# Universidad de las Ciencias Informáticas
## FACULTAD 6


# Título: Arquitectura de Software basada en un Modelo de Negocio para la Plataforma de Servicios Bioinformáticos.

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

**Autora:** Yohanka Lázara Aranguren Reyes

**Tutor:** Ing. Edel Moreno Lemus

**Co-Tutores:** Msc. Yosdenis Urrutia Badillo
Ing. Cecilia Milián Delgado

**Consultantes:** Ing. Armando Robert Lobo

La Habana, junio de 2012
**"Año 54 de la Revolución"**

*En lo tocante a ciencia, la autoridad de un millar no es superior al humilde razonamiento de un hombre.*

DECLARACIÓN DE AUTORÍA

Declaro que soy la única autora del trabajo ***"Arquitectura de Software basada en un Modelo de Negocio para la Plataforma de Servicios Bioinformáticos."*** y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Yohanka Lázara Aranguren Reyes

_____

**Autora**

Ing. Edel Moreno Lemus

_____

**Tutor**

Yosdenis Urrutia Badillo

_____

**Co-Tutor**

Ing. Cecilia Milían Delgado

_____

**Co-Tutora**

DATOS DE CONTACTO

**Autora**

Yohanka Lázara Aranguren Reyes
Universidad de las Ciencias Informáticas
La Habana, Cuba
Correo Electrónico: ylaranguren@estudiantes.uci.cu

**Tutor**

Ing. Edel Moreno Lemus
Correo Electrónico: emoreno@uci.cu
Ingeniero en Ciencias Informáticas de la Universidad de las Ciencias Informáticas 2011
Categoría Docente: Adiestrado

**Co-Tutores**

Msc. Yosdenis Urrutia Badillo
Correo Electrónico: yosdenis@uci.cu
Máster en Administración de Negocio, Universidad de La Habana 2009
Categoría Docente: Asistente.

Ing. Cecilia Milián Delgado
Correo Electrónico: cmilian@uci.cu
Ingeniero en Ciencias Informáticas de la Universidad de las Ciencias Informáticas 2011
Categoría Docente: Adiestrado

RESUMEN

Las nuevas tecnologías han supuesto una revolución para el comercio dando lugar a los "Modelos de Negocio". Estos modelos abarcan distintas posibilidades de compra de bienes y servicios. El impacto de la informática en este campo es enorme, permitiendo la creación productos, aplicaciones y/o soluciones en las ramas de la educación, la salud, la industria, entre otros sectores; comercializables mediante la prestación de servicios o venta de software. Con el fin de realizar productos y/o aplicaciones que se puedan comercializar en el mercado actualmente se desarrolla una Plataforma Servicios Bioinformáticos en la Universidad de las Ciencias Informáticas.

La presente investigación tiene como objetivo definir una arquitectura de software para la plataforma basada en un modelo de negocio que permita comercializar dichos servicios en el mercado. El éxito se enmarca en el empleo de un modelo que pueda detallar una arquitectura de software adecuada a los requisitos y la utilización de los principales servicios desde cualquier lugar estando disponible las 24 horas sin necesidad de realizar un despliegue de la aplicación o instalación de alguna herramienta. Como resultado de este trabajo se identificaron los requisitos no funcionales y se propone la arquitectura de la plataforma para responder al modelo de negocio definido. Dicha plataforma presentará mediante un portal web una gran variedad de servicios bioinformáticos para que biólogos puedan acceder a características avanzadas, sin tener que manejar conceptos complejos. Finalmente se presentan los resultados de la evaluación de la arquitectura.

**Palabras Claves:**

Plataforma de Servicios Bioinformáticos, Arquitectura de Software y Modelos de Negocio.

ÍNDICE

# INTRODUCCIÓN

El comercio es una actividad que se ha venido desarrollando desde hace muchos años para intercambiar bienes, servicios e información mediante la cual no solo se beneficia el hombre sino también la sociedad. En la actualidad constituye una actividad fundamental de la economía por lo que es una prioridad para el desarrollo de cualquier país.

En Cuba, el sustento de la economía estará basado dentro de quince años aproximadamente en las producciones intelectuales. En este sentido la Informática está dando grandes pasos para revolucionar el sector económico, rama que puede aportar mucho al país por concepto de producción de software. Concibiendo este campo como un apoyo a la economía de nuestro país se hace necesario, una proyección comercial para autofinanciamientos futuros, según se indica en el nuevo modelo de gestión económica y social. De ahí la importancia de seleccionar para cada producto/servicio cuyo destino final sea su venta un modelo de negocio que sirva de guía para establecer estrategias de negociación beneficiosas para ambas partes: empresa y cliente. Sin embargo, uno de los problemas que presenta este campo en la actualidad es la constante evolución de las tecnologías. Esto se ve reflejado en el proceso de desarrollo de software, por lo que se pretende lograr que los sistemas sean cada vez más flexibles, heterogéneos y de fácil integración. Para alcanzar este objetivo es preciso definir una arquitectura capaz de asegurar el correcto funcionamiento del software satisfaciendo las necesidades del cliente y priorizando la rentabilidad de las soluciones informáticas.

A través del modelo de negocio se puede conocer las necesidades del mercado, hacia qué sector va a estar dirigido el producto, qué posición tiene la entidad respecto a su competencia y cuáles serían las ventajas y desventajas del producto final tanto para el cliente como para la empresa. Partiendo de ahí se diseña una arquitectura de software que garantice el desempeño del sistema y responda a los resultados ofrecidos por el modelo de negocio y además marque las pautas para el proceso de comercialización.

La Universidad de las Ciencias Informáticas (UCI) es un centro productivo educacional, cuyo objetivo es producir software y servicios informáticos a partir de la vinculación estudio – trabajo, la cual juega un papel fundamental para cumplir este objetivo. El Centro de Tecnología de Gestión de Datos (DATEC) perteneciente a la UCI, es un centro que provee soluciones integrales, productos y servicios relacionados con las tecnologías de Gestión de Datos y Bioinformática. DATEC cuenta con cuatro

departamentos, tales como: Integración de Soluciones, Postgrees, Almacenes de Datos y Bioinformática.

El departamento de bioinformática está desarrollando una plataforma compuesta por los servicios bioinformáticos con el propósito que sean consumidos a través de un portal web; permitiendo a los usuarios acceder a características avanzadas sin tener que manejar conceptos complejos (1). No obstante, el departamento presenta dificultad para comercializar dichos servicios, al no tener identificados los clientes potenciales a quien está dirigida dicha plataforma, no conocer las necesidades reales de los mismos ni el comportamiento de los servicios en el mercado porque se centra en el desarrollo del producto y sin tener un enfoque de venta. Esto pudiera limitar que la plataforma no sea rentable, proporcione pérdidas en la inversión de dicha solución y la insatisfacción para el equipo de desarrollo desde el punto de vista profesional. Teniendo como origen un modelo de negocio para comercializar dichos servicios podemos definir una arquitectura de software que permita dar soporte a los servicios bioinformáticos.

Debido a la situación planteada anteriormente se identifica como ***problema de investigación*** lo siguiente: ¿Cómo influir en la Arquitectura de Software según un modelo de negocio para la Plataforma de Servicios Bioinformáticos?

La investigación tiene como ***objeto de estudio:*** los Modelos de Negocio enmarcado en el ***campo de acción:*** la Arquitectura de Software basada en un modelo de negocio.

Como ***objetivo general:*** Definir la arquitectura de software para la Plataforma de Servicios Bioinformáticos basada en un modelo de negocio.

Desglosados en los siguientes ***objetivos específicos:***

- Proponer un modelo para comercializar los Servicios Bioinformáticos.
- Describir la arquitectura de software para la Plataforma de Servicios Bioinformáticos basada en el modelo de negocio propuesto.
- Evaluar la arquitectura de software de la Plataforma de Servicios Bioinformáticos.

Al utilizar un modelo de negocio como herramienta para la Arquitectura de Software se hace necesario definir el ***alcance de esta investigación***, ya que un modelo de negocio visto desde el punto de vista económico demanda un estudio más profundo del mercado para poder determinar oferta-demanda de

la plataforma, fijar precios, entre otros factores que no son el objeto de estudio de este trabajo. Por lo que esta investigación comenzará desde el estudio de los diferentes modelos de negocios ya existentes hasta la influencia de un modelo que permita la descripción de la Arquitectura de Software.

Par dar cumplimiento a los objetivos trazados, se plantean las siguientes ***tareas de la investigación:***

- Identificación de un modelo de negocio para comercializar los Servicios Bioinformáticos a través de las tendencias de dichos modelos.
- Valoración de los principales mercados y competidores del producto Plataforma de Servicios Bioinformáticos.
- Diseño de la propuesta arquitectónica que cumpla con los requerimientos para la aplicación y que garantice el desarrollo paralelo y la reutilización de componentes de la misma.
- Diseño de las vistas del sistema.
- Selección de los posibles métodos de validación basados en la Arquitectura.
- Aplicación de los métodos seleccionados para evaluar la arquitectura de software definida.

En la cual se han de utilizar los siguientes **Métodos:**

**Teóricos:**

- Análisis y Síntesis: para el procesamiento de la información y arribar a las conclusiones de la investigación, así como para precisar las características del modelo arquitectónico propuesto.
- Histórico – Lógico: Para determinar las tendencias actuales en el desarrollo de los modelos de negocio y los enfoques arquitectónicos.
- Inducción y deducción: A partir del estudio de distintos estilos y patrones de arquitectura, así como del estudio de los diferentes frameworks de desarrollo para aplicaciones web específicas.

**Empíricos:**

- Observación: Para la percepción selectiva y sistemática de las restricciones y propiedades del sistema, y para el control de la evolución de la arquitectura inicial.

El presente trabajo de diploma se estructura en tres capítulos:

**Capítulo 1: Fundamentación Teórica**

Constituye la base teórica de la investigación realizada. Se describen los principales conceptos relacionados con el término de Modelo de Negocio. Se realiza un estudio de las diferentes modelos de negocios empleados en el proceso de comercialización dentro de la industria de software.

**Capítulo 2: Propuesta de Arquitectura**

En este capítulo se hace una propuesta de arquitectura basada en los elementos fundamentales en los que influye el modelo de negocio. Se analizan diferentes conceptos de Arquitectura de Software. Se describen los requisitos funcionales y no funcionales que debe cumplir el sistema así como la representación de la descripción de la arquitectura de software a través de la vistas de la arquitectura.

**Capítulo 3: Evaluación de la Arquitectura**

Muestra un estudio de la factibilidad de la arquitectura definida. Se expone lo referente a los principales métodos basados en la arquitectura utilizados para evaluar la arquitectura a nivel mundial. Se describe además el método seleccionado para aplicarlo a la plataforma, logrando así la validación de la arquitectura de software definida.

Finalmente se muestran las principales conclusiones a las que se arriba al terminar la investigación y se proponen recomendaciones para el departamento de Bioinformática. También se presentan las fuentes bibliográficas consultadas y los anexos que apoyan el desarrollo de esta investigación

# CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

## 1.1 Introducción

En este capítulo se introducen conceptos y aspectos acerca de los modelos de negocio con el objetivo de poder comercializar los servicios bioinformáticos que brinda el portal web de la plataforma. Se realizará un estudio de los factores externos e internos que afecten el posicionamiento de estos servicios en el mercado, como también de los principales mercados y competidores. Además se describen los elementos fundamentales que influyen en la arquitectura de software.

## 1.2 Antecedentes Históricos

Para la realización del presente trabajo se tuvo en cuenta antecedentes relacionados con la parte socio política del país, según Dotres (2), la alta dirección del Ministerio de Salud Pública, propuso: "…desarrollar en el Sistema Nacional de Salud un modelo integral de informatización a los diferentes niveles del mismo, así como el acceso a la información como proceso que apoye y potencie decisivamente la asistencia médica, la docencia, la investigación, la higiene y la epidemiología, la industria médico-farmacéutica, la economía y administración de salud, que se extienda de forma integral a todas las instituciones del país para alcanzar un Sistema Integrado de Gestión que será herramienta básica en la materialización de las estrategias y programas de Salud". Existencia del estudio de la industria del software a nivel mundial y su caracterización en América Latina y Cuba del autor Vismar Santos Hernández (3) donde realiza un análisis de la organización de esta industria en algunos países del mundo y ofrece los resultados arrojados por grandes consultoras del tema a nivel mundial. Una intención marcada del Gobierno de Cuba de desarrollar la economía de manera sustentable expresado en los Lineamientos de la Política Económica y Social del Partido y la Revolución del 18 de abril del 2011 y además en el departamento de Bioinformática se realizó un estudio donde se seleccionaron los servicios que pueden formar parte de la plataforma.

## 1.3 Modelos de Negocio

Los modelos de negocios han sido definidos y categorizados de muchas maneras diferentes. A continuación se mencionan algunas las definiciones de varios autores que llevaron el camino de esta investigación.

Joan Magretta (4), define el modelo de negocio como “historias que explican cómo la empresa trabaja”.

Según Amit y Zott (5), “un modelo de negocio explicita el contenido, la estructura y el gobierno de las transacciones designadas para crear valor al explotar oportunidades de negocio”.

Chesbrough and Rosenbloom (6), presentan una definición más detallada y operativa al indicar que las funciones de un modelo de negocio son: articular la proposición de valor; identificar un segmento de mercado; definir la estructura de la cadena de valor; estimar la estructura de costes y el potencial de beneficios; describir la posición de la empresa en la red de valor y formular la estrategia competitiva.

