tráfico sea mucho menor logrando un mayor rendimiento. Como tradeoff se tiene que según la cantidad de usuarios que utilizarán el servidor se decide si montar o no un servidor streaming para los videos, esto es necesario para mejorar la eficiencia del servidor cuando hay un gran número de usuario, pero requiere configuraciones adicionales o incluso puede ser necesario un servidor solamente dedicado a esta tarea.

La evaluación de la arquitectura demostró que la utilización del patrón MVC y de los framework Symfony y jQuery, dan cumplimiento a atributos de calidad como son la escalabilidad, modificabilidad y mantenibilidad de la aplicación, logrando esto uno de los principales objetivos. El uso de AJAX para mejorar la interactividad, trasladando la vista al cliente, logra una gran eficiencia en la aplicación; aunque hay que analizar bien el orden y el momento de carga de los archivos para que no se vea afectada la navegabilidad del usuario. Se detectó, que para que el entorno pueda ser visto desde múltiples navegadores hay que seguir este aspecto bien de cerca a la hora de incorporar el diseño y cada unas de las funcionalidades del software. La seguridad e integridad de los datos, están aseguradas por las potencialidades del gestor de bases de datos y el sistema de seguridad proporcionado por Symfony. El diseño modular permite un bajo acoplamiento entre los componentes del sistema, logrando que sea modificable y mantenible.

Conclusiones

En este capítulo se realizó la evaluación de la arquitectura antes definida. Se aplicó el método ATAM y la técnica basada en escenarios con el instrumento Árbol de Utilidades. Igualmente se identificaron los posibles puntos de sensibilidad o riesgos asociados que viabilizaron el perfeccionamiento de las decisiones arquitectónicas optadas con anterioridad.

Conclusiones Generales

Con la culminación del presente trabajo de diploma se concluye que el resultado de la puesta en práctica de la propuesta es satisfactorio, esto fue posible gracias a aspectos como la investigación que se llevó a cabo, que incluye el análisis de los puntos más importantes de la Arquitectura de Software para la realización de aplicaciones web. Dicha investigación, junto a los conocimientos adquiridos durante el proceso de desarrollo, hicieron posible realizar la definición de la arquitectura de la versión multiplataforma de la colección “El Navegante”, haciendo uso de las herramientas y tecnologías adecuadas. Con la utilización de la técnica de evaluación basada en escenarios y el método de evaluación ATAM se realizó la evaluación de la arquitectura, lo que ayudó a identificar los posibles puntos de riesgo así como constatar que dicha arquitectura fuera óptima y cumpliera con las metas trazadas, contribuyendo a la organización y a tener un mayor grado de confiabilidad en el sistema que se espera obtener como resultado.

Recomendaciones

A partir del trabajo realizado se sugieren las siguientes recomendaciones al proyecto que dará continuidad al desarrollo de la Colección:

- Durante el ciclo de vida de la Colección, continuar el refinamiento constante de la arquitectura propuesta.
- Optimizar la configuración de los servidores Apache2 y PostgreSQL para aprovechar mejor sus características de balanceo de carga, seguridad y capacidad de procesamiento.
- En etapas posteriores realizar la evaluación tardía de la arquitectura propuesta, garantizando así, la obtención de un resultado completo y detallado de los riesgos que puedan afectar la arquitectura.

Referencias Bibliográficas

1. **Jr, Frederick Brooks.** *The mythical man-month.* s.l. : Addison-Wesley, 1975.

2. **Perry, Dewayne y Wolf, Alexander.** *Foundations for the study of software architecture.* s.l. : ACM SIGSOFT Software Engineering, 1992.

3. *Introducción a la Arquitectura de Software.* **Reynoso, Carlos Billy.** 2005.

4. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* **Billy, Carlos y Kicillof, Nicolás.** 2004.

5. *Arquitectura Base sobre la Web.* **Pimentel, Luis y Perez, Yosev.** 2008.

6. **Dávila, Jose A Vela.** Jose A Vela Dávila. [En línea] Ponencia en Seminario, 2008. http://www.cimat.mx/Eventos/seminariotecnologias08/javd.pdf-.

7. *Metodologías de Desarrollo de Software.* **Sanchez, María A Mendoza.** 2004.

8. *Lenguajes de Descripción de Software.* **Reynoso, Carlos y Kicillof, Nicolás.** 2004.

