Universidad de las Ciencias Informáticas

Facultad 6

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

ARQUITECTURA DE SOFTWARE PARA EL SERVIDOR DE GESTIÓN DE POSTGRESQL

AUTORA:

Raiza Ramírez Ramos

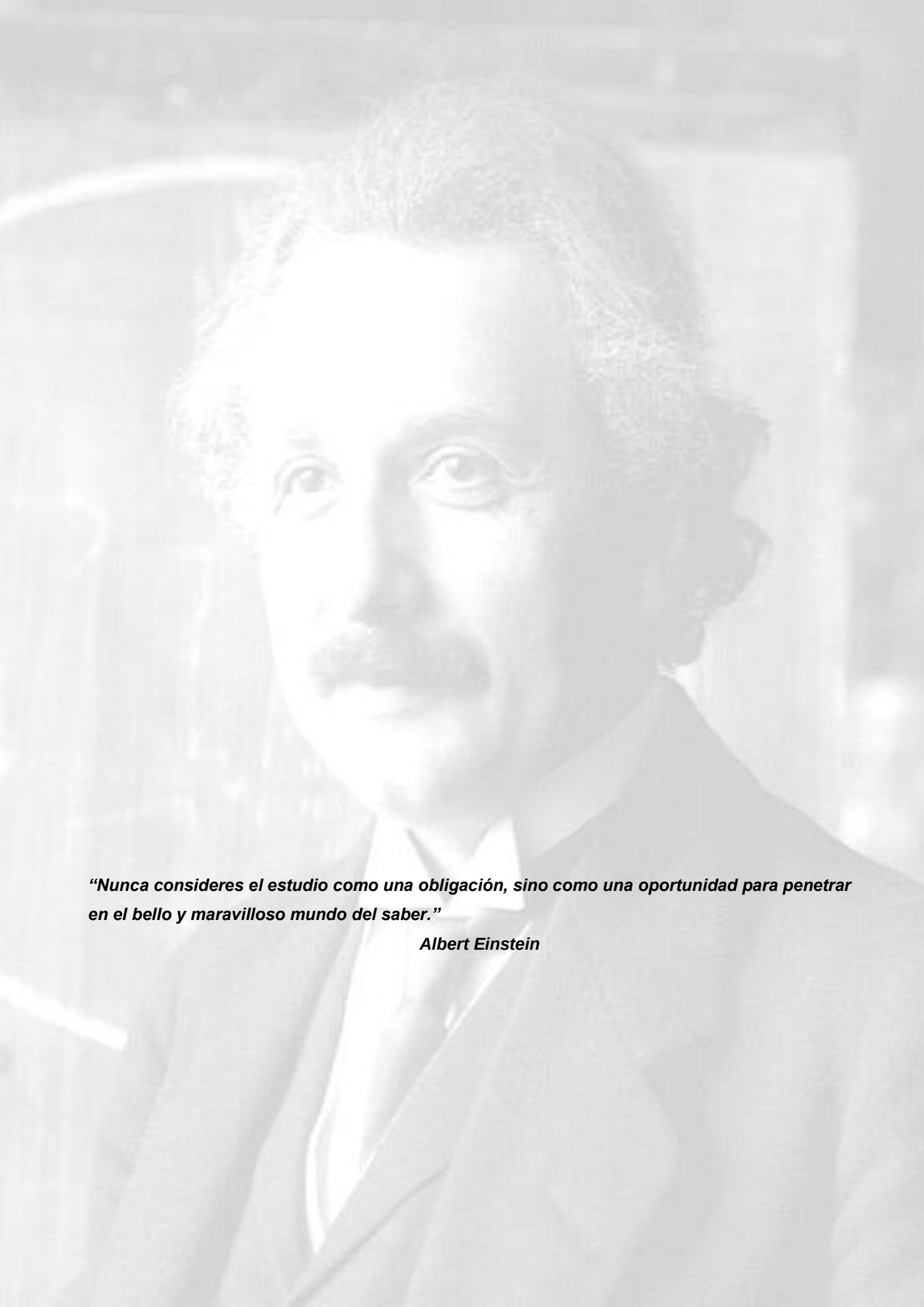
TUTOR:

Ing. Yunior Mesa Reyes

Ing. Glennis Tamayo Morales

La Habana, junio de 2011

"Año 53 de la Revolución"



“Nunca consideres el estudio como una obligación, sino como una oportunidad para penetrar en el bello y maravilloso mundo del saber.”

Albert Einstein

DECLARACIÓN DE AUTORÍA

Declaro ser autor del presente trabajo de diploma 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 _____.

Raiza Ramírez Ramos

_____

[Firma Autora]

Ing. Yunior Mesa Reyes

_____

[Firma Tutor]

Ing. Glennis Tamayo Morales

_____

[Firma Tutora]

DATOS DE CONTACTO

TUTOR:

Yunior Mesa Reyes: Ingeniero en Ciencias Informáticas

Correo: ymreyes@uci.cu

TUTORA:

Glennis Tamayo Morales: Ingeniera en Ciencias Informáticas

Correo: gtamayo@uci.cu

AGRADECIMIENTOS

A nuestro Comandante en Jefe Fidel Castro Ruz por ser el autor intelectual de la Universidad de las Ciencias Informáticas.

A mi mamá por ser la mejor madre del mundo entero, por la confianza que siempre ha depositado en mí, por su amor, su dedicación, preocupación y consejos constantes, que han sido la causa principal de mis esfuerzos para lograr ser lo que soy hoy ingeniera, para que siempre se sienta orgullosa de mí.

A mi hermana por ser mi ejemplo a seguir en todo lo que siempre me he propuesto en la vida, por todo el apoyo que me ha dado a lo largo de toda mi vida y en especial en mi tesis, ya que siempre ha sido mi tutora personal, además de su cariño que siempre me ha dado fuerzas para seguir adelante.

A mi papá por su cariño, preocupación, por guiarme y apoyarme durante estos años y confiar en que podía lograrlo.

A mi novio Edelquis por ser el mejor compañero que he tenido todo este tiempo, porque en los momentos que más lo he necesitado ha estado ahí para mí aunque sea desde lejos, por toda su ayuda en la realización de mi tesis y en especial por su amor que en este año ha sido de gran ayuda para mí.

A mi familia en general que se que me quieren mucho y nunca han dudado de mí.

A mi suegra, mi cuñado Lisdan, así como sus familias, que me apoyaron y siempre se han preocupado mucho por mí.

A mi amiga Mariam y los chicos UCI (Lester, Leiser, Arniel, Edgar) que a pesar de no conocernos de toda la vida han sido grandes amigos en todo momento.

A mis compañeros que me permitieron entrar en su vida durante estos 5 años de convivencia, en especial a mi grupo 6107 que significaron mucho para mí, ya que a pesar de habernos separado hemos seguido siendo un grupo como Maray, Vania, Jorge, Alejandro, Mario, Hector, Edelis, Aylenis, Bety Gago, Lara, Abel, entre otros, .

A mis compañeros Dariel, Maybel y Ele, ya que a pesar de haber compartido muy poco con ellos me han demostrado mucho cariño y preocupación.

A mi compañera de cuarto Katy, ya que ha sabido lidiar con mis cambios de humor que son tan frecuentes, además de soportarme y siempre estar ahí para ayudarme en lo que necesite, también por las charlas nocturnas que siempre me daba sobre su novio.

A mis tutores que siempre me han dado su apoyo incondicional, así como su ayuda en la realización de esta tesis.

Al tribunal que con sus sugerencias, preguntas y consejos me han ayudado a ganar confianza en mí misma para poder hoy ser la ingeniera que soy.

En especial a todos los presentes en este día tan especial para mí.

DEDICATORIA

Dedico este trabajo especialmente:

A mi mamá y a mi hermana por ser lo más importante en mi vida, ya que siempre han estado presentes apoyandome y queriéndome cuando lo he necesitado, a mi papá por quererme mucho, a mi novio por ayudarme y quererme tanto; a su familia por aceptarme con tanto cariño, especialmente a mi suegra; a Vicente por preocuparse todos estos años y no olvidarse nunca de mí a pesar de la distancia y a todos aquellos que con su apoyo me alentaron en el desarrollo de la Tesis.

RESUMEN

En el presente Trabajo de Diploma se realiza la Arquitectura del Servidor de Gestión para PostgreSQL Naire. El desarrollo de este sistema surge debido a que la administración de bases de datos es una tarea compleja, esto se debe a que en ocasiones a los administradores se les dificulta realizar esta actividad por el número de base de datos bajo su responsabilidad. El desarrollo de este sistema se realizará mediante la implementación de varios componentes, estos permitirán monitorear, planificar tareas y generar reportes de las base de datos.

Para el desarrollo de la investigación se realizó un estudio bibliográfico de los patrones, estilos de diseño, lenguajes de descripción de la arquitectura, herramientas, tecnologías, metodologías y métodos existentes. Se propuso el rol de arquitecto de software para la metodología XP donde se especifica las responsabilidades, actividades y artefactos para este. Se realizó la propuesta de arquitectura mediante las 4+1 vista arquitectónicas. Además, se realizó la evaluación de la arquitectura propuesta a través del método ARID y la técnica Basada en Escenarios obteniendo resultados satisfactorios.

PALABRAS CLAVES

Arquitectura de Software, Monitoreo, Servidor de Gestión, Planificador de Tareas, Reporte.

ÍNDICE DE CONTENIDO

INTRODUCCIÓN

La Arquitectura de Software es una práctica de apenas unos pocos años de trabajo constante, sin embargo, ya se ha convertido en un factor de vital importancia para lograr que las aplicaciones informáticas tengan un alto nivel de calidad.

En Cuba, en los últimos años, ha habido un progreso en el desarrollo de software, principalmente las empresas han comenzado la automatización e informatización de varias áreas donde el trabajo con las tecnologías se desempeña con mayor calidad. Es por ello que el Gobierno Cubano tiene como una de sus tareas principales desarrollar la industria del software con el objetivo de informatizar la sociedad e insertarse en el mercado a nivel mundial. En este contexto surge la Universidad de las Ciencias Informáticas (UCI), que tiene como uno de sus objetivos principales producir software y servicios informáticos. Con el transcurso del tiempo esta universidad se ha convertido en la vanguardia de las empresas desarrolladoras de software en Cuba y ha sido un pilar fundamental en la informatización de algunos sectores de la sociedad. Persiguiendo este objetivo, surgen en la UCI los centros de desarrollo, conceptualmente, pequeñas empresas enfocadas a la producción de software en una rama determinada. Entre ellos se encuentra el Centro de Tecnologías de Gestión de Datos (DATEC) el cual posee como base fundamental 3 áreas temáticas dentro de las que se encuentra PostgreSQL, la cual tiene como metas fundamentales: contribuir a la soberanía tecnológica potenciando tecnologías de bases de datos libres, proveer soluciones integrales y consultorías relacionadas con la explotación de PostgreSQL. Esta área se encuentra estructurada en varios sub-proyectos como son la Comunidad Técnica Cubana de PostgreSQL, Herramientas de Administración y el Servidor de Gestión (Naire). Este último surge debido a que la administración de las bases de datos ha sido por mucho tiempo una tarea compleja. Una de las causas de esta complejidad, en ocasiones, está dada por el número de bases de datos a administrar, por lo que los administradores se enfrentan a varios problemas dentro de los cuales pueden mencionarse, monitorear el rendimiento de las bases de datos, planificar las tareas que se ejecutarán en varios servidores y analizar los reportes sobre el funcionamiento de las bases de datos. Por ello se decidió implementar un servidor de gestión que de solución a estos problemas, mediante el desarrollo de los componentes monitoreo, planificador de tareas y reportes.

Teniendo en cuenta lo analizado se plantea como **problema de la investigación**: ¿Cómo integrar los componentes de monitoreo, planificador de tareas y reportes para garantizar que el Servidor de Gestión Naire sea un sistema robusto, flexible y mantenible?

La investigación tiene como **objeto de estudio** la arquitectura de software, enmarcado en el **campo de acción** la arquitectura de software en los servidores de gestión de bases de datos.

Se plantea como **objetivo general** de la investigación: definir la arquitectura de software para el Servidor de Gestión de PostgreSQL.

En correspondencia con el objetivo general, se plantean como **objetivos específicos**:

- Elaborar el marco teórico de la investigación.
- Definir el rol de arquitecto de software para la metodología XP.
- Describir la arquitectura de software del Servidor de Gestión de PostgreSQL.
- Evaluar la arquitectura de software diseñada para el Servidor de Gestión de PostgreSQL.

Para lograr el cumplimiento de estos objetivos se realizan las siguientes **tareas investigativas:**

- Investigación sobre el rol del arquitecto en el desarrollo de software.
- Describir las actividades y artefactos a generar del rol de arquitecto de software para la metodología XP.
- Estudio y selección del estilo y los patrones de diseño.
- Fundamentación de la utilización y aplicación de los patrones de diseño.
- Definición de las herramientas y tecnologías a utilizar para el desarrollo.
- Diseño de las vistas de historia de usuario, lógica, despliegue, implementación y de procesos.
- Análisis y selección de los métodos de evaluación basados en la arquitectura.
- Evaluación de la arquitectura propuesta a través de los métodos seleccionados.

El presente documento está estructurado en tres capítulos:

Capítulo 1: Fundamentación Teórica.

En el capítulo se plantean los principales conceptos relacionados con el objeto de estudio y que resultan importantes para entender posteriormente la solución propuesta. Se realizará un análisis del estado del arte de la arquitectura de software, partiendo del estudio de los diferentes conceptos existentes definidos por varios autores, así como la importancia del rol arquitecto de software. Además, se definen los estilos de diseño, patrones y lenguaje de descripción arquitectónica a utilizar.

Capítulo 2: Arquitectura del Sistema.

En el capítulo se realiza una propuesta de solución al problema planteado. Se analizan los requisitos del sistema, se escogen las tecnologías y herramientas informáticas para dar soporte al desarrollo de

la aplicación y se define el rol arquitecto de software para la metodología XP. Se representa la descripción de la arquitectura de software a través de las vistas arquitectónicas.

Capítulo 3: Evaluación de la arquitectura.

En el capítulo se presentan distintos métodos para evaluar un diseño arquitectónico y a partir del método seleccionado se hace un análisis de la solución propuesta, logrando así la validación de la arquitectura de software definida.

1 CAPÍTULO FUNDAMENTACIÓN TEÓRICA

En el capítulo se plantean los principales conceptos relacionados con el objeto de estudio y que resultan importantes para entender posteriormente la solución propuesta. Se realizará un análisis del estado del arte de la arquitectura de software, partiendo del estudio de los diferentes conceptos existentes definidos por varios autores, así como la importancia del rol arquitecto de software. Además, se definen los estilos de diseño, patrones y lenguaje de descripción arquitectónica a utilizar.

1.1 Arquitectura de software

Existen gran variedad de conceptos y definiciones sobre la Arquitectura de Software expuesto por varios autores. En la presente investigación se analizan aquellos dados por instituciones reconocidas, así como de autores de prestigio mundial, los cuales han servido de base para el desarrollo del presente trabajo.

La IEEE Std 1471-2000 plantea: “La arquitectura del software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución”. [1]

Según Len Bass, Paul Clements y Rick Kazman: “La arquitectura de software de un sistema de programa o computación es la estructura de las estructuras del sistema, la cual comprende los componentes del software, las propiedades de esos componentes visibles externamente, y las relaciones entre ellos.” [2]

Philippe Kruchten plantea: “La arquitectura de software, tiene que ver con el diseño y la implementación de estructuras de software de alto nivel. Es el resultado de ensamblar un cierto número de elementos arquitectónicos de forma adecuada para satisfacer la mayor funcionalidad y requerimientos de desempeño de un sistema, así como requerimientos no funcionales, como la confiabilidad, escalabilidad, portabilidad, y disponibilidad.” [3]

Después del análisis realizado sobre los diferentes conceptos, para la presente investigación se define a la arquitectura de software como el diseño de más alto nivel de la estructura de un sistema, de sus

componentes y de las relaciones entre estos, es llevar las necesidades del usuario a un sistema usando las tecnologías y componentes precisos para su total funcionamiento.

1.1.1 Importancia y objetivos

Importancia de la arquitectura de software:

La importancia de la arquitectura de software abarca todo el proceso para dar distribución y orden a los elementos de un sistema informático, desde los requerimientos que debe satisfacer el mismo y las restricciones a las que está sujeto, hasta las propiedades no funcionales y las decisiones de diseño que gobiernan esta estructura. También es donde los profesionales aportan todos sus conocimientos, creatividad y experiencia para crear la mejor propuesta de solución que se dará al cliente, así como sus preocupaciones principales de lo que esperan del mismo. Resulta vital en el éxito final del sistema como una entidad operacional, ya que facilita el camino a la construcción de un sistema exitoso.

La importancia de la arquitectura de software se evidencia en los siguientes aspectos: [4]

- **Comunicación mutua:** Los desarrolladores de un software pueden utilizar la arquitectura de software como base para crear un entendimiento mutuo y comunicarse entre sí. Esto es muy importante para tomar futuras decisiones y formar un consenso común respecto a estas.
- **Decisiones tempranas de diseño:** Mediante la arquitectura de software se toman decisiones tempranas del diseño sobre un sistema, lo cual tiene un peso importantísimo para la mitigación de riegos potenciales, evitando que ocurran futuros desastres a gran escala.
- **Restricciones constructivas:** Una descripción arquitectónica proporciona diagramas que brindan una representación del sistema, indicando los componentes y las dependencias entre ellos, los cuales constituyen una guía para el desarrollo.
- **Reutilización:** La arquitectura de software promueve la reutilización a gran escala de una cantidad importante de componentes y frameworks. Esto reduce los costos de diseño y la cantidad de código se simplifica.
- **Evolución:** La arquitectura de software estima los posibles cambios y los costos de las modificaciones a las que se puede someter un sistema, permitiendo comprender el grado de mejoramiento que este puede alcanzar.
- **Análisis:** Las descripciones arquitectónicas aportan nuevas oportunidades para el análisis, incluyendo verificaciones de consistencia del sistema, conformidad con las restricciones impuestas por un estilo, conformidad con atributos de calidad, análisis de dependencias y análisis específicos de dominio y negocios.