Linder y Cantrell de Accenture (7) definen “un modelo de negocio operativo es la lógica nuclear de la organización para crear valor. El modelo de negocio de una empresa orientada a los beneficios y explica cómo ésta hace dinero”.

Osterwalder, Pigneur y Tucci (8) repasaron distintas definiciones y terminan aportando una: “un modelo de negocio es una herramienta conceptual que contiene un conjunto de elementos y sus relaciones que nos permite expresar la lógica de negocio de una empresa específica. Es la descripción del valor que una empresa ofrece a uno o varios segmentos de clientes y de la arquitectura de la empresa y su red de socios para crear, comercializar, y aportar este valor a la vez que genera un flujo rentable y sostenible de ingresos.”

En conclusión, los modelos de negocio se definen como la forma en que las empresas obran con el fin de obtener ganancias y beneficios, y un público objetivo al que quiere servir. En su configuración abarcan elementos como la metas, estrategias, procesos, tecnología y estructura de la organización concebidas para competir exitosamente en el mercado.

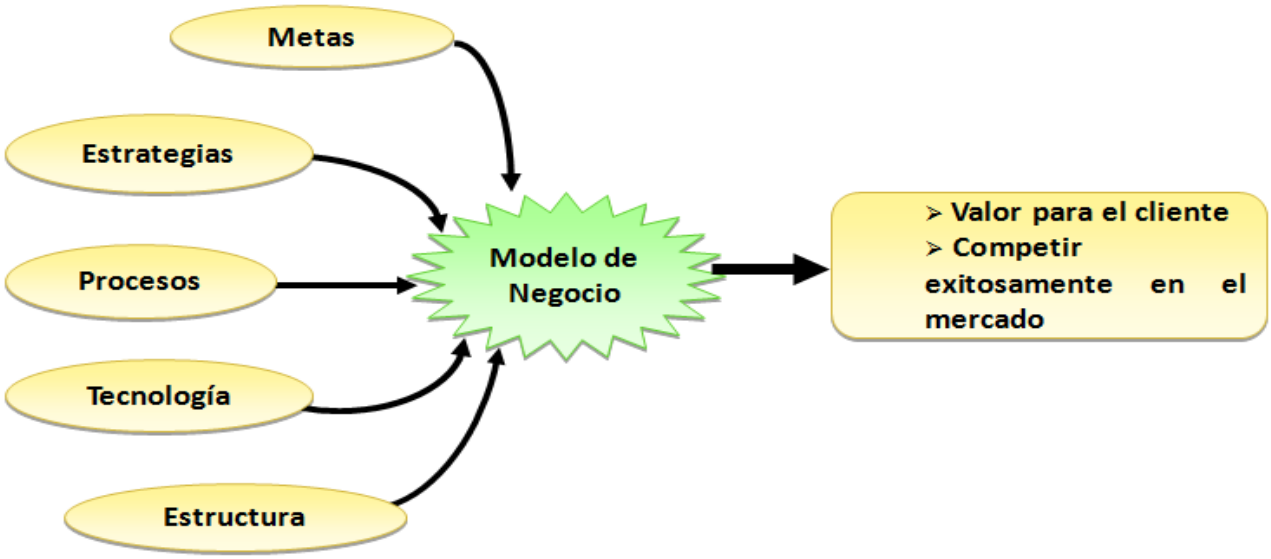


Figura 1: Elementos del Modelo de Negocio. Fuente: Elaboración Propia

Las ***metas*** definen hacia dónde se va y hacia dónde se quiere llegar y una vez definidas, se trazan las ***estrategias*** que permitirán definir los objetivos de la empresa. Las organizaciones o entidades tienen como principio un enfoque por ***procesos*** o gestión de procesos para gestionar la calidad y poder comercializar en el mercado, teniendo en cuenta la ***tecnología*** haciendo énfasis en el conjunto de herramientas necesarias para realizar un proyecto.
La empresa define la ***estructura*** de la organización para que los procesos sean más ágiles. Todos estos elementos son entradas del modelo de negocio que se transforman en valor agregado, teniendo como salida un cliente satisfecho y mejor competitividad en el mercado.

## 1.4 Tipos de Modelos de Negocio

Con la revolución de la red de redes, han surgido diferentes Modelos de Negocio. Entre ellos se encuentra el más básico y antiguo conocido como el **modelo del tendero**, consiste en instalar un negocio en el lugar donde deberían encontrarse los clientes potenciales y allí desplegar la oferta de productos y/o servicios. (9)

El **modelo del cebo y el anzuelo**, se basa en la oferta de un producto básico a bajo precio, incluso soportando pérdidas (el cebo), para después cobrar precios excesivos por los recambios o insumo asociados (el anzuelo). Podemos verlo en el negocio de las impresoras, que tienen un costo muy bajo en comparación al de los cartuchos de tinta.

**La subasta**, los compradores y vendedores se reúnen o son conectados en línea. Los precios son determinados según cuánto los compradores estén dispuestos a pagar. La ventaja para el vendedor radica en la competencia que se establece entre los compradores por hacerse del producto subastado, llegando a pagar un precio superior, en la mayoría de los casos, al valor de mercado.

En el mundo del software se utilizan los Modelos de Negocio que presenta Michael A. Cusumano en su libro “El negocio del software” (10), estos son:

- **Modelo de Negocios en Base a Venta de Licencias de Software:** es uno de los modelos de mayor rentabilidad porque una vez que el software ha sido desarrollado toda la venta incremental tiene costo muy bajo. Su principal desventaja está dada por la capacidad de inversión de la empresa en promocionar su producto. Las licencias de software pueden ser vendidas al público en general o a las empresas. Si los clientes son empresas se debe

asegurar que durante el primer año se ganen 5 o 6 clientes importantes que puedan servir como referencias de ventas en el futuro.

- **Modelo de Negocios en Base a Servicios:** este modelo ofrece soluciones a medida de los clientes. Sus ventajas fundamentales son: no necesita una inversión en promoción tan importante y un buen servicio es difícil de imitar por las compañías competidoras, por lo tanto, si su organización alcanza un buen nivel de servicios sus barreras de protección serán altas. Su desventaja está dada en el costo de brindar este servicio es sumamente alto porque es muy intensivo en mano de obra especializada.

- **Modelo Híbrido:** En este último las empresas generan una parte significativa de sus ingresos por venta de servicios y otra parte por ventas de licencias. Estas organizaciones no tienen muchos conocimientos en la innovación de software pero reciben ingresos por productos lanzados en el pasado y que han sido altamente aceptados por el mercado.

Además de los modelos ya expuestos anteriormente existen también diferentes modelos basados en Internet según Michael Rappa (11), tales como:

- **Modelos de Negocio basados en publicidad:** es una extensión del modelo tradicional de transmisión de medios. El medio de transmisión, en este caso es un sitio Web, provee contenidos y servicios mezclados con mensajes publicitarios en la forma de banners. Los banners pueden ser la principal o única fuente de ingresos para el medio. El cual puede ser un creador de contenidos o distribuidor de contenidos creados en cualquier lugar. El modelo de publicidad solo funciona cuando el volumen de tráfico por visitas es muy grande o altamente especializado.

- **Modelo directo o fabricante:** está fundamentado en la capacidad de la Web para permitir a un fabricante (una compañía que crea un producto o servicio) alcanzar a los compradores directamente y por lo tanto comprimir el canal de distribución. Este puede estar basado en la eficiencia, mejora del servicio al cliente y un mejor entendimiento de las preferencias del mismo.

- **Modelo de afiliación:** provee oportunidades de compra en cualquier lugar donde los usuarios puedan estar navegando. Esto se logra ofreciendo incentivos financieros a los sitios asociados

que se afilian. El modelo de afiliación está inherentemente bien adaptado para la Web, lo cual explica su popularidad. Algunas variaciones incluyen, intercambio de banners, pago por clic, y programas de compartición de ganancias.

- **Modelo de suscripción:** Los usuarios pagan para acceder al contenido al sitio pudiendo ser tanto a través de suscripciones regulares o por visita/consulta. Los modelos basados en esta fuente de ingresos tienen que enfrentarse a la fuerte resistencia de los usuarios a pagar por los contenidos.

- **Modelo de corretaje o brokerage:** Los corredores o brokers son creadores de mercados, ellos atraen a compradores y vendedores y facilitan las transacciones. Los brokers frecuentemente juegan un rol en los mercados de negocio a negocio (B2B), negocio a consumidor (B2C), consumidor a consumidor (C2C). Usualmente un broker carga una cuota o comisión por cada transacción que propicia. La fórmula para las cuotas puede variar.

El software libre no se queda atrás y también ha propuesto diversos modelos como: (12)

- **Modelo de Comercialización con licencias duales:** Una licencia dual es un Modelo de Negocio en el que una compañía que comercializa un determinado producto aporta a sus licencias dos modelos a escoger: código abierto y código "cerrado" o propietario. Aquí, el cliente puede elegir el tipo de licencia con las condiciones de una licencia de código abierto, como la de tipo BSD o la GPL, y otra con licencia de código cerrado. Este modelo es ampliamente utilizado para productos como Open Office, Trolltech y MacOS X. Se ha comprobado que es exitoso porque las licencias duales estimulan la introducción del software de código abierto en los negocios mientras impulsa su desarrollo con los ingresos que brinda la venta de licencias propietarias.

- **Modelo de Desarrollo de componentes comerciales para productos de software libre:** Se basa en el desarrollo bajo licencia libre de aplicaciones específicas, con el objetivo de que se desarrollen determinados componentes comerciales que cubren determinadas necesidades de un cliente. Se basa en repartir paquetes compuestos de un conjunto de componentes probados e integrados. Es utilizado por empresas como Zope Corporation, Ximian y OpenSystems.
- **Modelo de Software propietario sobre Software libre:** Consiste en construir software propietario utilizando herramientas libres, y después vender sus servicios, a través de Internet.

Con este modelo muchas empresas intentan la integración con el software libre, utilizando piezas de código cerrado para tratar de mejorar la compatibilidad y permitir la ejecución de su software sobre sistemas operativos libres. Se aplica en Google y Oracle.

Después de los modelos expuestos anteriormente surgen nuevos modelos enfocados a al software como servicio, tales como (13):

- **Modelo de Proveedor de Servicios de Aplicaciones (ASP):** empresas que bridan servicios de software a sus clientes a través de Internet. Con el fin de lograr un ahorro en los gastos de la empresa relacionados con la instalación, operación y mantenimiento de aplicaciones de software. De esta manera los clientes pueden utilizar el software mediante navegadores.
- **Modelo Software como Servicio (SaaS):** Se basa en contratar un servicio completo que permita el acceso a una aplicación en internet según las necesidades de la empresa, evitando gastos en un contrato de licencia, en instalación y soporte al software. Una de las diferencias entre ASP y SaaS, es que la mayoría de aplicaciones ASP son programas de cliente-servidor con una interfaz Web HTML, mientras que las SaaS están diseñadas únicamente para el entorno Web, lo que permite que sea la más usada por los clientes.

**Origen y características de SaaS**

Con el desarrollo de la Internet surgió el concepto ASP (Proveedor de Servicios de Aplicaciones) (14), que no es más que una forma de acceder a aplicaciones y servicios de software a cambio de una cuota periódica, evitando gastos a las empresas de instalación, despliegue, mantenimiento, contrato de licencias, etcétera. Se basaba en una plataforma Cliente - Servidor donde solo hacían uso de algunas páginas HTML estáticas que no estaban diseñadas para su uso web. Cada cliente accedía a su propia base de datos por lo que la administración de la infraestructura era compleja y las actualizaciones se debían coordinar adecuadamente con todos los clientes. Por ello, este modelo de negocio no se desarrolló plenamente.

Con el desarrollo de la arquitectura orientada a servicios (SOA) se vuelve a definir el concepto Software como Servicio o bajo demanda, superando todas las insuficiencias del modelo de negocio ASP.

Entre sus principales características se encuentran:

- El software se hospeda en el servidor del proveedor.
- Controla la gestión de los clientes, gestión de proyectos y puntos de venta.
- El software es propiedad del proveedor.

- El cliente paga una cuota por tener acceso al software en línea.
- Introduce el concepto “Multitenancy”, que no es más que la capacidad de dar servicio con una misma infraestructura (servidor de BD y aplicaciones) a varios suscriptores, a diferencia de ASP que utiliza generalmente uno para cada cliente, lo cual permite lograr (con SaaS) una escalabilidad sin límites.

**Diferencias de SaaS con ASP.**

Con SaaS desaparecen las licencias, siendo el pago por consumo o uso el modo de facturación empleado. Existe el hecho de que muchos ASP necesitan del despliegue de módulos adicionales del lado del cliente, cosa que no sucede en SaaS donde todas las aplicaciones se encuentran en los servidores del proveedor. SaaS da la posibilidad de hostear las aplicaciones, dando servicio a muchos clientes, por lo que su infraestructura es pública, frente a la utilizada con ASP que es privada. Quizás el aspecto más significativo es el concepto introducido por SaaS de “Multitenancy”, que no es más que la capacidad de dar servicio con una misma infraestructura (servidor de BD y aplicaciones) a varios suscriptores, a diferencia de ASP que utiliza generalmente uno para cada cliente, lo cual permite lograr (con SaaS) una escalabilidad sin límites. Esto trae consigo una serie de factores que acentúan aún más las diferencias entre uno y otro, por ejemplo: es mucho más fácil actualizar y dar mantenimiento a una infraestructura que da servicio a muchos clientes (SaaS), que hacerlo para cada conjunto de servidores que se tenga por cliente (ocurre muchas veces en ASP), por lo que el período con que se realizan modificaciones en la aplicaciones en favor de los afiliados equidista mucho de uno a otro modelo. A la vez, es más probable que se produzcan errores durante el proceso en ASP, por el simple hecho de que se han de realizar operaciones en cada uno de los servidores que se tienen destinados para cada cliente, incrementando la posibilidad de que ocurra algún fallo.

