UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 5

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas.

COMPONENTE FEEDBACK PARA EL SISTEMA INTEGRAL DE PERFORACIÓN DE POZOS.

Autor: Yoan Manuel Pérez Lohuis.

Tutores: Ing. David Tavares Cuevas.

Ing. Omar Martínez Díaz.

Ing. Yordan Gallardo Avilés.

La Habana julio, 2012.

DECLARACIÓN DE AUTORÍA

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

_____

Yoan Manuel Pérez Lohuis

Autor

_____

Ing. David Tavares Cuevas

Tutor

_____

Ing. Omar Martínez Díaz

Tutor

_____

Ing. Yordan Gallardo Áviles

Tutor

DATOS DE CONTACTO

TUTORES

Ing. David Tavares Cuevas.

Ciudadanía: Cubano.

Institución: Universidad de las Ciencias Informáticas (UCI).

Título: Ing. en Ciencias Informáticas.

Categoría Docente: Instructor.

E-mail: dtavares@uci.cu.

Graduado de la UCI, con 5 años de experiencia en el desarrollo de sistemas industriales relacionados con la perforación de pozos.

Ing. Omar Martínez Díaz.

Ciudadanía: Cubano.

Institución: Universidad de las Ciencias Informáticas (UCI).

Título: Ing. en Ciencias Informáticas.

Categoría Docente: Recién graduado en adiestramiento(RGA).

E-mail: omarmd@uci.cu.

Graduado de la UCI, con dos años de experiencia.

Ing. Yordan Gallardo Áviles.

Ciudadanía: Cubano.

Institución: Universidad de las Ciencias Informáticas (UCI).

Título: Ing. en Ciencias Informáticas.

Categoría Docente: Recién graduado en adiestramiento(RGA)

E-mail: ygaviles@uci.cu.

Graduado de la UCI, con 4 años de experiencia en proyectos de perforación de pozos de petróleo.

AGRADECIMIENTOS

A Dios por acompañarme todos los días.

A mi madre por ser mi amiga, aliada y mejor ejemplo, gracias por todo el apoyo en esta tesis y en la vida. Sin ella no hubiera sido posible.

A José Alejandro, porque he aprendido y heredado su fuerza, organización y entrega, por todo el cariño que me ha dado. Siempre ha sido un padre para mí.

A mis abuelitas Delia y Marta por su constante preocupación y aliento.

A Rosy, mi futura esposa, por su infinita paciencia, tierna compañía y su inagotable apoyo. Gracias por compartir mi vida y mis logros, esta tesis también es suya.

A Irene de la Caridad Niubo Jorge por su incondicional y oportuna colaboración.

A mis tutores Omar Martínez Díaz y Yordan Gallardo Áviles, por su enseñanza y dedicación en la elaboración de este trabajo.

A Yoan Hernández, Programador de la Casa de las Américas, por brindar sus conocimientos, que se convirtieron en una parte importante de este trabajo.

A la vida, por rodearme de personas maravillosas que hacen posible todos mis sueños.

A todas las personas que han estado a mi lado, ayudándome y brindándome amor, apoyo y amistad.

RESUMEN

La comunicación entre los usuarios que explotan una aplicación, y el equipo de desarrolladores de la misma, es necesaria para lograr un producto de óptima calidad. El equipo de desarrollo se retroalimenta de la opinión de los clientes que explotan su herramienta, con el fin de lograr desarrollar una aplicación lo más cercana al gusto de quienes la explotan. En el presente trabajo de diploma se tiene como objetivo desarrollar una herramienta web, en forma de plugins, que permita la comunicación entre el equipo de desarrollo, su aplicación, y los clientes que la explotan.

Se emplea como metodología de desarrollo RUP. Los artefactos se generaron usando como lenguaje de modelado UML y auxiliados por el Visual Paradigm como herramienta CASE. Como Entorno Integrado de Desarrollo se utilizó el NetBeans en su versión 7.0.1. Como servidor Web y de Base de Datos, se utilizó Apache y PostgreSQL.

Esta herramienta cuenta con tres módulos para los desarrolladores conocer el comportamiento que tienen las aplicaciones que utilizan en esta herramienta, como son Symfony, Apache y Postgres, siendo un módulo por aplicación. Y un cuarto módulo para los clientes, siendo este el más importante, debido a que es el encargado de gestionar todas las sugerencias emitidas por lo clientes, permitiéndoles comunicarse con el equipo de desarrolladores.

Palabras Clave:

Feedback (retroalimentación), gestión de errores, seguimiento de errores, mejora.

TABLA DE CONTENIDO

INTRODUCCIÓN

Las tecnologías de la comunicación se encargan del estudio, desarrollo, implementación, almacenamiento y distribución de la información mediante la utilización de hardware y software como medio de un sistema informático. Las tecnologías de la información y las comunicaciones son una parte de las tecnologías emergentes que habitualmente suelen identificarse con las siglas TICs (1). Las TICs, impulsadas por un vertiginoso avance científico y sustentado por el uso generalizado de las potentes y versátiles tecnologías, conllevan a cambios que alcanzan todos los ámbitos de la actividad humana.

Las TICs abarcan diferentes ramas, entre ellas se encuentra la rama informática en la que intervienen los dispositivos donde el hardware y el software están interconectados el uno con el otro, y esta a su vez, se divide en otras ramas entre las que se encuentran algoritmos y estructuras de datos, inteligencia artificial, ingeniería de software, base de datos, comunicación y seguridad, entre otras. De ellas, una de las más importantes es la comunicación y seguridad debido a que en esta rama se desarrolla completamente todo lo relacionado con la seguridad informática de los ordenadores. Dentro de toda la seguridad informática que se le puede brindar a los ordenadores se encuentra el sistema de seguimiento y control de errores, que es un sistema que permite mantener un control total de todos los errores que puede arrojar una aplicación. A nivel internacional existen varias aplicaciones que responden a estas características, y empresas que las utilizan como una medida de eficiencia y seguridad.

Entre los centros de desarrollo de mayor importancia en Cuba se encuentra la Universidad de las Ciencias Informáticas (UCI), la cual ha alcanzado un alto prestigio a nivel nacional e internacional. La UCI, se ha convertido en uno de los cimientos fundamentales en la industria del software cubano y a su vez en un eslabón imprescindible para la economía del país.

En la Facultad 5 se encuentra el Centro de Desarrollo de Informática Industrial (CEDIN), en el cual se ha desarrollado el Sistema de Manejo Integral de Perforación de Pozos (SIPP), el cual

responde a las necesidades del Centro de Investigaciones de Petróleo (CEINPET) y la Dirección de Intervención y Perforación de Pozos (DIPP), entidad responsable de todos los pozos en perforación en tierra del territorio cubano. Está destinado a los pozos de petróleo en perforación, con el objetivo de gestionar todo el flujo de la información que es generado en estos.

Actualmente, el SIPP, mediante la aplicación que lo representa **Maximus Drill Pro** se encuentra desplegado en varios pozos. Como resultado de su explotación, esta aplicación arroja errores, los cuales no se registran ni están estandarizados, lo que imposibilita el entendimiento entre el cliente y el equipo de desarrolladores para futuras mejoras de la aplicación. Las fallas detectadas no son gestionadas, así como tampoco las sugerencias que los clientes proponen debido a la nula comunicación entre la aplicación, los clientes que la explotan y el equipo de desarrolladores.

Teniendo en cuenta la situación problémica anterior, se plantea el siguiente **problema de investigación**:

¿Cómo facilitar y/o contribuir a mejorar la comunicación entre la aplicación **Maximus Drill Pro**, el cliente y el equipo de desarrollo? A partir del problema planteado, se toma como **objeto de investigación**: Técnicas y procesos para la comunicación entre los sistemas industriales y el equipo de desarrollo. Y el **campo de acción**: Comunicación entre la aplicación **Maximus Drill Pro** y el equipo de desarrollo.

El **Objetivo General** que se persigue es diseñar e implementar un componente que permita la comunicación entre el cliente, la aplicación **Maximus Drill Pro** y el equipo de desarrolladores.

De esta manera, y para dar cumplimiento al objetivo trazado, se plantea la siguiente **idea a defender:** con el desarrollo del componente se logrará la comunicación entre el cliente, la aplicación **Maximus Drill Pro** y el equipo de desarrolladores.

Para dar cumplimiento al objetivo planteado, se proponen las siguientes **tareas de investigación** a realizar:

- ✓ Elaboración del marco teórico de la investigación a partir del estado del arte actual sobre el tema.
- ✓ Determinar la metodología, tecnologías y herramientas a utilizar para el desarrollo de la solución.
- ✓ Confección del modelo de dominio.
- ✓ Identificación de los requerimientos.
- ✓ Diseño e implementación del componente.
- ✓ Realización de pruebas al componente para las validaciones.

Métodos científicos de la investigación: Los métodos científicos utilizados en esta investigación son los métodos empíricos y los métodos teóricos. Dichos métodos ayudan a mantener un trabajo más organizado, lo que posibilita un mayor conocimiento y entendimiento de todo lo relacionado con el sistema, la evolución que ha tenido tanto en Cuba como en el mundo, así como el conocimiento de bibliografías fundamentales para que el desarrollo de esta investigación sea el más propicio.

Métodos Empíricos: Crean las condiciones para ir más allá de las características fenomenológicas y superficiales de la realidad. Posibilitan el conocimiento del estado del arte del fenómeno, su evolución en una etapa determinada, su relación con otros fenómenos (2).

- ✓ **Observación:** Este método se ha utilizado para comunicarse con la aplicación con el fin de recoger de forma directa algunas características específicas de esta.
- ✓ **Entrevista y Encuesta:** Estos métodos han sido utilizados para obtener una información adecuada y acertada a través de preguntas a los líderes de proyectos y a los profesionales que explotan la aplicación.
- ✓ **Observación bibliográfica:** Este método se ha utilizado en la mayor parte de la investigación pues permite el análisis de la información mediante otros tutores y trabajos existentes.

Métodos Teóricos: Permiten estudiar las características del objeto de investigación que no son observables directamente. Facilitan la construcción de modelos e hipótesis de investigación (2).

- ✓ **Análisis histórico-lógico:** Este método se utilizó para investigar la información que se tiene hasta ahora sobre los diferentes sistemas de seguimientos y gestión de errores y considerar los aspectos fundamentales para la realización de la investigación en curso.
- ✓ **Analítico-sintético:** Este método permitió procesar toda la información y dividir el problema en porciones más pequeñas con el fin de hacerlo más fácil de entender, y darle solución a cada porción del mismo por separado.
- ✓ **Modelación:** Este método se ha utilizado para la representación de los modelos de base de datos, tanto el físico, como el lógico así como el diseño de los casos de prueba.
- ✓ **Consulta bibliográfica**: Este método permitió la consulta de toda la bibliografía existente.

Estructura del contenido:

A continuación se muestra la estructura del presente trabajo, incluyendo una síntesis de los capítulos:

Capítulo1: “Fundamentación Teórica”. En este capítulo se darán a conocer los conceptos fundamentales para un mejor entendimiento del entorno en el que se desarrolla la aplicación **Maximus Drill Pro**. Se dejarán expuestas las herramientas que se utilizarán para la confección de este trabajo, así como comparaciones con otras herramientas, haciendo énfasis en aquellas que se utilizarán en la solución. Además se plantean las características que permiten a estas herramientas ser las más propicias para la implementación del componente.

Capítulo 2: “Características del Sistema”. En este capítulo se realizará una descripción completa del componente a implementar con el fin de lograr un mejor entendimiento por parte del equipo de desarrolladores. Para esto, se especificarán los requisitos funcionales y no funcionales que presenta el componente. También se declararán los actores y sus vínculos con los casos de uso, por medio de un diagrama de casos de uso del sistema. Se brindará una detallada descripción de

cada uno de ellos, y los patrones de casos de uso a utilizar. Además se mostrará el diagrama de modelo de dominio y se realizará una breve descripción del mismo.

Capítulo 3: “Análisis y diseño del Sistema”. En este capítulo se describe el sistema desde la perspectiva de Ingeniería de Software, utilizando diferentes diagramas como el diagrama de despliegue, el diagrama de clases persistentes y el modelo de datos de la base de datos. También se mostrará el diagrama de clases del diseño, con una descripción de cada paquete que lo conforma, explicando los patrones de diseño y los estilos utilizados, además de explicar cómo se adaptan dichos estilos y patrones al diseño del sistema.

Capítulo 4: “Implementación y Validación de los Resultados”. En este capítulo se describirá la implementación del componente a partir del diagrama de componentes. También se presentarán varias pantallas de la aplicación donde se muestra el sistema dando cumplimiento a cada uno de los requisitos funcionales. Además se listarán las pruebas que se le realizaron al sistema para verificar su integridad y ajuste a los requerimientos planteados por el cliente.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA.

1.1. Introducción.

En el presente capítulo se darán a conocer los conceptos fundamentales para un mejor entendimiento del entorno en el que se desarrolla la aplicación **Maximus Drill Pro**. Se dejarán expuestas las herramientas que se utilizarán para la confección de este trabajo, así como comparaciones con otras herramientas, haciendo énfasis en aquellas que se utilizarán en la solución. Además se plantean las características que permiten a estas herramientas ser las más propicias para la implementación del componente

1.2. Sistema de Manejo Integral de Perforación de Pozos (SIPP).

El Sistema de Manejo Integral de Perforación de Pozos (SIPP) es un proyecto que tiene como objetivo desarrollar soluciones informáticas orientadas al área de negocio de la perforación dentro de la Industria del Petróleo, integrándose de manera tal que brinde la cobertura a todo el proceso. Esto se logra dividiendo el negocio en subprocesos y enfocando estos desarrollos hacia ellos.

El proyecto SIPP ha desarrollado la aplicación Maximus Drill Pro, que es la herramienta encargada de gestionar la información consolidada por los expertos en los pozos de perforación.

1.2.1. Conceptos asociados.

A continuación se darán algunos conceptos relacionados con el componente a desarrollar para una mejor comprensión:

Mejora: Acción o efecto de mejorar, que le supera en calidad, hacer avanzar algo hacia el bien, superioridad, ventaja (3).