- **Administración:** La experiencia demuestra que los proyectos exitosos consideran una arquitectura viable como un logro clave del proceso de desarrollo industrial. La evaluación crítica de una arquitectura conduce típicamente a una comprensión más clara de los requerimientos, las estrategias de implementación y los riegos potenciales.

Algunos de los objetivos que se buscan alcanzar al diseñar una arquitectura de software son los siguientes: [5]

- Comprender y mejorar la estructura de las aplicaciones complejas.
- Reutilizar dicha estructura (o partes de ella) para resolver problemas similares.
- Planificar la evolución de la aplicación, identificando las partes que cambian y las que permanecen constantes de la misma, así como los costos de los posibles cambios.
- Analizar la corrección de la aplicación y su grado de cumplimiento respecto a los requisitos.
- Permitir el estudio de algunas propiedades específicas del dominio.

1.2 Rol arquitecto de software

El Arquitecto de Software es el principal tomador de decisiones respecto a la manera en que será construida la aplicación por los programadores del equipo. Éste tiene la responsabilidad general de conducir las principales decisiones técnicas, expresadas como la arquitectura de software. Por su importancia una serie de entidades han dedicado parte de sus estudios a definir elementos significativos en los que el arquitecto desempeña un papel fundamental.

Características de un Arquitecto de Software por IBM: [6]

- El arquitecto tiene competencias técnicas y de liderazgo, debe tener la autoridad para tomar decisiones técnicas, ayudar a armar el equipo y a organizar el trabajo, además constantemente comunica el valor de lo que se está haciendo.
- El arquitecto debe tener conocimiento del proceso de desarrollo, ya que este garantiza que todos los miembros del equipo trabajen de manera coordinada. Un buen proceso define las funciones claramente. Dado que el arquitecto participa a diario con muchos de los miembros del equipo, es importante para el arquitecto entender sus funciones y responsabilidades.
- Entiende el dominio del negocio, el cual es un área de conocimiento o actividad caracterizada por un conjunto de conceptos y terminología entendida por los profesionales del área y permite imaginar requisitos probables y anticipar cambios.

- Tiene conocimiento tecnológico, pero no necesariamente experticia profunda. Dado que la tecnología cambia con cierta frecuencia, es esencial que el arquitecto se mantenga actualizado con los cambios tecnológicos.
- Tiene competencias de diseño.
- Tiene suficiente competencia de desarrollo como para comunicarse con el equipo.
- Es un buen comunicador.
- Es un negociador, debe explicar a los clientes del proyecto las consecuencias de sus opciones o alternativas de arquitectura.

Características de un Arquitecto de Software por Microsoft: [6]

- Poseer fuerte visión para los negocios el cual consiste en entender los costos de capital operacional y considerar cada uno de estos mientras se crea la solución. Leer estados financieros, tener conversaciones con funcionarios financieros y tener una comunicación acertada con los dueños de negocios para justificar los proyectos y calcular el rendimiento de un proyecto.
- Pensamiento visionario, ya que durante la participación en un proyecto, el arquitecto debe considerar y proyectar la tecnología en el futuro, visionando los cambios que se producen en los negocios de los clientes, y la mejor manera de aprovechar las ventajas de la solución tecnológica actual en el futuro.
- Investigar nuevas tecnologías, debe estar en continua investigación de nuevas tendencias en tecnología y las aplicaciones empresariales.
- Los arquitectos seleccionan y usan metodologías en los proyectos, entienden el funcionamiento del framework y cómo la solución será desarrollada, el comportamiento antes y después del despliegue. Entienden el ciclo de vida de un proyecto y de una solución.
- Seguir y divergir a la vez, los arquitectos deben tener la capacidad de personalizar o modificar frameworks y las metodologías utilizadas para lograr una solución a un problema o requisito de negocio.
- Poder desarrollar rápidamente un profundo conocimiento en una tecnología, ganando profundidad en múltiples tecnologías anteriores, el arquitecto puede asociar o transferir la capacidad de aprender otros métodos para investigar y para ganar rápidamente experiencia en nuevas tecnologías.
- Los arquitectos deben colaborar en el proceso de indagación de la información para llegar a una solución, pero pueden empezar a trabajar con información limitada y conforme el proyecto

progresa, tomar decisiones de compensación o equilibrio con el fin de mantener una solución que cumpla con los objetivos y continuar satisfaciendo las exigencias de negocio que al principio fueron identificadas.

1.3 Servidores de gestión

Los servidores de gestión son utilizados para administrar, gestionar y monitorear un Sistema de Gestión de Base de Datos (SGBD). Constituyendo una completa solución para administradores de bases de datos para la captura en tiempo real del rendimiento de la base de datos, las vistas de estadísticas del servidor, generación de reportes automáticos y proveer sugerencias para la configuración del servidor en producción.

En la actualidad los servidores de gestión más utilizados son los desarrollados por las empresas SpringSource y la EnterpriseDB. A continuación se exponen los elementos más importantes de cada uno de ellos.

1.3.1 Postgres Plus HQ

La empresa EnterpriseDB tiene como uno de sus productos a ofertar Postgres Plus HQ. Este está construido sobre PostgreSQL en el mundo del código abierto más avanzado en base de datos relacional totalmente disponible en la actualidad, suministrando conectores de cliente, la replicación, el almacenamiento en caché y otros módulos de nivel empresarial. [7]

La siguiente es una lista de los componentes individuales de Postgres Plus HQ: [7]

- **Postgres Plus HQ Agente:** Este programa se ejecuta en cada equipo que se desea supervisar. El Agente es responsable de detectar automáticamente los componentes de software que se ejecutan en la máquina y presentar la información al servidor. También detecta los cambios que ocurren en la configuración del sistema. Además de recopilar las métricas y enviarlas al servidor para su posterior análisis.

- **Postgres Plus HQ Server:** Es una aplicación JBoss (servidor de aplicaciones que queda automáticamente instalado cuando se instala Postgres Plus HQ) que recibe la información de los recursos y métricas de los Agentes y almacena los datos en la base de datos. Además Postgres Plus HQ Server permite configurar y administrar Postgres Plus HQ.

El Servidor de Gestión Postgres Plus HQ puede adquirirse bajo los términos de código abierto, pero presenta dependencia directa con las políticas de la empresa en cuanto al soporte que se le brinda a

esta aplicación.[8] No siendo esta una solución factible para Cuba ya que no contribuye a la soberanía tecnológica.

1.3.2 Hyperic HQ

La empresa SpringSource tiene como uno de sus productos a ofertar Hyperic HQ que administra y monitorea sus aplicaciones e infraestructuras, asegurando al mínimo el tiempo que pueda estar sin servicios su aplicación y notifica inmediatamente la degradación del rendimiento o la no disponibilidad del servicio. Es una plataforma basada en Java para monitorear y administrar recursos de software. Los elementos clave de la arquitectura de Hyperic HQ son los Servidores HQ, que proporcionan una administración y persistencia centralizada, y el Agente HQ, que facilita el monitoreo y control por plataforma. [9]

Figura 1: Arquitectura de Hyperic HQ

Agente HQ

Hyperic HQ ejecuta un Agente en cada máquina que desea administrar. Desde el primer inicio, el agente auto-descubre el software que se encuentra corriendo en la máquina, y periódicamente vuelve a escanear para cambios de configuración. Los Agentes HQ acumulan disponibilidad, utilización, desempeño y métricas usadas. Además realizan rastreo de accesos, eventos y le permiten realizar funciones de control, como iniciar y detener servidores web. Estos envían el inventario de datos y métricas que acumularon a un Servidor Central HQ.

El Agente se lleva a cabo principalmente en Java, con pequeñas porciones de código en C portátil. Funciona en Linux, Windows, Solaris, HP / UX, Mac OS X, y AIX. El agente está diseñado para tener una memoria compacta y la huella de utilización de la CPU.

Servidor HQ

El Servidor HQ recibe inventario y métrica de datos de los Agentes y los almacena en la base de datos. Además, determina una variedad de información de tipo específico, por ejemplo, la arquitectura de una plataforma, la memoria RAM, velocidad de la CPU, la dirección IP y el nombre de dominio. Proporciona facilidades para administrar su inventario de software, ya que implementa el inventario y modelo de acceso, agrupando los recursos de su software en maneras útiles para permitir el monitoreo y administración. El Servidor HQ detecta cuando las alertas se disparan y realiza las notificaciones que se definan. También procesa acciones que se hayan iniciado para la interfaz de usuario. Proporciona servicios de autentificación, utilizando una máquina interna o un servicio de autentificación externa. Tiene incorporado un servidor de aplicaciones JBoss, y un servidor PostgreSQL, de igual forma soporta otro tipo de bases de datos para almacenar su información como MySQL y Oracle.
El Servidor HQ es una aplicación J2EE distribuido que se ejecuta en la parte superior del código abierto JBoss Application Server. Está escrito en Java y de código en C portable y se ejecuta en Linux, Windows, Solaris, OS X, y HPUX.

Esta herramienta de origen estadounidense está sujeta a las normas de exportación de EE.UU, las cuales prohíben las operaciones de exportación de la misma a cualquier persona que sea ciudadano o esté al servicio del gobierno de Cuba.[10] La herramienta Hyperic HQ propone elementos de gran fortaleza, como su arquitectura basada en agentes, que proporciona mayor organización al proceso de monitoreo. Por esta característica se propone basar la arquitectura física del Servidor de Gestión Naire en la herramienta Hyperic HQ.

1.4 Estilos y patrones

En algunas bibliografías referidas al estudio de los patrones arquitectónicos se les conoce de la misma manera a los patrones y estilos, pero la mayoría de los autores los ven de manera separada. La diferencia radica en el nivel de abstracción en que se aplican. Los estilos están en un plano de abstracción más alto, definiendo como configurar la arquitectura, mientras los patrones están más cercanos al diseño, incluso podría decirse que más cercano a algo físico, pues estos pueden representarse mediante código en un lenguaje de programación determinado.

Algunos patrones coinciden con los estilos hasta en el nombre con que se los designa. Los elementos por los que difieren son mostrados en la tabla 1. [11]

Tabla 1: Diferencias entre Estilos y Patrones Arquitectónicos.

Elemento a comparar	**Estilo Arquitectónico**	**Patrón Arquitectónico**
Definición	Son una categorización de sistemas.	Son soluciones generales a problemas comunes.
Representación	Sólo describe el esqueleto estructural general para aplicaciones.	Existen en varios rangos de escala, comenzando con patrones que definen la estructura básica de una aplicación.
Solución	Expresan técnicas de diseño desde una perspectiva que es independiente de la situación actual de diseño.	Expresa un problema recurrente de diseño muy específico, y presenta una solución para él, desde el punto de vista del contexto en el que se presenta.

Una vez analizados los elementos esenciales de los estilos y patrones de arquitectura, se concluye que existen afinidades entre ambos conceptos, pero difieren en que los patrones se refieren más a prácticas de reutilización y los estilos corresponden a teorías sobre la estructura de los sistemas.

1.4.1 Estilos

Los estilos arquitectónicos indican los tipos de componentes y conectores involucrados en una arquitectura. Los diferentes estilos de las arquitecturas tienen sus fortalezas y debilidades, ciertos estilos hacen que sea más fácil o más difícil trabajar con diferentes obstáculos.

Cada estilo describe una categoría del sistema que contiene un conjunto de componentes que realizan una función requerida por el sistema, un conjunto de conectores que posibilitan la comunicación, la coordinación y la cooperación entre los componentes; también describen las restricciones que definen como se pueden integrar los componentes que forman el sistema y modelos semánticos que permiten al diseñador entender las propiedades globales de un sistema para analizar las propiedades conocidas de sus partes constituyentes.

Algunos de los principales estilos arquitectónicos que se usan en la actualidad están divididos por clases, las que se exponen a continuación: [12]

- **Estilos de flujo de datos:** Esta familia de estilos enfatiza la reutilización. Es apropiada para sistemas que implementan transformaciones de datos en pasos sucesivos.
    - **Arquitecturas en tubería y filtros:** Una tubería (pipeline) es una popular arquitectura que conecta componentes computacionales a través de conectores, de modo que las

computaciones se ejecutan a la manera de un flujo. Los datos se transportan a través de las tuberías entre los filtros, transformando gradualmente las entradas en salidas. El sistema tubería y filtros percibe como una serie de transformaciones sobre sucesivas piezas de los datos de entrada. Los datos entran al sistema y fluyen a través de los componentes. Cada paso se ejecuta hasta completarse antes que se inicie el paso siguiente.

- **Estilos de llamada y retorno:** Esta familia de estilos enfatiza la escalabilidad. Son los estilos más generalizados en sistemas en gran escala. El sistema se constituye de un programa principal que controla el sistema y varios subprogramas que se comunican con éste mediante el uso de llamadas.
    - **Modelo Vista Controlador (MVC):** En ocasiones se le define más bien como un patrón de diseño o como práctica recurrente. Un propósito común en numerosos sistemas es el de tomar datos de un almacenamiento y mostrarlos al usuario. Luego que el usuario introduce modificaciones, las mismas se reflejan en el almacenamiento. Dado que el flujo de información ocurre entre el almacenamiento y la interfaz, una tentación común, un impulso espontáneo (hoy se llamaría un anti-patrón) es unir ambas piezas para reducir la cantidad de código y optimizar el rendimiento. Sin embargo, esta idea es antagónica al hecho de que la interfaz suele cambiar, o acostumbra depender de distintas clases de dispositivos; la programación de interfaces de HTML, además, requiere habilidades muy distintas de la programación de lógica de negocios. El estilo conocido como MVC separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes:
        - Modelo: 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).
        - Vista: Maneja la visualización de la información.
        - Controlador: Interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado.
    - **Arquitecturas en capas:** Es una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. En la práctica, las capas suelen ser entidades complejas, compuestas de varios paquetes o subsistemas. En un estilo en capas, los conectores se

definen mediante los protocolos que determinan las formas de la interacción. Las restricciones topológicas del estilo pueden incluir una limitación, más o menos rigurosa, que exige a cada capa operar sólo con capas adyacentes y a los elementos de una capa entenderse sólo con otros elementos de la misma.

- **Estilos centrados en datos:** Esta familia de estilos enfatiza la integridad de los datos. Se estima apropiada para sistemas que se fundan en acceso y actualización de datos en estructuras de almacenamiento.
    - **Arquitecturas de pizarra o repositorio:** En esta arquitectura hay dos componentes principales: una estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre él. Estos sistemas se han usado en aplicaciones que requieren complejas interpretaciones de proceso de señales (reconocimiento de patrones, reconocimiento de habla, etc.), o en sistemas que involucran acceso compartido a datos con agentes débilmente acoplados. También se han implementado estilos de este tipo en procesos en lotes de base de datos y ambientes de programación organizados como colecciones de herramientas en torno a un repositorio común.

Luego de realizarse un estudio detallado de los principales estilos existentes, se propone como estilo a utilizar el de Llamada y Retorno, en especial el estilo arquitectónico MVC por todas las características que estos presentan, así como la ventaja de la adaptación al cambio. La separación de las capas en presentación, lógica de negocio y acceso a datos es fundamental para el desarrollo de arquitecturas consistentes y reutilizables, lo que al final resulta en un ahorro de tiempo en el desarrollo de posteriores proyectos. También la facilidad que presenta la separación de la vista del modelo, ya que no existe una dependencia directa de estos, proporcionando así que la interfaz de usuario pueda mostrar múltiples vistas de los mismos datos simultáneamente. Al existir la separación de vistas, controladores y modelos es más sencillo realizar labores de mejora como:

- Agregar nuevas vistas.
- Agregar nuevas formas de recolectar las órdenes del usuario.
- Modificar los objetos de negocios bien sea para mejorar el performance o para migrar a otra tecnología.
- Las labores de mantenimiento también se simplifican y se reduce el tiempo necesario para ellas.

- Las correcciones solo se realizan en un solo lugar, debido a la integración de la presentación e implementación de la lógica del negocio.

1.4.2 Patrones

El patrón es una descripción del problema y la esencia de su solución, de forma que la solución pueda re-utilizarse en diferentes situaciones, no constituye una especificación detallada, es una solución adecuada a un problema común. El uso de patrones mejora la calidad del sistema y la selección de estos constituyen una decisión fundamental de diseño en el desarrollo de un software. Un arquitecto reutiliza patrones para definir la estructura y el comportamiento interno y externo de un sistema.

Las características de un buen patrón son:

- **Debe solucionar un problema:** Los patrones capturan soluciones, no solo principios o estrategias abstractas.
- **Debe ser un concepto probado:** Ser soluciones demostradas, no teorías o especulaciones.
- **La solución no es obvia:** Muchas técnicas de solución de problemas tratan de hallar soluciones por medio de principios básicos. Los mejores patrones generan una solución a un problema de forma indirecta.
- **Describe participantes y relaciones entre ellos:** Los patrones no sólo describen módulos sino estructuras del sistema y mecanismos más complejos.

Los elementos que debe contener un patrón son: [13]

- **Nombre:** Tiene que tener un nombre significativo. Sería muy incontrolable tener que describir el patrón cada vez que se utilizan en una discusión.
- **Problema:** Describe cuándo aplicarlo, explica el problema, su contexto y la lista de precondiciones que deben encontrarse.
- **Solución:** Describe los elementos que lo componen: clases, objetos, relaciones, responsabilidades y colaboraciones.
- **Consecuencias:** Describe los costos y beneficios de aplicarlo. Incluye el impacto sobre la flexibilidad, extensibilidad y portabilidad del sistema.

Entre las ventajas del uso de patrones, se pueden encontrar: [14]

- Permiten la reutilización de soluciones arquitectónicas de calidad.
- Son de gran ayuda para controlar la complejidad de un diseño.
- Facilitan la documentación de diseños arquitectónicos.

- Proporcionan un vocabulario común que mejora la comunicación entre diseñadores.