9. *Applying UML and Patterns -An Introduction to Object- Oriented Analysis and Design and Iterative Development.* **Larman, Craig.** 2005, Vol. tercera edición.

10. **Gustavo F Torrealday.** Torrealday Infomática. [En línea] http://www.torrealday.com.ar/articulos/articulo006.htm.

11. **Álvarez, Miguel A.** [En línea] 2004. http://www.desarrolloweb.com/articulos/que-es-html.html.

12. **Monteiro, Juliana Lázaro.** desarrolloweb.com. [En línea] http://www.desarrolloweb.com/articulos/26.php.

13. mozilla <developer center>. [En línea] 2008. https://developer.mozilla.org/index.php?title=Es/Gu%C3%ADa_JavaScript_1.5/Concepto_de_JavaScript.

14. Ruby. A Programmer's Best Friend. [En línea] http://www.ruby-lang.org/es/about/.

15. **Der, Christian Van.** [En línea] 2003. http://www.maestrosdelweb.com/editorial/phpintro/.

16. **Gutiérrez, Javier J.** [En línea] http://www.lsi.us.es/~javierj/investigacion_ficheros/Framework.pdf.

17. CakePHP. [En línea] http://cakephp.org/.

18. symfony.es. [En línea] http://www.symfony.es/que-es-symfony/.

19. **Potencier, Fabien.** *Symfony guía definitiva 1.2.* 2009.

20. [En línea] http://netbeans.org.

21. [En línea] 2009. http://uvfdatabases.wordpress.com/2009/02/07/terminos-de-repaso-introductorios-a-bds/.

22. PostgreSQL. [En línea] http://www.postgresql.org/about/.

23. **Clements, Paul, Kasman, Rick y klein, Mark.** *Evaluating Software Architecture: Methods and Case Studies.* s.l. : Addison-Wesley, 2002.

24. **Erika Camacho, Fabio Cardeso, Gabriel Nuñez.** *Arquitecturas de Software.* 2004.

25. **Bosch, Jan.** *Design and Use of Software Architectures.* s.l. : Addison Wesley, 2000.

26. **Barbacci, Mario, y otros.** *Quality Attributes. Technical Report.* Carnegie Mellon University. Pittsburgh, Pennsylvania 15213 : s.n., 1995.

27. **Puebla, Yoan Arlet Carrascoso, Gómez, Enrique Chaviano y Vega, Anisleydi Céspedes.** Procedimiento para la evaluación de arquitecturas de software basadas en componentes. [En línea] 2009. http://www.gestiopolis.com/administracion-estrategia/procedimiento-para-la-evolucion-de-las-arquitecturas-de-software.htm.

28. **Salvador Gómez, Omar.** *Evaluando Arquitecturas de Software. Parte 1. Panorama General.* s.l. : Brainworx S.A, 2007.

29. *Introducción a Extreme Programming .* **Escribano, Gerardo Fernández.** 2002.

30. **Souders, Steve.** *Even Faster Web Sites.* s.l. : O'REILLY, 2009.

31. —. *High Performnace Web Sites: Essential Knowledge for Frontend Engineers.* s.l. : O'REILLY, 2009.

32. **Stefanov, Stoyan.** *Object-Oriented JavaScript.* 2008.

33. **White, Alexei.** *JavaScript Programmer's Reference.* s.l. : Wiley Publishing, 2009.

34. **BIBEAULT, BEAR y KATZ, YEHUDA.** *jQuery in Action.* s.l. : Manning Publications, 2008.

35. **OLSSON, TOMMY y O'BRIEN, PAUL.** *THE ULTIMATE CSS REFERENCE.* s.l. : SitePoint, 2008.

36. **Holzner, Steve.** *Design Patterns For Dummies.* s.l. : Wiley Publishing, 2006.

37. **Harmes, Ross y Diaz, Dustin.** *Pro JavaScript™ Design Patterns.* 2008.

38. **Fowler, Martin.** *Patterns of Enterprise Application Architecture .* s.l. : Addison Wesley, 2003.

39. **Len Bass, Paul Clements, Rick Kazman.** *Software Architecture in Practice, Second Edition.* s.l. : Addison Wesley, 2003.

40. **Martínez, Amparo Navasa.** *Marco de trabajo para el desarrollo de arquitecturas software orientado a aspectos.* 2008.

41. **James McGovern, Scott W. Ambler, Michael E. Stevens, James Linn, Vikas Sharan, Elias K. Jo.** *Practical Guide to Enterprise Architecture, A.* s.l. : Prentice Hall PTR, 2003.