**La Computación en la Nube y SaaS**

Existen muchas definiciones sobre el Cloud Computing o **Computación en la Nube** (15)**,** este término se puede definir como una tecnología o modelo de prestación de servicios a través de Internet que permite utilizar aplicaciones sin necesidad de instalar algún software o herramienta.

El término nube se refiere a las tecnologías como la visualización, técnicas de programación que permite que una misma ejecución de una aplicación dé servicio a varios clientes y/o habilidades para la escalabilidad, balanceo de carga y rendimiento óptimo, para conseguir ofrecer el recurso de una manera rápida y sencilla. La Computación se basa en las funcionalidades que debe cumplir la aplicación.

El Cloud Computing se divide en tres niveles o capas según los servicios que ofrecen, los cuales son (15):

- **IaaS:** Infraestructura como Servicio, se encuentra en la capa inferior y se basa en contratar almacenamiento y capacidad de procesos (CPU). Es una solución basada en virtualización en la que se paga por consumo de recursos: espacio en disco utilizado, tiempo de CPU, espacio en base de datos, transferencia de datos. Ejemplo de IaaS: Los web service de Amazon y Azure de Microsoft.
- **PaaS:** Plataforma como Servicio, es la capa intermedia y la que ofrece todo lo necesario para soportar el ciclo de vida completo de construcción y pone en marcha las aplicaciones y servicios web completamente disponibles en la Internet. Ejemplo de PaaS: Google App Engine
- **SaaS:** Software como Servicio (Software as a Service) se encuentra en el nivel más externo o en la capa más alta y se caracteriza por brindar servicios mediantes una aplicación en Internet, en dependencias de las necesidades de la empresa o cliente. Ejemplos: Salesforce y Basecamp.

## 1.5 Características de la Plataforma de Servicios Bioinformáticos

La Plataforma de Servicios Bioinformáticos es un sistema que estará compuesto por una aplicación web y los servicios bioinformáticos que tienen software ya terminados tales como: AlasSIRNA-Design1.0, T-arenal 2.0 y BioSyS 1.0. El objetivo que se persigue es unir todos los servicios en una misma aplicación que se configurará según las necesidades del cliente.

**Módulos que componen la Plataforma de Servicios Bioinformáticos**

1. **alassiRNA-desing:** permite la toma de decisiones a científicos en el estudio de enfermedades, así como para la creación de medicamentos efectivos a partir del genoma humano y la creación de ARN de transferencia artificial llamado siRNA.

2. **BioSyS:** su objetivo fundamental es permitir que los investigadores realicen simulaciones a sistemas biológicos que estén descritos por ecuaciones diferenciales ordinarias. Estas simulaciones se pueden hacer de forma local o distribuida. Además el sistema almacena la información en una base de datos para el meta-análisis al conjunto almacenado.

3. **T-arenal 2.0:** es una multiplataforma implementado en Java que permite que computadoras con diferentes arquitecturas de hardware y sistemas operativos puedan ser utilizadas para la realización de tareas distribuidas a partir de una interfaz gráfica de usuario desde la cual se gestionan todas las tareas que se realicen de forma distribuida.

4. **Almacenamiento Distribuido:** tiene como objetivo: Integrar al Portal de Servicios Bioinformáticos de la UCI a un Sistema de Archivos Distribuidos (SAD) que utilice el espacio disponible de las estaciones de trabajo, para almacenar los datos biológicos. Dicho sistema utilizaría el espacio disponible de las estaciones de trabajo a partir de una interfaz Web que permitiría la gestión de los datos almacenados en el SAD y una fachada que posibilita la comunicación entre estos dos componentes.

5. **Módulo Básico:** su principal objetivo es implementar las funcionalidades básicas de la Plataforma de Servicios Bioinformáticos, como por ejemplo: Edición de Secuencias, Visualizar Gráficamente la Información obtenida en Secuencia, Visualizar en 3D, entre otras funcionalidades.

## 1.6 Estudio de los principales mercados y competidores

Con este estudio se identificarán elementos significativos que según la investigación realizada están incidiendo en el macro entorno (oportunidades y amenazas) en el que se encuentra el grupo de bioinformática de la UCI. Adicionalmente, se hará énfasis en los segmentos de mercado que se sugiere esté dirigida la Plataforma de Servicios Bioinformáticos.

**OPORTUNIDADES**

1. La actual coyuntura político-social favorable que estimula la comercialización para el autofinanciamiento y sostenibilidad de proyectos que promuevan el desarrollo en todas sus facetas del país de forma organizada y armónica, (aspiración del 6to. Congreso del PCC).

2. Infraestructura nacional de organizaciones en la esfera de la informática, entre las que se pueden mencionar: Citmatel, Desoft SA, Avante, SoftCal y recientemente la Universidad de Ciencias Informáticas (UCI).

3. Reconocimiento de la Bioinformática como un campo crucial para el desarrollo científico del país, a pesar de ser considerada una ciencia emergente donde convergen diferentes áreas del conocimiento tales como la biología, la química, la física y la informática.

**AMENAZAS**

1. Poca infraestructura tecnológica a nivel de país, para acceder a servidores internacionales.

2. Poca disponibilidad financiera del país para destinar gran parte de su presupuesto económico para el desarrollo de redes tecnológicas.

3. El bloqueo económico, comercial y financiero de Estados Unidos contra Cuba impide el acceso a su mercado y nos obliga a invertir varias veces más recursos al tener que recurrir a proveedores y clientes más distantes. La industria de la Bioinformática tampoco está exenta.

El entorno indica que resulta positiva la idea de la creación de una plataforma para brindar servicios bioinformáticos que pueden contribuir al desarrollo del país en esta importante esfera y también pudiera aportar recursos financieros al país. Con esta plataforma se lograra aprovechar las redes establecidas en la universidad y con ello utilizar mejor el potencial existente, minimizando la limitante nacional al respecto. Puede beneficiarse a su vez la infraestructura en organizaciones para avanzar en alianzas estratégicas con estos centros para el desarrollo en la creación de software, si fuera propicio.

A continuación se realiza una segmentación geográfica del mercado, tomando en consideración los estudios realizados.

**Segmentación del mercado y caracterización**

**I-Mercado nacional**

Para una posible comercialización de los servicios bioinformáticos en el mercado nacional el producto tendría como principales beneficiarios al Centro de Inmunología Molecular y el Centro de Ingeniería Genética y Biotecnología. También se pueden beneficiar de los servicios de la plataforma las entidades siguientes:

- Instituto Pedro Kuri (IPK)
- Centro Nacional de Sanidad Agropecuaria (CENSA)

- Instituto de Ciencia Animal (ICA)
- Labiofam
- Universidad de Camagüey y Holguín

Las entidades anteriormente mencionadas presentan restricciones en el acceso a servidores internacionales que brindan servicios como el Docking y poco conocimiento en este campo.

**II- Mercado internacional**

Se define como mercado internacional a los países que arrojan mejores resultados y desarrollo tecnológico en la informática y que hegemónicamente dominan el mercado, atendiendo a su relación con Cuba.

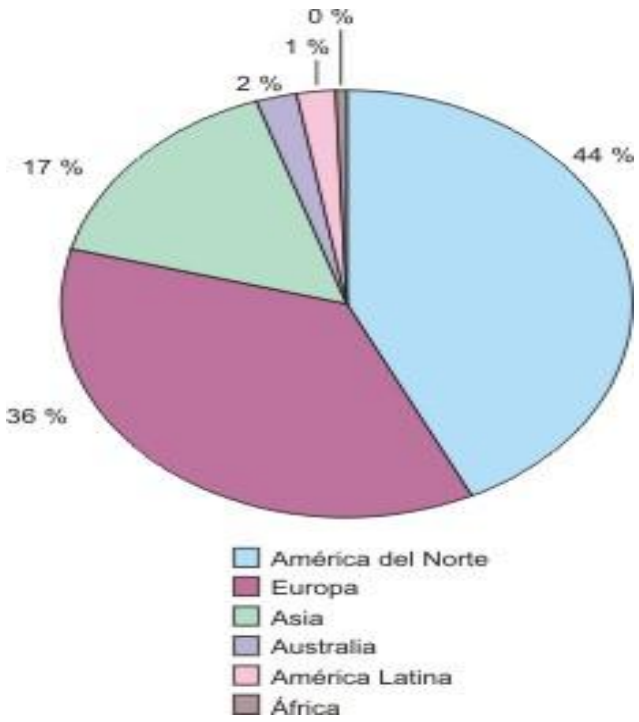


Figura 2. Producción científica de artículos bioinformáticos en el Web of Science. Fuente: Artículos. Bioinformática: en busca de los secretos moleculares de la vida.

**II.1 - Países de Latinoamérica**

La industria del software en Latinoamérica tiene una participación del 3.9 % del gasto total en Tecnología de la Información del mundo, siendo Chile el de mayor fuerza en este sentido, luego le suceden países como Uruguay, Panamá, Costa Rica y Brasil. Ver figura 2.

**II.2 – Asia**

La zona asiática apunta a un ligero incremento en este sentido, pero debido a su alta población los indicadores muestran que aun no son suficientes los resultados obtenidos. La India ha venido emergiendo como una gran potencia en la producción de software en los últimos tiempos, del 2009 al 2010 creció un 24 % por conceptos de subcontratación de servicios y exportaciones de la industria del software y la información.

**II.3 Europa**

Europa como región, según datos del Foro Económico Mundial, sigue ocupando una posición relevante en los rankings en la red de disponibilidad de las nuevas tecnologías y en específico de las infraestructuras para el desarrollo.

| Ranking | Economía | Valor |
|---|---|---|
| 1 | Dinamarca | 5.85 |
| 2 | Suecia | 5.84 |
| 3 | Singapur | 5.67 |
| 4 | Suiza | 5.58 |
| 5 | Finlandia | 5.53 |
| 6 | Islandia | 5.50 |
| 7 | Noruega | 5.49 |
| 8 | Holanda | 5.48 |

Tabla 1. Disponibilidad de las nuevas tecnologías europeas. Fuente: Foro Económico Mundial 2009-2010

**II.4 –** América del Norte

Es la que marca el desarrollo del software a nivel global, a pesar que de la actualidad está marcada por una fuerte crisis global que tuvo su comienzo en la esfera inmobiliaria de los Estados Unidos de América y se ha extendido a todo el planeta con consecuencias para todos los países. La industria de las TICs no es inmune a la crisis y según el Foro Económico Mundial (2009-2010) la industria crecerá en un 2.9 % por debajo del 4.9% estimado.

| Ranking | Economía | Valor |
|---|---|---|
| 1 | Estados Unidos | 5.68 |
| 2 | Canadá | 5.41 |

Tabla 2: Disponibilidad de las nuevas tecnologías en América del Norte 2008-2009. Fuente: Foro Económico Mundial 2009-2010

**Clientes meta:**

1- El mercado nacional ya que ayuda al desarrollo e informatización de la sociedad cubana en la rama de la Bioinformática, sustituye importaciones y protege el patrimonio científico de la nación.

2- Países miembros del ALBA, que consumen servicios profesionales de la salud mediante convenios y asociaciones. Sin embargo no existe un desarrollo en la rama de la Bioinformática.

3- África: a través de la promoción y ventas de los servicios bioinformáticos asociados a la rama de la salud, ya que se mantiene en el último lugar del mundo.

**Principales competidores**

DockingServer: conocido como Acoplamiento Molecular, es una web que está integrada por software de química computacional. La figura muestra las dificultades que presenta para los países subdesarrollados ya que no pueden hacer grandes gastos en estos servicios que se destinan a nuevos estudios de tal envergadura.

DOCKINGSERVER

Home | Pricing | Getting Started | Consulting Services | References | Forum | About Us | Sign in | Register

Pricing and Payment Options

Complete List of Available Service Plans

| Features | Guest Account | Free Registration | 3 days | 1 month | Annual | Annual-group |
|---|---|---|---|---|---|---|
| Subscribe | Start Now | Register | Register | Register | Register | Contact us |
| Prices | Free | Free | $15 | $79.90 | $559.90 | $2999.90 |
| Max. docking calculations / day | Unlimited | 2 | 10 | 100 | Unlimited | Unlimited |
| Max. # of runs in docking calculations | 2 | 10 | 255 | 255 | 255 | 255 |
| Max. # of dedicated processors | ✗ | ✗ | 2 | 2 | 2 | 2 |
| Processing multi-ligand .sdf files | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ |
| Quantum chemical ligand preparation with MOPAC 2009 | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ |
| Quantum chemical protein preparation with MOPAC 2009 | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ |
| Job priority in the queue | lowest | low | medium | high | highest | highest |
| Max. storage space | ✗ | 100 MB | 500 MB | 1000 MB | 2000 MB | 2000 MB |
| Storage guarantee | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ |
| 24-hour email support | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ |
| Forum access | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Number of users | ✗ | 1 | 1 | 1 | 1 | 10 |

Figura 3: Tabla de precios y servicios para realizar Acoplamiento Molecular en DockingServer.