Los patrones pueden dividirse o clasificarse en:

- **Patrones de arquitectura:** Relacionados a la interacción de objetos dentro o entre niveles arquitectónicos, problemas arquitectónicos, adaptabilidad a requerimientos cambiantes, modularidad y acoplamiento.
- **Patrones de diseño:** Fueron construidos dado los problemas con la claridad de diseño, multiplicación de clases y adaptabilidad a requerimientos cambiantes.
- **Patrones de análisis:** Usualmente específicos de aplicación.
- **Patrones de proceso o de organización:** Tratan todo lo concerniente con el desarrollo o procesos de administración de proyectos, o técnicas, o estructuras de organización.
- **Patrones de idioma:** Regulan la nomenclatura en la cual se escriben, se diseñan y desarrollan los sistemas.

Patrones de Arquitectura

Los patrones de arquitectura expresan el esquema fundamental de organización para sistemas de software. Proveen un conjunto de subsistemas predefinidos, especifican sus responsabilidades e incluyen reglas y guías para organizar las relaciones entre ellos. Además, ayudan a especificar la estructura fundamental de una aplicación, así como a conseguir una propiedad específica en el sistema global.

El libro "*Pattern-Oriented Software Architecture: A System of Patterns*" de F. Buschmann, propone una serie de patrones, algunos de los que mencionan son: [15]

- **Tubería y filtros:** Provee una estructura para sistemas que procesan un flujo de datos. Cada etapa del proceso es encapsulada como un filtro. Los datos se pasan entre filtros adyacentes mediante tubos. Re-combinando filtros se obtienen familias de sistemas relacionados.
- **Pizarra o repositorio:** Útil para sistemas en que no se conoce una solución o estrategia determinista. Varios subsistemas especializados ensamblan su conocimiento para construir una posible solución parcial.
- **Capas:** Permite estructurar aplicaciones que se pueden descomponer en grupos de sub-tareas, donde cada grupo está en un determinado nivel de abstracción.
- **Corredor:** Permite estructurar sistemas distribuidos desacoplados que interaccionan mediante invocación de servicios remotos. Un componente Corredor es responsable de coordinar la

comunicación, así como de transmitir resultados y excepciones.

- **Modelo-Vista-Controlador (MVC, Model-View-Controller por sus siglas en inglés):** Divide una aplicación interactiva en tres componentes: el modelo contiene la información y funcionalidad principal, las vistas muestran información al usuario y el controlador gestiona la entrada de usuario. Un mecanismo de propagación de cambios asegura la consistencia entre el modelo y la interfaz de usuario.
    - **Modelo:** Define la lógica de negocio, la base de datos pertenece a esta capa. Encapsula los datos y las funcionalidades. Es independiente de cualquier representación de salida y/o comportamiento de entrada. Esta es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura la integridad de estos y permite derivar nuevos datos.
    - **Vista:** Es lo que utilizan los usuarios para interactuar con la aplicación, los gestores de plantillas pertenecen a esta capa. Este presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.
    - **Controlador:** Recibe las entradas, traducidas a solicitudes de servicio para el modelo. Es un bloque de código que realiza llamadas al modelo para obtener los datos y se los pasa a la vista para que los muestre al usuario. Este responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista.

Patrones de Diseño

Según Gamma: *"Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí adaptada para resolver un problema de diseño general en un contexto particular."* [16]

Existen patrones que describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Estos son los conocidos Patrones de Software para la asignación General de Responsabilidad (**GRASP** por sus siglas en inglés). [17]

Experto

Problema: ¿Cómo asignar responsabilidades, de la forma más eficiente?

Solución: Asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad.

Explicación: Experto es un patrón que se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos. Con él no se pretende designar una idea oscura ni extraña; expresa simplemente la "intuición" de que los objetos hacen

cosas relacionadas con la información que poseen.

Creador

Problema: ¿Quién debería ser el responsable de crear una nueva instancia de alguna clase?

Solución: La responsabilidad de crear una instancia de la clase A se le dará a aquella clase B, en los siguientes casos:

- B agrega los objetos A.
- B contiene los objetos A.
- B registra las instancias de los objetos A.
- B utiliza específicamente los objetos A.
- B tiene los datos de inicialización que serán transmitidos hacia A cuando este objeto sea creado (B es experto en la creación de A).

Explicación: El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento.

Bajo Acoplamiento

Problema: ¿Cómo dar soporte a una dependencia escasa y a un aumento de la reutilización?

Solución: Asignar una responsabilidad para mantener bajo acoplamiento.

Explicación: El Bajo Acoplamiento es un principio que no se puede descartar durante las decisiones del diseño, como meta principal a lograr siempre. Se puede catalogar como un patrón evaluativo que el diseñador aplica al juzgar sus decisiones de diseño.

El bajo acoplamiento estimula la asignación de responsabilidades de forma tal que la inclusión de éstas no incremente el acoplamiento, creando clases más independientes y con mayor resistencia al impacto de los cambios, que aumentan la productividad y la posibilidad de reutilización.

Alta Cohesión

Problema: ¿Cómo mantener la complejidad dentro de los límites manejables?

Solución: Asignar una responsabilidad de modo que la cohesión siga siendo alta (la cohesión es una medida de cuan relacionadas y enfocadas están las responsabilidades de una clase, además de que una alta cohesión garantiza que clases con responsabilidades estrechamente relacionadas no realicen un trabajo enorme).

Explicación: Como el patrón Bajo Acoplamiento, también Alta Cohesión es un principio que se debe

tener presente en todas las decisiones de diseño: es la meta principal que ha de buscarse en todo momento. Es un patrón evaluativo que el desarrollador aplica al valorar sus decisiones de diseño.
Ahora, existen formas de calificar el grado de cohesión, siendo estas: muy baja, baja, moderada y alta.
Un diseño con muy baja cohesión, es aquel en el que una clase es la responsable de operaciones correspondientes a áreas funcionales muy heterogéneas, sucediendo algo parecido con un diseño de baja cohesión, ya que en este caso la clase tiene la responsabilidad exclusiva de una tarea compleja dentro de un área funcional. En ninguno de los dos casos anteriores se delega o distribuyen las responsabilidades, dichas clases abarcan el volumen de las responsabilidades a realizar sin importar su complejidad o heterogeneidad.
Un diseño con un nivel de cohesión baja, presenta problemas a la hora de comprender, reutilizar o conservar dichas clases, además de constantes afectaciones dadas por cambios en dicho diseño.
Un nivel moderado de cohesión implica que la clase posee un peso ligero pues agrupa áreas que están relacionadas lógicamente con el concepto de clases pero no entre ellas.
El alto nivel de cohesión, es presentado por las clases que tienen responsabilidades moderadas en un área funcional y colaboran con otras para llevar a cabo las tareas.

Controlador

Problema*:* ¿Quién debería encargarse de atender un evento del sistema?
Solución: Asignar la responsabilidad del manejo de un mensaje de los eventos del sistema a una clase que represente una de las siguientes opciones: Controlador de fachada, controlador de tareas o controlador de casos de uso.
Explicación: La mayor parte de los sistemas reciben eventos de entrada externa, los cuales generalmente incluyen una interfaz gráfica para el usuario operado por una persona. Otros medios de entrada son los mensajes externos entre ellos un conmutador de telecomunicaciones para procesar llamadas o las señales procedentes de sensores como sucede en los sistemas de control de procesos.

Además, existen cuatro patrones GRASP adicionales:

- Fabricación Pura.
- Polimorfismo.
- Indirección.
- No hables con extraños.

El libro "*Design Patterns*", escrito por GOF (*Gang of Four*, "Pandilla de los Cuatro"), recopila una serie de patrones de diseños, agrupados en tres categorías: de creación, de estructura y de comportamiento. [18]

Patrones de Creación

- **Abstract Factory:** Proporciona una interfaz para crear familias de objetos o que dependen entre sí, sin especificar sus clases concretas.
- **Builder:** Separa la construcción de un objeto complejo de su representación, de forma que el mismo proceso de construcción pueda crear diferentes representaciones.
- **Factory Method:** Define una interfaz para crear un objeto, pero deja que sean las subclases quienes decidan qué clase instanciar. Permite que una clase delegue en sus subclases la creación de objetos.
- **Prototype:** Especifica los tipos de objetos a crear por medio de una instancia prototípica y crear nuevos objetos copiando este prototipo.
- **Singleton:** Garantiza que una clase sólo tenga una instancia y proporciona un punto de acceso global a ella.

Patrones Estructurales

- **Adapter:** Convierte la interfaz de una clase en otra distinta que es la que esperan los clientes. Permite que cooperen clases que de otra manera no podrían por tener interfaces incompatibles.
- **Bridge:** Desvincula una abstracción de su implementación, de manera que ambas puedan variar de forma independiente.
- **Composite:** Combina objetos en estructuras de árbol para representar jerarquías de parte todo. Permite que los clientes traten de manera uniforme a los objetos individuales y a los compuestos.
- **Decorator:** Añade dinámicamente nuevas responsabilidades a un objeto, proporcionando una alternativa flexible a la herencia para extender la funcionalidad.
- **Facade:** Proporciona una interfaz unificada y de alto nivel para un conjunto de interfaces de un subsistema lográndose que sea más fácil de usar.
- **Proxy:** Proporciona un sustituto o representante de otro objeto para controlar el acceso a éste.

Patrones de Comportamiento

- **Chain of Responsibility:** Evita acoplar el emisor de una petición a su receptor, al dar a más de un objeto la posibilidad de responder a la petición. Crea una cadena con los objetos receptores y pasa la petición a través de la cadena hasta que esta sea tratada por algún objeto.
- **Command:** Encapsula una petición en un objeto, permitiendo así parametrizar a los clientes con distintas peticiones, encolar o llevar un registro de las peticiones y poder deshacer las operaciones.

- **Interpreter:** Dado un lenguaje, define una representación de su gramática junto con un intérprete que usa dicha representación para interpretar las sentencias del lenguaje.
- **Iterator:** Proporciona un modo de acceder secuencialmente a los elementos de un objeto agregado sin exponer su representación interna.
- **Mediator:** Define un objeto que encapsula cómo interactúan un conjunto de objetos. Promueve un bajo acoplamiento al evitar que los objetos se refieran unos a otros explícitamente y permite variar la interacción entre ellos de forma independiente.
- **Memento:** Representa y exporta el estado interno de un objeto sin violar la encapsulación, de forma que éste puede volver a dicho estado más tarde.
- **Observer:** Define una dependencia de uno a muchos entre objetos, de forma que cuando un objeto cambia de estado se notifica y actualizan automáticamente todos los objetos.
- **State:** Permite que un objeto modifique su comportamiento cada vez que cambia su estado interno. Parecerá que cambia la clase del objeto.
- **Strategy:** Define una familia de algoritmos, encapsula uno de ellos y los hace intercambiables. Permite que un algoritmo varíe independientemente de los clientes que lo usan.
- **Template Method:** Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos. Permite que las subclases redefinan ciertos pasos del algoritmo sin cambiar su estructura.
- **Visitor:** Representa una operación sobre los elementos de una estructura de objetos. Permite definir una nueva operación sin cambiar las clases de los elementos sobre los que opera.

Luego de realizarse un estudio detallado de los patrones existentes, se propone como patrón a utilizar el MVC que se encuentra implícito en el estilo escogido, ya que presenta ventajas de gran utilidad para la realización del proyecto como son la separación del Modelo de la Vista, donde se separan los datos de su representación visual. Facilita agregar nuevos tipos de datos según sea requerido por la aplicación ya que son independientes del funcionamiento de las otras capas. También facilita el mantenimiento en caso de errores, ofrece maneras más sencillas para probar el correcto funcionamiento del sistema y permite el escalamiento de la aplicación en caso de ser requerido. Además utilizar los principales patrones GRASP ya que presentan como principal ventaja el grado de organización y ayuda que le proporcionan al programador en la implementación del sistema, en la organización de las clases, así como la importancia que se les pueda dar a las mismas. Estos también en su mayoría son muy utilizados en los frameworks de desarrollo por lo que constituye de vital importancia la utilización de los mismos.

1.6 Lenguaje de descripción de la arquitectura

Un Lenguaje de descripción de la arquitectura (ADL por sus siglas en inglés) es un lenguaje que provee características para modelar un sistema de software a nivel arquitectónico. Permite la descripción, análisis y reutilización de arquitecturas de software. Un ADL genuino tiene que proporcionar propiedades de composición, abstracción, re-usabilidad, configuración, heterogeneidad y análisis, lo que excluiría a todos los lenguajes convencionales de programación. Mediante un ADL, la arquitectura de un sistema se define como un conjunto de componentes, conectores y las conexiones entre ellos.

Características importantes de los ADLs [19]

- **Composición:** Permiten la representación del sistema como composición de una serie de partes.
- **Configuración:** La descripción de la arquitectura es independiente de la de los componentes que formen parte del sistema.
- **Abstracción:** Describen los roles o papeles abstractos que juegan los componentes dentro de la arquitectura.
- **Flexibilidad:** Permiten la definición de nuevas formas de interacción entre componentes.
- **Reutilización:** Permiten la reutilización tanto de los componentes como de la propia arquitectura.
- **Heterogeneidad:** Permiten combinar descripciones heterogéneas.
- **Análisis:** Permiten diversas formas de análisis de la arquitectura y de los sistemas desarrollados a partir de ella.

Existe gran variedad de ADLs, los principales de estos se mencionan a continuación:

- **Acme:** Soporta el mapeo de especificaciones arquitectónicas entre diferentes ADLs. Objetivo principal el intercambio entre arquitecturas e integración de ADLs.
- **Aesop:** Construye ambientes de desarrollo que soporta estilos. Tiene como objetivo la exploración de las bases formales de la arquitectura de software, el desarrollo del concepto de estilo arquitectónico y la producción de herramientas útiles a la arquitectura
- **Rapide:** Lenguaje de descripción de propósito general. El comportamiento se define vinculando observación de acciones externas con iniciación de acciones públicas.
- **Darwin:** ADL orientado a arquitecturas dinámicas, su desarrollo consiste en instanciar declaraciones de componentes y establecer relaciones entre servicios.

- **UniCon:** Centrado en el apoyo a la variedad de piezas y estilos arquitectónicos que se encuentran en el mundo real y en la construcción de sistemas de las descripciones de su arquitectura.

Sin embargo, como todos los lenguajes especializados, los ADLs solamente pueden ser comprendidos por expertos del lenguaje y son inaccesibles para los especialistas tanto de las aplicaciones como del dominio. Esto hace que sea difícil analizarlos desde una perspectiva práctica. Las notaciones como lenguaje unificado de modelado (UML) constituyen las más comúnmente usadas para la descripción arquitectónica, además de poseer un propósito general y comprensible.

1.6.1 Lenguaje unificado de modelado

El Lenguaje Unificado de Modelado (UML por sus siglas en inglés), como su propio nombre indica es un lenguaje que permite modelar, construir y documentar los elementos que forman un sistema de software orientado a objetos. Un lenguaje proporciona un vocabulario y unas reglas para permitir una comunicación. En este caso, este se centra en la representación gráfica de un sistema. Se ha convertido en poco tiempo en una notación estándar no sólo para la comunidad de investigadores en ingeniería del software, sino también para la industria. Cada vez hay más artículos, documentos técnicos, libros, etc., que hablan de UML y exponen experiencias de utilización de este lenguaje en el análisis, diseño y organización de sistemas complejos.

Los objetivos de UML son muchos, pero se pueden sintetizar sus funciones: [20]

- **Visualizar:** UML permite expresar de una forma gráfica un sistema de forma que otro lo puede entender.
- **Especificar:** UML permite especificar cuáles son las características de un sistema antes de su construcción.
- **Construir:** A partir de los modelos especificados se pueden construir los sistemas diseñados.
- **Documentar:** Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión.

Ventajas de utilizar UML:

- El uso de una notación única ayuda a mantener la coherencia entre las diferentes vistas del sistema y facilita su trazabilidad.
- Independiente del proceso de desarrollo.
- Su difusión, utilización y aceptación es cada vez más grande.

- Es un estándar que sigue unas revisiones.
- Hay gran soporte de herramientas para el lenguaje como lo son Rational Rose, Rhapsody y Visual Paradimg por solo mencionar algunos.
- Mayor rigor en la especificación.
- Permite realizar una verificación y validación del modelo realizado.
- Se pueden automatizar determinados procesos y permite generar código a partir de los modelos y a la inversa (a partir del código fuente generar los modelos). Esto permite que el modelo y el código estén actualizados, con lo que siempre se puede mantener la visión en el diseño, de más alto nivel, de la estructura de un proyecto.

1.7 Conclusiones del capítulo

En este capítulo se han abordado temas relacionados con la arquitectura de software, patrones y los lenguajes de descripción de arquitectura, los principales estilos arquitectónicos, así como un estudio relacionado con el objeto de investigación, con el propósito de proveer una solución arquitectónica que permita conformar un Servidor de Gestión de PostgreSQL. Finalmente, se propone por las características del sistema y las necesidades del grupo de desarrollo seguir la línea del estilo arquitectónico llamada y retorno y el patrón de arquitectura MVC. Además, se propone aplicar el conjunto de patrones de diseño GRASP y UML como lenguaje de modelado.

.

2
CAPÍTULO
ARQUITECTURA DEL SISTEMA

En el capítulo se realiza una propuesta de solución al problema planteado. Se analizan los requisitos del sistema, se escogen las tecnologías y herramientas informáticas para dar soporte al desarrollo de la aplicación y se define el rol arquitecto de software para la metodología XP. Se representa la descripción de la arquitectura de software a través de las vistas arquitectónicas.

2.1 Organización estructural del sistema

En la Figura 1 se muestra la forma más general en que se propone organizar el sistema, para ofrecer una mejor idea y facilitar la comprensión de la arquitectura propuesta.