42. **Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, Judith Stafford.** *Documenting Software Architectures: Views and Beyond.* s.l. : Addison Wesley, 2002.

43. **Frank Buschmann, Regine Meunier,Hans Rohnert,Peter Sornmerlad,Michael Stal.** *PATTERN-ORIENTED SOFTWARE ARCHITECTURE Volumen 1.* s.l. : JOHN WILEY & SONS, 1996.

44. **Echevarría, Javier.** Revista Iberoamericana de Educación, nº 24. [En línea] Septiembre-Diciembre de 2000. http://reddigital.cnice.mec.es/6/Documentos/documento.php?tipo=2&documento=3.

Glosario de Términos

AJAX: Javascript Asíncrono y XML: no es una tecnología por sí misma, es un término que describe un nuevo modo de utilizar conjuntamente varias tecnologías existentes. Esto incluye: HTML o XHTML, CSS, Javascript, DOM, XML, XSLT, y el objeto XMLHttpRequest. Cuando estas tecnologías se combinan en un modelo AJAX, es posible lograr aplicaciones web capaces de actualizarse continuamente sin tener que volver a cargar la página completa. Esto crea aplicaciones más rápidas y con mejor respuesta a las acciones del usuario.

JSON: (*Javascript Object Notation*) el formato de intercambio de datos basado en objetos Javascript es ampliamente utilizado por la facilidad para codificar y decodificar y por su poco tamaño.

WYSIWYG: (Viene del inglés "What you see is what you get"). Característica que tienen algunos editores de texto, o editores en general, que lo que ves en la pantalla es el resultado final que obtendrás en la impresión o en el visionado final. Generalmente usado en editores de texto o HTML.

PDA: Ordenador de pequeño tamaño cuya principal función era en principio la de mantener una agenda electrónica, aunque cada vez más se va confundiendo con los ordenadores de mano y de palma, y se está integrando con otros dispositivos como los teléfonos móviles. Se le denomina también ordenador de bolsillo.

Multiplataforma: Programa o dispositivo que puede utilizarse sin inconvenientes en distintas plataformas de hardware y sistemas operativos. Un programa en lenguaje Java posee esta característica.

API: Conjunto de especificaciones de comunicación entre componentes del software, cuyo propósito consiste en proporcionar un conjunto de funciones de uso general.

XML: Lenguaje desarrollado por el W3 Consortium para permitir la descripción de información contenida en el WWW a través de estándares y formatos comunes, de manera que tanto los usuarios de Internet como programas específicos (agentes) puedan buscar, comparar y compartir información en la red. El formato de XML es muy parecido al del HTML aunque no es una extensión ni un componente de éste.

CRUD: en el área de la informática es el acrónimo de Crear, Obtener, Actualizar y Borrar los datos de una base de datos.

Cookies: Archivo de texto que se graba en el ordenador del visitante del cual se sirven los servidores web para guardar información acerca del cliente de un sitio. Sirve para identificar a visitantes recurrentes.

SVN: es un sistema de control de versiones usado para que varios desarrolladores puedan trabajar en un mismo proyecto en forma más o menos ordenada. Tiene una arquitectura cliente servidor con controles de concurrencia para cuando varios desarrolladores están trabajando en el mismo.

ACID (Atomicity, Consistency, Isolation and Durability): es un término usado en bases de datos para definir un conjunto de características para garantizar la consistencia de la información. Estas características son Atomicity, Consistency, Isolation and Durability: Atomicidad, Consistencia, Aislamiento y Durabilidad en español.

ORM (Mapeo objeto-relacional): es una técnica para convertir datos del lenguaje de programación orientado a objetos al utilizado en una base de datos relacional, esto facilita el uso de las características propias de la orientación a objetos.

Clases: Lección que imparte el profesor a los estudiantes.

Software Educativo: Se refiere a los programas educativos o didácticos, conocidos también como programas por ordenador, creados con la finalidad específica de ser utilizados para facilitar los procesos de enseñanza y aprendizaje.

Anexos

Anexo 1
Módulo Ejercicios

Caso de Uso:	Realizar ejercicio
Resumen:	El caso de uso inicia cuando el usuario selecciona la opción que le permite realizar los ejercicios, el sistema muestra los contenidos y permite seleccionar los que el usuario desee, además seleccionar la cantidad de ejercicios que desea realizar mediante un tipo de selección (secuencial, aleatoria o asignados) y realizar los ejercicios.