Geneious Pro: es un producto competidor para la plataforma compuesto por funciones bioinformáticas como alineación y análisis de secuencia, creación de árboles filogénicos, incluye funciones para la traducción de proteínas extracción, estructura de visualización en 3D, análisis de datos bio-moleculares u otras funciones básicas. Ofrece una descarga gratuita por 15 días para que todos los usuarios puedan hacer uso del producto. Está valorado en los $ 795.00 USD.

El SNP-Vista, es una aplicación que brinda servicios públicos de visualización que ayuda a los científicos en el análisis de la nueva secuencia de datos relacionados con la enfermedad encaminado al descubrimiento de genes asociados y datos para el estudio de recombinación homóloga en las poblaciones microbianas.

Para la Simulación y Análisis de Sistemas Biológicos existen muchos productos similares como E-Cell, ByoDyn, BioSPICE entre otras herramientas para la simulación y modelación de procesos.

A pesar del avance y desarrollo de la Bioinformática de forma global posee desventajas para el consumo de sus productos debido al carácter privativo del software, la constante actualización de la licencia y productos que finalmente tienen poca de frecuencia de uso y el gasto excesivo que origina en un período corto de tiempo. La plataforma estará compuesta por un portal web donde el usuario accederá a los servicios solicitados pagando una cuota mensual por el mismo y ahorra en este sentido.

El departamento de Bioinformática tiene identificados una serie de servicios para desarrollar, tales como:

1. Visualizar un Polimorfismos de Nucleótido Simple (SNP) influyentes en el silenciamiento de un microRNA (pequeña partícula de Ácido Ribonucleico).
2. Búsqueda y visualización de los nombres de los transcritos contenidos en un gen especificado por el usuario y su análisis.
3. Buscar, seccionar y mostrar los identificadores de los miRNAs maduros contenidos en un miRNA especificado por el usuario y su análisis.
4. Búsqueda de gen por identificador.
5. Crear archivo fasta con información del genoma humano.
6. Análisis y simulación de sistemas biológicos.
7. Visualización de moléculas en 3D.
8. Mostrar índice de hidrofobicidad.
9. Mostrar segmento de transmembrana.
10. Análisis de Secuencias
11. Mostrar marcos abiertos de lecturas
12. Acoplamiento molecular

Basado en el estudio de los principales competidores se proponen los siguientes servicios a comercializar:

- Análisis de Secuencias: incluyendo dentro del mismo alinear y editar secuencias, y mostrar información gráfica de las mismas
- Visualizar un Polimorfismos de Nucleótido Simple (SNP) influyentes en el silenciamiento de un microRNA (pequeña partícula de Ácido Ribonucleico): muestra aquellos SNPs que tienen influencia en el poder silenciador de un miRNA determinado.
- Análisis y Simulación de Sistemas Biológicos: realiza un análisis de simulaciones por reglas,

bifurcaciones, estabilidad entre otras.

- Acoplamiento Molecular: este servicio permite realizar cálculos de acoplamientos, diseñar fármacos como también la evaluación de los resultados de los investigadores.

Desde un punto de vista arquitectónico dichos servicios se convierten en casos de uso arquitectónicamente significativos por ser los de más oferta-demanda en el mercado.

Como estrategia de comercialización se puede incluir el servicio Foro Bioinformático para que los usuarios registrados puedan compartir información e interactuar entre sí, logrando con esto atraer a más clientes a los servicios que brinda el portal.

Para realizar una promoción de los servicios bioinformáticos el principal medio sería la Internet, además de los eventos de ciencias y tecnologías que se realizan en nuestro país. El producto se puede presentar en eventos tales como:

- Fórum de Ciencia y Técnica a nivel de Universidad
- UCIENCIA
- Feria de Productos UCI
- Jornada Científica Estudiantil
- Evento Policlínico-Hospital
- Feria Informática Habana

## 1.7 Fundamentación de la propuesta de modelo de negocio seleccionado

A partir de las características de los modelos de negocio existentes en este trabajo se selecciona SaaS. Entre otras razones es mucho más fácil actualizar y dar mantenimiento a una infraestructura que da servicio a muchos clientes y no requerir el despliegue de módulos adicionales por parte de este. La posibilidad de acceder a las aplicaciones, dando servicio a muchos clientes es otro de los factores considerados.

Se tuvo en cuenta además la segmentación y caracterización del mercado de la Plataforma de Servicios Bioinformáticos y la facilidad que brinda SaaS para que los clientes accedan a los servicios.

Las características de dicha plataforma también facilita la adopción de este modelo de negocio por su características web y la ya existencia de servicios.

**Ventajas del SaaS para los clientes**

- No necesita instalación

Indistintamente del Sistema Operativo que tenga, solo se necesita un navegador web para conectarse a la aplicación.

- La inversión inicial es casi nula

Ahorro en Hardware y Licencias. El sistema de pago se hace mediante alquiler, solo se paga por el uso dado a la aplicación, se comparte el gasto con los demás usuarios. El Software Saas y el Cloud Computing ponen a la disposición del cliente potentes aplicaciones con un gasto mínimo.

- Mayor disponibilidad y seguridad de los datos

Gestionar la seguridad de una red y de los datos en la actualidad es situación bastante compleja porque cada día existen nuevas amenazas a la seguridad e integridad de la información; es por eso que la mayoría de la empresas que ofrecen software como servicio brindan un SLA (acuerdo a nivel del servicio) a la medida para cada tipo de empresa o cliente.

- Mantenimiento vía conexión remota ágil y rápida

Solo se tiene que pagar una cuota mensual y mantener la red a punto para poder acceder a Internet. El cliente cuenta con un servicio rápido y profesional que se dedica a realizar tareas proactivas, administrativas, reactivas, de ajustes y soporte, logrando así obtener el máximo provecho de las soluciones implementadas.

- Accesible desde cualquier lugar y en cualquier momento

El acceso vía Internet permite una gran accesibilidad desde el ordenador, el usuario podrá acceder a su cuenta ya que estos servicios online están siempre disponibles, 24 horas al día y los 7 días de la semana. No hace falta estar en la oficina para acceder a ellos.

- Financieramente permite pasar de un modelo de costes fijos (clásico de licencia) hacia un modelo orientado a costes variables. El servicio siempre puede cancelarse en cualquier momento o disminuir el número de usuarios habilitados y pagar menor cuota.

**Ventajas de SaaS para las empresas**

- No hay licencias por número de usuarios, ya que solo tiene que pagarse por usar el servicio.

- El soporte técnico se realiza en línea, por lo que su costo y tiempo de respuesta es menor, solo debe tener acceso a la red.
- Para lograr aplicaciones que cumplan con las necesidades de las empresas en cuanto a funcionalidades del software y que puedan estar orientadas a la Web, utiliza la arquitectura orientada a servicios (SOA) para mejorar la agilidad de las aplicaciones.
- Mayor facilidad para llevar estadísticas de diverso tipo sobre sus clientes y su negocio, incluyendo la posibilidad de adquirir datos y perfiles de los clientes en un formato que facilita su inmediata sistematización y su aprovechamiento como "inteligencia comercial".

**Desventajas de SaaS**

1. Se puede llegar a un alto grado de dependencia del proveedor.
2. Posible incumplimiento de los acuerdos sobre el nivel de servicio, que guarda relación con el grado de confianza en el proveedor. Se pone en manos de un proveedor el funcionamiento y servicio de una aplicación de la cual se conocen sus ventajas pero se desconoce si el proveedor podrá cumplir con el nivel de servicio acordado. (9)

## 1.8 El modelo de negocio en la Arquitectura Software

La propuesta de SaaS como modelo de negocio determina que se debe tener en cuenta el cumplimiento de:

1- Requisitos funcionales del sistema: basado en el estudio de mercado ya que brinda los principales servicios que se pueden comercializar y realiza un estudio enfocado a las necesidades del cliente en el mercado. Influye en el diseño de las vistas arquitectónicas siendo estos considerados como casos de usos arquitectónicamente significativos.

2- Requisitos no funcionales que garanticen mitigar los principales factores que dificultan la adopción de SaaS por los clientes. Estos son: (16)

- rendimiento
- disponibilidad
- seguridad
- escalabilidad
- confiabilidad

Los requisitos asociados con la apariencia y la usabilidad son importantes para lograr el éxito en un entorno SaaS, como también el soporte ya que es otro factor importante que demandan los clientes. En la figura 4 se muestra la relación modelo de negocio con arquitectura.

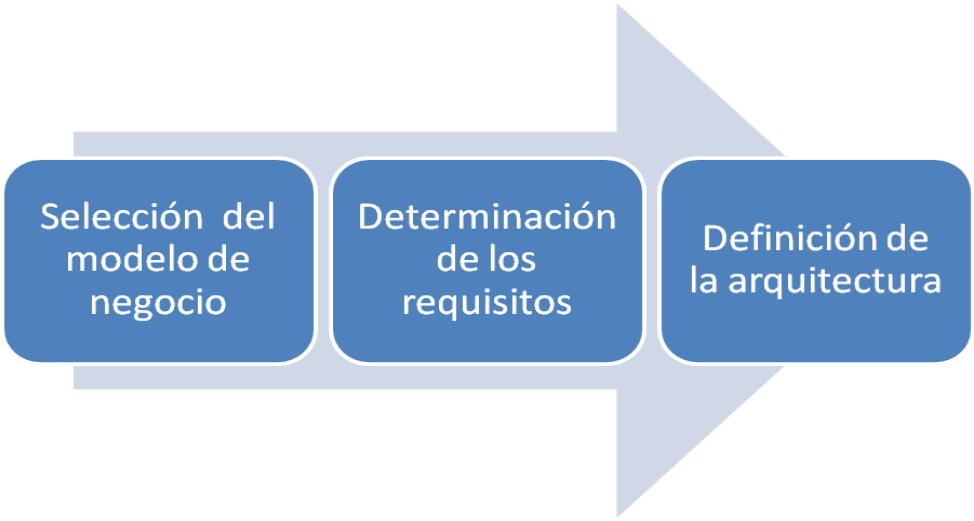


Figura 4: Definición de la arquitectura a partir del modelo de negocio.

### 1.9 Conclusiones del Capítulo

En este capítulo se abordaron los elementos teóricos que sustentan el problema científico y los objetivos del trabajo, se tuvieron en cuenta el estado del arte de los diferentes modelos de negocio existentes y se eligió SaaS para la Plataforma de Servicios Bioinformáticos, con el objetivo de comercializarlo. El modelo de negocio se toma en consideración desde su fase inicial del proyecto.

# CAPÍTULO 2: PROPUESTA DE ARQUITECTURA

## 2.1 Introducción

En el capítulo anterior se definió en el modelo SaaS, cuáles eran los competidores, los clientes hacia el cuál va a estar dirigido los servicios bioinformáticos y los requerimientos que debe cumplir la plataforma. En este capítulo se presenta una propuesta de arquitectura de software (AS) basada en el modelo de negocio seleccionado para la Plataforma de Servicios Bioinformáticos que ayuden a la toma de decisiones y al mismo tiempo proporcionar conceptos y un lenguaje común que permitan la comunicación entre el equipo de desarrollo que participa en este proyecto.

## 2.2 Definición de Arquitectura de Software

Muchos han sido los autores que han abordado el tema de AS y al mismo tiempo han realizado diferentes definiciones por lo que se mencionan las que de alguna forma llevaron el camino a lo que se considera hoy Arquitectura de Software.

| Autores | Definiciones |
|---|---|
| *IEEE Std 1471-2000 (17)* | *"La arquitectura es la organización fundamental de un sistema incorporada en sus componentes, en sus relaciones mutuas y en el entorno, y los principios que guían su diseño y evolución".* |
| *PRESSMAN, 2002 (18)* | *"La arquitectura es una descripción de los subsistemas y los componentes de un sistema informático y las relaciones entre ellos (...)".* |
| Philippe Kruchten (19) | *"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".* |
| Len Bass, Paul Clements y Rick Kazman (20) | *"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".* |

| | |
|---|---|
| RUP (21) | *"Arquitectura es la estructura de los componentes más significativos de un sistema interactuando a través de interfaces con otros componentes conformados por componentes sucesivamente pequeños e interfaces".* |

Tabla 3. Conceptos de Arquitectura de Software. Fuente: Elaboración Propia.

La Arquitectura de Software ha sido conceptualizada de varias formas pero de manera general se puede decir que es una descripción de los subsistemas y componentes (computacionales) de un sistema de software y las relaciones entre ellos. Constituye una sub-disciplina de la ingeniería del software. Pasa por todos los flujos de trabajo, se va perfeccionando, refinando a medida que vayamos realizando tareas particulares a lo largo del ciclo de vida del software y tiene como objetivo reutilizar tanto los componentes como los patrones estructurales.

## 2.3 Estilos y Patrones Arquitectónicos

Los estilos arquitectónicos son una generalización y abstracción de los patrones, representan los componentes y las relaciones entre ellos, define las reglas del diseño para su construcción y las restricciones del sistema; describe sistemas completos y son una guía para la organización del software. Están vinculados a la forma más general en la que está organizado un sistema de software específicamente determina el vocabulario de componentes y conectores que pueden ser utilizados en instancias de este estilo, con un conjunto de restricciones en las descripciones arquitectónicas.