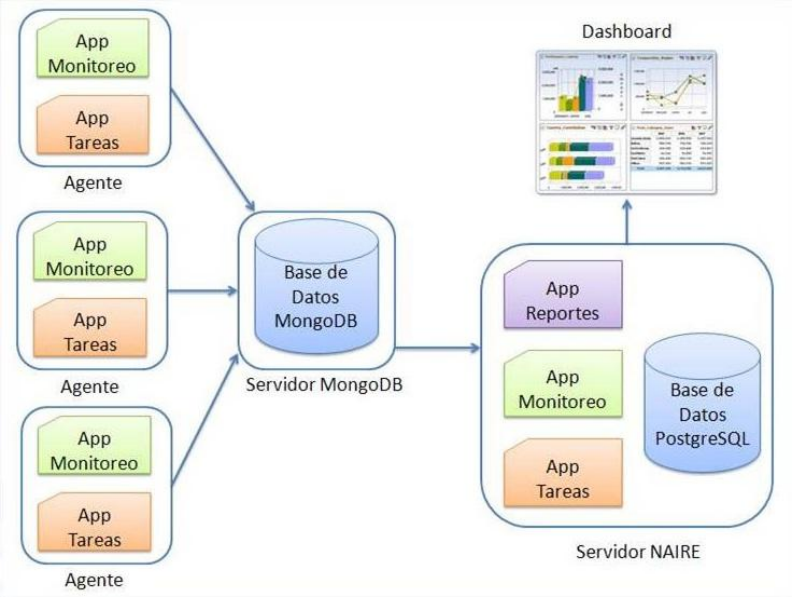


Figura 2: Organización estructural general del sistema.

El sistema se encontrará dividido en 3 partes, el servidor de aplicaciones, el servidor de base de datos y el agente o servidor a monitorear.

- **Servidor de Aplicación:** Proveerá el sistema central de gestión y persistencia de los datos, visualización de los reportes y monitoreo de las estadísticas del servidor basado en una aplicación web.
- **Servidor de BD:** Proveerá el sistema de gestión de datos en forma de documentación, monitoreando los mismos en tiempo real.
- **Agente:** Proveerá las bases para el monitoreo y control de los servicios de la base de datos en los diferentes servidores que estarán bajo la mira del administrador.
- **App Reporte:** Se encargará de brindar una interfaz al usuario que provea información de indicadores o métricas que permitan facilitar el análisis de estos datos para una correcta toma de decisiones.
- **App Monitoreo:** Se encargará de monitorear en tiempo real los servidores de bases de datos verificando indicadores como el consumo de recursos del servidor, tiempo de respuesta y comportamiento de las bases de datos en una determinada tarea o consulta, con el objetivo de conseguir la mayor cantidad de información posible y obtener datos suficientes para realizar análisis comparativos que permitirán evaluar el desempeño de los servidores y conocer el rendimiento de su infraestructura.
- **App Planificador de Tareas:** Se encargará de planificar una secuencia de tareas y la asignación de los recursos necesarios para alcanzar un objetivo al cumplirse las tareas asignadas a cada servidor de base de datos.

Para garantizar la disponibilidad de los datos se realizará para el servidor MongoDB una replicación de datos, mediante el conjunto de réplica que es un mecanismo de recuperación ante fallos automáticos unido a la replicación asincrónica de datos, el cual consiste en declarar un nodo maestro único (llamado primario) y uno o más esclavos (llamados secundarios), en caso de que la conexión a este nodo principal caiga, otro de los nodos secundarios tomaría su lugar. Esto asegura una mayor disponibilidad de la aplicación, ya que una vez que la conexión entre el servidor de aplicaciones y el servidor de base de datos maestro se encuentre afectada, el sistema re-direccionará sus peticiones de datos a aquel servidor que en ese momento sea el primario. [21]

En el servidor de aplicaciones se realizarán copias de seguridad sistemáticas asegurando así la disponibilidad de los datos. Estas copias serán incrementales, que consisten en copiar todos los elementos que han sido modificados desde la copia de seguridad anterior. Este tipo de copia es más eficaz que una copia de seguridad completa porque se centra específicamente en los archivos modificados y requiere menos espacio de almacenamiento. Ver figura 2

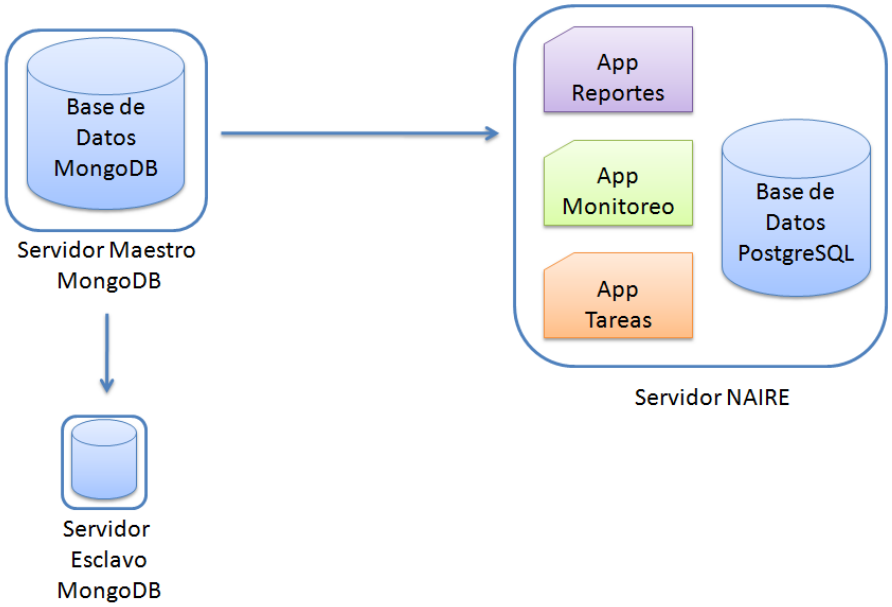


Figura 3: Organización estructural del funcionamiento del sistema.

2.2 Restricciones arquitectónicas

El sistema estará constituido por un grupo de componentes que interactúan entre sí, gestionando los datos correspondientes a su estudio, lo cual incluye insertar, modificar y realizar determinados reportes sobre los datos. A continuación se mencionan las Historias de Usuarios (HU) del sistema por componentes que tienen un impacto significativo en la arquitectura:

Componente de monitoreo cuenta con 11 HU, 8 significativas:

- HU1: Autenticar usuario.
- HU2: Administrar conexión a MongoDB.
- HU3: Administrar archivo log.
- HU4: Capturar uso de espacio en disco.
- HU5: Capturar uso de espacio de la memoria.
- HU6: Capturar cantidad de conexiones.
- HU7: Capturar cantidad de procesos activos por el servidor PostgreSQL.
- HU8: Capturar cantidad de transacciones.

- **Componente de reportes cuenta con 27 HU, 12 significativas:**
  - HU1: Buscar servidor de BD a monitorear.
  - HU2: Buscar BD de los servidores a monitorear.
  - HU3: Generar Reporte de métrica.

- HU4: Imprimir reporte de métrica.
- HU5: Exportar reporte de métrica.
- HU6: Consultar Información de métricas.
- HU7: Definir modo de autenticación.
- HU8: Generar alertas del sistema.
- HU9: Representar un histórico de las alertas de los servidores.
- HU10: Configurar parámetros de alertas de los sistemas.
- HU11: Eliminar las trazas del sistema.
- HU12: Representar histórico de valores de las métricas de un servidor determinado.

- **Componente planificador de tarea cuenta con 15 HU, 6 significativas:**
  - HU1: Insertar tarea.
  - HU2: Modificar tarea.
  - HU3: Eliminar tarea.
  - HU4: Listar por usuario.
  - HU5: Generar reporte por día.
  - HU6: Mostrar estadísticas.

Los requisitos no funcionales que se describen a continuación constituyen la base que debe sustentar la arquitectura. La reutilización de dicha arquitectura garantizará el aprovechamiento de los recursos tecnológicos con los que se cuenta.

Requerimientos de Software:

Requerimientos para los servidores:

- Debe tener como sistema operativo GNU/Linux, con la distribución Debian.

Requerimientos para el cliente:

- Los clientes tendrán acceso al sistema a través de cualquier navegador Web, o sea, para la navegación del sistema se realizará con el uso de navegadores: Internet-Explorer 5 o superior, Mozilla Firefox 3.0 o superior, preferentemente el Mozilla Firefox.
- La aplicación debe ejecutarse sobre cualquier Sistema Operativo que soporte las versiones mínimas de los navegadores que se presentaron anteriormente.

Requerimientos de Hardware:

Requerimientos mínimos para el servidor:

- Microprocesador Pentium superior a 3.0 GHz

- 512 MB RAM o superior.
- 4 GB de espacio en disco como mínimo.

Requerimientos mínimos para la conexión del cliente:

- Microprocesador Pentium superior a 3.0 GHz.
- 256 MB RAM o superior.

Requerimientos de Seguridad:

El sistema deberá contar con un mecanismo de seguridad basado en el modelo de Autenticación, Autorización y Auditoria (Authentication, Authorization and Accounting, AAA) con autenticación de firma única (Single Sign On).

- La autenticación será la primera acción del usuario en el sistema y consistirá en suministrar un nombre de usuario único y una contraseña que debe ser de conocimiento exclusivo de la persona que se autentica. Si el usuario autenticado no se encuentra registrado se debe reportar un error de acceso.
- Las trazas del sistema contendrán un texto descriptivo de las acciones realizadas así como el día, mes, año, hora, minuto, segundo en que se registra. Cada petición de usuario a un componente, autorizada o no, será registrada en las trazas del sistema.
- El sistema permitirá a las personas el acceso a los datos que se correspondan con el rol que desempeñen en el sistema. Los administradores podrán crear cuentas de usuarios.

Requerimientos de Rendimiento:

El sistema deberá tener un tiempo de respuesta de 2 segundos para un total de 450 usuarios conectados concurrentemente, garantizando de esta forma la agilidad del mismo.

Requerimientos de Soporte:

El sistema debe tener un manual de usuario con el objetivo de guiar al usuario durante el uso de la aplicación. Los servicios de instalación del sistema serán responsabilidad del administrador en la entidad que sea utilizado, además se encargará del mantenimiento que se realizará cada 6 meses.

Requerimientos de Disponibilidad:

Se debe garantizar el funcionamiento de la aplicación durante las 24 horas del día y los siete días de la semana, con el menor tiempo posible de recuperación ante fallos, para ello el servidor de base de datos tendrá un servidor esclavo al cual se replicará en tiempo real cualquier cambio en el servidor principal o maestro.

Requerimientos de Usabilidad:

La aplicación informática debe garantizar un acceso fácil y rápido, contando con un menú que satisfaga las necesidades de los usuarios. Este podrá ser usado por cualquier persona que posea conocimientos básicos en el manejo de una computadora y del ambiente web.

Estándares generales de base de datos y codificación

El uso de los estándares generales de base de datos y codificación permite una mejor comunicación entre los diseñadores y desarrolladores creando las condiciones para la re-usabilidad y el mantenimiento del sistema. Los estándares generales quedaron bien explícitos, cada uno de los aspectos fundamentales posee una breve descripción y ejemplos para una rápida adaptación y comprensión por parte del equipo de desarrolladores. [Ver artefactos definidos en el grupo: Estándares de Codificación y Estándares de Base de Datos].

2.3 Tecnologías y herramientas informáticas

Las tecnologías y herramientas informáticas son de gran importancia a la hora de desarrollar un software, posibilitando obtener un mejor resultado en poco tiempo, siempre y cuando se realice un amplio estudio de las herramientas y lenguajes a utilizar que se correspondan a las características del software que se va a realizar.

2.3.1 Sistema operativo

GNU/Linux es un sistema operativo que se distribuye bajo la licencia General Public License (GPL). El sistema lo forman el núcleo del sistema, un gran número de programas y librerías que hacen posible su utilización, estas características proveen una gran seguridad, estabilidad y un buen desempeño.

Ubuntu es una distribución principalmente enfocada a ordenadores de escritorio aunque también proporciona soporte para servidores, esta se centra en facilitar al máximo una mayor comodidad al usuario, es decir, fomentar la versión de escritorio. Está basado en Debian GNU/Linux, concentrando su objetivo en la facilidad de uso, la libertad de uso, los lanzamientos regulares (cada 6 meses) y la facilidad en la instalación.

Debian es una distribución GNU/Linux que ofrece un sistema operativo que intenta centrarse en que su versión estable, o sea, sin ningún tipo de fallos y que las actualizaciones estén más que probadas, normalmente se dice que la rama estable de Debian está destinada únicamente a servidores en producción. Presenta una gran comunidad de desarrolladores la cual da soporte a todas sus versiones

tanto estables como de prueba, haciendo énfasis en la primera. Tiene un ciclo de desarrollo el cual está enfocado principalmente a la estabilidad del sistema de manera general.

Después de un estudio realizado se escoge como sistema operativo Debian 6.0, ya que presenta como la ventaja más importante que está principalmente enfocado para servidores. Debian es una de las distribuciones GNU/Linux más maduras y estables. Presenta una comunidad de usuarios que la soportan, donde es posible encontrar multitud de información, ya que es de las más activas.

2.3.2 Control de versiones

El control de versiones es el arte de administrar los cambios que sufre la información. Es una actividad cuyo objetivo primordial es facilitar, informar y mantener estructurados los avances, retrocesos y modificaciones que sufre el software mientras es desarrollado. Los sistemas de control de versiones facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas.
Los principales sistemas de control de versiones de código abierto existente son el Sistema de Versiones Concurrentes (CVS por sus siglas en inglés) y Subversion.

El **CVS** es una aplicación que implementa un sistema de control de versiones, difundido bajo la licencia GPL. Este sistema utiliza una arquitectura cliente-servidor: un servidor guarda la(s) versión(es) actual(es) del proyecto y su historia, los clientes se conectan al servidor para sacar una copia completa del proyecto, trabajar en esa copia y entonces ingresan sus cambios, incrementándose automáticamente los números de versión de todos los ficheros implicados.

Subversion (**SVN**) es un software de sistema de control de versiones diseñado específicamente para reemplazar al popular CVS. Una característica importante de SVN es que, a diferencia de CVS, los archivos versionados no tienen cada uno un número de revisión independiente. En cambio, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en un instante determinado. Posibilita a cierto nivel, que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomentando así la colaboración. Se puede progresar más rápidamente sin un único conducto por el cual deban pasar todas las modificaciones. [22]

Las ventajas que presenta SVN son en su mayoría mejoras a CVS: [23]

- El seguimiento de la historia de cambios sufridos por los archivos involucra copias y cambios de nombre.

- Las modificaciones en varios archivos y carpetas son atómicas, es decir, si por algún motivo uno de los archivos que son parte de la actualización no puede ser modificado entonces toda la operación es cancelada. Esto permite tener siempre datos consistentes.
- Al realizar operaciones de sincronización entre los datos de los programas clientes y el servidor SVN sólo se transmiten aquellos archivos que han sufrido cambios, no todos los archivos. Esto redunda en un ahorro del uso del tráfico de red.
- La creación de ramas y etiquetas es una operación más eficiente, ya que por cada rama no crea una nueva copia en el depósito de archivos, sino que utiliza un árbol diferencial de cambios con el cual conoce qué archivos pertenecen a una rama.
- Permite bloquear archivos o carpetas individualmente, para evitar que sean editados por más de un usuario. Esto es usado generalmente en la edición de archivos binarios.
- Siguiendo con los archivos binarios, es capaz de mantener el control de las diferencias entre archivos binarios, con lo cual consigue ahorrar espacio en el repositorio de archivos.
- Se integra fácilmente a un servidor web Apache, y por tanto, puede utilizar sus opciones para la definición de controles avanzados y de navegación del depósito de archivos vía web.

A continuación se muestra la comparación realizada entre los sistemas de control de versiones, en los que se reflejaron aspectos importantes para los mismos. [24]

Tabla 2: Comparación entre las herramientas de control de versiones.

Sistemas de versión de controles	Operaciones con el repositorio	Mover, renombrar y copiar ficheros		Administración de repositorios remotos		Acceso al repositorio	
		Mover/ renombrar	Copiar	Clonar a local un repositorio remoto	Propagar cambios de un repositorio a otro	Permisos acceso	Forma de acceso remoto
CVS	No realiza	No realiza	No realiza	No realiza	No realiza	Limitado	Servidor propio/ Tunel SSH
SVN	Si realiza	Si realiza	Si realiza	Si realiza	Si realiza	Si realiza	WebDAV/ Propio/ Tunel SSH

Después de realizar un estudio de ambos sistemas de control de versiones se selecciona SVN por todas las ventajas antes mencionadas, ya que constituyen mejoras al propio sistema de control de versiones CVS.

2.3.3 Herramienta CASE

Las herramientas CASE (Computer Aided Software Engineering), son un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un Software.

ArgoUML es un entorno gráfico de diseño, desarrollo y documentación de software orientado a objetos, escrita en Java y publicada bajo la Licencia BSD. Permite crear y guardar la mayoría de los nueve diagramas UML compatibles con los estándares de la versión 1.4 de este lenguaje, así como generar diagramas UML a partir de código Java compilado, genera ficheros PNG, GIF, JPG, SVG, EPS desde diagramas. No es conforme completamente a los estándares UML y carece de soporte completo para algunos tipos de diagramas incluyendo los Diagrama de secuencia y los de colaboración.

Visual Paradigm for UML Community Edition (VP-UML CE) es una herramienta CASE para modelamiento UML muy potente, gratuita, fácil de instalar, utilizar y actualizar. Te permite dibujar todo tipo de diagramas UML, revertir código fuente a modelos UML, generar código fuente desde los diagramas UML y mucho más. Incluye los objetos más recientes de UML además de diagramas de casos de uso, diagramas de clase, diagramas de componentes, reversa instantánea para Java, C++,Python, XML, entre otros, ofrece soporte para Rational Rose, integración con Microsoft Visio, además permite generar reportes y documentación en HTML/PDF.
El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de y proyectos de este tipo.

Se escoge como herramienta CASE Visual Paradigm, esta utiliza UML como lenguaje de modelado, cuenta con abundante documentación, permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Es una herramienta que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue.

2.3.4 Lenguaje de Programación

Un lenguaje de programación es un idioma artificial para expresar acciones que pueden ser llevadas a cabo por máquinas como las computadoras. Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo

de comunicación humana. Está formado por un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones.