Caso de Uso:	Revisar Ejercicio
Resumen:	El caso de uso inicia cuando el usuario selecciona la opción que le permite revisar la respuesta de un ejercicio, el sistema evalúa la respuesta e informa la evaluación al usuario de forma afectiva, si la respuesta no es correcta muestra un nivel de ayuda y permite analizar la respuesta incorrecta, cuando se agotan los intentos el sistema permite analizar la respuesta correcta, si la respuesta es correcta el sistema brinda la opción de saber más, luego de revisado el ejercicio el usuario puede ver la respuesta dada por él y la respuesta correcta cuando lo desee.

Caso de Uso:	Consultar Resultados
Resumen:	El caso de uso inicia cuando el usuario selecciona la opción terminar la realización de ejercicios, el sistema carga la pantalla que permite consultar los resultados de los ejercicios realizados, el sistema permite elegir el usuario a analizar los resultados, mostrar la calificación general obtenida, el tiempo total de trabajo y muestra tres áreas que puntualizan los resultados.

Caso de Uso:	Mostrar estadísticas
Resumen:	El caso de uso inicia cuando el usuario selecciona la opción que le permite mostrar las estadísticas, el sistema muestra la efectividad de las respuestas realizadas, la cantidad de cuestionarios interactivos que faltan por responder y una gráfica de barra donde se representa la cantidad de ejercicios por categoría de calificación a partir de los

	ejercicios realizados.

Caso de Uso:	Gestionar ejercicio
Resumen:	El caso de uso se inicia cuando el actor selecciona la opción que le permite realizar una acción sobre los ejercicios. El actor puede adicionar, ver, modificar y eliminar un ejercicio. En caso de que seleccione la opción de adicionar ejercicio, el sistema dará la posibilidad de insertar los datos que se necesitan para crear el ejercicio. Si el actor elige la opción de modificar el ejercicio, el sistema mostrará los datos que pueden ser editables dentro del mismo, y una vez realizados los cambios, guardará las modificaciones. Si el actor elige la opción de eliminar ejercicio, el sistema mostrará los datos a eliminar. El sistema permite ver una vista previa del ejercicio, terminando así el caso de uso.

Caso de Uso:	Gestionar Saber más.
Resumen:	El caso de uso se inicia cuando el actor selecciona la opción que le permite realizar una acción sobre un Saber Más. El actor puede adicionar y eliminar Saber Más. En caso de que seleccione la opción de adicionar Saber Más el sistema dará la posibilidad de insertar los datos que se necesitan para crear el Saber Más. Si el actor elige la opción de eliminar Saber Más, el sistema eliminará los datos seleccionados por el actor. El sistema permite ver una vista previa de los Saber Más luego de cada acción, terminando así el caso de uso.

Caso de Uso:	Consultar Saber Más
Resumen:	El caso de uso se inicia cuando el actor accede a la opción de consultar Saber Más. El sistema brinda la posibilidad de seleccionar el tema y los Saber Más a mostrar, listar todos los elementos y cancelar la operación en cualquier momento. El sistema brinda un resumen de los datos de Saber Más, los mismos pueden estar ordenados de forma ascendente o descendente. El sistema permite ver un elemento de la lista mostrada, modificar los datos de un elemento de la lista, eliminar un elemento y realizar una nueva búsqueda. El sistema permite además consultar el resumen de los datos de los elementos de la lista de coincidencias y seguidamente el caso de uso termina.

Módulo Temas

Caso de Uso:	Mostrar palabra caliente
Resumen:	El caso de uso se inicia cuando el actor selecciona una palabra caliente. El sistema muestra la información asociada a la palabra caliente seleccionada y el caso de uso termina.

Caso de Uso:	Mostrar artículo
Resumen:	El caso de uso se inicia cuando el actor selecciona un artículo. El sistema muestra la información asociada al artículo seleccionado y el caso de uso termina.

Caso de Uso:	Mostrar medias asociadas
Resumen:	El sistema muestra las medias asociadas a un elemento del contenido. El actor consulta las medias y el caso de uso termina.