A continuación se engloban una serie de estilos propuestos por Shaw y Garlan (22), quienes define las siguientes clases de estilos:

- **Estilos de Flujo de datos**
  - Tuberías y Filtros
- **Estilos centrados en datos**
  - Arquitecturas de Pizarra o repositorio
- **Estilos de Llamada y Retorno**
  1. Arquitectura en Capas
  2. Arquitectura Orientada a Objetos
  3. Arquitectura basada en Componentes

- **Estilo de Código Móvil**

  1. Arquitectura de Máquinas Virtuales

**3. Estilos heterogéneos**

  6. Sistema de control de procesos
  7. Arquitectura basada en Atributos

- **Estilos Peer To Peer**

  1. Arquitectura basadas en Eventos
  2. Arquitecturas Orientas a Servicios (SOA)

**Arquitecturas Orientadas a Servicios (SOA):** Es lo suficientemente flexible, elegante y ágil garantizando las soluciones que las empresas han anhelado siempre. Es una arquitectura de software que construye toda la topología de la aplicación como una topología de interfaces, implementaciones y llamados a interfaces. Es una relación entre servicios y consumidores de servicios, ambos lo suficientemente amplios como para representar una función de negocio completa.

**Ventajas:** Son varios los beneficios técnicos de una implementación de SOA entre ellos la re-potenciación del software anterior, conectividad, facilidad de mantenimiento, reducción de tamaño de proyectos, alta escalabilidad, reutilización real de los programas y mejora en tiempos de respuesta.

**Desventajas:** Dentro de los campos donde no se aconseja introducir SOA cabe mencionar aquellos donde frente a la flexibilidad se prefiera una centralización de la información, y ya se cuente con los programas y aplicaciones para ello. Otro campo donde SOA puede resultar menos atractiva es la relacionada con temas que requieran alta seguridad, resulta más complejo hacerlo con múltiples procesos independientes preparados fácilmente para compartir información que con un reducido número dentro de una aplicación monolítica.

**Fundamentación del estilo seleccionado.**

Para la Plataforma de Servicios Bioinformáticos se seleccionó dentro de la familia de estilos: el estilo **Peer to Peer** escogiendo dentro del mismo la Arquitectura Orientada a Servicios, ya que presenta dos objetivos fundamentales: uno desde el punto de vista empresarial y un segundo objetivo desde el

punto de vista de la tecnología (23).

1- Desde el punto de vista empresarial

Cuando una institución decide hacer uso de una arquitectura SOA es porque tiene objetivos específicos de negocio que cubrir, reducir costes, aumentar ingresos, mejorar la productividad, comunicación inter-empresarial con varias empresas y ajustar los sistemas a los requerimientos del negocio. También se puede decir que una arquitectura SOA consiste en una forma de modularizar los sistemas y aplicaciones en componentes de negocio que pueden combinar y re-combinar con interfaces bien definidas para responder a las necesidades de la empresa.

Con el uso de entornos orientados a servicios las diferentes instituciones pretenden mejorar la interacción con los clientes, proveedores, empleados y también reducir el ROI (Return of Investment) retorno de la inversión, es decir, conseguir una mayor rentabilidad de las inversiones tecnológicas.

1.1 Beneficios para el negocio

- Eficiencia. Transforma los procesos de negocio en servicios compartidos con un menor coste de mantenimiento.
- Capacidad de respuesta. Rápida adaptación y despliegue de servicios, clave para responder a las demandas de clientes y empleados.
- Adaptabilidad. Facilita la adopción de cambios añadiendo flexibilidad y reduciendo el esfuerzo.

2 - Desde el punto de vista tecnológico

Las arquitecturas SOA pretenden concebir las aplicaciones desde otro punto de vista, una aplicación orientada a servicios combina datos en tiempo real con otros sistemas capaces de fusionar los procesos de negocio.

Las aplicaciones basadas en SOA utilizan tecnología totalmente estándar como es XML y Servicios Web para la mensajería. Estándares como SOAP, Web Services Description Language (WSDL) y Business Process Execution Language (BPEL), estandarizan así la compartición de información, el modelo de integración de procesos y la cooperación entre aplicaciones.

Realizando aplicaciones orientadas a servicio se pueden conectar aplicaciones heterogéneas con el aumento de flexibilidad que supone, y un punto muy importante es que permite que las organizaciones interactúen cuando realmente lo requieran, sin necesidad de tener conexiones permanentes.

Como una arquitectura SOA se basa en estándares, el tiempo de aprendizaje de utilización de las tecnologías sobre las que se apoya se reduce drásticamente.

2.2 Beneficios Tecnológicos

- Reduce la complejidad gracias a la compatibilidad basada en estándares frente a la integración punto a punto.
- Reutiliza los servicios compartidos que han sido desplegados previamente.
- Integra aplicaciones heredadas limitando así el coste de mantenimiento e integración.
- Beneficios en el desarrollo, ya que las aplicaciones son reutilizables, más fácil de mantener y tienen la capacidad de ampliación de las funcionalidades del sistema, exponiéndolas de una forma segura.

Además según SaaS, SOA ayuda de manera más eficiente a la entrega de aplicaciones a medida que compiten cada vez más sobre el precio con los proveedores de software empaquetado y permite ser utilizado como un punto de integración de las aplicaciones SaaS y otros componentes o módulos que deseen forma parte de la plataforma, lo que concuerda también con la arquitectura propuesta por el departamento.

**Patrones Arquitectónicos**

Los patrones definen la estructura de un sistema software, los cuales a su vez se componen de subsistemas con sus responsabilidades, proporcionan un esqueleto sobre el cual se implementan las funcionalidades deseadas en un sistema. Además permite dar una respuesta a los requisitos no funcionales tales como confiabilidad, extensibilidad, reutilización, facilidad de uso; en el sentido de proporcionar criterios o elementos para la selección.

**Clasificación de los patrones**

**Patrones de arquitectura**

Se basa en la interacción de objetos dentro o entre los niveles arquitectónicos, se aplican durante la fase de diseño inicial para resolver problemas arquitectónicos, adaptabilidad a requerimientos

cambiante, modularidad y acoplamiento. Define la estructura básica de una aplicación, provee un subconjunto de subsistemas predefinidos, incluyendo reglas y pautas para su organización y constituye una plantilla de construcción (24).

- **Modelo Vista Controlador (MVC):** es un patrón de arquitectura de software que se emplea para modular la interfaz de usuario, las reglas del negocio y el control de eventos, es decir separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes:

  1. Modelo: Administra el comportamiento y los datos del dominio de aplicación, responde a los requerimientos de la información (formulados desde la vista) y a las instrucciones de cambiar de estado (formulados desde el controlador).

  2. Vista: Maneja la visualización de la información.

  3. Controlador: Interpreta las acciones del ratón, informando al modelo y/o a la vista para que cambien según resulte apropiado.

**Ventajas de MVC**

- Se pueden mostrar distintas variantes de display simultáneamente, porque la vista no depende del modelo (ni el modelo de la vista).
- La interfaz tiende a cambiar más rápido que las reglas de negocios. Agregar nuevos tipos de vista no afecta al modelo.
- El modelo es reusable con distintas vistas (una vista web y una con interfaz de ventanas).
- División clara de trabajo entre los miembros de un equipo, que estará formado por personas con distintos niveles de especialización.
- Soporte de vistas múltiples. Dado que la vista se halla separada del modelo y no hay dependencia directa del modelo con respecto a la vista, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente.
- Adaptación al cambio. Los requerimientos de interfaz de usuario tienden a cambiar con mayor rapidez que las reglas de negocios. Los usuarios pueden preferir distintas opciones de representación, o requerir soporte para nuevos dispositivos. Dado que el modelo no depende de las vistas, agregar nuevas opciones de presentación generalmente no afecta al modelo.

**Desventajas del MVC**

- Puede aumentar un poco la complejidad de la solución. Como está guiado por eventos, puede ser algo más difícil de depurar.
- Si hay demasiados cambios en el modelo, la vista puede caer bajo un diluvio de refrescos, a menos que se prevea programáticamente (por ejemplo, actualizando varias modificaciones en lotes)

Para la arquitectura propuesta se recomienda seguir como patrón arquitectónico el MVC debido a que organiza el código en base a su función; separa el código en tres capas principales lo cual facilita el proceso donde el usuario interactúa con la interfaz de usuario, el controlador recibe la notificación de la acción solicitada y gestiona el evento y accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario, el controlador además delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo. Permitiendo de esta forma un perfecto acoplamiento entre el modelo y la vista característica que se observa generalmente en aplicaciones Web.

**Patrones de diseño**

Se emplean durante la fase de diseño detallado para solucionar problemas de claridad del diseño, multiplicación de clases, adaptabilidad a requerimientos cambiantes. Mejoran el mantenimiento de sistemas ya que proveen una especificación explícita de clases e interacción entre objetos, proveen un vocabulario para discutir y comunicar decisiones de diseño en término de estructuras de clases en lugar de objetos y ayudan a elegir diseños alternativos que hacen un sistema reutilizable y evitan alternativas que comprometan la reutilización (24).

Dentro de los patrones de diseño se encuentran:

**Patrones GRASP** (Patrones de Asignación de Responsabilidades): comunican los principios fundamentales de asignación de responsabilidades en el diseño orientado a objetos. Dentro de ellos se destacan:

**- 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. Si estas asignaciones de

responsabilidades se hacen en la forma adecuada, los sistemas tienden a ser más fáciles de entender, mantener y ampliar, lo que nos ofrece la garantía de poder reutilizar los componentes en futuras aplicaciones.

Beneficios:

- Se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta un bajo acoplamiento, lo que favorece el hecho de tener sistemas más robustos y de fácil mantenimiento.
- El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clases "sencillas" y más cohesivas que son fáciles de comprender y mantener.

**- Creador**

- Problema: ¿Quién debería ser el responsable de crear una nueva instancia de alguna clase?
- Solución: La responsabilidades 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).

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.

Beneficios:

- Brinda un soporte al bajo acoplamiento, lo cual supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización. Es probable que el acoplamiento no aumente, pues la clase creada tiende a ser visible a la clase creador, debido a las asociaciones actuales que llevaron a elegirla como el parámetro adecuado.

**- 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.

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.

Beneficios:

- No se afectan por cambios en otros componentes.
- Fáciles de entender por separado.
- Fáciles de reutilizar.

**- 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).

Una clase con mucha cohesión presenta beneficios tales como: facilidad de mantenimiento, comprensión y uso. Su alto grado de funcionalidad, combinada con una reducida cantidad de operaciones, también simplifica el mantenimiento y los mejoramientos. La ventaja que significa una gran funcionalidad también soporta un aumento de la capacidad de reutilización.

**- 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.

Beneficios:

- Mayor potencial de los componentes reutilizables. Garantiza que la empresa o los procesos de dominio sean manejados por la capa de los objetos de dominio y no por la de la interfaz.
- Reflexionar sobre el estado del caso de uso.

**Patrones GOF** (Gang of Four o Pandilla de los cuatro) son utilizados en innumerables ocasiones cuando se desarrollan aplicaciones informáticas, se clasifican según su propósito en creacionales, estructurales y de composición, mientras que respecto a su ámbito se clasifican en clases y objetos:

**Respecto a su propósito**

- **Patrones creacionales**: El objetivo de estos patrones es abstraer el proceso de instanciación y ocultar los detalles de cómo los objetos son creados o inicializados. Proporcionan a los programas una mayor flexibilidad para decidir que objetos usar. Solucionan problemas de creación de instancias. Ayudan a encapsular y abstraer dicha creación. Patrón de Fábrica Abstracta (Abstract Factory), Patrón Constructor (Builder),Patrón del Método de Fabricación (Factory Method), Patrón Prototipo (Prototype), Patrón de Instancia Única (Singleton).

- **Patrones estructurales**: Describen como las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades. Estos objetos adicionados pueden ser incluso objetos simples u objetos compuestos. Solucionan problemas de composición (agregación) de clases y objetos. Patrón Adaptador (Adapter), Patrón Puente (Bridge), Patrón Compuesto (Composite), Patrón Decorador (Decorator), Patrón de Fachada (Facade), Patrón de Peso Mosca o ligero (FlyweigthPatrón Proxy (Proxy)).

- **Patrones de comportamiento:** Ayudan a definir la comunicación e iteración entre los objetos de un sistema. El objetivo de este patrón es reducir el acoplamiento entre los objetos. Dan soluciones respecto a la interacción y responsabilidades entre clases y objetos, así como los algoritmos que encapsulan. Patrón de Cadena de Responsabilidad (Chain of Responsibility),Patrón de Comando (Command),Patrón Intérprete (Interpreter), Patrón Iterador (Iterator), Patrón Mediador (Mediator),Patrón Recuerdo (Memento), Patrón Observador (Observer),Patrón de Estado (State), Patrón de Estrategia (Strategy), Patrón del Método Plantilla (Template Method), Patrón Visitante (Visitor).

Como patrones de diseño GOF se recomienda utilizar el patrón Fachada ya que es un patrón estructural que brinda una sencilla interfaz para conectar o servir de intermediaria con un sistema o varios sistemas más complejos con sus respectivas interacciones. Ayuda a reducir la complejidad del

mismo mediante una clase común que disminuye las dependencias y comunicaciones entre subsistemas. En la figura se presenta la estructura del mismo.

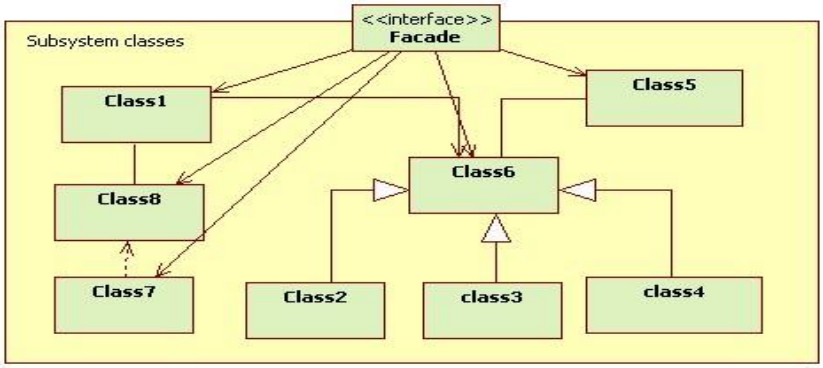


Figura 6: Estructura del patrón Fachada.