Componente de software: Un componente de software en tiempo de ejecución es un paquete dinámicamente vinculado con uno o varios programas manejados como una unidad y que son

accedidos mediante interfaces bien documentadas que pueden ser descubiertos en tiempo de ejecución. (4)

Gestión de errores: El vocablo “gestión” es la acción o efecto de administrar o gestionar grandes volúmenes de datos, manipulación automática de los mismos para obtener un aspecto esencial de ellos y presentación de los resultados, a fin de tomar una decisión; el vocablo error significa equivocación o falsedad, acto desacertado o sin tino (3). En su conjunto, “gestión de errores” es la acción o efecto de administrar las fallas que puedan ser arrojadas por una aplicación.

Seguimiento de errores: El vocablo “seguimiento” significa persecución, acción de seguir, vigilancia, observación detallada (3); significando “seguimiento de errores” la acción o efecto de observar detalladamente, seguir y vigilar las fallas que puedan ser arrojadas por una aplicación.

1.3. Sistema de seguimiento de errores.

En el ámbito informático un sistema de seguimiento de errores es una aplicación informática diseñada con el objetivo de ayudar a asegurar la calidad de software y apoyar a los programadores y otras personas involucradas en el desarrollo y uso de sistemas informáticos en el seguimiento de los defectos del software.

Estos tipos de sistemas están constituidos por diferentes componentes, de los cuales uno de los más importantes es la base de datos, donde se almacena todo un historial de las características de los errores de la aplicación. Estas características pueden ser muy variadas. Algunas de ellas pueden ser: descripción detallada del error, sugerencias por parte del cliente que explota dicho software, fecha en que fueron capturados los errores y las sugerencias. Otras características pueden estar relacionadas con el proceso de administración de la corrección del fallo como pueden ser la fecha probable de corrección del error y una posible ponderación del mismo. Esto último con el objetivo de darle solución a todos los errores de una manera ordenada, según el grado de importancia de cada error.

En todos estos sistemas se le da un seguimiento a cada error, desde que son lanzados hasta su solución final, con el fin de saber siempre en todo el ciclo de vida del error en qué estado se encuentra con respecto a la solución.

Algunos de estos estados pueden ser:

- ✓ Pendiente: cuando no exista todavía su solución.
- ✓ En trámite: cuando se esté desarrollando su solución.
- ✓ Solucionado: cuando ya se le haya dado solución.

Los administradores de este sistema serán los responsables de cambiar el estado a cada error cuando se le haga alguna modificación.

En el mundo, este tipo de sistemas se han convertido en un eslabón fundamental para lograr la comunicación entre el equipo de desarrolladores, la aplicación y los clientes que la explotan.

Entre los sistemas de seguimiento de errores más conocidos se encuentran **Bugzilla** y **Mantis**.

- ✓ **Bugzilla** es una herramienta de seguimiento de errores basada en Web, originalmente desarrollada y usada por el proyecto Mozilla. Lanzado como software de código abierto por Netscape Communications en 1998. Permite organizar en múltiples formas los defectos de software, permitiendo el seguimiento de múltiples productos con diferentes versiones, a su vez compuestos de múltiples componentes. Permite además categorizar los defectos de software de acuerdo a su prioridad y severidad, así como asignarles versiones para su solución. También permite anexar comentarios, propuestas de solución, designar a responsables a los que asignar la resolución y el tipo de solución que se aplicó al defecto, todo ello llevando un seguimiento de fechas en las cuales sucede cada evento y, si se configura adecuadamente, enviando mensajes de correo a los interesados en el error. Bugzilla utiliza un servidor HTTP y una base de datos para llevar a cabo su trabajo. Cada error puede tener diferente prioridad y encontrarse en diferentes estados, así como ir acompañado de notas del usuario o ejemplos de código que ayuden a corregir el error (5).

- ✓ **Mantis Bug Tracker** es un software que constituye una solución muy completa para gestionar tareas en un equipo de trabajo. Es una aplicación de código abierto hecha en PHP y Mysql, destaca por su facilidad y flexibilidad de instalar y configurar. Esta aplicación se utiliza para testear soluciones, hacer un registro histórico de alteraciones y gestionar equipos remotamente. Además es capaz de depurar errores de aplicaciones, sitios web y todo aquello que requiera un seguimiento y mejoras continuas y constantes. Esta aplicación permite la creación de diversas cuentas de usuario desde las cuales puedes informar de los bugs detectados. Con Mantis se puede dividir un proyecto en varias categorías, lo cual permite hacer un seguimiento más exacto de este. El flujo de trabajo también se puede configurar desde la propia herramienta, de forma que puede definirse quién puede causar problemas, quién puede analizarlos y quién puede atenderlos. Una de las características de Mantis es su gran abanico de posibilidades para su configuración, alguna de sus características son las nombradas a continuación:
  - ✓ Permite configurar la transición de estados (abierto, encaminado, testeado, devuelto, cerrado, reabierto, entre otros).
  - ✓ Puede especificar un número indeterminado de estados para cada tarea (abierta, encaminada, testeada, devuelta, cerrada, reabierta, ente otros).
  - ✓ Permite introducir diferentes perfiles (programador, probador, coordinador, visualizador, entre otros).

  Mantis incluye filtros, un sistema de búsqueda, tiene soporte para varios idiomas y también informa por correo electrónico de la resolución de los errores de los que se ha informado (6).

Para darle solución al problema planteado no se tuvo en cuenta ninguno de estos sistemas de seguimiento y control de errores debido a que no es lo que se quiere implementar. Estos sistemas han servido para tomar algunas ideas para la realización de este trabajo, como pueden

ser los estados de los errores, la prioridad de las sugerencias, la fecha en que fueron lanzados los errores y las sugerencias.

1.4. Componente feedback.

El proceso de compartir observaciones, preocupaciones y sugerencias con la intención de recabar información, a nivel individual o colectivo, es vital para intentar mejorar el funcionamiento de cualquier organización o de cualquier grupo formado por seres humanos. Este proceso es denominado **feedback**, conocido también como retroalimentación, y es un sistema de comunicación que se refiere a la capacidad del emisor para recoger las reacciones de los receptores ya sea mediante lenguaje verbal o no verbal, y de acuerdo con la actitud de estos, modificar su mensaje.

El **feedback** también está presente en numerosos espacios tecnológicos. En este sentido, gran parte de los aparatos y máquinas que utilizamos en nuestra vida cotidiana funcionan a través del sistema de **feedback**, ya que suponen el intercambio y traspaso permanente de datos de cualquier tipo. Un ejemplo claro de esta situación es la conexión a internet que, además de contar con un espacio virtual, necesita de un soporte técnico y físico a través del cual se mandan y reciben permanentemente datos de diversos tipos.

En términos informáticos, **feedback**, es un componente que se adhiere a una aplicación, con el fin de permitir una retroalimentación constante a través de la información que brinda el cliente. De esta manera la aplicación mejorará gradualmente en aras de lograr una aplicación óptima.

En el ámbito Web, realizar **feedback** está orientado a generar niveles de comunicación muy avanzados, de los cuales se esperan respuestas rápidas e interacción permanente.

Por lo anterior, es importante reconocer los diversos sistemas que puede utilizar una aplicación para recibir **feedback** o retroalimentación por parte del usuario.

Ejemplo de ellos son:

- ✓ Sistemas de Correo Electrónico.
- ✓ Sistemas de Encuestas o Votaciones.
- ✓ Sistemas de Foros.
- ✓ Sistemas de Chat.
- ✓ Sistemas de Simulación.

Es importante considerar que cada uno de ellos genera más información y conocimiento de la audiencia que utiliza la aplicación, por lo que siempre será interesante ver cómo reaccionan a los temas que se les planteen y, asimismo, administrar positivamente el mayor flujo de trabajo que pueden generar en una aplicación Web muy participativa (7).

El **feedback**, cuando es usado acertadamente contribuye a:

- ✓ Individualizar el aprendizaje.
- ✓ Diagnosticar dificultades o detectarlas en la práctica, así como predecir posibles niveles de rendimiento.
- ✓ Orientar y corregir, la intensidad y adecuación de los programas de entrenamiento-intervención, y la continua adaptación a estos, adecuando su acción para alcanzar paulatinamente los objetivos propuestos.
- ✓ Motivar al sujeto que aprende, si la información que se le ofrece es adecuada. Esto es mucho más necesario si los objetivos a alcanzar se sitúan a un largo plazo (8).

Variables que determinan la eficacia y validez del **feedback**:

- ✓ Frecuencia y Número de ensayos sobre los que se informa.
- ✓ Momento de Presentación.
- ✓ Tareas Intermedias.
- ✓ Grado de Precisión.
- ✓ Retirada del Conocimiento de los Resultados (8).

Para poder suministrar datos de manera ordenada y científica, no es suficiente con recogerlos, sino que se hace necesario confeccionar un programa que sistematice todo el proceso que va desde la detección de deficiencias o imperfecciones a la corrección de estas. Este proceso es denominado registro y recogida sistemática de la información.

El proceso podría definirse en 2 etapas:

- ✓ **Registro**: A la hora de observar la conducta se pierde en ciertos casos información, bien por la rapidez con la que suceden las acciones, bien a la complejidad de la misma o bien a la dificultad del manejo de los instrumentos que registran dicha conducta. Para eludirlos en conclusión, hay que elegir un buen instrumento adecuado a la tarea y al espacio donde se va a desarrollar el registro, que al final ofrezca una información fiable, válida, objetiva, veraz y útil.

- ✓ **Análisis y Categorización**: Previamente al proceso de análisis y categorización se requiere una delimitación del objeto de estudio y una selección de lo que conceptualmente se considera más relevante. La descripción detallada es el primer requisito para la construcción de categorías (8).

1.5. Herramientas y metodologías de desarrollo.

1.5.1. Entorno de Desarrollo Integrado (IDE):

Un Entorno de Desarrollo Integrado (IDE), es un programa informático compuesto por un conjunto de herramientas de programación. Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación; consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI) (9).

Se escogió como IDE a utilizar **NetBeans IDE 7.0.1.** Este IDE es libre, permite crear aplicaciones Web con PHP 5, además viene con soporte para el **framework Symfony**, facilitando el desarrollo de aplicaciones Web haciendo uso de este **framework**. También incluye un módulo de

Subversion, permitiendo desde el mismo entorno manipular las versiones del código durante la implementación.

1.5.2. Lenguajes de Programación.

1.5.2.1 PHP 5

Como lenguaje de programación del lado del servidor se escogió PHP 5. PHP es un lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas.

Principales Características:

- ✓ Orientado al desarrollo de aplicaciones web dinámicas con acceso a información almacenada en una base de datos.
- ✓ El código fuente escrito en PHP es invisible al navegador web y al cliente.
- ✓ Capacidad de conexión con la mayoría de los motores de base de datos que se utilizan en la actualidad, destaca su conectividad con Mysql y PostgreSQL.
- ✓ Capacidad de expandir su potencial utilizando módulos.
- ✓ Es libre, por lo que se presenta como una alternativa de fácil acceso para todos.
- ✓ Manejo de excepciones.
- ✓ Muestra una mejoría en cuanto al rendimiento que ofrece en los ordenadores (10).

1.5.2.2 HYPERTEXT MARKUP LANGUAGE (HTML).

HTML es el lenguaje de marcado predominante para la elaboración de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes (11).

1.5.2.3 CSS.

CSS es un lenguaje de hojas de estilos en cascada creado para mejorar la interfaz gráfica de las aplicaciones web. Con el uso de CSS se hace posible separar la presentación del contenido. La

creación de hojas de estilo es fácil pues solo se necesita tener un conocimiento básico de HTML para poder diseñar un estilo propio que sea agradable para el cliente. (12).

1.5.2.4 JAVASCRIPT

Como lenguaje del lado del cliente se utiliza JavaScript. Este es un lenguaje de programación interpretado, dinámico y con soporte de funciones anónimas (13). Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas, en bases de datos locales al navegador. Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM) (14).

1.5.3. Framework de Desarrollo.

Un **framework** de desarrollo es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, con base a la cual otro proyecto de software puede ser más fácilmente organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Las principales ventajas de utilizar un framework en la realización de una aplicación web son las siguientes:

- ✓ El desarrollo rápido de aplicaciones. Los componentes incluidos en un **framework** constituyen una capa que libera al programador de la escritura de código de bajo nivel.
- ✓ La reutilización de componentes software. Los **frameworks** son los paradigmas de la reutilización.
- ✓ El uso y la programación de componentes que siguen una política de diseño uniforme. Un **framework** orientado a objetos logra que los componentes sean clases que pertenezcan a

una gran jerarquía de clases, lo que resulta en bibliotecas más fáciles de aprender a usar (15).

Symfony es un framework que ha sido diseñado para optimizar el desarrollo de las aplicaciones web. Está desarrollado completamente en PHP 5, se puede integrar a los ORM (Object-Relational Mapping) Propel y Doctrine los cuales crean una capa de abstracción de la base de datos que permite a los desarrolladores realizar las consultas SQL de una forma más sencilla. Separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web; además de proporcionar varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación, es compatible con la mayoría de gestores de bases de datos, entre ellos PostgreSQL. Es fácilde extender, lo que permite su integración con las bibliotecas de otros fabricantes y presenta una potente línea de comandos que facilitan la generación de código, lo cual contribuye a ahorrar tiempo de trabajo. Provee un gran número de características integradas, tales como, el uso de **templates** (plantillas) y **helpers** (ayudadores), manejo de **cache** y URL inteligente, abstracción de la Base de Datos y decenas de **plugins** que permiten extender las funcionalidades del **framework**.

El uso del framework Symfony, unido a Propel, posibilita el uso de las tablas de la base de datos como objetos, lo que facilita la manipulación de los datos en la lógica de negocio contenida en el modelo. La abstracción de la base de datos provee de un conjunto de clases de acceso a datos ordenadas que contribuye a la organización de las clases del modelo. La estructura propuesta de MVC (Modelo-Vista-Controlador), aplicando el patrón controlador frontal por defecto, permite que las peticiones a la aplicación se hagan a través de un único punto de acceso, dejando a las clases controladoras la única responsabilidad de la lógica de la aplicación. De esta manera se aligera el trabajo de los controladores y estas clases pueden desempeñar mejor su responsabilidad como clases (16).