Caso de Uso:	Gestionar palabra caliente
Resumen:	El caso de uso se inicia cuando el gestor de contenido selecciona la opción que le permite realizar una acción sobre una palabra o frase caliente. El gestor de contenido puede incluir o eliminar una palabra caliente. En caso de que seleccione la opción de incluir una nueva palabra caliente, el sistema establecerá una relación entre dicha palabra y el elemento del glosario de términos que le corresponda; convirtiéndola así en palabra caliente. Si el actor elige la opción de eliminar una palabra caliente, el sistema descartará la relación existente entre la palabra caliente y el elemento del glosario de términos correspondiente, terminando así el caso de uso.

Caso de Uso:	Gestionar producto
Resumen:	El caso de uso se inicia cuando el gestor de contenido selecciona la opción que le permite realizar una acción sobre un producto. El gestor de contenido puede incluir, modificar o eliminar un producto. En caso de que seleccione la opción de incluir un producto, el sistema dará la posibilidad de insertar los datos necesarios. Si el gestor de contenido elige la opción de modificar un producto, el sistema

	mostrará los datos que pueden ser editables dentro del producto, y una vez realizados los cambios, guardará las modificaciones. En caso de seleccionar la opción eliminar, el sistema mostrará un mensaje de confirmación una vez aceptado el mensaje se eliminará el producto; terminado así el caso de uso.

Caso de Uso:	Gestionar contenedor
Resumen:	El caso de uso se inicia cuando el gestor de contenido selecciona la opción que le permite realizar una acción sobre un contenedor. El gestor de contenido puede incluir, modificar o eliminar un contenedor. En caso de que seleccione la opción de incluir un contenedor, el sistema dará la posibilidad de insertar los datos necesarios. Si el gestor de contenido elige la opción de modificar un contenedor, el sistema mostrará los datos que pueden ser editables dentro del contenedor, y una vez realizados los cambios, guardará las modificaciones. En caso de seleccionar la opción eliminar, el sistema mostrará un mensaje de confirmación, una vez aceptado el mensaje se eliminará el contenedor; terminado así el caso de uso.

Caso de Uso:	Gestionar artículo
Resumen:	El caso de uso se inicia cuando el gestor de contenido selecciona la opción que le permite realizar una acción sobre un artículo. El gestor de contenido puede incluir, modificar o eliminar un artículo. En caso de que seleccione la opción de incluir un artículo, el sistema dará la posibilidad de insertar los datos necesarios, y opcionalmente, de realizar una asociación entre una media y dicho artículo. Si el gestor de contenido elige la opción de modificar un artículo, el sistema mostrará los datos que pueden ser editables dentro del artículo, así como asociar y disociar medias a dicho artículo de ser necesario; una vez realizados los cambios, guardará las modificaciones. En caso de seleccionar la opción eliminar el sistema mostrará un mensaje de confirmación una vez aceptado el mensaje se eliminará el artículo. Terminado así el caso de uso.

Módulo Resultados

Caso de Uso:	Iniciar Resultados

Resumen:	El caso de uso se inicia cuando el usuario selecciona la acción iniciar resultados, el sistema muestra una interfaz permitiéndole al usuario el tipo de análisis a realizar.

Caso de Uso:	Realizar tipo de análisis
Resumen:	El caso de uso se inicia cuando el usuario selecciona la acción realizar tipo de análisis, el sistema muestra una interfaz permitiéndole al usuario la selección del mismo, realiza un filtrado de la escuela, grado y grupo y un listado con los estudiantes que han realizado ejercicios, y un tamaño de muestra para la selección aleatoria.

Caso de Uso:	Realizar análisis de contenidos
Resumen:	El caso de uso se inicia cuando el usuario selecciona la acción realizar análisis de los contenidos específicos. El sistema muestra una interfaz para la selección del tipo de análisis a realizar, así como la selección de los contenidos a analizar, muestra una tabla con los resultados de los mismos y una gráfica con la comparación de la efectividad de los contenidos.

Caso de Uso:	Realizar análisis del historial
Resumen:	El caso de uso se inicia cuando el usuario selecciona la acción realizar análisis del historial. El sistema muestra una interfaz para la selección del tipo de análisis a realizar, también permite la selección de las fechas en que desea analizar los contenidos, muestra además una tabla con los resultados del análisis y una gráfica con el análisis de la efectividad.

Caso de Uso:	Realizar análisis integral
Resumen:	El caso de uso se inicia cuando el usuario selecciona la acción realizar análisis integral. El sistema muestra una interfaz para la selección del tipo de análisis a realizar, también permite la selección de las asignaturas que desea analizar, muestra además una tabla con los resultados de este análisis y una gráfica de comparación de efectividad de las asignaturas.