**Ventajas que presenta este patrón**

- Protege al cliente de los componentes del subsistema, así reduce el número de objetos con los que él se relaciona y hace más fácil usar el subsistema.

- Promueve un acoplamiento débil entre el cliente y el subsistema. Esto permite modificar componentes en el subsistema sin que afecte al cliente.

- Evita que se use clases del subsistema si la aplicación no la necesita.

## 2.4 Ambiente de desarrollo

El departamento de Bioinformática tiene concebido su entorno de desarrollo compuesto por: como metodología de desarrollo Open UP, utilizando UML para describir la arquitectura y como lenguaje de modelado y Visual Paradigm como herramienta CASE. Lenguaje de programación del lado servidor Java y del lado cliente java script, utilizando Eclipse como entorno de desarrollo y Subversión como sistema de control de versiones. Frameworks de desarrollo Spring y Hibernate por los beneficios que presenta. Gestor de BD PostgreSQL y el servidor de aplicaciones Tomcat.

## 2.5 Metas y restricciones arquitectónicas

La plataforma estará compuesta por varios servicios bioinformáticos de los cuales los más significativos estarán basados en el estudio de mercado propuesto por el modelo de negocio SaaS,

explicado en el capítulo 1. Dichos servicios se convierten en los requerimientos o requisitos funcionales (RF) que de cumplir el sistema y a su vez interpretados como casos de uso arquitectónicamente significativos. A continuación se mencionan los RF:

Analizar secuencias.

- RF1: Alineación de secuencias.
- RF2: Editar secuencias.
- RF3: Visualizar información gráfica de secuencia.
- RF4: Visualizar SNPs influyente en el silenciamiento de un miRNA.
- RF5: Analizar y Simular Sistemas Biológicos.
- RF6 Acoplar Moléculas.
- RF7: Compartir información. Este hace referencia al servicio Foro Bioinformático propuesto en el capitulo anterior pero se define como compartir información ya que los casos de uso se escriben en infinitivo.
- Autenticar Usuario.
- Gestionar Usuario.

Dentro de los requerimientos arquitectónicos anteriormente mencionados, también se insertan un grupo de requisitos no funcionales (RNF) que la plataforma debe cumplir; que son propiedades que la hacen atractiva, usable, rápida y confiable. Basados en las necesidades del cliente según el modelo SaaS, identificados en el epígrafe 1.5 de capítulo 1. Seguidamente se explican cada uno de ellos:

1. **RNF de Rendimiento**
   1.1 Los tiempos de respuesta del sistema debe ser menor de 1 minuto garantizando de esta forma la agilidad del mismo.
   1.2 Las páginas deberán ser lo más ligeras posibles, con pocas imágenes de calidad aceptable y el sistema debe requerir un consumo mínimo de recursos.

1.3 La eficiencia de la aplicación estará determinada en gran medida por el aprovechamiento de los recursos que se disponen mediante el uso del patrón MVC y la velocidad de las consultas a la base de datos.

**2. RNF de Disponibilidad**

2.1 La disponibilidad del sistema debe ser continua con un nivel de servicio para los usuarios de 7 días por 24 horas, garantizando un esquema adecuado que permita ante una posible falla de la solución en cualquiera de sus componentes, contar con una contingencia o generación de alarmas.

2.2 Debe contemplar requerimientos de confiabilidad ante recuperaciones, en caso de fallas de algún componente, no debe haber pérdida de información.

**3. RNF de Confiabilidad**

3.1 La información manejada por el sistema será objeto de cuidadosa protección contra la corrupción y estados inconsistentes. Deberán existir mecanismos de chequeo de integridad.

3.2 Se debe crear de copias de respaldo que puedan restaurar el sistema en caso de fallo crítico o pérdida total de la información.

**4. RNF de Seguridad**

4.1 Para acceder a los servicios de la plataforma, 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.

4.2 El sistema mantendrá en todo momento las trazas que se corresponden con las diferentes situaciones críticas que se puedan ocurrir. Dichas trazas 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. Cada componente podrá registrar la información que considere deba formar parte de la misma.

4.3 Debe contar con protección contra acciones no autorizadas o que puedan afectar la integridad de los datos.

4.4 Se deben garantizar comunicaciones seguras entre los clientes y el servidor mediante el protocolo https.

**5. RNF de Escalabilidad**

5.1 El diseño debe contemplar el uso óptimo de recursos tales como conexiones a la base de datos.

5.2 Contemplar en el diseño la clara partición entre datos, recursos y aplicaciones para optimizar la escalabilidad del sistema.

5.3 Debe contemplar requerimientos de crecimiento para usuarios tanto internos como externos.

**6. RNF de Apariencia o Interfaz de usuario**

6.1 El sistema tiene que ofrecer una interfaz amigable y fácil de operar.

6.2 El diseño de la interfaz debe ser sencillo, con pocas entradas, permitiendo que no sea necesario mucho entrenamiento para que los usuarios puedan utilizar el sistema.

**7. RNF de Usabilidad**

7.1 El sistema podrá ser usado por aquellos usuarios que posean conocimientos básicos en el campo de la Bioinformática.

7.2 Debe ser capaz de tener una fácil navegación por todas las funcionalidades del sistema, desde cualquier página.

7.3 Se debe lograr un diseño adaptable, con la capacidad de poder soportar funcionalidades adicionales o modificar las funcionalidades existentes sin impactar el resto de los requerimientos contemplados en el sistema.

**8. RNF de Diseño e implementación**

8.1 Lenguaje de programación Java.

8.2 IDE de desarrollo Eclipse.

8.3 Herramienta CASE Visual Paradigm.

8.4 Gestor de bases de datos PostgreSQL.

8.5 Para el desarrollo del sistema se utiliza el Framework Hibernate para Java.

8.6 El diseño de cada uno de los módulos que conforman el sistema debe seguir el patrón en Capas y solo en los casos en que sea necesario violar la capa de control se seguirá el patrón MVC.

8.7 Legales: La plataforma con todos sus módulos y toda la documentación generada pertenecen al departamento de Bioinformática perteneciente al Centro de Tecnología y Gestión de Datos y a la Universidad de las Ciencias Informáticas.

8.8 Tecnología JSP para la creación de sitios web dinámicos.

**9. RNF de Hardware**

9.1 Un servidor de Base de datos como mínimo con 300 GB de disco duro ya que esta compuesto por 5 módulos los cuales cada uno requiere de 80 GB. El servidor estará compuesto por BD para usuarios, moléculas, secuencias y proteínas. Además de 2 GB de memoria RAM y un microprocesador Dual Core o superior.

9.2 Un servidor de aplicaciones como mínimo de 80 GB de disco duro, 2 GB de memoria RAM y un microprocesador Dual Core o superior.

9.3 Un servidor de servicios web como mínimo con 80 GB de disco duro, 2 GB de memoria RAM y un microprocesador Dual Core o superior.

**10. RNF de Software**

10.1 En los ordenadores de los usuarios debe estar instalado:

- La Máquina Virtual de Java (JVM siglas en inglés de Java Virtual Machine).
- Al menos un navegador web con los complementos para visualizar applets3 de Java.

10.2 Para el servidor de servicios web se deben instalar los programas definidos por el departamento tales como:

- Clustalw 2.0.12: para realizar alineamientos de secuencias con Clustalw.
- Muscle 3.7: para realizar alineamientos de secuencias con Muscle.
- T_Coffee 8.84: para realizar alineamientos de secuencias con T-Coffee.
- Dock 2.6: Es una herramienta que realiza el acoplamiento molecular.

Juntos con estos se proponen los siguientes programas:

- Jemboss 6.1.0-5: es una suite de herramientas para procesar datos biológicos. Es fácil para seleccionar el programa más adecuado y aplicar la correcta selección de las opciones para garantizar el mejor análisis de los datos biológicos; desde la extracción de una secuencia de bases de datos a la exportación de una representación gráfica de los datos en un manuscrito de su publicación.

- Readseq 1.0: es un programa para la conversión de formatos de secuencias moleculares, útiles para diversos programas de bioinformática y servicios. Está escrito en Java, aunque existe una versión en el lenguaje de programación C que sigue estando disponible en la red.

**11. RNF de Soporte**

11.1 El sistema estará bien documentado de forma tal que el tiempo de mantenimiento sea mínimo en caso de necesitarse.

11.2 El sistema debe permitir aumentar la capacidad de almacenamiento, sin dejar de prestar los servicios.

11.3 La capacidad de almacenamiento del sistema debe aumentar de manera progresiva, sin que este incremento afecte la funcionalidad del mismo.

## 2.6 Vistas arquitectónicas

Una vista es la representación de un conjunto de elementos del sistema y las relaciones asociadas entre ellos. El diseño arquitectónico estará basado a través de las 4+1 vistas de la arquitectura propuestas por Philippe Kruchten ya que el modelo de negocio SaaS está enfocado principalmente a satisfacer las necesidades del cliente y la vista de casos de usos (CU) se centra en este estudio. El resto de las vistas tales como: la lógica, la de implementación y de despliegue muestran el camino para el diseño, implementación y despliegue de la plataforma, respetando la metodología de desarrollo *Open Up* que propone las mismas vistas:

- **Vista de Casos de Usos**

La Vista de Casos de Usos es la vista rectora ya que recoge los casos arquitectónicamente significativos, los cuales contienen los servicios bioinformáticos basados en el modelo de negocio.
En la figura 5 se muestran como quedaría dicha vista.

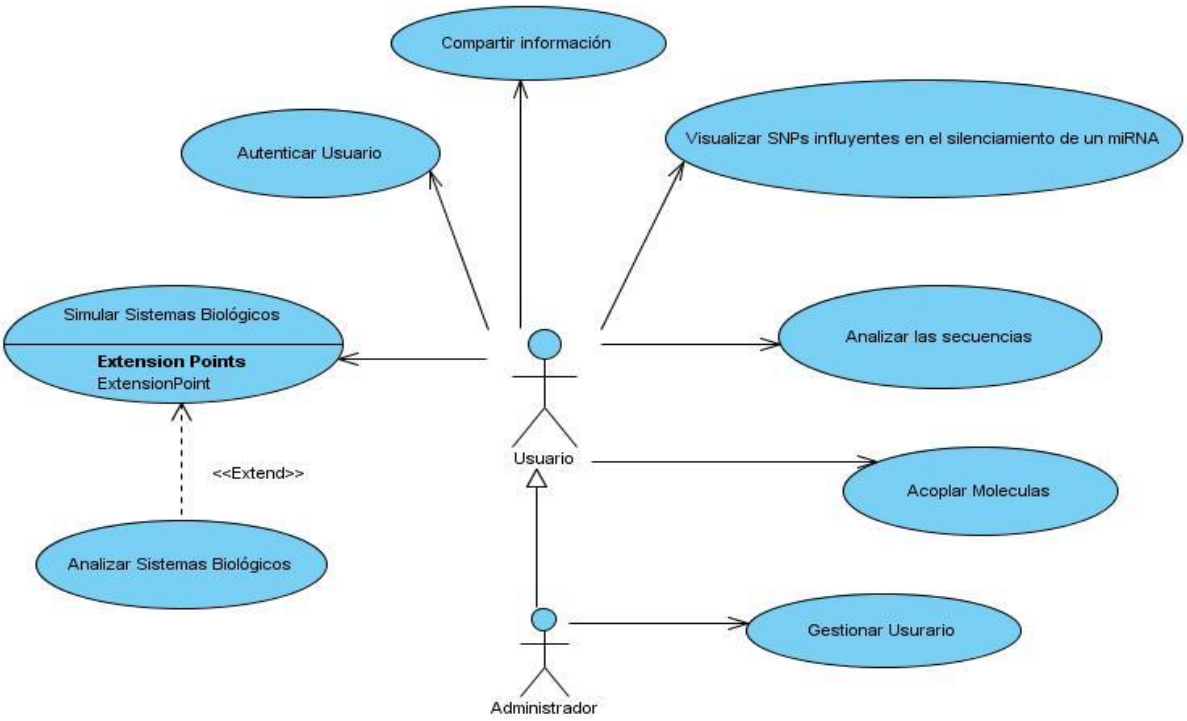


Figura 5: Vista de Casos de Usos. Fuente: Elaboración Propia.

**Usuario:** Puede ser un investigador o especialista con un perfil creado en la aplicación. Tiene acceso a todo lo publicado en el sitio. Tendrá privilegios extras según sean asignados por el administrador.

**Usuario administrador:** Encargado de gestionar toda la información de la aplicación. Establece los permisos pertinentes para los distintos tipos de usuarios.