Por todo lo descrito anteriormente y por su integración con **Netbeans**, permitiéndole todas las potencialidades antes descritas se utilizó **Symfony** como framework de desarrollo.

1.5.4. Sistema Gestor de Bases de Datos (SGBD)

El propósito general de los sistemas de gestión de bases de datos es manejar de manera clara, sencilla y ordenada un conjunto de datos que posteriormente se convertirán en información relevante para una organización.

Existen distintos objetivos que deben cumplir los SGBD:

- ✓ **Independencia.** La independencia de los datos consiste en la capacidad de modificar el esquema (físico o lógico) de una base de datos sin tener que realizar cambios en las aplicaciones que se sirven de ella.
- ✓ **Consistencia.** En aquellos casos en los que no se ha logrado eliminar la redundancia, será necesario vigilar que aquella información que aparece repetida se actualice de forma coherente, es decir, que todos los datos repetidos se actualicen de forma simultánea.
- ✓ **Seguridad.** La información almacenada en una base de datos puede llegar a tener un gran valor. Los SGBD deben garantizar que esta información se encuentra segura de permisos a usuarios y grupos de usuarios, que permitan otorgar diversas categorías de permisos.
- ✓ **Tiempo de respuesta.** Lógicamente, es deseable minimizar el tiempo que el SGBD demora en proporcionar la información solicitada y en almacenar los cambios realizados.

Existen en la actualidad un sin número de sistemas gestores de bases de datos (SGBD), la mayoría de estos son software propietario. Entre los gestores de software libre más usados se destaca PostgreSQL.

PostgresSQL8.4 es un sistema de gestión de base de datos relacional orientado a objetos y libre, publicado bajo la licencia BSD. Mediante un sistema denominado MVCC (Acceso concurrente multiversión, por sus siglas en inglés), PostgreSQL permite que mientras un proceso

escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. Cada usuario obtiene una visión consistente de lo último a lo que se le hizo *commit*. Esta estrategia es superior al uso de bloqueos por tabla o por filas común en otras bases, eliminando la necesidad del uso de bloqueos explícitos. (17)

La aplicación SIPP sobre la que se desea implementar el componente **feedback** utiliza como gestor de base de datos PostgreSQL, por ello es recomendable utilizarlo para garantizar la compatibilidad entre la base de datos del componente y la existente en la aplicación.

1.5.5. Servidor Web

Un servidor web o servidor HTTP es un programa que atiende y responde a las diferentes peticiones realizadas por los usuarios a través de los navegadores, usando el protocolo HTTP/ HTTPS pertenecientes a la capa de aplicación del modelo OSI.

Apache es un servidor web HTTP. Entre las principales ventajas que brinda Apache se encuentran:

- ✓ Corre en una multitud de Sistemas Operativos, como Unix, Microsoft Windows, Macintosh y otras lo que lo hace prácticamente universal.
- ✓ Es una tecnología de código fuente, abierto.
- ✓ Es un servidor altamente configurable de diseño modular.
- ✓ Trabaja con gran cantidad de Perl, PHP y otros lenguajes de script.
- ✓ Multi-hilos: esto permite atender varias peticiones al servidor concurrentemente. (18)

Apache se considera la plataforma web más utilizada del mundo, pues aumentan cada día el número de usuarios que permiten este código fuente abierto en su infraestructura. Es muy usado por la universidad, con amplia disposición de documentos e información, además de tener gran aceptación en toda la red.

Por lo anteriormente descrito y porque para el desarrollo de las aplicaciones web en el SIPP se definió que este sería el servidor a utilizar, se utilizó este servidor para garantizar así la compatibilidad.

1.5.6. Metodologías de desarrollo

Las metodologías de desarrollo surgen debido a la necesidad de utilizar una serie de patrones, guías, procedimientos, herramientas y soporte documental a la hora de desarrollar un producto de software. Existe una gran variedad de metodologías para la creación de un software (19). Un ejemplo de ellas es el Rational Unified Process (RUP).

RUP es un proceso dirigido por casos de uso, centrado en la arquitectura, y es iterativo e incremental.Este proceso de desarrollo está definido por 4 fases fundamentales:

- ✓ Inicio: en esta fase se determina la visión del proyecto.
- ✓ Elaboración: en esta fase se determinar la arquitectura óptima.
- ✓ Construcción: en esta fase se obtiene la capacidad operacional inicial.
- ✓ Transición: en esta fase el objetivo es llegar a obtener la liberación del proyecto.

RUP identifica 6 buenas prácticas que definen una forma efectiva de trabajar para los equipos de desarrollo de software.

- ✓ **Gestión de requisitos:** RUP brinda una guía para encontrar, organizar, documentar, y seguir los cambios de los requisitos funcionales y restricciones. Utiliza una notación de Caso de Uso y escenarios para representar los requisitos.
- ✓ **Desarrollo de software iterativo:** Desarrollo del producto mediante iteraciones con hitos bien definidos, en las cuales se repiten las actividades pero con distinto énfasis, según la fase del proyecto.
- ✓ **Desarrollo basado en componentes:** La creación de sistemas intensivos en software requiere dividir el sistema en componentes con interfaces bien definidas, que posteriormente serán

ensamblados para generar el sistema. Esta característica en un proceso de desarrollo permite que el sistema se vaya creando a medida que se obtienen o se desarrollan sus componentes.

- ✓ **Modelado visual (usando UML):** El modelado visual ayuda a mejorar la capacidad del equipo para gestionar la complejidad del software.
- ✓ **Verificación continua de la calidad:** Es importante que la calidad de todos los artefactos se evalúe en varios puntos durante el proceso de desarrollo, especialmente al final de cada iteración. En esta verificación las pruebas juegan un papel fundamental y se integran a lo largo de todo el proceso. Para todos los artefactos no ejecutables las revisiones e inspecciones también deben ser continuas.
- ✓ **Gestión de los cambios:** El cambio es un factor de riesgo crítico en los proyectos de software. Los artefactos software cambian no sólo debido a las acciones de mantenimiento posteriores a la entrega del producto, sino que durante el proceso de desarrollo, especialmente los cambios en los requisitos.

Por lo anteriormente descrito y por los conocimientos adquiridos en el transcurso de esta carrera se utilizará RUP como metodología de desarrollo para la realización de este trabajo.

1.5.7. Herramienta CASE

Las herramientas CASE (Computer Aided Software Engineering) están destinadas a aumentar la productividad en el desarrollo de software reduciendo tiempo y dinero. Estas herramientas pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, documentación o detección de errores entre otras. (20)

Entre las herramientas CASE más utilizadas se encuentran Rational Rose, Microsoft Project y Visual Paradigm, algunos de los beneficios más significativos de estas herramientas son los siguientes:

- ✓ **Facilidad para la revisión de aplicaciones**. Contar con un depósito central agiliza el proceso de revisión ya que éste proporciona bases para las definiciones y estándares para los datos.
- ✓ **Soporte para el desarrollo de prototipos de sistemas**. En ocasiones se desarrollan diseños para pantallas y reportes con la finalidad de mostrar la organización y composición de los datos, encabezados y mensajes. Los ajustes necesarios al diseño se hacen con rapidez para alterar la presentación y las características de la interfaz.
- ✓ **Generación de código**. La ventaja más visible de esta característica es la disminución del tiempo necesario para preparar un programa.
- ✓ **Mejora en la habilidad para satisfacer los requerimientos del usuario**. Tener los requerimientos correctos mejora la calidad de las prácticas de desarrollo. Las descripciones gráficas y los diagramas, así como los prototipos de reportes y la composición de las pantallas, contribuyen a un intercambio de ideas más efectivo.
- ✓ **Soporte interactivo para el proceso de desarrollo**. Las herramientas CASE soportan pasos interactivos al eliminar el tedio manual de dibujar diagramas, elaborar catálogos y clasificar.

Visual Paradigm es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. También proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML. Presenta licencia gratuita y comercial. Es fácil de instalar y actualizar y compatible entre ediciones.

Características principales:

- ✓ Soporte de UML versión 2.1.
- ✓ Modelado colaborativo con CVS y Subversión (control de versiones).
- ✓ Generación de código - Modelo a código, diagrama a código.
- ✓ Diagramas de flujo de datos.
- ✓ Soporte ORM - Generación de objetos Java desde la base de datos.
- ✓ Transformación de base de datos – De bases de datos existentes a diagramas de Entidad-Relación, y de diagramas de Entidad-Relación en tablas de base de datos.

- ✓ Distribución automática de diagramas – Reorganiza figuras y conectores de los diagramas UML.
- ✓ Importación y exportación de ficheros.

Para la realización de este trabajo se utilizó **Visual Paradigm,** ya que a diferencia de otras herramientas de este tipo, el equipo de desarrollo cuenta con una mayor experiencia con esta herramienta que con otras existentes, es multiplataforma y utiliza UML como lenguaje de modelado. El lenguaje de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML. Por todo esto es que se utilizó esta herramienta CASE.

1.6. Conclusiones Parciales.

Como resultado de la investigación realizada de las herramientas y tecnologías que existen en el mundo y por el amplio conocimiento que se tiene acerca de estas herramientas, se utilizó como metodología de desarrollo de software RUP y como herramienta CASE, el Visual Paradigm apoyándose en el lenguaje de modelado UML. Como IDE de desarrollo se usó el NetBeans IDE 7.0.1 por los módulos con los que viene incluido para Subversion y para Symfony. Como framework de desarrollo se utilizó Symfony 1.2.8 debido al aprovechamiento del ORM Propel que trae incluido y a la organización y seguridad que le brinda al proyecto así como a la aplicación respectivamente. Como gestor de base de datos y servidor web se seleccionó PostgreSQL 8.4 y Apache 2.0 respectivamente para garantizar la compatibilidad entre las aplicaciones existentes. Dejando establecidas todas las herramientas y metodologías utilizadas en el desarrollo de este trabajo se le da respuesta a la segunda tarea de investigación.

CAPÍTULO 2. CARACTERÍSTICAS DEL SISTEMA.

En el presente capítulo se realizará una descripción completa del componente a implementar con el fin de lograr un mejor entendimiento por parte del equipo de desarrolladores. Para esto se especificarán los requisitos que presenta el componente, tanto funcionales como no funcionales, se declararán los actores y sus vínculos con los casos de uso, por medio de un diagrama de casos de uso del sistema, así como una detallada descripción de cada uno de ellos, y los patrones de casos de uso a utilizar. Además se confeccionará el diagrama de modelo de dominio y se realizará una breve descripción del mismo

2.1. Propuesta del sistema.

Se propone la implementación de un componente **feedback** que permita la comunicación entre el cliente y el equipo de desarrollo, permitiendo gestionar los errores arrojados por la base de datos, el servidor web y la aplicación, así como las sugerencias emitidas por los clientes.

Permitirá a los clientes ingresar cualquier sugerencia que crean que mejorará la aplicación desde su punto de vista, con el objetivo de que la aplicación sea más agradable y lo más cercana al gusto de los que la explotan. Además de permitir modificar una sugerencia existente, dejándole a los usuarios cambiar solo los campos que tienen permitido modificar, así como eliminar una sugerencia existente siempre y cuando la sugerencia a eliminar haya sido realizada por el usuario que desea eliminar la sugerencia. Permitirá a los administradores tener un conocimiento del comportamiento que tienen las aplicaciones utilizadas en el lugar del despliegue, conociendo los errores que se están arrojando por cada una de esas aplicaciones, así como el grado de importancia que tiene cada error. Además de guardar la última línea analizada de cada fichero .log después de ser analizados, con el objetivo de empezar a analizar el fichero a partir de esa línea cuando se vuelve a realizar la acción, para no tener que analizar los mismos errores nuevamente.

2.2. Modelo de Dominio.

Cuando no existen procesos definidos, RUP propone realizar un modelo de dominio, que es un subconjunto del modelo de negocio. El modelo de dominio ayuda a comprender mejor los conceptos que utilizan los usuarios, los conceptos con los que trabajan y con los que deberá trabajar la aplicación. Se obtiene mediante la entrevista con los clientes y requiere la participación de expertos en el negocio. Se elabora en un diagrama de clases añadiéndole las relaciones existentes entre ellas y los atributos propios de cada clase.

2.2.1. Descripción del modelo de dominio.

En la figura 2.1 se muestra la descripción del negocio con el objetivo de facilitar la comprensión del sistema.

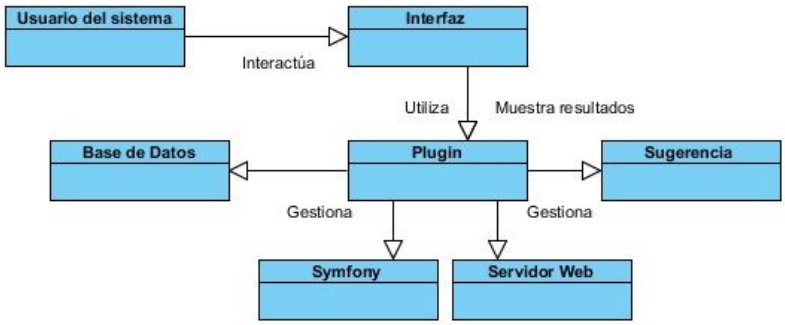


Fig 2.1. Modelo del Dominio

En la Fig 2.1 se pueden observar las siguientes clases conceptuales del dominio:

- ✓ **Usuario:** objeto encargado de iniciar las acciones referentes a la gestión de las sugerencias.
- ✓ **Interfaz:** objeto encargado de la interacción entre el cliente y la aplicación.
- ✓ **Plugin:** objeto encargado de comunicar la interfaz de los usuarios con toda la información existente en la base de datos.
- ✓ **Base de datos:** objeto encargado de gestionar los errores de la base de datos.