PHP es un lenguaje de script interpretado en el lado del servidor utilizado para la generación de páginas web dinámicas. Para su funcionamiento necesita tener instalado Apache o IIS con las librerías de PHP. Es fácil de aprender, se caracteriza por ser un lenguaje rápido, es multiplataforma, libre, por lo que se presenta como una alternativa de fácil acceso para todos. No requiere definición de tipos de variables ni manejo detallado del bajo nivel. Todo el trabajo lo realiza el servidor y no delega al cliente, por tanto puede ser más ineficiente a medida que las solicitudes aumenten de número. Dificulta la organización por capas de la aplicación.

JavaScript este es un lenguaje interpretado y no requiere compilación. Utilizado principalmente en páginas web. La mayoría de los navegadores en sus últimas versiones interpretan código JavaScript. Permite la programación de pequeños scripts, pero también de programas más grandes, orientados a objetos, con funciones y estructuras de datos complejas. Además, JavaScript pone a disposición del programador todos los elementos que forman la página web, para que éste pueda acceder a ellos y modificarlos dinámicamente. El código se ejecuta en el cliente por lo que el servidor no es solicitado más de lo debido; un script ejecutado en el servidor, sin embargo, sometería a éste a dura prueba y los servidores de capacidades más limitadas podrían resentirse de una continua solicitud por un mayor número de usuarios.

Python es un lenguaje de programación de alto nivel que últimamente está ganando gran popularidad entre los desarrolladores web. Este lenguaje está caracterizado por definir una sintaxis muy limpia y legible, a la vez que consigue tener una elasticidad perfecta para la web. Es un lenguaje de scripting independiente de plataforma y orientado a objetos, preparado para realizar cualquier tipo de programa, desde aplicaciones Windows a servidores de red o incluso, páginas web. Este lenguaje tiene tipado dinámico, lo que quiere decir que no hace falta preocuparse de declarar los tipos de datos de las variables. Es un lenguaje multiplataforma. Originalmente se desarrolló para Unix, aunque cualquier sistema es compatible con el lenguaje siempre y cuando exista un intérprete programado para él. Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo, lo que ofrece ventajas como la rapidez de desarrollo. La sintaxis de Python es concisa pero expresiva, lo que hace que haga falta menos código para llevar a cabo una tarea que en otros lenguajes más verbosos como Java, por ejemplo, 1 línea de Python suele equivaler a 10 líneas de Java.

Después de un estudio realizado sobre los lenguajes de programación se decidió optar por Python 2.6 ya que presenta ventajas que lo diferencia y hacen más eficiente que otros lenguajes analizados, estas son la cantidad de librerías que contiene, tipos de datos y funciones incorporadas en el propio lenguaje, que ayudan a realizar muchas tareas habituales sin necesidad de tener que programarlas desde cero. Es gratuito, incluso para propósitos empresariales, los programas se crean con sencillez y velocidad. Se puede desarrollar en varias plataformas, como Unix, Windows, Mac y otros. Un programa en Python puede tener de 3 a 5 líneas de código menos que su equivalente en Java o C. Además, se utiliza JavaScript para el lado del cliente ya que es un lenguaje de scripting seguro y fiable. Permite mejoras en la interfaz de usuario y páginas web dinámicas. Su uso en aplicaciones externas a la web, por ejemplo en documentos PDF y aplicaciones de escritorio es también significativo. Es el lenguaje de scripting por excelencia y sin lugar a dudas el más usado

2.3.5 Entorno de Desarrollo

Un entorno de desarrollo integrado (por sus siglas en inglés IDE) es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes, además de dedicarse en exclusiva a un sólo lenguaje de programación o bien poder utilizarse para varios.

Eclipse

Eclipse es un IDE para todo tipo de aplicaciones, inicialmente desarrollado por IBM, y actualmente gestionado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto. La característica clave de Eclipse es la extensibilidad. Eclipse es una gran estructura formada por un núcleo y muchos plugins que van conformando la funcionalidad final. La forma en que los plugins interactúan es mediante interfaces o puntos de extensión. Los lenguajes con proyectos más importantes son: Java, PHP y C/C++. Además, existen paquetes que ayudan también con lenguajes para Web como HTML y CSS.

Aptana

Al hablar de entornos multiplataforma muchos piensan en Netbeans, Visual Studio o en Eclipse, para este caso Aptana posee buenas cualidades, ya que muchos de los desarrollares de Python, Ruby, XHTML, PHP, ASP y plataformas como IPHONE y ADOBE AIR, han probado la comodidad de este IDE. Está basado en Eclipse, combinado con un conjunto de plugins que hacen que este entorno de desarrollo tenga muchas ventajas de personalización. Aptana es gratuito y está disponible para las

versiones de Windows, Linux y Mac. Además de contar con un portal de videos conforme a proyectos de las plataformas de desarrollo. Soporta las librerías más populares: Prototype, Scriptaculous, Dojo, MochiKit, Yahoo UI, Aflax, JQuery y Rico, pudiendo combinarlas fácilmente en la aplicación. Se integra con cualquier navegador y cuenta con documentación completa en línea de CSS, HTML y Javascript, al estilo de JavaDoc o PHPDoc.

Después de un estudio realizado se decidió optar por el IDE Aptana 3.0 el cual presenta ventajas que le son de gran utilidad a nuestra aplicación, por las características que el mismo presenta. Además de que el IDE por si solo presenta ventajas muy importantes a la hora de utilizarlo ya que permite editar fácilmente HTML, CSS y JavaScript, facilita el trabajo del programador ya que mientras se escribe el código JavaScript, indica los parámetros que requiere, en qué navegadores está soportada cada función, objeto y método. Éste permite escribir la documentación en ficheros separados del código JavaScript, por lo que el peso de los ficheros no se ve incrementado.

2.3.6 Framework base

Un framework o marco de trabajo de software es un diseño reutilizable para un sistema de software. Esto se expresa como un conjunto de clases abstractas y el modo en que sus instancias colaboran para un tipo específico de software. Un framework de desarrollo puede incluir programas de soporte, librerías de código, códigos script, u otras aplicaciones que ayuden al desarrollo. Utilizar un framework en el desarrollo de la aplicación permite aprovechar todo lo que una reutilización puede aportar desarrollo más rápido al incorporar al sistema componentes ya construidos, disminución de riesgos al utilizar un diseño y código ya probados.

Zend Framework (ZF) es un marco de trabajo de código abierto orientado a objetos, implementado en PHP 5. Tiene como objetivo simplificar el desarrollo web encapsulando al mismo tiempo las mejores prácticas de programación en PHP, y provee componentes para el patrón MVC. Entre los componentes que proporciona se encuentran aquellos que se utilizan con frecuencia en aplicaciones web, incluyendo la autenticación y autorización a través de listas de control de acceso (ACL), la configuración de aplicación, los datos de la memoria caché, el filtrado y validación de los datos proporcionados por el usuario, la internacionalización, las interfaces para Ajax, así como la composición y entrega de correo electrónico.

Django es un framework de desarrollo web de código abierto, escrito en Python, que cumple en cierta medida el paradigma del Modelo Vista Controlador. La meta fundamental es facilitar la creación de sitios web complejos. Además, pone énfasis en la re-usabilidad, la conectividad y extensibilidad de

componentes, el desarrollo rápido y del principio de DRY (del inglés Don't Repeat Yourself). Python es usado en todas las partes del framework, incluso en configuraciones, archivos, y en los modelos de datos. Django mantiene de forma rigurosa un diseño limpio en su propio código, y facilita que el programador siga las mejores prácticas de desarrollo web en las aplicaciones que crea.

Características:

- Django es 100% Python.
- Se incluye una interfaz de administración como parte del proyecto.
- Un mapeador objeto-relacional.
- Aplicaciones "enchufables" que pueden instalarse en cualquier página gestionada con Django.
- Una API de base de datos robusta.
- Un sistema incorporado de "vistas genéricas" que ahorra tener que escribir la lógica de ciertas tareas comunes.
- Un sistema extensible de plantillas basado en etiquetas, con herencia de plantillas.
- Un despachador de URLs basado en expresiones regulares.
- Soporte de internacionalización, incluyendo traducciones incorporadas de la interfaz de administración.
- Documentación incorporada accesible a través de la aplicación administrativa (incluyendo documentación generada automáticamente de los modelos y las bibliotecas de plantillas añadidas por las aplicaciones).

El patrón conocido como Modelo-Vista-Controlador (MVC) separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes. Django sigue este mismo principio, pero renombra el patrón a MTV, por el simple hecho de que el trabajo del controlador (C en MVC) es manejado por el framework por sí mismo, y la mayoría del trabajo en él ocurre en los modelos, las platillas y las vistas.

- **Modelo:** Una de las partes más potentes de Django, su modelo de datos. Cada uno de los modelos creados se mapean en diferentes tablas en la base de datos. Esto permite aislar la base de datos del código y olvidarte de los diferentes select y updates a veces tan tediosos.
- **Template:** La capa de presentación se basa en plantillas HTML. Django presenta un template engine y un template loader muy potente que permite presentar al usuario diversas páginas HTML usando una base como plantilla. Esto es posible porque en cada una de las plantillas se pueden introducir determinadas etiquetas Django que el template loader se encargará de interpretar.

- **Vista:** Puede llevar a la confusión, aunque Django lo llame vistas, estas son las que actúan como controlador. Escritas en puro código Python cada vista atenderá una petición HTML según el mapeo de URL.

Después de un análisis realizado se decidió utilizar como framework base Django v1.2 ya que coincide con el lenguaje de programación seleccionado con anterioridad, además de ser rápido de desarrollar. Está pensado para la eficiencia, es modular, sus bibliotecas hacen gran parte del trabajo, soporta varias bases de datos (MySQL, SQLite, Postgres). Potente mapeo objeto-relacional (ORM por sus siglas en ingles), infinita flexibilidad, encierra las mejores prácticas, diferencia el GET del POST, eficiencia SQL, sintaxis concisa y poderosa, opción de escribir SQL crudo cuando se necesita y permite el desarrollo en equipo.

2.3.7 Framework GUI

Un framework de interfaz gráfica de usuario (GUI por sus siglas en inglés) es un diseño reutilizable para un sistema de software, un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y las acciones disponibles en la interfaz. Su principal uso, consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computador.

Prototype es un framework escrito en JavaScript que nos permite manejar elementos de la página web de una forma muy sencilla y dinámica. Es por ello que nos es muy útil cuando interactuamos con formularios. Es una herramienta que implementa las técnicas AJAX. Está orientado a proporcionar al desarrollador de técnicas AJAX listas para ser usadas. El potencial de este es aprovechado al máximo si se desarrolla con Ruby On Rails, esto no quiere decir que no se puede usar desde otro lenguaje, solamente que demandara un "mayor esfuerzo" en el desarrollo.

jQuery es una biblioteca o framework de JavaScript, que permite simplificar la manera de interactuar con los documentos HTML, manejar eventos, desarrollar animaciones y agregar interacción con la tecnología AJAX a páginas web, es software libre y de código abierto. Al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en JavaScript que de otra manera requerirían de mucho más código, es decir, con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio. Lo que la hace tan especial es su sencillez y su reducido tamaño.

Ventajas: [25]

- Es ligero en comparación con otros marcos de JavaScript.

- Tiene una amplia gama de plugins disponibles para las diversas necesidades específicas.
- Ahorra muchas líneas de código.
- Hace transparente el soporte de nuestra aplicación para los navegadores principales.
- Provee de un mecanismo para la captura de eventos.

Después de realizarse un estudio sobre las herramientas existentes se decide usar jQuery v1.4.2, ya que tiene grandes ventajas, este provee un conjunto de funciones para animar el contenido de la página en forma muy sencilla. Tiene una gran comunidad de desarrollo, además de tener muchos plugins y proporcionar un buen soporte.

2.3.8 Servidor de aplicación (web)

Un servidor web es un programa que se ejecuta continuamente en una computadora, manteniéndose a la espera de peticiones de ejecución que le hará un cliente o un usuario de Internet. El servidor web se encarga de contestar a estas peticiones entregando como resultado una página web, algún tipo de servicio o información de acuerdo con los comandos solicitados.

Apache provee un mecanismo para ofrecer la aplicación Web a los usuarios. Es estable y tiene la capacidad para soportar aplicaciones críticas. Corre en una multitud de sistemas operativos, lo que lo hace prácticamente universal. Constituye una tecnología gratuita de código fuente abierta. Apache es un servidor altamente configurable de diseño modular. Es muy sencillo ampliar sus capacidades mediante la integración o la construcción de módulos. Permite personalizar la respuesta ante los posibles errores que se puedan dar en el servidor. Es posible configurar Apache para que ejecute un determinado script cuando ocurra un error en concreto.

Nginx es un servidor web/proxy inverso ligero de alto rendimiento. Es software libre, de código abierto y multiplataforma, por lo que corre en sistemas GNU/Linux, Solaris, Mac y Windows. Originalmente el sistema fue diseñado para poder atender las más de 500 millones de peticiones que recibían los servidores de Rambler (empresa dedicada a la construcción de autos). Es conocido por su estabilidad, gran conjunto de características, configuración simple, y bajo consumo de recursos.

Características básicas servidor web

- Servidor de archivos estáticos, índices y auto indexado.
- Proxy inverso con opciones de caché.
- Balance de carga.
- Tolerancia a fallos.

- Soporte de HTTP sobre SSL.
- Servidores virtuales basados en nombre y/o en dirección IP.
- Soporte para autenticación.
- Habilitado para soportar más de 10.000 conexiones simultáneas.

Se decide utilizar para la realización de esta aplicación el servidor de aplicaciones Nginx 0.8 ya que presenta grandes ventajas, es realmente fácil de configurar, su rendimiento y consumo de memoria son excepcionales, está disponible en variedad de plataformas, elimina la carga de trabajo de CPU asociada al procesamiento de SSL (Protocolo de Capa de Conexión Segura) y permite un balanceo de carga más simple ya que resuelve la necesidad de sesiones SSL persistentes.

2.3.9 Estándar de interoperabilidad

Son características de los ordenadores que les permite su interconexión y funcionamiento conjunto de manera compatible. Esto no siempre es posible, debido a los diferentes sistemas operativos y arquitecturas de cada sistema, pero los esfuerzos de estandarización están permitiendo que cada vez sean más los ordenadores capaces de inter-operar entre sí.

XML es un metalenguaje extensible de etiquetas. Permite definir la gramática de lenguajes específicos. Por lo tanto, XML es una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML. Se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

JSON es un formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo. Está basado en un subconjunto del Lenguaje de Programación JavaScript, es un formato de texto que es completamente independiente del lenguaje pero utiliza convenciones que son ampliamente conocidos por los programadores de la familia de lenguajes C, incluyendo C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos.

Características:

- JSON proporciona una mejor representación independiente del lenguaje de las estructuras de datos.

- JSON tiene una especificación más simple.
- JSON maneja adecuadamente actualización de contenido mixto.

Después de un estudio sobre los estándares existentes se escoge JSON ya que posee un mejor formato para intercambiar datos de forma ligera, organizada, de fácil acceso y comprensión por parte de los humanos y las máquinas, su procesamiento por parte de los ordenadores es rápido, se necesitan librerías muy pequeñas para trabajar con él dada su naturaleza es ideal para entornos Ajax, y existen parsers para este formato en más de 20 lenguajes de programación.

2.3.10 Sistema gestor de base de datos

Un Sistema Gestor de base de datos (SGBD) es un conjunto de programas que permiten crear y mantener una base de datos, asegurando su integridad, confidencialidad y seguridad. Tienen como principal propósito manejar de manera clara, sencilla y ordenada un conjunto de datos que posteriormente se convertirán en información relevante para una organización.

MongoDB es una base de datos de código abierto, de gran rendimiento, escalable y NoSQL orientada a documentos (esquemas de datos tipo JSON). Se puede usar desde lenguajes como PHP, Python, Perl, Ruby, JavaScript, C++ y muchos más. En MongoDB lo más parecido a una tabla son las colecciones, que vienen a ser una especie de listas donde se almacenan los diferentes objetos y sus atributos. Se abandona el enfoque relacional por bases de datos más orientadas a objetos y de esta manera es como se procesa la información. Combina la capacidad de escala con muchas de las características más útiles de las bases de datos relacionales, tales como índices secundarios, rangos, y la clasificación.

PostgreSQL es un poderoso sistema manejador de bases de datos para administrar grandes cantidades de datos, es hasta ahora la base de datos de código abierto más avanzada del mundo. Se ejecuta en la mayoría de los Sistemas Operativos incluyendo, GNU/Linux, varias versiones de UNIX y por supuesto Windows. PostgreSQL se ha preocupado por ser una solución real a los complejos problemas del mundo empresarial y a la vez mantener la eficiencia al consultar los datos. Con ese fin, se han desarrollado y añadido al PostgreSQL las más interesantes y útiles características que antes sólo podían hallarse en sistemas manejadores de bases de datos comerciales como Oracle, DB2 o Sybase. Tiene como características principales:

- Incorpora una estructura de datos arreglos.
- Incorpora funciones de diversa índole: manejo de fechas, geométricas, orientadas a operaciones con redes, etc.

- Permite la declaración de funciones propias, así como la definición de disparadores.
- Soporta el uso de índices, reglas y vistas.
- Incluye herencia entre tablas (aunque no entre objetos, ya que no existen), por lo que a este gestor de bases de datos se le incluye entre los gestores objeto-relacionales.
- Permite la gestión de diferentes usuarios, como también los permisos asignados a cada uno de ellos.

Para la selección de los sistemas gestores de base de datos se analizaron varios de ellos, escogiéndose por las características antes mencionadas PostgreSQL 8.4 ya que presenta ventajas como su costo de adquisición el cual es nulo, no presenta problemas de licencias y en las comunidades se brinda soporte gratuito y participan programadores del mismo motor de la base de datos. Además, se escoge el gestor MongoDB por ser una gran herramienta para el seguimiento de métricas en tiempo real por su rico pero sencillo sistema de consulta a la información contenida en la base de datos. Posee un balance perfecto entre rendimiento y funcionalidad, incorporando muchos de los tipos de consulta existentes en la actualidad. El rendimiento de MongoDB de actualización es muy bueno, lo que facilita realizar varias actualizaciones conjuntamente por solicitud de análisis.