A continuación se describen las casos de uso arquitectónicamente significativos correspondientes a los requerimientos funcionales del sistema; cuales fueron seleccionados porque el proceso que describen es de suma importancia para el mismo.

| Casos de Uso | Descripción |
|---|---|
| **Autenticar Usuario** | El objetivo de este caso consiste en concederles a los usuarios los permisos que tengan en el sistema. Este caso de uso pertenece al módulo básico. |
| **Gestionar Usuario** | Este caso de uso gestiona los usuarios del sistema, es decir |

| | |
|---|---|
| | crea, borra, actualiza y modifica los usuarios. Este caso de uso pertenece al módulo básico. |
| **Analizar Secuencia** | El sistema alinea las secuencias con el programa seleccionado y muestra el resultado de la operación. Este caso de uso pertenece al módulo básico. |
| **Visualizar SNPs influyente en el silenciamiento de un miRNA** | El usuario debe introducir el identificador (id) del miRNA, luego el sistema deberá mostrar los ids de los miRNAs maduros contenidos en dicho miRNA, y dar la posibilidad de escoger uno de ellos para el análisis. Este caso de uso pertenece al módulo alassiRNA-desing. |
| **Analizar y simular sistemas biológicos** | Gestiona la/las simulación/es que se realizan con un modelo matemático y un juego de valores pasados por parámetros. Este caso de uso pertenece al módulo Biosys. |
| **Acoplar Moléculas** | Carga los archivos entrados por el usuario y realiza los cálculos de acoplamiento molecular. Este caso de uso pertenece al módulo Biosys. |
| **Compartir información** | Permite a los usuarios intercambiar información y es conocer homólogos de este mismo campo. |

Tabla 4. Descripción de los casos de uso. Fuente: Elaboración propia.

- **Vista Lógica**

La vista lógica contiene las clases del diseño más importantes, organizadas por paquetes y subsistemas en capas de trabajo. Ver figura 6.

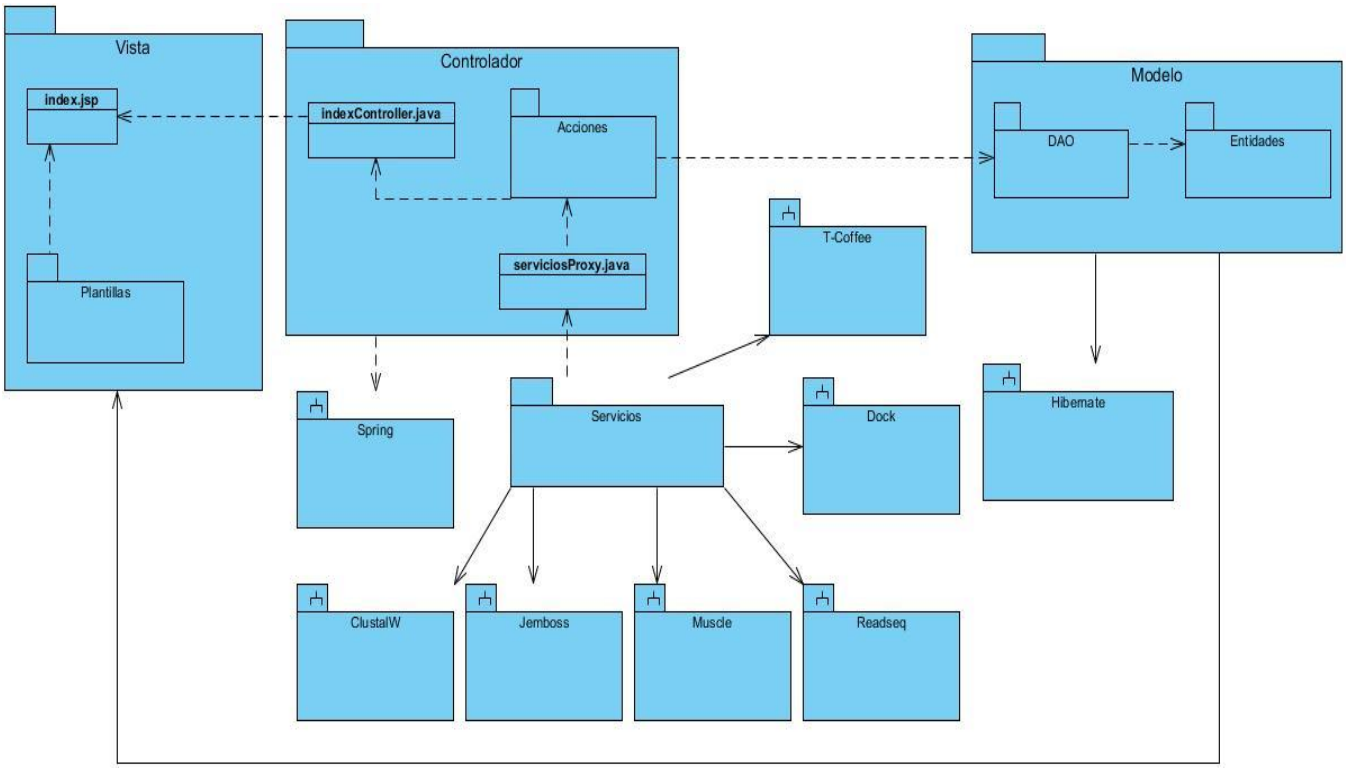


Figura 6: Vista Lógica General. Fuente: Elaboración Propia.

A continuación se expone la descripción de los paquetes y subsistemas del diseño que conforman la vista lógica.

**Paquete Vista**

Es la encarga de producir páginas que se muestran como resultado de las acciones. Agrupa todas las plantillas o páginas de cada módulo que se encargan de visualizar las variables definidas en el paquete del controlador. Constituyen la presentación de los datos de la acción que se están ejecutando. Para un mejor entendimiento del mismo, ver la siguiente figura.

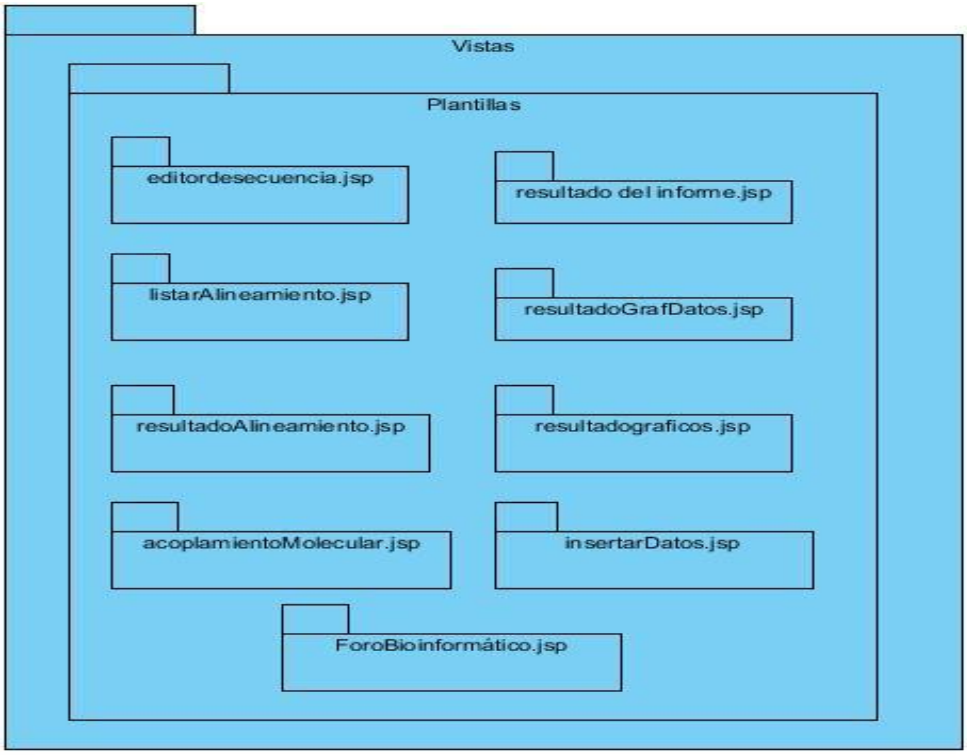


Figura 7: Vista Lógica del Paquete Vista. Fuente: Elaboración Propia.

- La pagina **index** es la principal y contiene los elementos que se muestran de forma idéntica para las páginas web a lo largo de toda la aplicación.
- **Paquete Plantillas:** Agrupa todas las plantillas o páginas de cada módulo que se encargan de visualizar las variables definidas en el paquete del controlador.

**Paquete Controlador**

Se encarga de realizar numerosas tareas, suele tener mucho trabajo entre los que se encuentran el manejo de las peticiones de la aplicación, el manejo de la seguridad, cargar la configuración de la aplicación y otras tareas similares. Incluyen el código específico del controlador de cada página y agrupa en el paquete Acciones las clases que implementan la lógica de la aplicación. Ver figura 8.

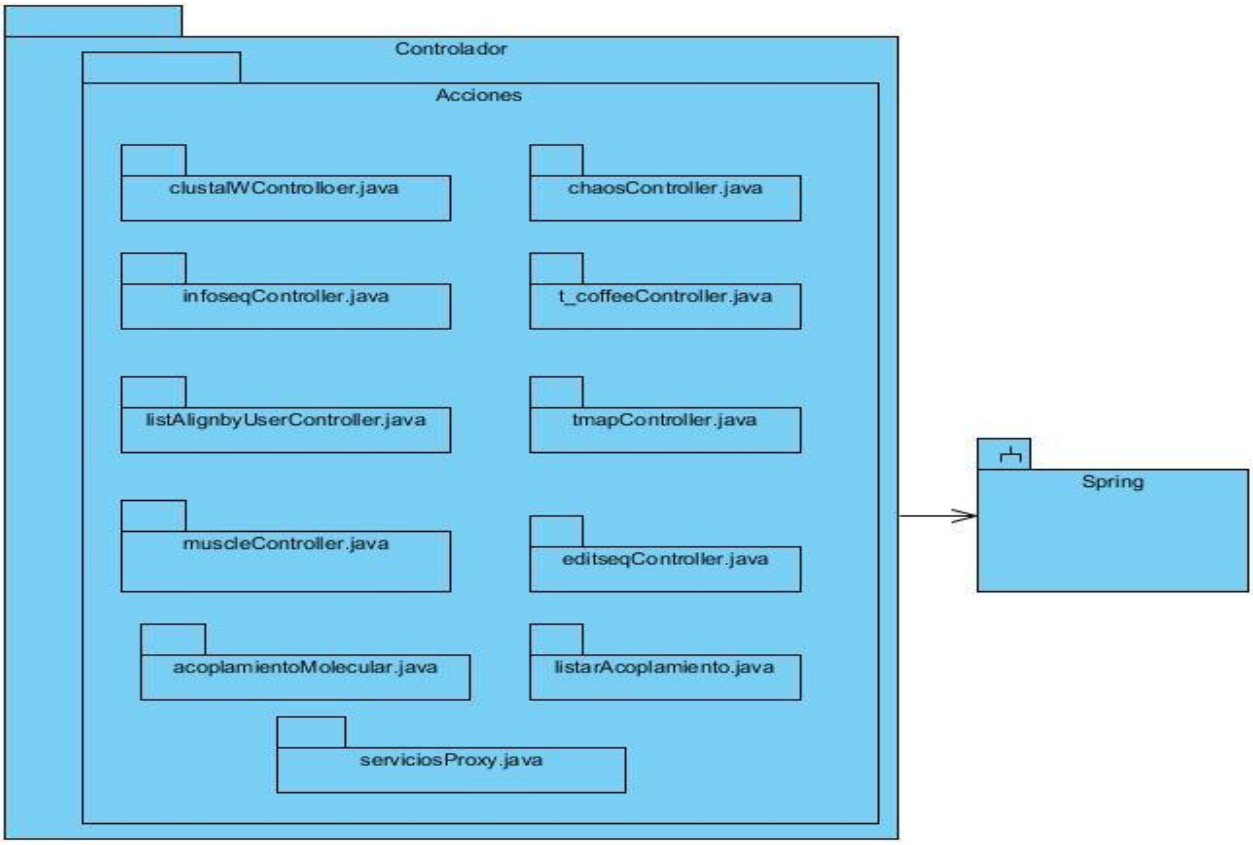


Figura 8: Vista Lógica del Paquete Controlador. Fuente: Elaboración Propia.

Establece una comunicación con la capa de presentación para recibir las solicitudes, procesarlas y presentar los resultados, además de interactuar con la capa de acceso a datos para persistir o recuperar información, por lo que define e implementa las funcionalidades que responden directamente a los requisitos de manera que se conserve la integridad del sistema y de los datos. Está constituida por los servicios y la fachada que permiten manipular la información que brindan los módulos.

- **Spring:** Framework de código abierto para el desarrollo de aplicaciones para la plataforma Java permite gestionar la seguridad de aplicación.
- **Paquete Acciones:** Agrupa las clases contrladoras de los módulos que integran la plataforma. .
- **Clase indexController:** clase controladora principal que gestiona la petición de los usuarios y es la encargada de seleccionar del paque acciones que clase dará respuesta a dicha solicitud.

**Paquete del modelo**

Contiene las clases que encapsulan la lógica del dominio, y se encargan del acceso a los datos almacenados en el gestor de base de datos. Recoge el código para la manipulación de los datos.