- ✓ **Symfony:** objeto encargado de gestionar los errores de la aplicación Maximus Drill Pro.
- ✓ **Servidor web:** objeto encargado de gestionar los errores del servidor web.
- ✓ **Mejoras:** objeto encargado de gestionar las sugerencias de mejoras emitidas por el cliente.

2.3. Reglas del negocio.

Las reglas del negocio son las acciones que median entre los datos y la gestión de éstos facilitando las tomas de decisiones de los gestores o directores. Son elementos constitutivos de uno o varios procesos de negocio. (21)

Para la realización del presente trabajo se tuvieron en cuenta las siguientes reglas del negocio:

- ✓ Una mejora recién creada, su estado será de Pendiente.
- ✓ No se podrá eliminar una sugerencia que su estado sea “En Trámite”, solo se podrá hacer si su estado es “Pendiente” o “Solucionada”.
- ✓ El cliente solo tendrá permiso a eliminar aquella mejora que haya sido realizada por él.
- ✓ Después de recoger todos los errores generados por el servidor web, el sistema gestor de base de datos y la aplicación, se guardará la última línea analizada de cada fichero log, con el objetivo de empezar a analizar desde esa línea en futuras búsquedas.

2.4. Requerimientos del sistema.

Un requerimiento es la condición o capacidad que debe exhibir o poseer un sistema o componente para satisfacer un contrato, estándar, especificación, u otra documentación formalmente impuesta, facilitando el entendimiento entre clientes y desarrolladores.

Estos pueden clasificarse en dos tipos:

- ✓ Requisitos funcionales

- ✓ Requisitos No funcionales

Seguidamente se exponen los requisitos funcionales y no funcionales, por los cuales se rige el desarrollo del componente.

2.4.1. Requisitos funcionales del sistema.

Son propiedades que el producto tiene que tener para satisfacer las solicitudes del cliente. Los requisitos funcionales contienen un nombre, un código único y un resumen. Esta información se utiliza para ayudar al lector a entender por qué el requisito es necesario, y para seguir al mismo durante todo el proceso de desarrollo del producto. La descripción de cada requisito debe ser clara y concisa.

En el componente a desarrollar se han identificado los siguientes requisitos funcionales:

RF1 Actualizar notificaciones.
RF2 Consultar notificaciones.
RF3 Eliminar notificaciones.
RF4 Modificar sugerencia.
RF5 Insertar sugerencia.

2.4.2. Requisitos no funcionales del sistema.

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como requisitos que ni describen información a guardar, ni funciones a realizar. Son las características que hacen al producto atractivo, usable, rápido y confiable.

A continuación se muestran requisitos no funcionales del componente.

2.4.2.1 REQUERIMIENTOS DE SOFTWARE

- ✓ **Software instalado en la estación de trabajo del cliente:** Sistema Operativo tanto Windows (XP o superior) como Linux (cualquier distribución). Navegador web Internet Explorer 5, Mozilla Firefox 3.0 o versiones superiores.
- ✓ **Software instalado en el Servidor Web:** Apache 2.0 y PHP 5.
- ✓ **Software instalado en el Servidor de Base de datos:** Postgres en su versión 8.4

2.4.2.2 REQUERIMIENTOS DE HARDWARE

- ✓ **Hardware de la estación de trabajo del cliente:** Procesador Pentium IV a 3.0 GHz, con 1 GB de memoria RAM
- ✓ **Hardware de la estación de trabajo del servidor web:** Procesador Pentium IV a 3.0 GHz, con 1 GB de memoria RAM y 250 GB de espacio libre en el disco duro.
- ✓ **Hardware de la estación de trabajo del servidor de base de datos:** Procesador Pentium IV, con 256 MB de memoria RAM y 500 GB de espacio libre en el disco duro.

2.4.2.3 REQUERIMIENTOS DE APARIENCIA O INTERFAZ EXTERNA:

La aplicación contará con una interfaz agradable a la vista del usuario y sin mucha información en las páginas, acercándose lo más posible a las aplicaciones de escritorio.

2.4.2.4 REQUERIMIENTOS DE SEGURIDAD

- ✓ **Confidencialidad:** existirán diferentes roles para restringir los permisos de acceso a determinados niveles de información.
- ✓ **Integridad:** la información solo podrá ser modificada por los usuarios autorizados al acceso a esta información, mediante la validación de estos datos en el servidor, evitando la corrupción y los estados inconsistentes.
- ✓ **Disponibilidad:** a los usuarios autorizados se les garantizará acceso a la información en el momento requerido.

2.4.2.5 REQUERIMIENTOS DE USABILIDAD

El componente puede ser usado por cualquiera de los clientes del SIPP, no se necesitan conocimientos avanzados de computación ni de aplicaciones web.

2.4.2.6 REQUERIMIENTOS DE PERSISTENCIA

- ✓ En la base de datos se almacenará permanentemente la información del sistema con el fin de poderle realizar un análisis en cualquier momento que se desee.

2.4.2.7 REQUERIMIENTOS DE RENDIMIENTO

- ✓ Tiempo de respuesta: menos de 3 segundos.
- ✓ Velocidad de procesamiento de la información: menos de 3 segundos.

2.4.2.8 REQUERIMIENTOS DE REUSABILIDAD

- ✓ Cada módulo del componente se desarrollará de manera que pueda ser reutilizado o capaz de funcionar por sí mismo y no como parte de un paquete general.

2.4.2.9 REQUERIMIENTOS DE PORTABILIDAD

- ✓ El componente podrá ser usado desde cualquier plataforma. Se podrá disponer del mismo tanto en el sistema operativo Linux como Windows.

2.4.2.10 LEGALES

- ✓ El componente será registrado en el Centro Nacional de Derecho de Autor a través de la Dirección de Servicios Legales de la UCI.

2.5. Modelo de Casos de Uso del Sistema.

El modelo de casos de uso describe las funcionalidades del sistema. Un caso de uso representa la interacción entre los diferentes usuarios y el sistema, cada usuario puede estar representado por uno o varios actores, los que pueden representar un individuo o un sistema externo.

Se identificaron como actores del sistema al cliente y al administrador.

Actor	Descripción
Cliente	Es la persona encargada de los procesos de gestión de las mejoras.
Administrador	Es la persona encargada de gestionar las notificaciones y de generar los reportes.

2.5.1. Diagrama de Casos de Uso del Sistema

Los patrones utilizados en el desarrollo del componente, son: el patrón CRUD (Create, Read, Update y Delete), este patrón propone identificar un caso de uso que modela las operaciones de crear, leer, actualizar y eliminar, sobre una entidad; y el patrón Cohesión adición. Quedando agrupados los 5 requisitos funcionales en los siguientes casos de uso. Ver Figura 2.2.

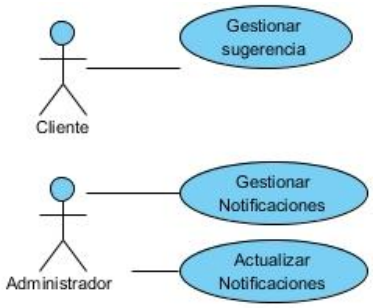


Figura 2.2 Diagrama de Casos de Uso del Sistema

2.5.2. Descripción textual de los casos de uso del sistema

Cada Caso de Uso posee una descripción de las acciones que realizará el sistema como respuesta a las peticiones del usuario. A continuación se relacionan las tablas correspondientes a

las descripciones de los Casos de Uso detectados y se argumentan los flujos operacionales de cada uno, los casos de uso más complejos se dividen en secciones.

2.5.2.1 DESCRIPCIÓN DEL CASO DE USO GESTIONAR SUGERENCIAS.

Caso de Uso	Gestionar Sugerencias.
Actores	Cliente (inicia).
Propósito	Permitir Insertar, listar, editar y eliminar datos acerca de una sugerencia.
Resumen	El cliente cuenta con la posibilidad de realizar diferentes acciones con las mejoras, estas acciones pueden ser: insertar una nueva mejora, modificar y eliminar una existente siempre y cuando haya sido realizada por él.
Referencias	RF3, RF4, RF5.
Precondiciones	El cliente debe estar previamente autenticado en el sistema.
Flujo Normal de los eventos	

Acciones del Actor	Respuesta del sistema
1. Inicia el caso de uso cuando decide ejecutar una de las siguientes acciones: Insertar una mejora. Editar una mejora. Eliminar una mejora.	2. El sistema en dependencia de la opción que seleccione el cliente realiza las siguientes operaciones: Si el cliente selecciona la opción de insertar una mejora ir a la sección “Insertar Mejora”. Si el cliente selecciona la opción de actualizar una mejora ir a la sección “Actualizar Mejora”. Si el cliente selecciona la opción de eliminar una mejora ir a la sección “Eliminar Mejora”.
Sección “Insertar Mejora”	
Acciones del Actor	Respuesta del Sistema
	1- Se despliega un diálogo donde se muestran los datos que debe tener una mejora, descripción y prioridad.
2- El cliente ingresa los datos requeridos para la creación de una nueva mejora y presiona el botón	3- El sistema inserta la nueva mejora y refresca el área donde se muestran todas las mejoras. Finalizando así el caso de uso.

aceptar.	
Flujo Alterno 1 “Insertar Mejora”	
2.1- El cliente ingresa o no los datos requeridos para le creación de una nueva mejora y presiona el botón cancelar.	2.2- El sistema regresa al área donde se muestran las mejoras y no realiza ninguna acción.
Sección “Eliminar Mejora”	
Acciones del Actor	Respuesta del Sistema
	1- El sistema pide confirmación de la acción a realizar.
2- El cliente confirma que desea eliminar la mejora seleccionada.	3- El sistema verifica el estado de la mejora, si está en pendiente lo elimina y actualiza el área donde se muestran las mejoras. Finalizando así el caso de uso.
Flujo Alterno 1 “Eliminar Mejora”	
2.1- El cliente confirma que no desea eliminar la mejora.	2.2- El sistema no realiza la acción de eliminar la mejora y concluye así la sección.

Flujo Alterno 2 “Eliminar Mejora”	
	3.1- El sistema verifica el estado de la mejora, si no es “Pendiente”, muestra un mensaje informando al usuario que no puede realizar esta acción debido a que el estado de la mejora no es de “Pendiente”.
Sección “Modificar Mejora”	
Acciones del Actor	Respuesta del Sistema
	1- El sistema verifica que el estado de la mejora sea “Pendiente” y despliega un dialogo donde se muestran los datos que pueden ser modificados por el cliente, la descripción y la prioridad.
2- El cliente introduce los nuevos datos y presiona el botón modificar.	3- El sistema valida los datos introducidos.
	4- Si los datos introducidos son correctos se actualiza el área donde se muestran las mejoras con los nuevos datos. Finalizando así el caso de uso.

Flujo Alterno 1 “Modificar Mejora”	
	1.1- El sistema verifica el estado de la mejora, si no es “Pendiente”, muestra un mensaje informando al usuario que no puede realizar esta acción debido a que el estado de la mejora no es de “Pendiente”.
Flujo Alterno 2 “Modificar Mejora”	
2.1- El cliente introduce los datos y presiona el botón cancelar.	2.2- El sistema cancela la operación y deja la mejora con los datos anteriores, concluyendo así la sección.
Flujo Alterno 3 “Modificar Mejora”	
	4.1- Si los datos introducidos son incorrectos, el sistema mostrará un mensaje indicando que campo es el incorrecto.

2.5.2.2 DESCRIPCIÓN DEL CASO DE USO CONSULTAR NOTIFICACIONES.

Caso de Uso	Consultar Notificaciones.

Actores	Administrador (inicia).
Propósito	Permitir consultar las notificaciones existentes en la base de datos, de Symfony, Apache o Postgres.
Resumen	El administrador cuenta con la posibilidad de poder ver todas las notificaciones existentes.
Referencias	RF2
Precondiciones	El administrador debe estar previamente autenticado en el sistema.
Flujo Normal de los eventos	
Acciones del Actor	Respuesta del sistema
1- Inicia el caso de uso cuando selecciona la opción “Consultar Notificaciones”.	2- El sistema muestra las notificaciones existentes con todas sus características. Finalizando así el caso de uso.

2.5.2.3 DESCRIPCIÓN DEL CASO DE USO ACTUALIZAR NOTIFICACIONES.

Caso de Uso	Actualizar Notificaciones.

Actores	Administrador (inicia).
Propósito	Permitir agregar las últimas notificaciones a la base de datos.
Resumen	El administrador cuenta con la posibilidad de poder agregar a la base de datos las últimas notificaciones de la base de datos, del servidor web y de Symfony.
Referencias	RF1
Precondiciones	1- El administrador debe estar previamente autenticado en el sistema. 2- El administrador debe haber realizado el caso de uso “Consultar Notificaciones”.
Flujo Normal de los eventos	
Acciones del Actor	Respuesta del sistema
1- Inicia el caso de uso cuando selecciona la opción “Buscar Errores”.	2- El sistema busca los errores en los ficheros .log, los agrega a la base de datos, guarda la última línea analizada del fichero y muestra lo errores. Finalizando así el caso de uso.

2.6. Consideraciones Parciales.

En el desarrollo de este capítulo se elaboró el modelo de dominio, dándole cumplimiento a la tercera tarea de la investigación. Se identificaron las funcionalidades necesarias para el correcto funcionamiento de la aplicación, agrupadas en 3 casos de uso arquitectónicamente significativos, y se identificaron además los requisitos no funcionales del sistema. Se realizó el diagrama de casos de uso del sistema, así como la descripción de cada caso de uso, para obtener un mayor entendimiento del problema a resolver.

CAPÍTULO 3. ANÁLISIS Y DISEÑO DEL SISTEMA.

En el presente capítulo se describe el sistema desde la perspectiva de Ingeniería de Software, utilizando diferentes diagramas como el diagrama de despliegue, el diagrama de clases persistentes y el modelo de datos de la base de datos. También se mostrará el diagrama de clases del diseño, con una descripción de cada paquete que lo conforma, explicando los patrones de diseño y los estilos utilizados, además de explicar cómo se adaptan dichos estilos y patrones al diseño del sistema.