Caso de Uso:	Controlar navegación
Resumen:	El caso de uso se inicia cuando el usuario decide cambiar la interfaz. El sistema muestra en la interfaz tres botones permitiéndole al usuario realizar la navegación deseada (siguiente, anterior, terminar).

Caso de Uso:	Realizar análisis de trazas
Resumen:	El caso de uso se inicia cuando el usuario selecciona el módulo “Resultados”. El sistema muestra el listado de todos los estudiantes que interactuaron con el sistema y sus trazas por el hiperentorno.

Caso de Uso:	Registrar traza del módulo Temas
Resumen:	El caso de uso se inicia cuando un estudiante selecciona el módulo “Temas”. El sistema registra los datos de la interacción del mismo con el módulo y termina el caso de uso.

Caso de Uso:	Registrar traza del módulo Ejercicios
Resumen:	El caso de uso se inicia cuando un estudiante selecciona el módulo “Ejercicios”. El sistema registra los datos de la interacción del mismo con el módulo y termina el caso de uso.

Caso de Uso:	Registrar traza del módulo Juegos
Resumen:	El caso de uso se inicia cuando un estudiante selecciona el módulo “Juegos”. El sistema registra los datos de la interacción del mismo con el módulo y termina el caso de uso.

Caso de Uso:	Registrar traza del módulo “Juegos” con ejercicios
Resumen:	El caso de uso se inicia cuando un estudiante selecciona el juego que desea realizar. El sistema registra los datos de la interacción del mismo con cada uno y termina el caso de uso.

Caso de Uso:	Registrar traza del módulo “Juegos” sin ejercicios
Resumen:	El caso de uso se inicia cuando un estudiante selecciona el juego que desea realizar. El sistema registra los datos de la interacción del mismo con cada uno y termina el caso de uso.

Caso de Uso:	Registrar traza del módulo Mediateca
Resumen:	El caso de uso se inicia cuando un estudiante selecciona el módulo “Mediateca”. El sistema registra los datos de la interacción del mismo con el módulo y termina el caso de uso.

Caso de Uso:	Listar estudiantes
Resumen:	El caso de uso se inicia cuando el usuario selecciona el módulo “Resultados”. El sistema muestra el listado de todos los estudiantes que interactuaron con el sistema y termina el caso de uso.

Módulo General

Caso de Uso:	Autenticarse
Resumen:	El caso de uso se inicia cuando el usuario ejecuta la aplicación, la misma muestra la interfaz de autenticación donde el usuario especifica cómo es que se va a autenticar (invitado, profesor o estudiante), una vez realizada la acción el usuario puede navegar por la aplicación según sus permisos.

Caso de Uso:	Gestionar información del estudiante
Resumen:	El caso de uso se inicia cuando el profesor selecciona la opción que le permite realizar una acción sobre el estudiante, la cual puede ser adicionar, modificar o eliminar un estudiante. Si el profesor decide adicionar un estudiante el sistema será capaz de mostrar una interfaz que permite introducir los datos del estudiante, si decide modificar, mostrará una interfaz con los campos que son editables, y si decide eliminar un estudiante permitirá seleccionar cuál es el estudiante deseado, el sistema notificará cuando las acciones tengan o no éxito.

Caso de Uso:	Buscar y mostrar contenido en el producto
Resumen:	El caso de uso se inicia cuando el usuario ejecuta la acción de búsqueda. El sistema muestra una interfaz que da la posibilidad al usuario de introducir el texto que quiere buscar, así como seleccionar el criterio de búsqueda (cualquier palabra, todas las palabras o la frase exacta) y donde quiere realizar específicamente la misma. Una vez realizada la búsqueda el sistema muestra un interfaz donde enumera los resultados hallados, especifica el módulo donde fue encontrado el texto con un pequeño resumen, y permite seleccionar y acceder al lugar específico donde se encontró la palabra o frase buscada, permite navegar por los resultados y una vez lo desee el usuario terminar la búsqueda.

Caso de Uso:	Gestionar gestor de contenido
Resumen:	El caso de uso se inicia cuando el administrador del sistema selecciona la opción que le permite realizar una acción sobre los gestores de contenidos, la cual puede ser adicionar, modificar o eliminar uno de éstos. Si el administrador decide adicionar un gestor de contenido el sistema será capaz de mostrar una interfaz que permite introducir los datos del mismo, si decide modificar mostrará una interfaz con los campos que son editables, y si decide eliminar uno de estos gestores, permitirá seleccionar el deseado y lo eliminará, el sistema notificará cuando las acciones tengan o no éxito.