2.3.11 Metodología de desarrollo

Para definir la metodología de desarrollo del proyecto PostgreSQL Empresarial se realizó un estudio intensivo de las metodologías ágiles existentes, donde se escogió Extreme Programming (XP), ya que es la metodología que más se adapta a las características del proyecto. [26]

XP es una de las metodologías de desarrollo de software más exitosa en la actualidad utilizada para proyectos de corto plazo y pequeños equipos. La metodología consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto.

Esta metodología se basa en: [27]

- Pruebas Unitarias: Se basa en las pruebas realizadas a los principales procesos, de tal manera que adelantándonos en algo hacia el futuro, podamos hacer pruebas de las fallas que pudieran ocurrir. Es como si nos adelantáramos a obtener los posibles errores.
- Re-fabricación: Se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.

- Programación en pares: Una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento.

Lo fundamental en este tipo de metodología es: [27]

- La comunicación, entre los usuarios y los desarrolladores.
- La simplicidad, al desarrollar y codificar los módulos del sistema.
- La retroalimentación, concreta y frecuente del equipo de desarrollo, el cliente y los usuarios finales.

Ventajas: [28]

- Se consiguen productos usables con mayor rapidez.
- Mejora continua de los procesos y el equipo de desarrollo.
- El proceso de integración es continuo, por lo que el esfuerzo final para la integración es nulo. Se consigue integrar todo el trabajo con mucha mayor facilidad.
- Se atienden las necesidades del usuario con mayor exactitud. Esto se consigue gracias a las continuas versiones que se ofrecen al usuario.
- Se consiguen productos más fiables y robustos contra los fallos gracias al diseño de los tests de forma previa a la codificación.
- Cada componente del producto final ha sido probado y satisface los requerimientos.
- Obtenemos código más simple y más fácil de entender, reduciendo el número de errores.
- Gracias a la filosofía de la programación en parejas, se consigue que los desarrolladores apliquen las buenas prácticas que se les ofrecen con la XP.
- Gracias a la refactorización es más fácil el modificar los requerimientos del usuario.
- Minimiza los costos frente a cambios.
- Conseguimos tener un equipo de desarrollo más alegre y motivado. Las razones son la comunicación entre los miembros del equipo que consigue una mayor integración entre ellos.

Esta metodología es la que mejor se ajusta a las necesidades y condiciones de un equipo de trabajo pequeño y que cuenta con un corto período de tiempo para el desarrollo de una solución final, siendo estas las características del proyecto [26]. Todo esto es posible gracias a la sencillez que presenta XP, tanto en su aprendizaje como en su aplicación, reduciendo los costos de implantación en un equipo de desarrollo. Se está preparado para el cambio, significando una reducción en su coste. La planificación

es más transparente para los clientes ya que conocen las fechas de entrega de las funcionalidades vitales para su negocio. Permite definir en cada iteración cuáles son los objetivos de la siguiente y tener una retroalimentación por parte de los usuarios. La presión está a lo largo de todo el proyecto y no en una entrega final.

Arquitectura de software en XP

En investigaciones realizadas sobre cómo se comporta la arquitectura de software en XP se obtuvo como resultado que en esta metodología no se define un rol de arquitecto para el equipo de desarrollo. En XP la arquitectura se define mediante una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo. Una metáfora es una historia que describe cómo debería funcionar el sistema.

Por la importancia que tiene definir una arquitectura para Naire, con el objetivo de lograr la integración de los componentes de Reporte, Planificador de Tareas y Monitoreo, se propone el rol arquitecto de software para incluirse en la metodología de desarrollo XP en el caso particular del entorno de trabajo del Servidor de Gestión. Definiéndose las siguientes actividades:

- Analizar la arquitectura.
- Describir la distribución que va a tener el sistema.
- Priorizar las funcionalidades del sistema.
- Seleccionar las herramientas y metodologías necesarias para la elaboración de la solución.
- Representar la arquitectura mediante las vistas arquitectónicas.
- Evaluar la arquitectura propuesta.
- Responder sobre las inquietudes relacionadas con la selección de herramientas y ambientes de desarrollo.

Cada rol por sí mismo genera una serie de artefactos que van a ser los productos tangibles del mismo, ya que mediante estos queda constancia del trabajo realizado por el arquitecto, por lo que se define como artefacto:

- Documento de arquitectura de software: El propósito principal de este documento es describir la arquitectura del proyecto proporcionando la información necesaria para estructurar el sistema desde el más alto nivel de abstracción. En este se describe la estructura del sistema en cuanto a los elementos, los conectores, las configuraciones y sus restricciones. Visualizar los elementos arquitectónicamente significativos en el desarrollo del sistema, dando a conocer los

diferentes patrones y artefactos que lo componen, así como las herramientas y metodologías a utilizar en la implementación del sistema.

2.4 Vistas arquitectónicas

Las vistas de la arquitectura constituyen un extracto de los modelos que están en la línea base de la arquitectura y se representa a través de 4+1 vistas arquitectónicas. Una vista es la representación de un conjunto coherente de elementos arquitectónicos y sus relaciones, en este sentido, la vista describe parte de la arquitectura del sistema. El modelo 4+1 se percibe hoy como un intento de reformular una arquitectura estructural y descriptiva en términos de objeto y de UML. De acuerdo con la metodología que se está llevando a cabo en el proyecto que no define las vistas de arquitectura, se decidió definir de acuerdo con las necesidades y características del mismo como vistas de arquitectura a las siguientes:

- Vista de historia de usuario.
- Vista lógica.
- Vista de despliegue.
- Vista de implementación.
- Vista de procesos.

Vista de Historias de usuarios (Rectora)

En esta vista se muestra la percepción que tiene el usuario de las funcionalidades del sistema mediante la representación de los actores e historias de usuarios más importantes. Si el sistema se hace extenso entonces se debe organizar en paquetes, lo cual facilitaría la comprensión de la vista.

Para priorizar las historias de usuarios del sistema, es necesario determinar cuáles son necesarias para el desarrollo en las primeras iteraciones y cuáles pueden dejarse para más tarde. Los resultados se recogen en la presente vista de la arquitectura.

Muy alta: Constituyen las historias de usuarios más importantes para los usuarios, porque cubren las principales tareas o funciones que el sistema ha de realizar, definen la arquitectura básica.

Alta: Sirven de apoyo a las historias de usuarios que tengan una prioridad muy alta, involucran funciones secundarias y tienen un impacto más modesto sobre la arquitectura, pero deben implementarse pronto porque responden a los requerimientos de interés para los usuarios.

Media: No son claves para la arquitectura y complementan historias de usuarios con prioridad muy alta y alta.

Baja: Responden a funcionalidades que pueden o no estar en la aplicación, pero que no son imprescindibles en las primeras versiones

Figura 4: Organización por paquete.

Vista Lógica

La vista lógica representa los elementos de diseño más importantes para la arquitectura del sistema. Esta describe las clases más importantes, su organización en paquetes y subsistemas. Representa un conjunto arquitectónicamente significativo de paquetes de alto nivel, subsistemas de diseño e interfaces.

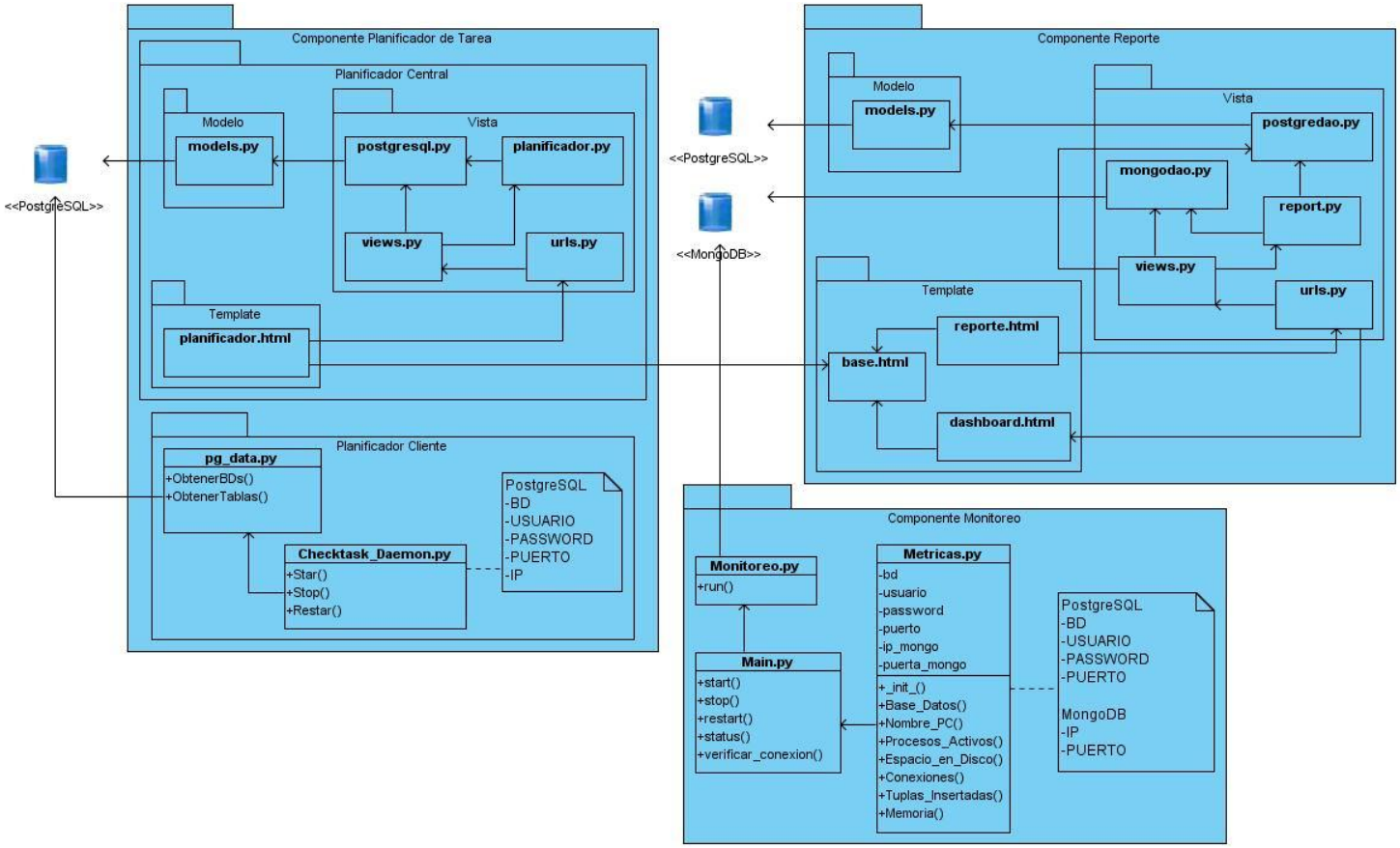


Figura 5: Vista Lógica General.

Paquete Modelo

Contiene las clases que se encargan del acceso a los datos almacenados, cada uno de los modelos creados se mapean en diferentes tablas en la base de datos, esto permite aislar la base de datos del código.

Paquete Template

Contiene las clases .html que son elementos que se muestran de forma idéntica a lo largo de toda la aplicación, es decir, las plantillas que serán presentadas al usuario. Por este motivo, la vista se separa en dos, la base y la plantilla. La clase base contiene los elementos que se muestran de forma idéntica para las páginas web a lo largo de toda la aplicación y la plantilla que contiene la información que se va a mostrar al usuario.

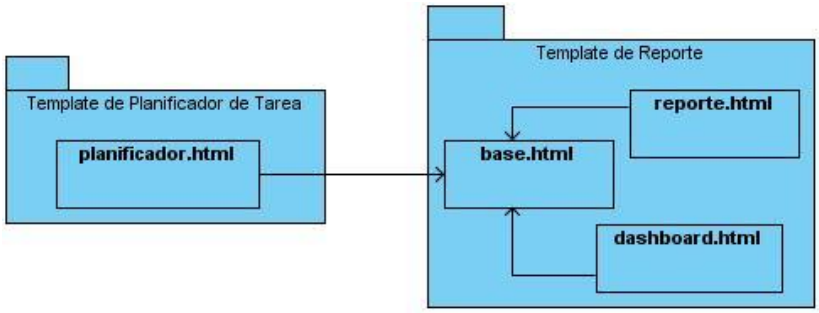


Figura 6: Paquete Template.

Paquete Vista

Agrupa las clases que implementan la lógica de la aplicación, cada vista atenderá una petición HTML según el mapeo de urls.py.

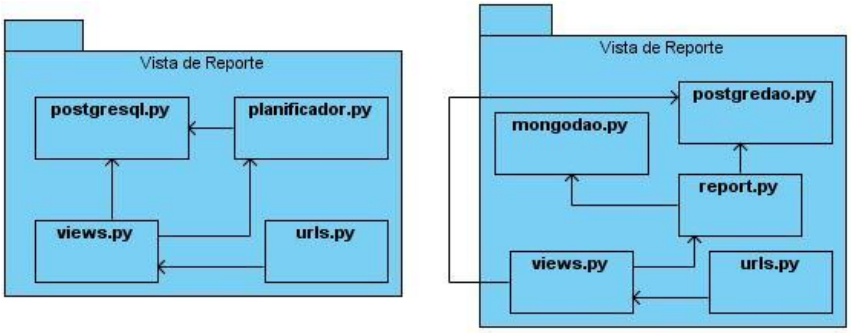


Figura 7: Paquete Vista.

Vista Despliegue

La vista de despliegue define la arquitectura física del sistema por medio de nodos interconectados. Estos nodos son elementos hardware sobre los cuales pueden ejecutarse los elementos software. Describe los principales nodos físicos, ordenadores, así como los dispositivos que se necesitan para configurar la plataforma que pueda soportar la implementación del sistema. Describe la situación de los componentes en una posible implantación del sistema de acuerdo con los requisitos.

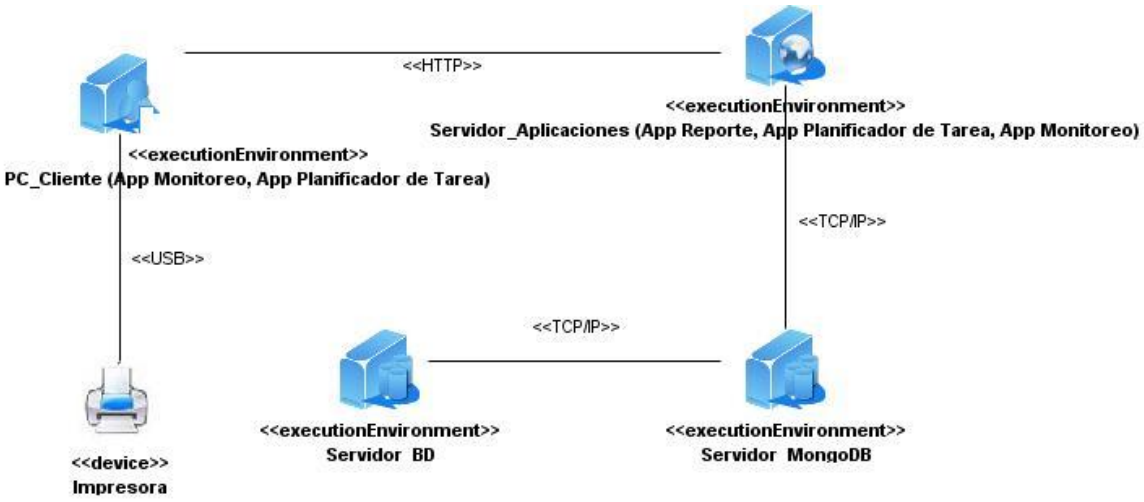


Figura 8: Vista de Despliegue.

PC_Cliente: Se refiere a las estaciones de trabajo que el usuario utilizará para acceder a la aplicación Web y transcribir sus datos.

Impresora: Este dispositivo estará a disposición de la estación de trabajo con la aplicación cliente para visualizar en una hoja impresa los datos requeridos por el cliente.

Servidor_Aplicaciones: Se refiere al servidor utilizado para la publicación de la aplicación y para lograr la conexión del sistema con la PC Cliente. Es el responsable de ejecutar el código de las páginas del servidor, en el cual se encuentran los componentes de Monitoreo, Planificador de Tarea y Reporte, además de una base de datos en PostgreSQL para la autenticación de los usuarios, así como guardar los reportes realizados.

Servidor_BD: Se refiere al servidor que radica en cada nodo regional donde se van a estar centralizados los datos recopilados, en el cual se encontrarán los componentes de Monitoreo y Planificador de Tarea.

Servidor_MongoDB: Se refiere al servidor donde van a estar centralizados los datos recopilados y el cual va a enviar al servidor aplicaciones para que los muestre.

Protocolos de Comunicación

Conexión HTTP: Es el protocolo utilizado entre los browsers de los clientes y el servidor web. Este elemento de la arquitectura representa un tipo de comunicación no orientado a la conexión entre clientes y servidor.

Conexión TCP/IP: Es un protocolo que realiza una labor de intermediario entre internet y el computador personal. El Transmission Control Protocol (TCP por sus siglas en inglés) se encarga de

fragmentar y unir los paquetes y el Internet Protocol (IP por sus siglas en inglés) tiene como misión hacer llegar los fragmentos de información a su destino correcto.

Conexión USB: Es el protocolo para la conexión entre la estación de trabajo y el dispositivo impresora.

Vista Implementación

La vista de implementación muestra el software como los elementos físicos que lo integran: componentes, ficheros, librerías. Contiene la organización de los módulos en términos de paquetes y capas, pueden incluirse también la trazabilidad de la vista lógica. Es representada por un diagrama de componentes o especificaciones de paquetes que son básicamente un subconjunto del modelo de despliegue. Refleja la organización de módulos de software dentro del entorno de desarrollo. Esta vista toma en cuenta los requerimientos que facilitan la programación, los niveles de reutilización y las limitaciones impuestas por el entorno de desarrollo.