3.1. Arquitectura del Sistema

La arquitectura de software incluye los aspectos estáticos y dinámicos más significativos del sistema. Es el conjunto de decisiones significativas sobre la organización de un sistema, la selección de los elementos estructurales y sus interfaces de los cuales el sistema está compuesto junto con su comportamiento. Describe los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente. La misma se representa en 4+1 vistas arquitectónicas y entre ellas se encuentran la vista lógica, de despliegue, de implementación, de procesos y de casos de uso.

3.1.1. Patrón de arquitectura. Modelo Vista Controlador.

Buschmann define un patrón arquitectónico como aspectos fundamentales de la estructura de un sistema de software. Ellos especifican un conjunto predefinido de subsistemas con sus responsabilidades y una serie de recomendaciones para organizar los distintos componentes. Los patrones de arquitectura ofrecen el esquema fundamental de organización para sistemas de software, incluyendo reglas y guías para organizar las relaciones entre ellos.

El Patrón Modelo Vista Controlador es un patrón de arquitectura de las aplicaciones software que separa la lógica de negocio de la interfaz de usuario, facilitando el avance por separado de ambos aspectos, e incrementando la reutilización y la flexibilidad.

Para la creación del componente **feedback** se hará uso del patrón Modelo Vista Controlador (MVC Model View Controller) por sus siglas en inglés. Básicamente se trata de dividir el programa en tres módulos que se relacionan entre ellos de una manera muy concreta:

- ✓ El modelo: Administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).
- ✓ La vista: implementa una interfaz predefinida para la aplicación y configura a quien le llegan los eventos que se producen sobre sus elementos.
- ✓ El controlador: Controla el flujo entre la vista y el modelo (los datos).

Tanto la vista como el controlador dependen del modelo, el cual no depende de las otras clases. Esta separación permite construir y probar el modelo, independientemente de la representación visual. Ver Fig 3.1.

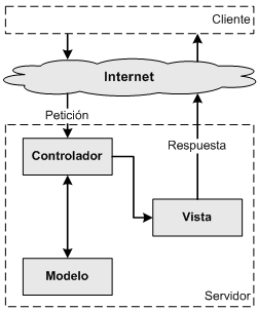


Fig 3.1: Patrón Modelo Vista Controlador.

Este patrón consta de tres etapas en su ciclo de vida. La primera se inicia cuando el usuario realiza una petición a la aplicación, el controlador recibe la petición del cliente y decide a quien debe encargar esta tarea.

El modelo es el encargado de realizar las operaciones necesarias sobre la información que maneja para cumplir con las peticiones del controlador. Una vez realizadas las operaciones

necesarias, devuelve al controlador la información resultante, y este las muestra en las vistas. Entre todas las ventajas que ofrece la aplicación de este patrón arquitectónico se encuentran las siguientes:

- ✓ Comprender el sistema.
- ✓ Organizar el desarrollo.
- ✓ Fomentar la reutilización.
- ✓ Hacer evolucionar el sistema.

3.2. Implementación del MVC de Symfony.

Symfony toma lo mejor de la arquitectura MVC y lo implementa de forma tal que la creación de aplicaciones sea lo más sencilla posible. El framework Symfony crea un controlador frontal y un **layout** comunes para cada aplicación. El controlador frontal es el único punto de entrada de la aplicación, es el encargado de cargar la configuración y determina la acción que debe ejecutarse.

Las clases necesarias para acceder al modelo de datos de cada aplicación son generadas por el framework también, por defecto la librería que se encarga de esta tarea es la Propel, aunque pueden utilizarse otras, como Doctrine. Symfony garantiza una abstracción a la base de datos, pues se realiza de forma nativa mediante PDO (PHP Data Object), esto posibilita que si se cambia el sistema gestor de bases de datos, no es necesario cambiar ni una sola línea de código.

Para la parte de la vista, el framework Symfony cuenta con plantillas para cada una de las acciones, además de componentes y slots, los que facilitan el desarrollo de una vista agradable y por pequeños pedazos, mejorando el mantenimiento; además de la corrección de errores. (22)

3.2.1. Patrones de diseño.

Un patrón es una unidad de información nombrada, instructiva e intuitiva que agrupa exitosas soluciones probadas a un problema recurrente dentro de un cierto contexto. El objetivo de los

patrones es crear un lenguaje común para comunicar experiencia sobre los problemas y sus soluciones. Muchos autores han dado conceptos acerca de que es un patrón. (23)

Un buen patrón debe contar con las siguientes partes:

- ✓ Nombre del patrón.
- ✓ Clasificación del patrón.
- ✓ Intención.
- ✓ Implementación.
- ✓ Código de ejemplo.
- ✓ Usos conocidos.
- ✓ Patrones relacionados.

Según Grady Booch "Una arquitectura orientada a objetos bien estructurada está llena de patrones. La calidad de un sistema orientado a objetos se mide por la atención que los diseñadores han prestado a las colaboraciones entre sus objetos. Los patrones conducen a arquitecturas más pequeñas, más simples y más comprensibles".

En la creación del componente feedback se hizo uso de los siguientes patrones GRASP:

- ✓ Creador
- ✓ Alta Cohesión
- ✓ Bajo Acoplamiento
- ✓ Experto

3.2.1.1 PATRÓN CREADOR.

El patrón **Creador** pertenece al conjunto de patrones GRASP. Guía la asignación de responsabilidades relacionadas con la creación de objetos. El propósito fundamental de este patrón es encontrar un creador que se debe conectar con el objeto producido en cualquier evento.

El empleo del mencionado patrón se evidencia cuando se crean objetos de las clases del modelo. En la Fig. 3.2 puede observarse un fragmento de código que muestra la utilización de este patrón, donde es la clase **feedback** la encargada de crear las instancias de **SugerenciaCliente** y **Notificación**.

```
public static function saveSferror($f, $u, $t, $d, $c)
{
  $not = new Notificacion();
  $not->setContenido($d);
  $not->setFecha($f);
  $not->setUsuario($u);
  $not->save();

  $sf = new ErrorSf();
  $sf->setTipo($t);
  $sf->setClase($c);
  $sf->setNotificacionId($not->getId());
  $sf->save();
}
```

Fig 3.2: Ejemplo del uso del patrón Creador.

3.2.1.2 PATRÓN ALTA COHESIÓN

La cohesión define la fuerza con la que interactúan los elementos de un sistema. Cada elemento del diseño debe realizar una labor única dentro del sistema, no desempeñada por el resto de los elementos y auto-identificable.

La clase *actions* tiene la responsabilidad de definir las acciones para las plantillas y colabora con otras para realizar diferentes operaciones e instanciar objetos, entre otras. Proporciona diferentes funcionalidades que se encuentran estrechamente relacionadas, garantizando que el software sea flexible frente a grandes cambios. Ver Fig. 3.3.

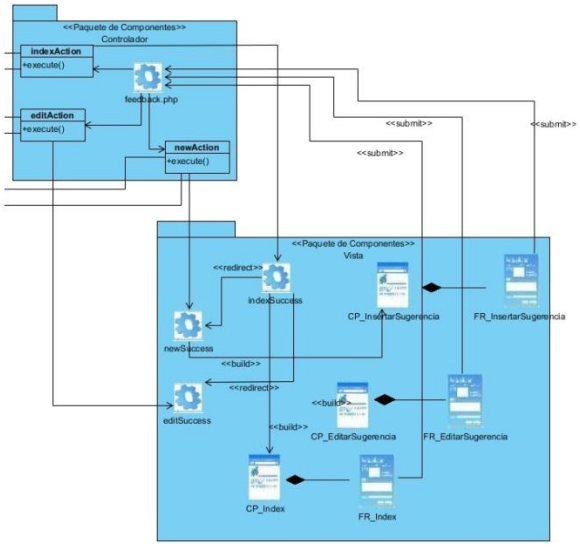


Fig. 3.3 Ejemplo del uso del patrón Alta Cohesión

3.2.1.3 PATRÓN BAJO ACOPLAMIENTO.

Debe existir pocas dependencias entre las clases, logrando que estén lo menos posible relacionadas entre sí, de forma tal que si se produce algún cambio en alguna clase, afecte lo menos posible al resto.

Un ejemplo de este patrón se puede apreciar en las clases controladoras: las acciones. Estas clases heredan solamente de sfAction para lograr un bajo acoplamiento de clases. Ver Fig. 3.4.

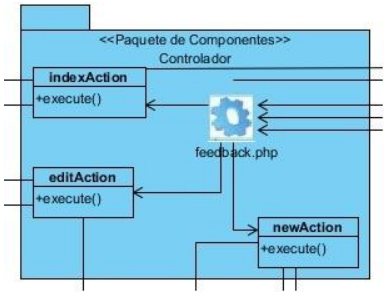


Fig. 3.4 Ejemplo del uso del patrón bajo acoplamiento.

3.2.1.4 PATRÓN EXPERTO.

El patrón GRASP experto en información es el principio básico de asignación de responsabilidades. Consiste en asignar la responsabilidad de creación de un objeto o la implementación de un método al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad.

El framework Symfony utiliza Propel como ORM (Object-Relational mapping) para lograr una capa de abstracción en el modelo. Encapsula toda la lógica de los datos y son generadas las clases con todas las funcionalidades comunes de las entidades.

Se generan 4 clases por cada tabla en la base de datos. Para la tabla sugerencia_cliente, por ejemplo, se generan las clases SugerenciaCliente, SugerenciaClientePeer, BaseSugerenciaCliente y BaseSugerenciaClientePeer, donde las dos últimas son las encargadas del trabajo con la base de datos, pues contienen los métodos y atributos necesarios para la realizar dicha función. En la figura 3.5 se muestra la forma en que estas clases manipulan los datos, a través de un fragmento de código.

```
public static function saveSugerenciaCliente( sfParameterHolder $values){

 $user = sfContext::getInstance()->getUser();

 $not = new Notificacion();
 $not->setContenido( $values->get("descripcion"));
 $not->setUsuario( ($user->hasAttribute("nombreUsuario"))?$user->getAttribute("nombreUsuario"):"user" );
 $not->setFecha(date("c"));
 $not->save();

 $sc = new SugerenciaCliente();
 $sc->setNotificacionId($not->getId());
 $sc->setEstado("Pendiente");
 $sc->setPrioridad($values->get("prioridad"));
 $sc->setUrl("http://".$_SERVER["SERVER_NAME"].$_SERVER["REQUEST_URI"]);

 $sc->save();
}
```

Ver Fig. 3.5. Ejemplo del uso del patrón experto.

3.2.1.5 PATRÓN CONTROLADOR.

La función de este patrón es asignar la responsabilidad de contralar el flujo de eventos del sistema, a clases específicas. Es el intermediario entre la interfaz y las clases que la implementan

por lo que delega la responsabilidad de realizar operaciones del sistema a otras clases, como respuestas a eventos. Este patrón es muy común cuando se utiliza el framework Symfony. El propio framework crea un controlador frontal encargado de manejar todas las peticiones web siendo el único punto de entrada de la aplicación, quedando en evidencia el uso de este patrón cuando se tiene una acción para cada template, encargándose esta de controlar todo un único proceso ya sea de eliminación, inserción o búsqueda de procesos. Ver Fig. 3.6.

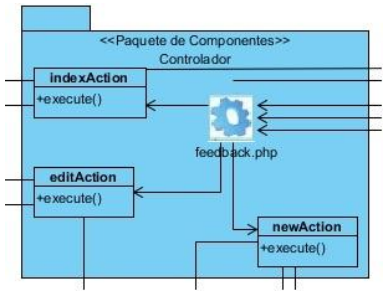


Fig. 3.6 Ejemplo del uso del patrón controlador.

3.3. Modelo del diseño.

El modelo de diseño es un modelo de objetos que describe la realización de los casos de uso centrándose en como los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar. Además, el modelo de diseño sirve de abstracción de la implementación del sistema y es, de ese modo, utilizado como una entrada fundamental de las actividades de implementación. (24)

3.3.1. Diagrama de clases del diseño.

Lo que se debe implementar es representado en los diagramas de clases del diseño. Estos diagramas representan la parte estática del sistema a través de las clases y sus relaciones.

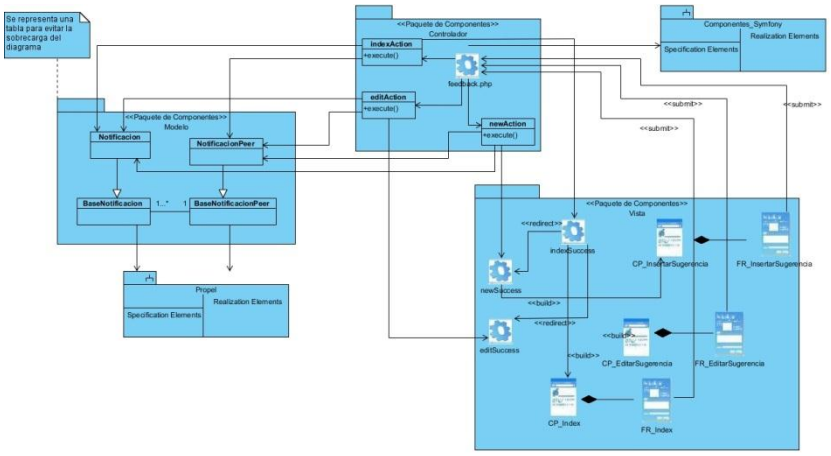


Fig 3.7: Diagrama de clases del diseño.

3.3.2. Diagramas de secuencia.

Un diagrama de secuencia muestra un conjunto de mensajes, dispuestos en una secuencia temporal. Se usa para mostrar la secuencia del comportamiento de un caso de uso y la interacción de un conjunto de objetos a través del tiempo, estos representan con más claridad el flujo de las acciones que debe realizar el sistema. Como resultado del diseño se obtuvo un diagrama de secuencia por cada escenario de los cuatro casos de uso. A continuación se muestran cuatro diagramas de secuencia, siendo estos los más significativos. El resto de los diagramas se encuentran en el documento complementario.