Módulo mediateca

Caso de Uso:	Consultar Mediateca
Resumen:	El caso de uso se inicia cuando el usuario selecciona la imagen que representa el módulo Mediateca, el sistema muestra las imágenes que metafóricamente representan las diferentes secciones y permite su selección. El usuario selecciona una sección, el sistema muestra los elementos que la integran y permite visualizar o reproducir uno de ellos, terminando así el caso de uso.

Caso de Uso:	Controlar navegación

Resumen:	Permitir al usuario navegar entre las distintas páginas de una sección, de un tema y de sus elementos y seleccionar las opciones siguiente, anterior, primera y última. El usuario selecciona una de estas opciones y culmina el caso de uso.

Caso de Uso:	Gestionar videos
Resumen:	El caso de uso se inicia cuando el gestor de contenido selecciona Videos. El sistema muestra el listado de videos existentes y permite adicionar un nuevo video, modificar los datos del video y eliminarlo. El gestor de contenido selecciona una de estas opciones y finaliza el caso de uso.

Caso de Uso:	Gestionar imágenes
Resumen:	El caso de uso se inicia cuando el gestor de contenido selecciona Imágenes. El sistema muestra el listado de las imágenes existentes y permite adicionar una nueva imagen, modificar los datos de la imagen y eliminarla. El gestor de contenido selecciona una de estas opciones y finaliza el caso de uso.

Caso de Uso:	Gestionar sonidos
Resumen:	El caso de uso se inicia cuando el gestor de contenido selecciona Sonidos. El sistema muestra el listado de sonidos existentes y permite adicionar un nuevo sonido, modificar los datos del sonido y eliminarlo. El gestor de contenido selecciona una de estas opciones y finaliza el caso de uso.

Caso de Uso:	Gestionar personalidades
Resumen:	El caso de uso se inicia cuando el gestor de contenido selecciona

	Imágenes. El sistema muestra el listado de las imágenes existentes y permite adicionar una nueva personalidad, modificar los datos de la personalidad y eliminarla. El gestor de contenido selecciona una de estas opciones y finaliza el caso de uso.

Caso de Uso:	Gestionar diaporamas
Resumen:	El caso de uso se inicia cuando el gestor de contenido selecciona Diaporamas. El sistema muestra el listado de diaporamas existentes y permite adicionar un nuevo diaporama, modificar los datos del diaporama y eliminarlo. El gestor de contenido selecciona una de estas opciones y finaliza el caso de uso.

Caso de Uso:	Gestionar animaciones
Resumen:	El caso de uso se inicia cuando el gestor de contenido selecciona Imágenes. El sistema muestra el listado de las imágenes existentes y permite adicionar una nueva animación, modificar los datos de la animación y eliminarla. El gestor de contenido selecciona una de estas opciones y finaliza el caso de uso.

Caso de Uso:	Gestionar glosario
Resumen:	El caso de uso se inicia cuando el gestor de contenido selecciona Glosario. El sistema muestra el listado de palabras del glosario existentes y permite adicionar una nueva palabra al glosario, modificar los datos de una palabra del glosario o eliminarla. El gestor de contenido selecciona una de estas opciones y finaliza el caso de uso.

Módulo Juegos

Caso de Uso:	Comenzar juego
Resumen:	El caso de uso es inicializado por el usuario, cuando éste selecciona en la interfaz que muestra el sistema la imagen que identifica a un

	juego determinado, luego el sistema carga el juego, los datos del jugador, muestra las áreas de interés para el juego y permite jugar el mismo.

Caso de Uso:	Controlar tiempo
Resumen:	El caso de uso es inicializado cuando el jugador comienza a interactuar con las actividades del juego, el sistema comienza a llevar el conteo del tiempo en segundos desde que el usuario comienza las actividades del juego y hasta que éste culmina, dando como resultado el cálculo del tiempo total que demoró en segundos.

Caso de Uso:	Evaluar jugador
Resumen:	El caso de uso es inicializado cuando el usuario una vez terminada las actividades del juego selecciona la opción “Revisar”, el sistema muestra un mensaje de notificación informando si la respuesta es correcta o no y otorga la clasificación en dependencia de la respuesta y guarda el tiempo transcurrido en realizar la actividad.