La vista de implementación del sistema queda constituida por un grupo de componentes agrupados por paquetes, estos representarán cada una de las clases más significativas de los componentes del sistema.

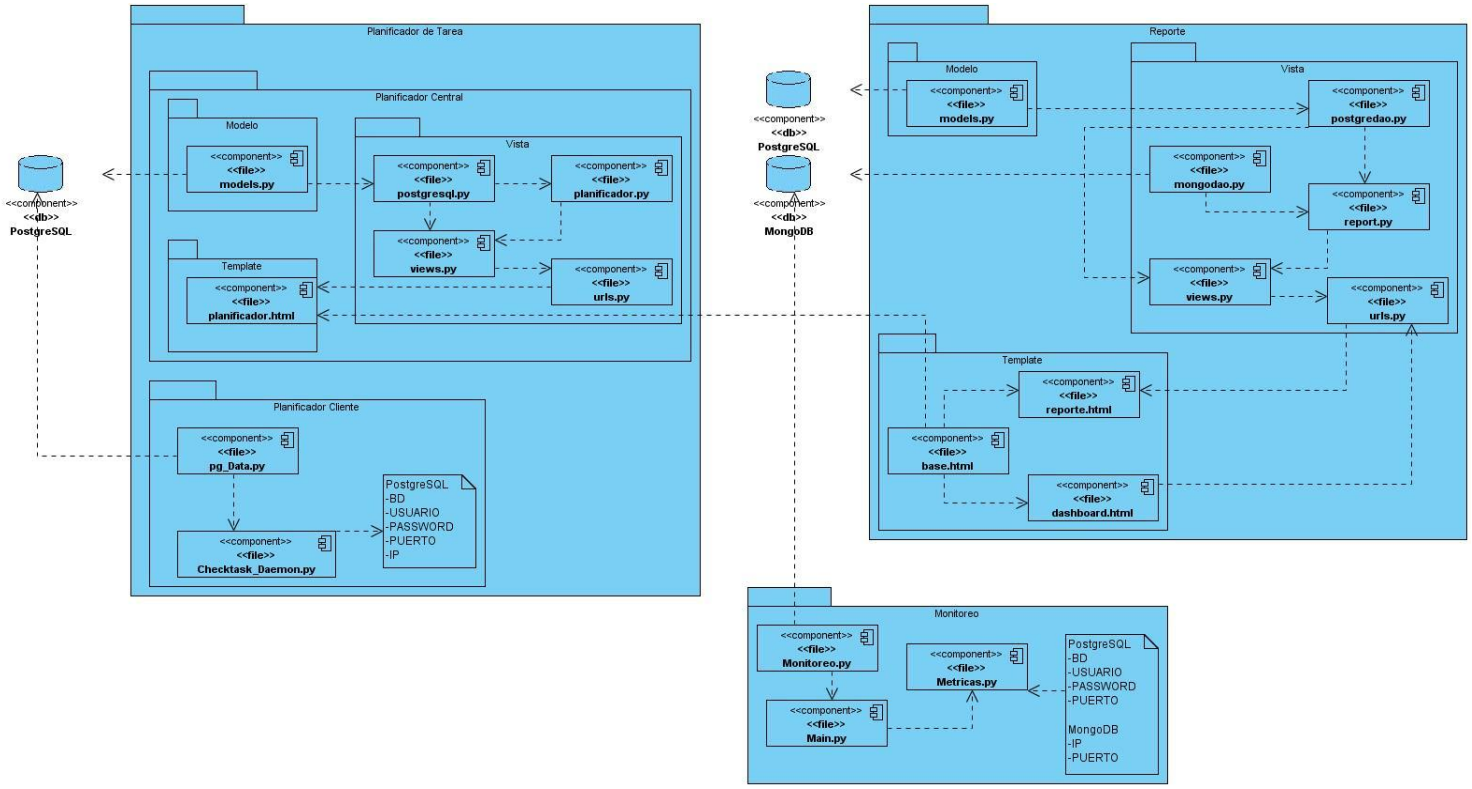


Figura 9: Vista de Implementación detallada.

Componente Reporte:

Urls: Es la encargada mediante una petición http de realizar el re-direccionamiento a la vista correspondiente.

Views: Es el componente controlador el cual define cual es la información que va a ser mostrada.

Report: es la encargada de la realización de todas las consultas sobre MongoDB y obtener la información necesaria de la misma.

Mongodao: Es la conectora entre el sistema y la base de datos MongoDB.

Postgredao: Es la encargada de la realización de todas las consultas sobre PostgrSQl, así como la obtención de todas las informaciones necesarias para la construcción de reportes y autenticación de usuario entre otras funcionalidades.

Models: Cada uno de los modelos creados se mapean en diferentes tablas en la base de batos. Esto permite aislar la base de datos del código.

Dashboard.html: Muestra las tablas, las graficas y los informes de cada uno de los servidores.

Reporte.html: Es donde se genera un histórico de todos los reportes de métricas por servidores.

Base.html: Es la plantilla global que almacena el código HTML común a todas las páginas de la aplicación, para no tener que repetirlo.

Componente Planificador de Tarea:

Urls: Es la encargada mediante una petición http de realizar el re-direccionamiento a la vista correspondiente.

Views: Es la clase controladora la cual define cual es la información que va a ser mostrada.

Planificador: Es la clase donde se gestionan las tareas de cada uno de los servidores.

PostgreSQL: Es la encargada de la realización de todas las consultas sobre PostgreSQL y obtener la información que se va a mostrar.

Models: Cada uno de los modelos creados se mapean en diferentes tablas en la base de datos. Esto permite aislar la base de datos del código.

Reporte: Es donde se genera un histórico de todos los reportes de métricas por servidores.

Planificador.html: Es el listado de las tareas como lo hace el dashboard.

Checktask_Daemon: Es la encargada de definir las tareas que se ejecutarán en los servidores.

PG_Data: Es donde se van a llamar a todas las tareas que se van a mostrar concurrentemente.

Componente Monitoreo:

Métricas: Es la clase encargada de definir las métricas que se recogen de los servidores a monitorear.

Monitoreo: Es donde se van a llamar a todas las métricas que se van a mostrar concurrentemente.

Vista de Procesos

La vista de proceso especifica qué operaciones son ejecutadas por cada una de las clases identificadas en la vista lógica y el flujo de colaboración entre ellas, muestra los principales procesos que el sistema debe tener integrado. Estos procesos se encuentran agrupados por módulos en dependencia de su funcionalidad.

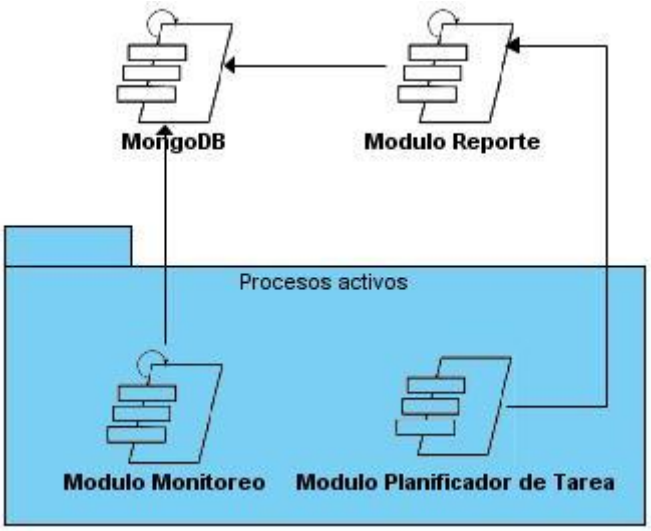


Figura 10: Vista de Procesos.

Esta vista se describe mediante los módulos que conforman el sistema:

Modulo Planificador de tarea: Se encargará de planificar las tareas de las bases de datos en los diferentes hosts en que estará desplegado.

Modulo de Monitoreo: Se encargará de realizar el cálculo de las diferentes métricas como son uso de espacio en disco, procesos activos, estadísticas del servidor y las bases de datos en general.

Modulo de Reporte: Se encargará de proveer el sistema central para la visualización de los reportes y monitoreo de las estadísticas de rendimiento y funcionamiento del servidor además de la planificación de las tareas.

2.5 Conclusiones del capítulo

En este capítulo se presentaron los requisitos del sistema, los cuales tuvieron gran importancia para la realización de la propuesta de solución. A partir de las 26 HU significativas para la arquitectura quedaron definidas las vistas: Lógica, Despliegue, Implementación, de Procesos y de HU las cuales serán la línea base de la arquitectura para el proceso de desarrollo de la solución. Además de analizarse diferentes herramientas entre las que se seleccionaron como metodología de desarrollo XP, para el modelado Visual Paradigm para UML6.4, el lenguaje de programación escogido es Python 2.6,

como IDE de desarrollo Aptana Studio 3.0, para la realización de la aplicación el framework Django v1.2, el servidor de aplicación web Nginx y para la gestión de la base de dato PostgreSQL 8.4 y MongoDB.

3
CAPÍTULO
EVALUACIÓN DE LA ARQUITECTURA

En el capítulo se presentan distintos métodos para evaluar un diseño arquitectónico y a partir del método seleccionado se hace un análisis de la solución propuesta, logrando así la validación de la arquitectura de software definida.

La arquitectura es un artefacto decisivo en la calidad del software que se desarrolla. Su evaluación permite mitigar los diferentes riesgos asociados con el desarrollo del software. Además, de mejorar la visión de los procesos críticos y validar las decisiones de diseño que se tomaron. Así como valorar los atributos de calidad sin esperar a que el software se construya.

3.1 Atributos de calidad

Los atributos de calidad son los aspectos del sistema, que en general, no afectan directamente a la funcionalidad necesaria, sino que definen la calidad y las características que el sistema debe soportar. Estas características o atributos se conocen con el nombre de atributos de calidad, los cuales se definen como las propiedades de un servicio que presta el sistema a sus usuarios. [29]

Los atributos de calidad se pueden clasificar en dos categorías: [11]

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

Dentro de los atributos de calidad observables vía de ejecución se pueden encontrar:

- **Confiabilidad:** Es la habilidad del sistema de continuar operando sobre el tiempo. Es usualmente medida en tiempo promedio entre fallas.
- **Disponibilidad:** Es la porción de tiempo en que el sistema está levantado y corriendo. Se mide como el tiempo transcurrido entre fallas, así como, cuán rápido el sistema está apto para reanudar y quedar operativo ante una falla.

- **Seguridad Interna:** Es la medida de la habilidad del sistema de resistirse al uso no autorizado y negar los servicios, mientras los provee a usuarios legítimos.
- **Seguridad Externa:** Ausencia de consecuencias catastróficas en el ambiente. Es la medida de ausencia de errores que generan pérdidas de información.
- **Funcionalidad:** Es la habilidad del sistema de hacer el trabajo para el cual fue construido.

Dentro de los atributos de calidad no observables vía de ejecución se pueden encontrar:

- **Modificabilidad:** Es la habilidad de realizar cambios al sistema en forma rápida y a bajo costo.
- **Portabilidad:** Es la habilidad del sistema de correr sobre diferentes ambientes. Estos ambientes pueden ser de hardware, de software o una combinación de ambos. Portabilidad es un caso particular de modificabilidad.
- **Variabilidad:** Es la capacidad de la arquitectura de ser expandida o modificada para producir nuevas arquitecturas. Variabilidad es importante cuando la arquitectura se va a utilizar como piedra fundamental de toda una familia de productos relacionados, como ser una línea de producto.
- **Integridad:** Es la ausencia de alteraciones inapropiadas de la información.
- **Configurabilidad:** Posibilidad que se otorga a un usuario experto a realizar ciertos cambios al sistema.
- **Escalabilidad:** Es el grado con el que se pueden ampliar el diseño arquitectónico, de datos o procedimental.
- **Mantenibilidad:** Es la capacidad de someter a un sistema a reparaciones y evolución. Capacidad de modificar el sistema de manera rápida y a bajo costo.
- **Reusabilidad:** Es la capacidad de diseñar un sistema de forma tal que su estructura o parte de sus componentes puedan ser reutilizados en futuras aplicaciones.

3.2 Métodos de evaluación de arquitecturas de software

Kazman proponen que la existencia de un método de análisis de arquitecturas de software hace que el proceso sea repetible, y ayuda a garantizar que las respuestas correctas con relación a la arquitectura pueden hacerse temprano, durante las fases tempranas de diseño; es en este punto donde los problemas encontrados pueden ser solucionados de una forma relativamente poco costosa. De manera similar, un método de evaluación sirve de guía a los involucrados en el desarrollo del sistema, en la búsqueda de conflictos que puede presentar una arquitectura, y sus soluciones. Por esta razón,

resulta conveniente estudiar los métodos de evaluación de arquitecturas de software propuestos hasta el momento. [30]

3.2.1 Método de Análisis de Arquitecturas de Software (SAAM)

El Método de Análisis de Arquitecturas de Software (SAAM por sus siglas en inglés) fue el primer método de evaluación basado en escenarios que surgió. Este método fue originalmente creado para el análisis de la modificabilidad de una arquitectura, pero en la práctica ha demostrado ser muy útil para evaluar de forma rápida distintos atributos de calidad, tales como modificabilidad, portabilidad, escalabilidad e integrabilidad.

Se enfoca en la enumeración de un conjunto de escenarios que representan los cambios probables a los que estará sometido el sistema en el futuro. Como entrada principal, es necesaria alguna forma de descripción de la arquitectura a ser evaluada y las salidas de la evaluación son las siguientes:

- Una proyección sobre la arquitectura de los escenarios que representan los cambios posibles ante los que puede estar expuesto el sistema
- Entendimiento de la funcionalidad del sistema, e incluso una comparación de múltiples arquitecturas con respecto al nivel de funcionalidad que cada una soporta sin modificación

Con la aplicación de este método, si el objetivo de la evaluación es una sola arquitectura, se obtienen los lugares en los que la misma puede fallar, en términos de los requerimientos de modificabilidad. Para el caso en el que se cuenta con varias arquitecturas candidatas, el método produce una escala relativa que permite observar qué opción satisface mejor los requerimientos de calidad con la menor cantidad de modificaciones.

3.2.2 Método de Análisis de Acuerdos de Arquitectura (ATAM)

El Método de Análisis de Acuerdos de Arquitectura (ATAM por sus siglas en inglés) obtiene su nombre no solo porque nos dice cuán bien una arquitectura particular satisface las metas de calidad, sino que también provee ideas de cómo esas metas de calidad interactúan entre ellas, como realizan concesiones mutuas entre ellas.

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

análisis repetible y ayuda a asegurar que las preguntas acerca de una arquitectura serán contestadas en forma temprana, cuando es relativamente económico corregir problemas.

Por tanto, lo que se pretende hacer con ATAM a demás de mejorar la documentación, es registrar los posibles riesgos, los no riesgos, los puntos de sensibilidad y los puntos de desventajas que encontramos en el análisis de la arquitectura.

Riesgos: Las decisiones de arquitectura que podría crear problemas en el futuro para algunos atributos de calidad.

No riesgos: Las decisiones arquitectónicas que sean adecuadas al atributo de calidad que afectan.

Desventaja: Las decisiones de arquitectura que tienen un efecto en más de un atributo de calidad.

Puntos de sensibilidad: Una propiedad de uno o más componentes, y/o las relaciones entre componentes, fundamental para el logro de un determinado requisito de atributo de calidad.

3.2.3 Método de Análisis de Diseños Intermedios (ARID)

El método de Revisiones Activas para Diseños Intermedios (ARID por sus siglas en inglés) es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. El método ARID se basa en las mejores cualidades de los métodos basados en escenarios (ATAM o SAAM) y las revisiones activas de diseños (ADRS).

En ADR, los involucrados en la evaluación reciben documentación detallada y llenan sus cuestionarios de forma independiente. En ATAM se evalúa toda la arquitectura. Dadas las anteriores características y la necesidad de realizar evaluaciones tempranas de los diseños de la arquitectura, el método ARID es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. Las revisiones de diseño activas y los ATAM tienen características útiles para resolver el problema de evaluar diseños preliminares. ARID surge de combinar las mejores cualidades de los métodos ADRS y los métodos basados en escenarios. Este recoge de ADR, la fidelidad de las respuestas que se obtiene de los involucrados en el desarrollo y del ATAM la idea de que sean los involucrados con el sistema los encargados de generar los escenarios.

La Tabla 2 presenta las fases y los pasos que involucra el método de evaluación ARID, con una breve descripción de cada uno. [11]

Tabla 3: Pasos del método de evaluación ARID.

Fase 1: Actividades Previas	
1. Identificación de los encargados de la revisión.	Los encargados de la revisión son los ingenieros de software que se espera que usen el diseño, y todos los involucrados en el diseño. En este punto, converge el concepto de encargado de revisión de ADR e involucrado del ATAM.

2. Preparar el informe de diseño.	El diseñador prepara un informe que explica el diseño. Se incluyen ejemplos del uso del mismo para la resolución de problemas reales. Esto permite al facilitador anticipar el tipo de preguntas posibles, así como identificar áreas en las que la presentación puede ser mejorada.
3. Preparar los escenarios base.	El diseñador y el facilitador preparan un conjunto de escenarios base. De forma similar a los escenarios del ATAM y el SAAM, se diseñan para ilustrar el concepto de escenario, que pueden o no ser utilizados para efectos de la evaluación.
4. Preparar los materiales.	Se reproducen los materiales preparados para ser presentados en la segunda fase. Se establece la reunión, y los involucrados son invitados.
Fase 2: Revisión	
6. Presentación del diseño.	El líder del equipo de diseño realiza una presentación, con ejemplos incluidos. Se propone evitar preguntas que conciernen a la implementación o argumentación, así como alternativas de diseño. El objetivo es verificar que el diseño es conveniente.
7. Lluvia de ideas y establecimiento de prioridad de escenarios.	Se establece una sesión para la lluvia de ideas sobre los escenarios y el establecimiento de prioridad de escenarios. Los involucrados proponen escenarios a ser usados en el diseño para resolver problemas que esperan encontrar. Luego, los escenarios son sometidos a votación, y se utilizan los que resultan ganadores para hacer pruebas sobre el diseño.
8. Aplicación de los escenarios.	Comenzando con el escenario que contó con más votos, el facilitador solicita el seudo-código que utiliza el diseño para proveer el servicio, y el diseñador no debe ayudar en esta tarea. Este paso continúa hasta que ocurra alguno de los siguientes eventos: - Se agota el tiempo destinado a la revisión. - Se han estudiado los escenarios de más alta prioridad. - El grupo se siente satisfecho con la conclusión alcanzada. Puede suceder que el diseño presentado sea conveniente, con la exitosa aplicación de los escenarios, o por el contrario, no conveniente, cuando el grupo encuentra problemas o deficiencias.
9. Resumen.	Al final, el facilitador recuenta la lista de puntos tratados, pide opiniones de los participantes sobre la eficiencia del ejercicio de revisión, y agradece por su participación.