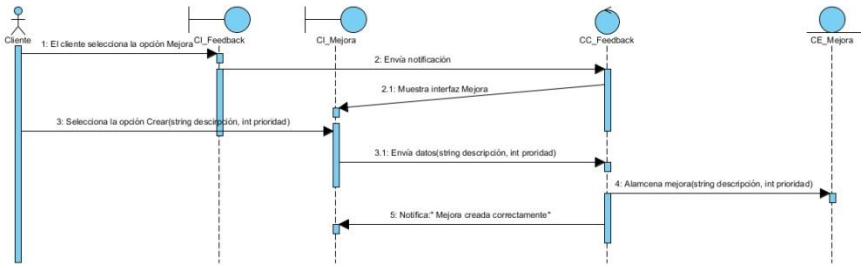


Fig. 3.7 Diagrama de secuencia: CU Gestionar sugerencia (Escenario: Crear sugerencia).

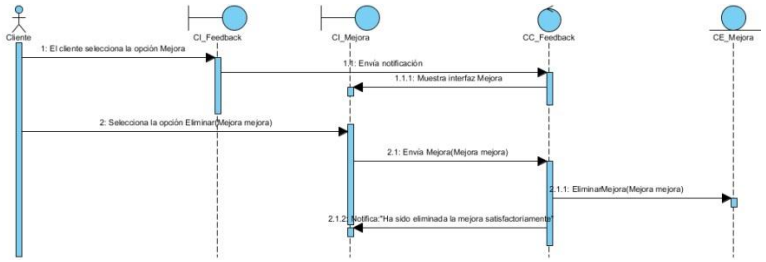


Fig. 3.8 Diagrama de secuencia: CU Gestionar sugerencia (Escenario: Eliminar sugerencia).

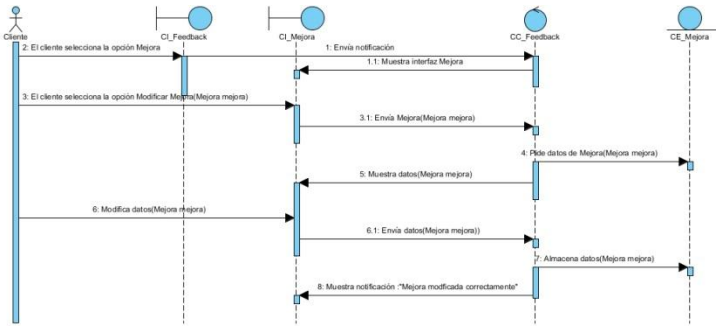


Fig. 3.9 Diagrama de secuencia: CU Gestionar sugerencia (Escenario: Modificar sugerencia).

3.4. Diseño de la base de datos.

El diseño de la base de datos es uno de los diagramas UML que no se debe pasar por alto debido a la importancia que se le atribuye dentro del desarrollo de cualquier sistema de gestión. El modelo de datos describe la representación de los datos en un sistema de gestión de una manera más abstracta. En este modelo se especifican las variables, los tipos de datos y la organización y relación que tendrán dentro del sistema. Este modelo sirve como guía para el diseño de la base de datos.

3.4.1. Diagrama de clases persistentes.

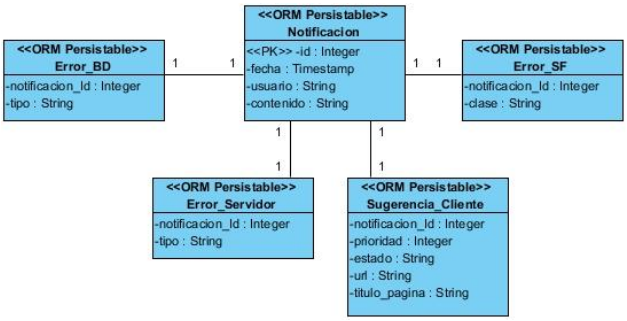


Fig. 3.10 Diagramas de clases persistentes.

3.4.2. Modelo de datos.

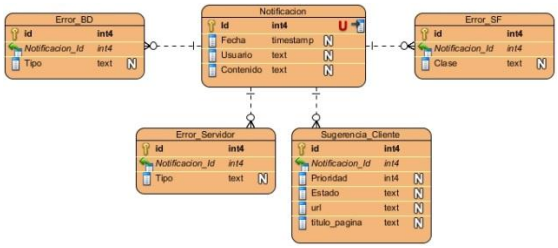


Fig. 3.11 Diagrama Entidad-Relación

3.5. Diagrama de despliegue.

Un diagrama de despliegue muestra la configuración de los elementos de procesamiento en tiempo de ejecución y los componentes de software y hardware, procesos y objetos que los ejecutan. La sintaxis de un diagrama de despliegue es un grafo de nodos unidos por conexiones de comunicación. Un nodo puede contener instancias de componentes software, objetos o procesos. Las instancias de componentes software pueden estar unidas por relaciones de dependencia, posiblemente a interfaces.

El diagrama de despliegue elaborado en esta investigación consta de tres nodos: el servidor de aplicaciones, el microprocesador que servirá de servidor de base de datos y la terminal de trabajo de los clientes. El protocolo de comunicación existente entre los servidores es TCP/IP, y entre la PC_Cliente y el servidor de aplicaciones es HTTP.

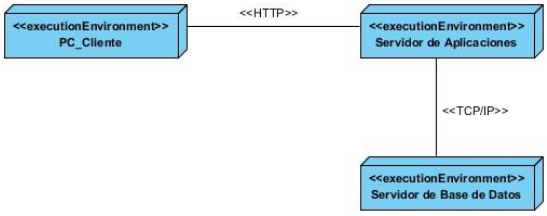


Fig. 3.12 Diagrama de despliegue.

3.6. Conclusiones parciales.

Con la culminación de este capítulo quedó elaborado el diseño del componente feedback para el SIPP. La aplicación de los patrones Modelo-Vista-Controlador (MVC) y demás patrones descritos anteriormente, facilitaron el diseño de las clases. Llegando a alcanzar una arquitectura estable y sólida, así como clases mejor diseñadas, modulares, flexibles y reutilizables. Se realizó un diagrama de secuencia por cada escenario de los casos de uso, el modelo de datos, el diagrama de despliegue del sistema y el diagrama de clases persistentes.

CAPÍTULO 4. IMPLEMENTACIÓN Y VALIDACIÓN DE LOS RESULTADOS.

En el presente capítulo se describirá la implementación del componente a partir del diagrama de componentes. También se presentarán varias pantallas de la aplicación donde se muestra el sistema dando cumplimiento a cada uno de los requisitos funcionales. Además se listarán las pruebas que se le realizaron al sistema para verificar su integridad y ajuste a los requerimientos planteados por el cliente.

4.1. Modelo de implementación.

En la etapa de implementación se comienza con el resultado de la etapa de diseño y se implementa el sistema en términos de componentes, es decir, ficheros de código fuente, ficheros de código binario, scripts, ejecutables y similares.

El objetivo principal de la etapa de implementación es desarrollar la arquitectura y el sistema como un todo. De forma más específica, los propósitos de la implementación son:

- ✓ Definir la organización del código.
- ✓ Planificar las integraciones del sistema necesarias en cada iteración.
- ✓ Implementar las clases y subsistemas encontrados durante el diseño.

El artefacto fundamental que se desarrolla en esta etapa es:

- ✓ Modelo de implementación, que incluye componentes, subsistemas y el producto.

4.1.1. Diagrama de componentes.

Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones. Los componentes representan todos los tipos de elementos software que entran en la fabricación de aplicaciones informáticas. Se definen cinco estereotipos estándar que se aplican a los componentes. Estos son:

- ✓ **Ejecutable:** componente que se puede ejecutar en un nodo.

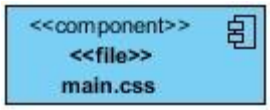
- ✓ **Biblioteca:** biblioteca de objetos estática o dinámica.
- ✓ **Tabla:** componente que representa una tabla de una base de datos.
- ✓ **Archivo:** componente que representa un documento que contiene código fuente o datos.
- ✓ **Documento:** componente que representa un documento.

Los componentes tienen las siguientes características:

- ✓ Tienen relaciones de traza con los elementos del modelo de implementación.
- ✓ Es normal que un componente implemente varios elementos como por ejemplo varias clases.
- ✓ Los componentes proporcionan las mismas interfaces que los elementos del modelo que implementan. (25)

4.1.1.1 DESCRIPCIÓN GENERAL DE LOS COMPONENTES.

JavaScript: Es el fichero donde se implementan las validaciones del lado del cliente, además de las funciones para la creación de campos dinámicos.

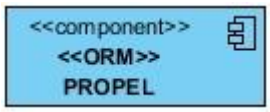


CSS: en estos ficheros es donde se le da formato al diseño de las interfaces. Los css son importantes para lograr una vista agradable y una uniformidad en la aplicación.

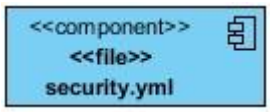
ActionClass: Son las clases controladoras donde está contenida toda la lógica de la aplicación. Estas son las encargadas de interactuar con el modelo y a la vez actualizar las vistas.

Success: Son las vistas de la aplicación que están encargadas de interactuar con el usuario, en los Success se muestra el resultado lógico de la aplicación de una forma comprensible por el usuario.

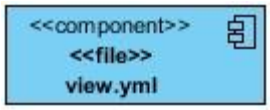
Model Class: Son las clases generadas por el ORM Propel. Estas contienen la lógica del negocio y el acceso a los datos (Peer), lo cual posibilita que el trabajo de consultas a la base de datos se haga de forma más rápida y sencilla.



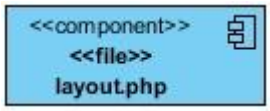
Propel: Symfony utiliza Propel como ORM y Propel utiliza Creole como capa de abstracción de base de datos.



Security Class: Son las clases que utiliza Symfony para restringir el acceso a una acción determinada. En esta clase se especifican los requerimientos de seguridad que el usuario debe cumplir para acceder a una acción o para todas.



View: En este fichero se establece la estructura de la vista por defecto, el nombre del layout, el título de la página, las hojas de estilo y los archivos JavaScript que se incluyen.



Layout: Contiene los elementos que son comunes para toda la aplicación o para parte de ella como es la cabecera y pie de página, normalmente son globales en toda la aplicación o en un gran número de páginas.

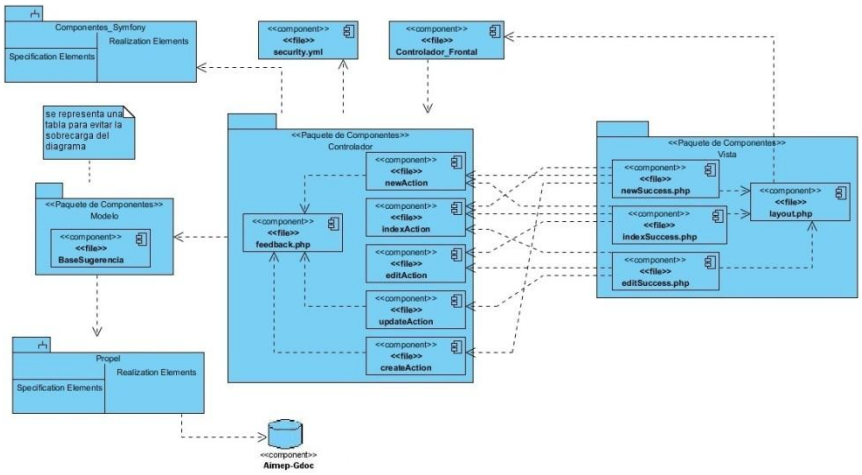


Fig 4.1 Diagrama de Componentes CU Consultar Notificaciones.

4.2. Segmentos principales del código fuente.

4.2.1. Segmento de código de la clase SugerenciaCliente.php perteneciente al modelo.

La clase **SugerenciaCliente.php** es creada por el ORM Propel para trabajar con la tabla de la base de datos, sugerencia_cliente. La misma hereda de la clase **BaseSugerenciCliente.php,** que es la que cuenta con los métodos y atributos que permiten la interacción con la base de datos sin necesidad de tener conocimientos básicos sobre ningún lenguaje SQL. Para darle cumplimiento a una de las reglas del negocio establecidas, de no poder eliminar una sugerencia que su estado sea “En Trámite”, es necesario redefinir el método **delete (),** como se muestra en el siguiente fragmento de código. Ver Fig. 4.3.

```
class SugerenciaCliente extends BaseSugerenciaCliente
{
  public function delete(PropelPDO $con = null)
  {
    if(($this->getEstado() != "En trámite") && ($this->getNotificacion()->getUsuario() == sfContext::getInstance()->getUser()))
    {
      try
      {
        parent::delete();
      }
      catch (Exception $exc)
      {
        throw new Exception(sfContext::getInstance()->getI18N()->__('CAN_NOT_DELETE_SUGGESTION_WITH_INSTANCE'));
      }
    }
    else
    {
      throw new Exception(sfContext::getInstance()->getI18N()->__('CAN_NOT_DELETE_ACTIVE_SUGGESTION'));
    }
  }
}
```