Caso de Uso:	Realizar actividad
Resumen:	El caso de uso es inicializado cuando el usuario selecciona realizar alguna de las actividades que permite avanzar en el juego, el sistema muestra una ventana con la actividad y da la posibilidad de revisar la misma, notificando con un mensaje si la respuesta es correcta o incorrecta.

Caso de Uso:	Gestionar contenido de los juegos
Resumen:	El caso de uso es inicializado cuando el profesor selecciona el juego al que le desea gestionar el contenido, el sistema muestra el contenido actual junto a las opciones de eliminar, insertar o modificar el contenido y actúa en dependencia de la opción seleccionada.

Caso de Uso:	Mostrar resultados del juego

Resumen:	El caso de uso comienza una vez el usuario culmine el juego, el sistema muestra un mensaje de notificación al usuario comunicándole que ha terminado, que observe los resultados y muestra una interfaz donde se puede observar un podio con los lugares y la puntuación que se recibe.

Módulo Profesor

Caso de Uso:	Gestionar escuela
Resumen:	El caso de uso se inicia cuando el profesor selecciona la opción Configuraciones, el sistema va a mostrar una interfaz que le permite realizar acciones sobre la escuela, las cuales pueden ser adicionar, modificar o eliminar. Si el profesor decide adicionar una escuela el sistema será capaz de mostrar una interfaz que permite seleccionar e introducir los datos de la escuela, si decide modificar mostrará los campos que son editables, y si decide eliminar una escuela permitirá seleccionar la misma y la eliminará, el sistema notificará cuando las acciones tengan o no éxito.

Caso de Uso:	Cambiar contraseña
Resumen:	El caso de uso se inicia cuando el profesor selecciona la opción Configuraciones, el sistema va a mostrar una interfaz que le permite cambiar la contraseña, el sistema da la posibilidad de realizar el cambio, especificando la vieja y solicitando la nueva contraseña, la cual pide que confirme para poder cerciorarse de que es correcta y acceder al cambio.

Caso de Uso:	Configurar revisión e historial
Resumen:	El caso de uso se inicia cuando el profesor selecciona la opción Configuraciones, el sistema va a mostrar una interfaz que permite determinar los niveles de resultados de los ejercicios (bien, mal o regular) así como determinar el tiempo que se van a almacenar estos resultados en la aplicación.

Caso de Uso:	Consultar contenido

Resumen:	En este caso de uso el sistema muestra una interfaz al entrar al módulo donde el profesor puede seleccionar el tema específico que desea consultar, y el sistema muestra el contenido seleccionado en otra interfaz.

Caso de Uso:	Consultar ejercicios resueltos
Resumen:	El caso de uso se inicia cuando el profesor selecciona la opción Visor de ejercicios, el sistema va a mostrar una interfaz que permite al profesor seleccionar ejercicios acerca del tema que desee y cuántos de éstos, de los cuales el sistema mostrará la respuesta.

Anexo 2

Para declarar una clase, se le pasa un solo parámetro a la función que será un objeto, donde la propiedad constructor tiene que ser una función, la cual va a ser el constructor de la clase, las propiedades que sean funciones serán métodos de la clase y las que no serán atributos compartidos para todas las instancias de la clase (si se modifica en una instancia de la clase se modifica en todas).

```
var Persona = $.clase({
      //constructor de la clase
constructor : function(pnombre)
{
   //nombre es un atributo de la clase Animal
  this.nombre=pnombre;
  this.apellido = “Marrero”;
},
//nombre y toString son métodos de la clase Animal
nombre : function()
{
  return this.nombre + this.apellido;
},
toString : function()
{
  return"Persona";
}
})
```

Cuando se va a declarar una clase que herede de otra el primer parámetro es la clase de la que se heredará y el segundo será un objeto con las mismas características que el anterior.

```
var Estudiante=$.clase(Persona,{
constructor : function(pnombre,pgrupo)
{
   //De esta forma, se llama al constructor de la clase padre. Esto se
   // tiene que hacer siempre o no se heredarán los atributos del padre;
  Ave.baseConstructor.apply(this,[pnombre]);
  this.grupo = pgrupo;
},
codigoEstudiante : function(){
return this.nombre() + this.grupo;
},
//aquí un ejemplo de cómo sobrescribir un método
toString : function(){
//Para referenciar a un método de la clase padre que se ha sobrescrito
  return Estudiante.padre.toString.apply(this) + "--Estudiante";
}
})
```