La Tabla 3 muestra la selección del método basado en arquitectura que se realizó a través de una comparación entre los métodos estudiados: SAAM, ATAM y ARID analizando aspectos relevantes como los atributos de calidad que contempla cada uno, los objetivos que analizan, las etapas o fases del proyecto en las que se aplican, así como sus principales enfoques. [11]

Tabla 4: Comparación entre métodos de evaluación.

	SAAM	**ATAM**	**ARID**
Atributos de calidad contemplados	• Modificabilidad • Funcionalidad	• Modificabilidad • Seguridad • Confiabilidad • Desempeño	• Conveniencia del diseño evaluado

Objetivos analizados	• Documentación • Vistas Arquitectónicas	• Estilos arquitectónicos • Documentación • Flujo de datos • Vistas Arquitectónicas	• Especificación de los componentes
Etapas del proyecto en las que se aplica	• Después de que la arquitectura cuenta con funcionalidad ubicada en módulos	• Después de que el diseño de la arquitectura ha sido establecido	• A lo largo del diseño de la arquitectura
Enfoques utilizados	• Lluvia de ideas para escenarios y articular los requerimientos de calidad • Análisis de los escenarios para verificar funcionalidad o estimar el costo de los cambios	• Utility Tree y lluvia de ideas para articular los requerimientos de calidad • Análisis arquitectónico que detecta puntos sensibles, puntos de balance y riesgos	• Revisiones de diseños, lluvia de ideas para obtener escenarios

3.3 Evaluación de la arquitectura definida para el Servidor de Gestión de PostgreSQL

Evaluar una arquitectura de software sirve para prevenir todos los posibles riesgos de un diseño que no cumple con los requerimientos de calidad y para saber que tan adecuada es la arquitectura de software diseñada para el sistema. Para la evaluación de la arquitectura propuesta, se utilizó el método, ya que evalúa tempranamente la arquitectura de software, obteniendo las zonas de falla del sistema, permitiendo así realizar cambios cuando estos no demanden demasiados esfuerzos. ARID constituye un método conveniente para la evolución de diseños parciales en las etapas tempranas del desarrollo y la técnica de evaluación basada en escenario (breve descripción de la interacción de alguno de los involucrados en el desarrollo del sistema). Este tipo de evaluación es simple de crear y entender, efectivo, poco costoso y no requiere mucho tiempo.

Con el objetivo de poder evaluar la arquitectura propuesta se seleccionaron una serie de atributos de calidad y un listado de riesgos que son los que se relacionarán con estos para poder realizar la evaluación.

Atributos de calidad seleccionados:

- Seguridad Interna.
- Disponibilidad.

Los riesgos asociados a los atributos seleccionados son:

- Intento de acceso no permitido.
- Problemas de conexión a la base de datos.

- Escalabilidad.
- Usabilidad.
- Pérdida de los datos debido a fallas del servidor.
- Añadir servicios o plugins al sistema.
- El sistema debe ser entendible y fácil de usar.

Tabla 5: Atributo Seguridad.

Atributo de calidad	**Perfil**	**Escenario**
Seguridad Interna	Seguridad	Intento de acceso no permitido.
Relación atributo – escenario		
Para el uso de la aplicación, el sistema gestiona la seguridad a través de la autenticación de usuario. La autenticación es la primera acción del usuario en el sistema y consiste en suministrar un nombre de usuario único y una contraseña que debe ser de conocimiento exclusivo de la persona que se autentica. Si la información proporcionada por el usuario es correcta el sistema muestra al usuario las funcionalidades que puede realizar en dependencia de los permisos que tenga. En caso contrario muestra un mensaje de error.		

Tabla 6: Atributo Disponibilidad.

Atributo de calidad	**Perfil**	**Escenario**
Disponibilidad	Disponibilidad	Problemas de conexión a la base de datos.
Relación atributo – escenario		
Ante la ocurrencia de algún tipo de falla en la conexión de la aplicación con la base de datos la misma re-direccionará sus pedidos al servidor esclavo, ya que en el mismo se encuentra toda la información de la base de datos principal, debido a que deben realizarse replicas constantes del servidor principal.		

Tabla 7: Atributo Disponibilidad.

Atributo de calidad	**Perfil**	**Escenario**
Disponibilidad	Disponibilidad	Posible pérdida de los datos debido a fallas del servidor.
Relación atributo – escenario		
Ante la ocurrencia de algún tipo de falla en los servidores MongoDB y Naire, la cual pueda traer consigo pérdida de los datos, el sistema realizaría una completa recuperación de los mismos, debido a la replicación de datos que se efectúa en el servidor MongoDB y a las copias de seguridad que se ejecutan en el servidor Naire. Esto garantiza al menos una copia fiel de los datos del sistema, minimizando el tiempo sin funcionamiento del servidor y manteniendo la disponibilidad de los datos.		

Tabla 8: Atributo Escalabilidad.

Atributo de calidad	**Perfil**	**Escenario**
Escalabilidad	Escalabilidad	Añadir servicios o plugins al sistema.
Relación atributo – escenario		
El servidor de aplicaciones permitirá la inclusión de nuevas funcionalidades al sistema. Esto no brindará ningún problema debido a la implementación del patrón arquitectónico MVC, pues se pueden realizar cambios en la aplicación, ya sea agregar nuevas funcionalidades a la vista o cambiar de base de datos, lo que no afecta la arquitectura del sistema.		

Tabla 9: Atributo Usabilidad.

Atributo de calidad	**Perfil**	**Escenario**
Usabilidad	Usabilidad	El sistema debe ser entendible y fácil de usar.
Relación atributo – escenario		

El sistema presenta al usuario una interfaz atractiva e interactiva, con un menú general que lo guía en la búsqueda de la información que necesita. Además de garantizar una estructura afín a todas las páginas, estableciendo regularidades en el orden de aparición de algunos campos de formularios. También consta con una ayuda para guiar a los usuarios en el manejo del mismo.

3.4 Conclusiones del capítulo

Después de realizado un análisis de los métodos de evaluación existentes se escogió el método ARID para evaluar la arquitectura propuesta, permitiendo así, evaluar la arquitectura en una etapa temprana del diseño. Además, fueron definidos un conjunto de escenarios, dentro de los cuales prevalecieron aquellos que se relacionaban con los atributos escogidos. Finalmente, se realizó la evaluación de la arquitectura, obteniéndose como resultado que las funcionalidades deseadas para el sistema fueron satisfactoriamente cumplidas, demostrando así que la arquitectura propuesta es la adecuada para la implementación del sistema.

CONCLUSIONES

Con el objetivo de definir la arquitectura de software para el Servidor de Gestión de PostgreSQL se realizó el estudio del marco teórico acerca de los principales conceptos, estilos y patrones relacionados con la arquitectura de software, soluciones similares, así como la definición del rol del arquitecto para la metodología que se utilizó. A partir de este análisis se desarrolló la propuesta y se seleccionaron como patrones arquitectónicos el Modelo Vista Controlador (MVC) y los principales patrones GRASP, garantizando principalmente la organización, la escalabilidad y seguridad de la aplicación.

Para la definición de la arquitectura se priorizaron las historias de usuario más significativas y se identificaron los principales requerimientos no funcionales del sistema. Se determinaron las herramientas, tecnologías y metodologías que se utilizaron. Se diseñó la propuesta de solución del sistema mediante las 4+1 vista arquitectónicas: la vista de Historia de Usuario, Lógica, de Despliegue, de Implementación y la de Procesos.

Finalmente, se evaluó la arquitectura propuesta mediante el método ARID y la técnica basada en escenarios, obteniéndose resultados satisfactorios. Esta evaluación permitió concluir que la propuesta de solución es la adecuada para la implementación del sistema.

RECOMENDACIONES

Luego de haber analizado los resultados del presente trabajo de diploma, se realizan las siguientes recomendaciones:

- Utilizar este trabajo como una guía para el desarrollo de nuevas aplicaciones informáticas con características similares.
- Aplicar otros métodos de evaluación de la arquitectura para aumentar la fortaleza del sistema.

REFERENCIAS BIBLIOGRÁFICAS

1. 1471-2000, I.S. (2000) *Recommended Practice for Architectural Description of Software-Intensive Systems*. **Volume**,
2. Bass, L., P. Clement, and R. Kazman, *Software Architecture in Practice, ed. A.-W. Professional.* 2003, Addison-Wesley.
3. Garlan, D., *Software Architecture: A Roadmap. En Anthony Finkelstein: The future engineering*. 2000.
4. Reynoso, C.B. (2004) *Introducción a la Arquitectura de Software Versión 1.0* **Volume**,
5. Fuentes, L., N. Gámez, and M. Pinto, *Desarrollo de Software Basado en Componentes.* 2007.
6. Ivis Rosa Vásquez Sierra, P.A.C., Sorey Bibiana García Zapata (2008) *El Rol del Arquitecto de Software* **Volume**,
7. DB, E. *EnterpriseDB The Enterprise PostgreSQL Company.* . 2011 [cited 2011 10 enero]; Available from: http://www.enterprisedb.com/docs/en/8.4/pphq/Postgres_Plus_HQ_Users_Guide-06.html
8. Corporation, E. *EnterpriseDB*. 2010 [cited 2011 15 abril]; Available from: http://enterprisedb.com/ba/limited-license-v2-2.
9. Marie McGarry, C.L. (2010) *Overview* **Volume**, Spring Source Hyperic
10. Source, S. *Spring Source*. 2011 [cited 2011 20 abril]; Available from: http://www.springsource.com/license/hyperic-enterprise.
11. Erika Camacho, F.C., Gabriel Nuñez (2004) *Guía de Estudio Arquitecturas de Software*. **Volume**,
12. Reynoso, C. and N. Kiccillof (2004) *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft Versión 1.0*. **Volume**,
13. Rosanigo, Z.B. (2000) *Modelos y Patrones.* **Volume**,
14. López, Y.M. and A.R.R. García, *Propuesta del diseño arquitectónico de la Plataforma bioGRATO.* . 2008.
15. Yorio, D. (2006) *Identificación y Clasificación de Patrones en el Diseño de Aplicaciones Móviles*. **Volume**,
16. García, Y. and G. Jasper (2008) *Diseño con Patrones.Diseño de Sistemas*. **Volume**,
17. Larman, C., *UML y PATRONES.* 1999.
18. Gamma, E., et al. (1995) *Design Patterns: Elements of reusable object-oriented software.* **Volume**,

19. Díaz, O.M. (2001) *Una Visión General a los Lenguajes de Descripción Arquitectónica*. **Volume**,
20. Orallo, E.H. (2010) *El Lenguaje Unificado de Modelado (UML).* **Volume**,
21. Kristina Chodorow, M.D., *MongoDB: The Definitive Guide*. 2010: O'Reilly Media.
22. Barcia, G., *Manejo de Control de Versiones con Google Code y NetBeans.* 2010.
23. Rosas, J.E.S. (2007) *Principios de Control de Versiones con Subversion*. **Volume**,
24. Manuel Resinas de Reyna, M.J.R.S. (2005) *Introducción a los Sistemas de Control de Versiones.* **Volume**,
25. R., E., *Ventajas de jQuery.* 2010.
26. Ing. Marianela Gutiérrez Rodríguez, I.Y.M.R., Ing. Glennis Tamayo Morales, Ing. Yudisney Vazquez Ortíz, *SELECCIÓN DE LA METODOLOGÍA DE DESARROLLO PARA EL PROYECTO POSTGRESQL EMPRESARIAL*. 2009.
27. Flower, M. (2010) *eXtreme Programming.* **Volume**,
28. José Carlos Cortizo Pérez, D.E.G.y.M.R.L. (2008) *eXtreme Programming.* **Volume**,
29. Edelquis Guillermo Geigel Echavarría, D.S.R., *Rediseño de la arquitectura de la Plataforma alasGrato.* 2010.
30. Kazman, R., Clements, P., Klein, M, *Evaluating Software Architectures. Methods and case studies. .* 2001.

BIBLIOGRAFÍA

1. PRESSMAN, R. S. Ingeniería del Software. Un enfoque práctico. 2005. vol. 1,
2. BASS, L.; CLEMENT, P., et al. Software Architecture in Practice. Addison-Wesley: 2003.
3. CLEMENTS, P.; KAZMAN, R., et al. Evaluating Software Architectures: Methods and Case Studies. Editado por: Wesley, A. 2000.
4. CAMACHO, E.; CARDESO, F., et al. Arquitecturas de Software. 2004, Disponible en: http://prof.usb.ve/lmendoza/Documentos/PS-6116/Guia%20Arquitectura%20v.2.pdf
5. REGALADO, Y. V. y CABANA, E. F. La arquitectura de software como disciplina científica. 2008, Disponible en: http://www.gestiopolis.com/administracion-estrategia/arquitectura-de-sofware-como-disciplina-cientifica.html
6. REYNOSO, C. y KICILLOF, N. Lenguajes de Descripción de Arquitectura (ADL) Versión 1.0 2004,
7. ZARAGOZA, F. J. O. Arquitectura de Referencia para Unidades de Control de Robots de Servicio Teleoperados. Doctoral, 2005.
8. MORALES, T. P. D. y RODRÍGUEZ, J. S. B. Diseño de un Datawarehouse para los Ensayos Clínicos que se gestionan en el Centro de Inmunología Molecular. 2010.
9. Nedelcu, C., Nginx HTTP Server. 2010.
10. Torre, A.d.l. *Lenguajes del lado servidor o cliente*. 2006 [cited; Available from: http://www.adelat.org/media/docum/nuke_publico/lenguajes_del_lado_servidor_o_cliente.html.
11. Valdés, D.P. *Los diferentes lenguajes de programación para la web*. 2010 [cited; Available from: http://www.maestrosdelweb.com/principiantes/los-diferentes-lenguajes-de-programacion-para-la-web/.

GLOSARIO DE TÉRMINOS

ADL: Lenguaje de descripción de arquitectura cuyas siglas se derivan de su nombre en inglés Architecture Descripction Language y su función como su nombre lo indica es describir una arquitectura de software.

ARID: Siglas del método de Revisiones Activas para Diseños Intermedios.

Atributos de calidad: Aspectos del sistema, que en general, no afectan directamente a la funcionalidad necesaria, sino que definen la calidad.

GNU/LGPL: "GNU Lesser General Public License" (Licencia Pública General Menor). Pretende garantizar la libertad de compartir y modificar el software libre, esto es para asegurar que el software es libre para todos sus usuarios. Esta licencia pública general se aplica a la mayoría del software de la "FSF Free Software Foundation" (Fundación para el Software Libre) y a cualquier otro programa de software cuyos autores así lo establecen.

Framework: Se conoce como marco de trabajo y es un conjunto de conceptos, metodologías y herramientas de administración y diseño para el desarrollo de forma estandarizada de una aplicación.

GUI: Interfaz gráfica para el usuario.

Herramientas CASE: Conjunto de aplicaciones informáticas orientadas al incremento de la productividad en el desarrollo de software, las siglas CASE vienen dadas por su nombre en inglés Computer Arded Software Engineering que se conoce como Ingeniería de Software Asistida por Computadoras.

IEEE: Asociación técnico-profesional mundial dedicada entre otras cosas a la estandarización, sus siglas vienen dadas por su nombre en inglés de The Institute of Electrical and Electronics Engineers, Instituto de Ingenieros Eléctricos y Electrónicos.

J2EE: es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en Lenguaje de programación Java, que se apoya ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones.

JBoss: es un servidor de aplicaciones J2EE de código abierto implementado en Java. Puede ser utilizado en cualquier sistema operativo para el que esté disponible Java. Implementa todo el paquete de servicios de J2EE.

JPG: Técnica de compresión de imágenes que casi no afecta la calidad, reduce el peso en bytes de las imágenes hasta 5%. Sus siglas vienen dadas por el diminutivo de su nombre en inglés JPEG (Joint Photographic Experts Group).

Open Source: Cualidad de algunos software de incluir el código fuente en la distribución del programa. En general se usa para referirse al software libre.

PC: Es la expresión estándar que se utiliza para denominar a las computadoras personales en general.

PNG: Formato gráfico muy completo especialmente pensado para redes. Sus siglas vienen dadas por su nombre en inglés Portable Network Graphics.

Portlets: son componentes modulares de las interfaces de usuario gestionadas y visualizadas en un portal web. Producen fragmentos de código de marcado que se agregan en una página de un portal.

PostgreSQL: Sistema Gestor de Base de Datos.

RUP: Metodología de desarrollo cuyas siglas vienen dadas por su nombre en inglés Rational Unified Process. Establece para el desarrollo de un software una serie de flujos de trabajo agrupados en cuatro fases fundamentales.

UML: Lenguaje visual para especificar, construir y documentar un sistema de software. Sus siglas vienen dadas por su nombre en inglés Unified Modeling Language.

VP-UML CE: Plataforma de modelado diseñada para el aprendizaje del modelado visual cuyas siglas vienen dadas por su nombre en inglés Visual Paradigm for UML Community Edition.

XML: Estándar de información cuyas siglas vienen dadas por su nombre en inglés eXtensible Markup Language.