Fig. 4.3 Método delete de la clase SugerenciaCliente.php

4.2.2. Segmentos de código de la clase feedback.php.

Segmentos de código que muestran el trabajo con los ficheros, la manera en que se abren y cierran los ficheros y directorios (ver Fig. 4.4), la forma en que se guarda la última línea analizada de cada fichero (ver Fig. 4.5), así como el trabajo que se le hace a la información contenida dentro. Ver Fig. 4.6.

```
$finder = new sfFinder();
$fcomp = fopen(sfConfig::get("app_dir_postgres_line"),'r') or die('Error');
$lcomp = fgets($fcomp);
fclose($fcomp);
$files = $finder::type('any')->name('*.log')->in(sfConfig::get('app_dir_postgres_log'));
foreach($files as $file)
{
  if(strstr($file,$lcomp))
  {
    $lcomp = "postgresql";
    $fecha = explode('-', $files[$pos]);
    $d = explode('_', $fecha[3]);
    $fecha_hora = $fecha[1].'-'.$fecha[2].'-'.$d[0].' '.$d[1][0].$d[1][1].':'.$d[1][2].$d[1][3].':'.$d[1][4].$d[1][5];
    $fopen = fopen($file, 'r') or die('Error');
    while (!feof($fopen))
    {
      $linea = fgets($fopen);
      $partes = array();
      if(strstr($linea,"ERROR") || strstr($linea,"FATAL") || strstr($linea,"PANIC"))
      {
        $partes = explode(" ",$linea,4);
        if (($partes[0] === "%t")) //%t FATAL:  autovacuum launcher started
        {
          $tipo = explode(':', $partes[1]);
          $not = new Notificacion();
          $not->setContenido($partes[3]);
        }
        else  //FATAL:  could not reattach to shared memory (key=124, addr=02A10000): 487
        {
          $tipo = explode(':', $partes[0]);
          $not = new Notificacion();
          $not->setContenido($partes[2]." ".$partes[3]);
        }
        $not->setFecha($fecha_hora);
        $not->setUsuario((sfContext::getInstance()->getUser()->hasAttribute("nombreUsuario"))?sfContext::getInstance()->getUser()->getAttribute("nor

        $bd = new ErrorBd();
        $bd->setNotificacionId($not->getId());
```

Fig. 4.4 Manera en que se abre cada fichero del directorio del servidor de base de datos.

```
unlink(sfConfig::get("app_dir_apache_line"));
$escribir = explode(" ", $ultimalinea);
$doc = fopen(sfConfig::get("app_dir_apache_line"), "a+") or die('Error');
fputs($doc, $escribir[0]." ".$escribir[1]." ".$escribir[2]." ".$escribir[3]." ".$escribir[4]);
fclose($doc);
```

Fig. 4.5 Manera en que se guarda la última línea analizada del fichero de apache

```
if(strstr($linea,"[error]"))
{
  $partes = explode("] ",$linea,4);
  $fecha = explode(" ", $partes[0]);
  $tipo = explode("[",$partes[1]);

  $not = new Notificacion();
  $not->setContenido($partes[3]);
  $not->setUsuario(($user->hasAttribute("nombreUsuario"))?$user->getAttribute("nombreUsuario"):"user" );
  $not->setFecha($fecha[4]."-".$mes[''.$fecha[1]]."-".$fecha[2]." ".$fecha[3]);

  $sw = new ErrorServidor();
  $sw->setNotificacionId($not->getId());
  $sw->setTipo($tipo[1]);
  $error[] = array($not,$sw);
}
```

Fig. 4.6 Manera con que se trabajan los datos del fichero de apache.

4.3. Pantallas principales de la aplicación.

4.3.1. Pantalla principal del componente.

El componente consta con una interfaz principal la cual gestiona las mejoras o sugerencias. Estas mejoras son listadas en una tabla y en la parte superior se ubican unos links que representan algunas de las distintas acciones que se pueden realizar en esta página como son: Ir a las pantallas principales de las áreas de los errores de la base de datos, Symfony y Apache. En la parte lateral de cada sugerencia se ubican botones que representan otras acciones que se pueden realizar desde esta página como son: eliminar o editar sugerencias existentes

Fig 4.7 Pantalla principal

A continuación se describen cada una de las funcionalidades que presenta la herramienta desarrollada.

- ✓ **Insertar sugerencia.** Dentro de las operaciones que puede realizar el cliente en el interacción con la aplicación esta la inserción de una nueva sugerencia. Para ello se necesita que se llenen los datos requeridos. En la figura 4.8 se representa dicha operación.

Fig. 4.8 Insertar

- ✓ **Editar sugerencia.** A la hora de editar una sugerencia se muestran los mismos campos que se muestran cuando se va a insertar una sugerencia pero esta vez con los valores de la sugerencia a editar en cada campo correspondiente. En la figura 4.9 se representa dicha operación.

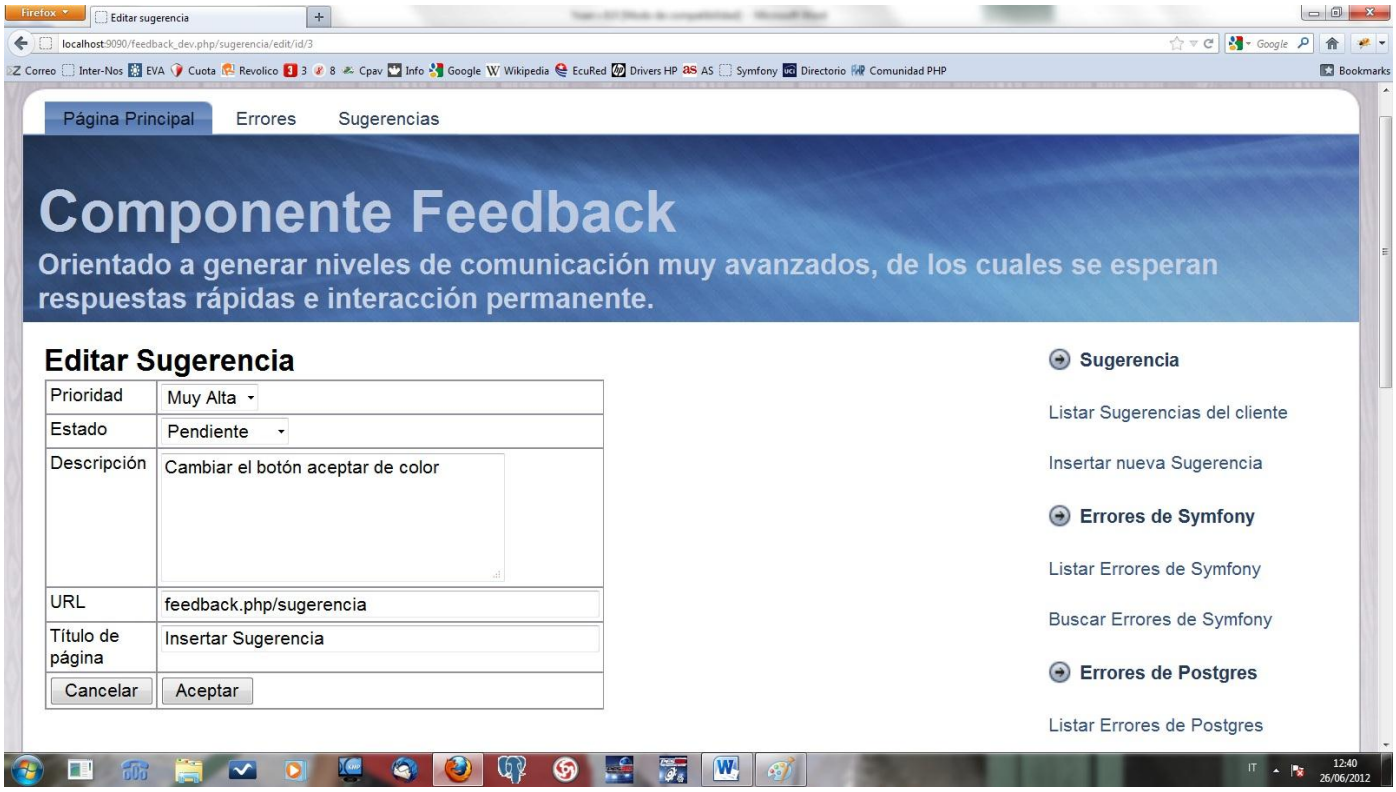


Fig. 4.9 Editar

- ✓ **Eliminar sugerencia.** Al eliminar el sistema muestra un dialogo para que el cliente confirme que desea realizar la acción de eliminar una sugerencia. Ver Fig 4.10

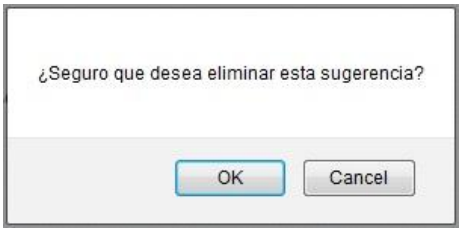


Fig 4.10: Mensaje de notificación de eliminar sugerencia.

4.3.2. Pantalla principal del área de Symfony.

En esta pantalla se mostrarán los errores de Symfony existentes en la base de datos, listados en una tabla. En la parte superior se encuentran unos links que representan las distintas acciones que se pueden realizar en esta página como son: ir hacia la pantalla principal del componente,

del servidor web, del servidor de base de datos, así como buscar más errores de Symfony en el fichero donde se almacenan.

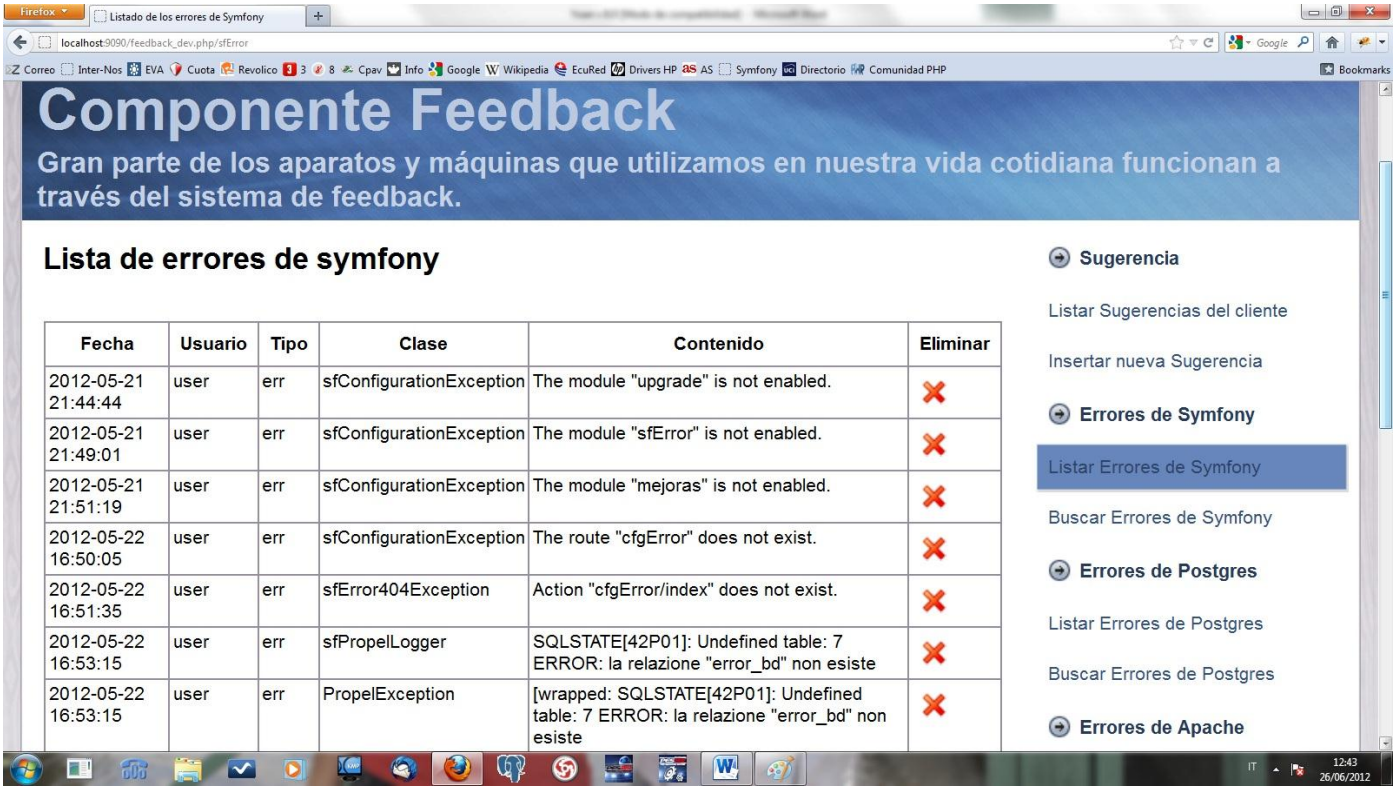


Fig 4.11 Pantalla principal del área de Symfony.

4.3.3. Pantalla principal del área del servidor de base de datos.

En esta pantalla se mostrarán los errores de la base de datos existentes en la base de datos, listados en una tabla. En la parte superior se encuentran unos links que representan las distintas acciones que se pueden realizar en esta página como son: ir hacia la pantalla principal del componente, del servidor web, de Symfony, así como buscar más errores de base de datos en el directorio donde se almacenan.

Fig 4.12 pantalla principal del área del servidor de base de datos.

4.3.4. Pantalla principal del área del servidor web.

En esta pantalla se mostrarán los errores del servidor web existentes en la base de datos, listados en una tabla. En la parte superior se encuentran unos links que representan las distintas acciones que se pueden realizar en esta página como son: ir hacia la pantalla principal del componente, del servidor de base de datos, de Symfony, así como buscar más errores de base de datos en el fichero donde se almacenan.

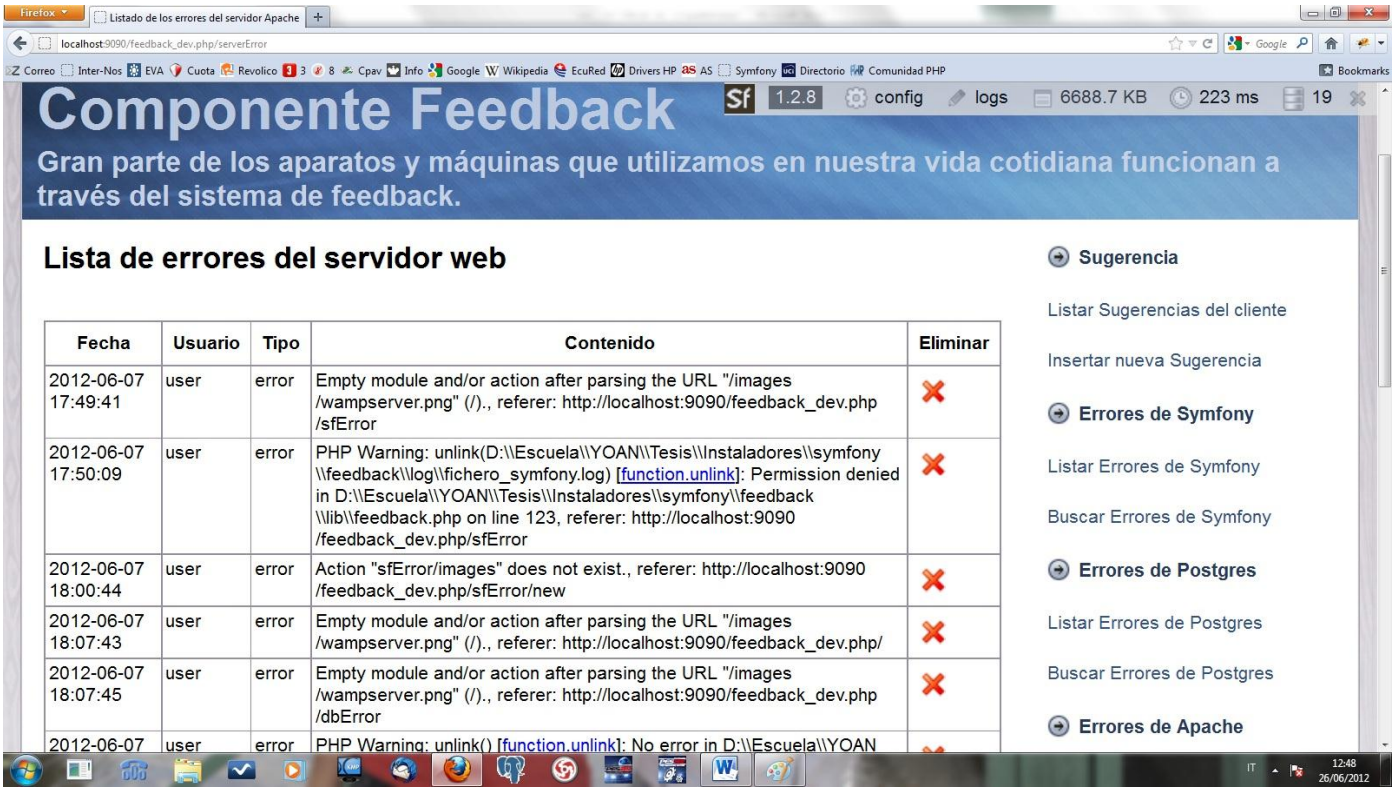


Fig 4.13 pantalla principal del área del servidor web.

4.4. Modelo de prueba.

Las pruebas del software constituyen un elemento crítico para la garantía de la calidad del software y representan una revisión final de las especificaciones, del diseño y de la codificación. Son los procesos de ejecución de un programa, con la intención de descubrir un error, teniendo éxito, si se descubre un error no detectado hasta entonces; por lo que un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.

Las pruebas que se llevan a cabo sobre la interfaz del software, son las de Caja Negra, también denominadas prueba de comportamiento. Cuando las pruebas aseguran que la operación interna se ajusta a las especificaciones y que todos los componentes internos se han comprobado de forma adecuada se hace referencia a las pruebas de Caja Blanca, denominada a veces prueba de caja de cristal.

4.4.1. Pruebas de integridad de la base de datos y de los datos.

Objetivos de la prueba	Comprobar que los procedimientos y métodos de acceso a la base de datos funcionan correctamente.
Técnicas	Invocar cada procedimiento o método de acceso a la base de datos con datos validos e inválidos. Inspeccionar la base de datos para asegurar que los datos son los previstos, todos los eventos de la base de datos ocurren adecuadamente, o revisar los valores devueltos para asegurar que la recuperación de datos es correcta.
Criterios de finalización	Todos los procedimientos y métodos de acceso funcionan como se diseñaron y sin ningún error en los datos.
Consideraciones	Ninguna.

4.4.2. Pruebas de funcionalidad.

Las pruebas de funcionalidad se deberían centrar en cualquier requisito que pueda ser trazado directamente de los casos de uso y reglas de negocio. El objetivo de estas pruebas es verificar la aceptación, procesamiento y recuperación de datos y la adecuada implementación de las reglas de negocio.

Objetivos de la prueba	Asegurar la navegación correcta de la aplicación, la entrada de datos, su procesamiento y recuperación.
Técnicas	Ejecutar cada caso de uso y flujo del caso de uso con datos válidos e inválidos para verificar lo siguiente: Cuando se utilizan datos correctos se obtienen los resultados esperados. Cuando se utilizan datos incorrectos se obtienen los mensajes de error o advertencias adecuadas. Cada regla de negocio se ha aplicado correctamente.
Criterios de finalización	Todas las pruebas planificadas se han ejecutado. Todos los defectos identificados se han considerado.
Consideraciones	Invocar cada método de acceso a la base de datos con datos válidos e inválidos. Inspeccionarla para asegurar que los datos son los previstos, o revisar los valores devueltos para asegurar que la recuperación de datos es correcta.

4.4.3. Pruebas de interfaz de usuario.

Las pruebas de interfaz de usuario verifican la interacción del usuario con el sistema software. El objetivo de esta prueba es asegurar que la interfaz de usuario permite al usuario acceder y navegar a través de toda la funcionalidad de la aplicación. Además, la prueba de interfaz de usuario garantiza que las interfaces de usuario cumplen los estándares.

Objetivos de la prueba	Verificar los siguientes objetivos: La navegación a través de la aplicación refleja adecuadamente las reglas de negocio y los requisitos incluyendo ventana a ventana, campo a campo y métodos de acceso (tabulador, movimientos del ratón y teclas de función). Las ventanas y sus características, como menús, tamaño, posición y estado cumplen los estándares.
Técnicas	Crear o modificar pruebas para cada ventana con el objetivo de verificar la correcta navegación y su estado.
Criterios de finalización	Cada ventana se ha verificado con éxito y es consistente con la referencia o con los estándares utilizados.
Consideraciones	Ninguna.

4.4.4. Pruebas de desarrollo.

Las pruebas de desarrollo miden tiempos de respuesta, índices de transacción y otros requisitos susceptibles al tiempo. El objetivo de estas pruebas es verificar y validar que los requisitos de rendimiento se han alcanzado.

Las pruebas de desarrollo normalmente se ejecutan varias veces usando cada vez un cargo de trabajo diferente. La prueba inicial se debería realizar con una carga normal y la segunda prueba con una carga extrema.

Objetivos de la prueba	Validar el tiempo de respuesta del sistema software para las transacciones diseñadas o funciones de negocio bajo las condiciones siguientes: Volumen de trabajo normal. El peor volumen de trabajo.
Técnicas	Usar los procedimientos de prueba definidas para las pruebas de funcionalidad. Modificar los ficheros de datos para incrementar el número de transacciones.
Criterios de finalización	Se han completado las pruebas sin ningún error y dentro de los tiempos de respuesta esperados.
Consideraciones	Ninguna.

4.5. Conclusiones parciales.

Al culminar el presente capítulo se le ha dado cumplimiento a la última tarea de la investigación trazada para el desarrollo de la aplicación, quedando totalmente implementado el componente para mejorar la comunicación entre el cliente y el equipo de desarrollo.

CONCLUSIONES

Con la realización de este trabajo se implementó una herramienta web en forma de plugins, que permite al cliente plantear sugerencias o mejoras al equipo de desarrolladores, con el fin de lograr una aplicación más cercana a las preferencias de los que la explotan, lográndose así una retroalimentación del equipo de desarrollo por parte del cliente. Permite también a los clientes editar o eliminar sugerencias o mejoras ya existentes. De esta manera se le dio cumplimiento al objetivo planteado al inicio de la investigación y se resolvió el problema descrito en la misma, además:

- ✓ Se realizó el diseño de un componente feedback, haciendo un correcto uso de los patrones de diseño.
- ✓ Permite al equipo de desarrollo, conocer el comportamiento de las herramientas con las que trabaja el cliente, mediante la búsqueda de errores o anomalías que presentan dichas herramientas.
- ✓ Se definió un modelo relacional de base de datos para la gestión de las sugerencias emitidas por los clientes y de los errores arrojados por las aplicaciones.

RECOMENDACIONES

Teniendo en cuenta la investigación realizada y los resultados obtenidos se recomienda que el módulo desarrollado sea integrado a la herramienta Maximus Drill Pro.

Al módulo propuesto en el presente trabajo se le pueden añadir mejoras con el propósito de perfeccionar su usabilidad, de manera que facilite y optimice el trabajo a los que lo explotan.

Entre estas se pueden mencionar las siguientes:

- Otra forma de recibir feedback por parte del cliente como correo, encuestas, foros, chat.
- Exportar los datos de las notificaciones a archivos .doc, .pdf o .xls.
- Realizar una búsqueda filtrada antes de exportar o guardar las notificaciones en la base de datos.
- Crear un módulo de configuración visual para que mediante este se puedan definir las direcciones de los ficheros a analizar.

BIBLIOGRAFÍA

1. Curso TICs. [En línea] http://gisellecepatics.blogspot.com/2008/10/significado-de-tics.html.
2. **Betancourt Rodríguez, Keyly y Rodríguez Martell, Frank.** *Diseño de pruebas de calidad para producto LIMS control de calidad del CIGB.* Ciudad de la Habana: Universidad de las Ciencias Informaticas : s.n., 2007.
3. *Diccionario Enciclopedico .* s.l. : Grijalbo, 1998.
4. **Ariza Rojas, Maribel y Molina García, Juan Carlos.** *INTRODUCCIÓN Y PRINCIPIOS BÁSICOS DEL DESARROLLO DE SOFTWARE BASADO EN COMPONENTES.* 2004.
5. http://www.bugzilla.org/. [En línea]
6. Mantis. [En línea] http://www.mantisbt.org/documentation.php.
7. Guía de Desarrollo Web. [En línea] http://www.guiaweb.gob.cl.
8. *El tratamiento de la información. La necesidad del feedback.* **Franco, Francisco Javier Fernandez.** Buenos Aires : s.n., 2002.
9. IDE. [En línea] http://foro.ignetwork.net/showthread.php?15188-IDE-Entorno-integrado-de-desarrollo-%28Concepto-importante%29.
10. **Mehdi Achour, Friedhelm Betz, Antony Dovga.** http://php.net/manual/es/index.php. [En línea] [Citado el: 11 de febrero de 2012.]
11. http://www.hooping.net/faq-caracteristicas.aspx. [En línea] [Citado el: 2 de marzo de 2012.]
12. *https://belenus.unirioja.es/~guprado/pagweb/caraccss.html.*
13. [En línea] http://www.ecured.cu/index.php/JavaScript.
14. http://javascriptexperts.blogspot.com/2009/05/javascript-objetos-dinamicos-closures.html. [En línea]
15. Framework. [En línea] http://es.answers.yahoo.com/question/index?qid=20070817175348AAgSFgE.
16. **Fabien Potencier, Francois Zaninotto.** *Symfony la guía definitiva.* 2007.
17. [En línea] http://www.ecured.cu/index.php/postgresql.
18. http://es.scribd.com/doc/52208534/29/CARACTERISTICAS-Y-VENTAJAS-DEL-APACHE. [En línea] 06 de febrero de 2012.
19. Metodologias de desarrollo de software. [En línea] http://www.google.com.cu/url?sa=t&rct=j&q=metodologias+de+desarrollo+de+software&source=web&cd=4&ved=0CEQQFjAD&url=http%3A%2F%2Fsolusoft-g11.googlecode.com%2Ffiles%2FMetodologias%2520de%2520desarrollo.pdf&ei=W6BXT__6GJHh0wGcl4S8Dw&usg=AFQjCNFZ9_K-b0YEM-tw.
20. [En línea] http://www.herramientas-case-proceso-desarrollo-software2.shtml.
21. **Francesch Díaz, Enrique y Joyanes Aguilar, Luis.** *MODELO DE ANÁLISIS DE REGLAS DE NEGOCIO PARA.* Madrid : s.n., 2007.
22. **Diaz, Omar Martinez y Garófalo Hernández, Daniel.** *Ambiente integrado para el modelado y ejecución de procesos de gestión de la información. Herramienta web para la modelación de los procesos.* La Habana : s.n., 2010.
23. **Fowler, Martin.** *Patterns of enterprise application architecture.*
24. **Jacobson, Ivar, Booch , Grady y Rumbaugh, James.** *El Proceso unificado de desarrollo de software.* Madrid : s.n., 2000.
25. http://www.dsi.uclm.es/asignaturas/42530/pdf/M2tema12.pdf. [En línea] [Citado el: 7 de febrero de 2012.]

GLOSARIO DE TÉRMINOS

A

Algoritmo: Es una lista que, dado un estado inicial y una entrada, propone pasos sucesivos para arribar a un estado final obteniendo una solución.

B

BSD: es la licencia de software otorgada principalmente para los sistemas Berkeley Software Distribution (BSD). Es una licencia permisiva como la licencia OpenSSL o la MIT License. La licencia BSD al contrario de la licencia GPL permite el uso del código fuente en software no libre.

C

Commit: sentencia SQL que finaliza la transacción de base de datos dentro de un gestor y pone visibles todos los cambios a otros usuarios.

Clase: Una clase es una construcción que se utiliza como un modelo (o plantilla) para crear objetos de ese tipo. El modelo describe el estado y el comportamiento que todos los objetos de la clase comparten.

D

Debugger: programa usados para probar y corregir los errores de una aplicación.

H

HTTP: Cada transacción de información realizada en la Web es realizada utilizando el protocolo HTTP, "HyperText Transfer Protocol" por sus siglas en inglés, o Protocolo de Transferencia de HyperTexto.

M

Módulo: Es una parte autónoma de un programa de ordenador.

T

TCP/IP: es un conjunto de protocolos. Siglas de “Protocolo de Control de Transmisión/Protocolo de Internet”.

P

PDO: PHP Data Object. Provee una capa de abstracción de acceso a bases de datos que permite al desarrollador abstraerse de la base de datos de una aplicación.

Propel: Es un ORM para PHP que facilita la labor de desarrollo de aplicaciones web, gracias a la capa que transforma el tratamiento de la BD mediante objetos, con la que se puede recuperar, insertar y modificar datos.