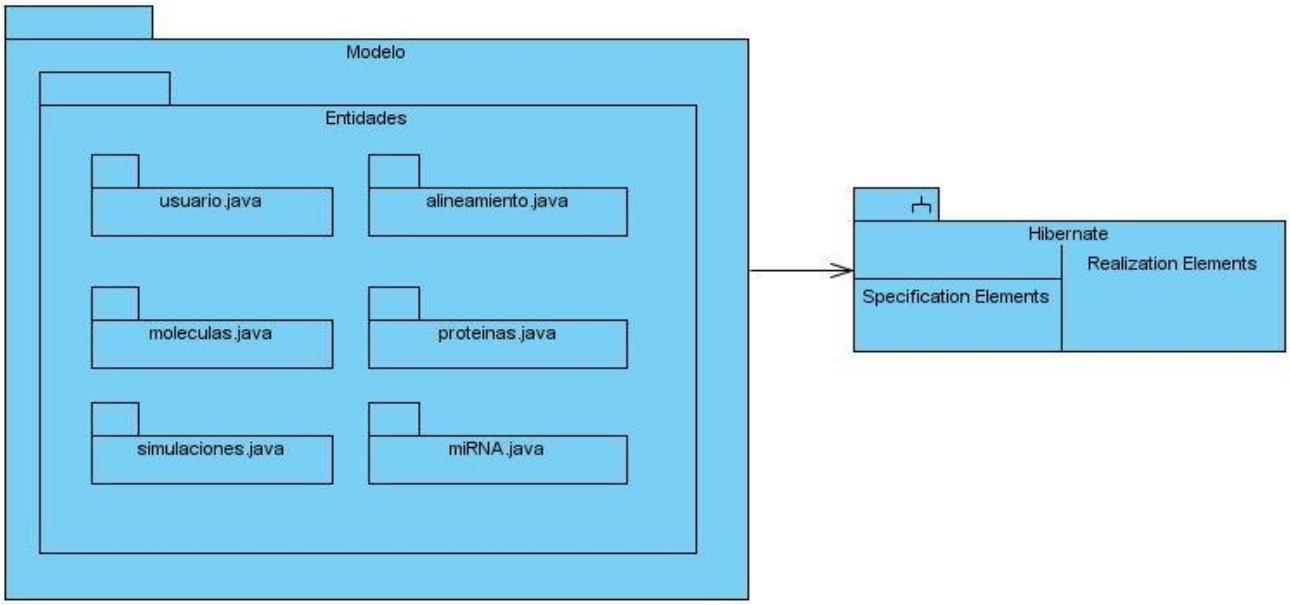


Figura 9: Vista Lógica del Paquete Modelo. Fuente: Elaboración Propia.

Es la encargada de manejar la información desde y hacia la base de datos. Contiene las clases del dominio representadas como clases entidades, los objetos que encapsulan la lógica de acceso a datos (DAO) y sus interfaces; así como las implementaciones de los DAOs usando el framework ORM Hibernate.

**Subsistema Hibernate:** Es un ORM que hace posible el manejo de la capa de persistencia de cualquier aplicación. Realiza el mapeo entre el mundo orientado a objetos de las aplicaciones y el mundo entidad-relación de las bases de datos en entornos Java.

- **Vista de Implementación**

La vista de implementación es representada por un diagrama de componentes o especificaciones de paquetes. En la vista de componentes de un sistema se representa la organización y las dependencias que existen entre los componentes que se van a utilizar para dar solución a nuestro problema como se muestra en la figura 10.

Un componente es un módulo software de nuestro sistema o un subsistema por sí mismo (código fuente, código binario o código ejecutable), contiene la implementación de una o varias clases y puede tener varias interfaces. Las interfaces representan la parte del componente que es pública y puede ser utilizada por otros componentes.

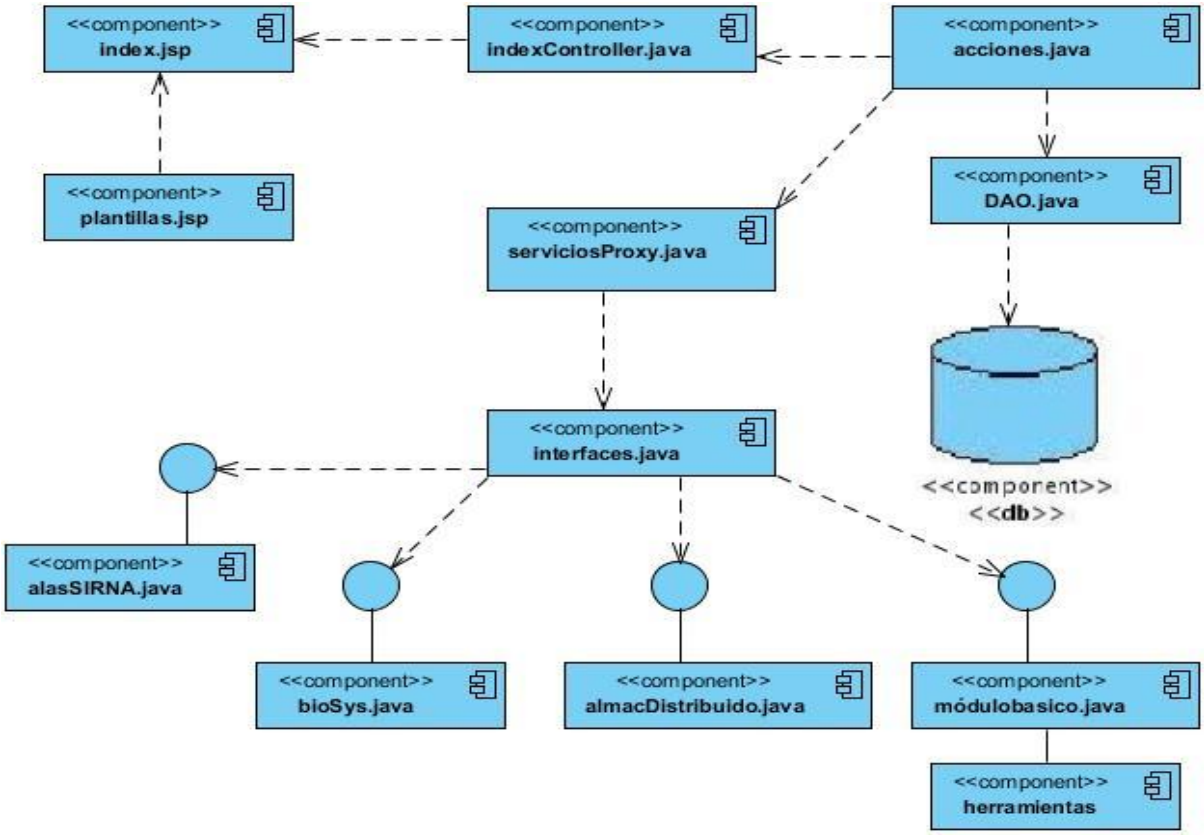


Figura 10: Vista de implementación. Fuente: Elaboración Propia.

**indexController:** Al recibir las peticiones del cliente, ejecutan código Java que realiza la lógica del sistema; manipula a los objetos DAO involucrados y decide que vista usar para el despliegue de los resultados.

**Interfaces:** Contiene la implementación de las interfaces que brindan los módulo que forman la integración del sistema, así como los componentes de cada módulo.

**serviciosProxy:** Componente que provee al sistema de una interfaz unificada y sencilla que hace de intermediaria entre el componente indexController y el grupo de interfaces más complejas que implementa el componente Interfaces.

- **Vista Despliegue**

La vista de despliegue describe los principales nodos físicos, ordenadores, así como los dispositivos que se necesitan para configurar la plataforma que pueda soportar la implementación del sistema. A pesar que SaaS no realiza despliegues externos se propone dicha vista la empresa o entidad que

desee utilizar este modelo de negocio. En la figura 10 se muestra como quedaría el despliegue de la plataforma para la empresa.

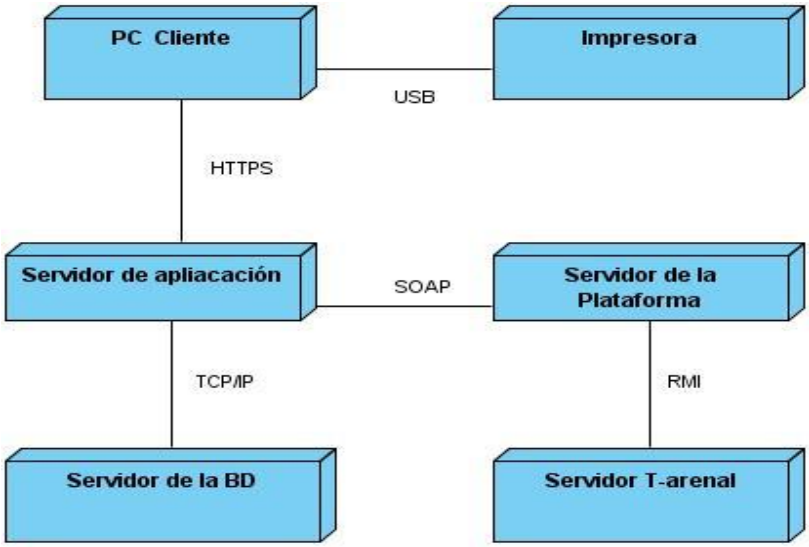


Figura 11: Vista Despliegue de la plataforma. Fuente: Elaboración Propia.

Se requiere al menos de una computadora (PC) con un navegador web, que permita realizar peticiones a través del protocolo de red HTTPS al servidor de aplicaciones, el cual encapsula la lógica de aplicación. Los datos persistentes del sistema serán almacenados en el servidor de bases de datos, al que se puede acceder utilizando el protocolo JDBC, por último las funcionalidades están accesibles en el servidor de servicios web o servidor de la plataforma a través de SOAP. Al servidor también estará conectado el servidor T-arenal el cual será el encargado de distribuir las peticiones a las PC Cliente T-arenal que estén asignadas a él para realizar los cálculos distribuidos, por medio del protocolo RMI.

## 2.9 Conclusiones del Capítulo

En este capítulo se definió cual será la arquitectura que se utilizará en la Plataforma de Servicios Bioinformáticos. Teniendo en cuenta las características de la misma fue seleccionado dentro de la familia de estilos: el estilo Peer to Peer dentro de este SOA. Luego del estudio de los patrones de arquitectura, se eligió el MVC, como patrón de diseño se propone los patrones GRASP dentro de ellos Bajo acoplamiento, Alta cohesión, Experto y Creador. Se plasmaron los requisitos no funcionales que harán que el funcionamiento de la plataforma sea el mejor posible y se explicaron detalladamente las diferentes vistas arquitectónicas: vista de CU, vista Lógica, vista de Procesos y la vista de Implementación

# CAPÍTULO 3: EVALUACIÓN DE LA ARQUITECTURA

## 3.1 Introducción

Evaluar una arquitectura de software es lo que permite prevenir todos los problemas de un diseño que no cumple con los requerimientos de calidad y para saber si la arquitectura diseñada para el sistema es la adecuada. En este capítulo se exponen los distintos métodos para evaluar la arquitectura y a partir del método seleccionado se hace un análisis de la solución propuesta.

## 3.2 Atributos de Calidad

Se entienden por atributos de calidad a las propiedades de un sistema que permiten a los usuarios e interesados en el mismo juzgar el estado en que se encuentra. Estos atributos son influenciados significativamente por la arquitectura de software de cada sistema, debido a las decisiones que se toman en favor de potenciar los de mayor relevancia de acuerdo al tipo de aplicaciones que se pretende desarrollar. Dadas sus características, son agrupados en dos clasificaciones:

| Atributos de calidad | Descripción |
|---|---|
| Disponibilidad | Es la medida en que un sistema es accesible para su uso. |
| Confidencialidad | Se trata como la ausencia de acceso no autorizado a la información. |
| Desempeño | Grado en que el sistema cumple con sus funciones, ateniéndose a ciertas condiciones de velocidad, consumo de memoria, entre otras. |
| Funcionalidad | Habilidad del sistema para realizar el trabajo para el cual fue concebido. |
| Confiabilidad | Da la medida de la robustez de un sistema para mantenerse operando satisfactoriamente a lo largo del tiempo. |
| Seguridad | Medida en que no se producen fallos o pérdidas de información en el sistema, a la vez que este es capaz de resistir ataques o usos indebidos mientras sirve a usuarios legítimos. |

Tabla 5. Ejemplos de atributos de calidad observables en tiempo de ejecución. Fuente: Elaboración propia.

| Atributos de calidad | Descripción |
|---|---|
| Configurabilidad | Posibilidad que se otorga a un usuario experto de realizar cambios en el sistema. |
| Mantenibilidad | Capacidad de modificar el sistema de manera rápida y a bajo costo. |
| Interoperabilidad | Medida en que partes de un sistema pueden trabajar con otro sistema externo. |
| Portabilidad | Habilidad del sistema para ser ejecutado en diferentes ambientes computacionales (de hardware, software o ambos). |
| Reusabilidad | Capacidad de diseñar un sistema de forma tal que parte de su estructura o componentes puedan ser reutilizados en futuras aplicaciones. |

Tabla 6: Ejemplos de atributos de calidad no observables en tiempo de ejecución. Fuente: Elaboración propia.

El llevar a cabo dichas pruebas, permite conocer qué tan presentes están los factores que han sido priorizados, así como información valiosa sobre los riesgos y fortalezas existentes en la arquitectura. Estos datos permiten poner en práctica las variantes arquitectónicas que promuevan el mayor nivel de aseguramiento posible que se espera obtener de cada uno de los atributos que determinan la calidad del sistema. Como resultado, además, mediante la observación de las relaciones y dependencias que existen entre dichos elementos, el arquitecto puede establecer un equilibrio en la medida que desee que el sistema posea cada una de estas cualidades.

## 3.3 Técnicas de evaluación de la Arquitectura

Existen diversas técnicas de evaluación de la arquitectura, las cuales se clasifican en cualitativas y cuantitativas. La técnica de evaluación cualitativa se utiliza cuando la arquitectura está en construcción, mediante las listas de verificación, escenarios o cuestionarios; mientras que la cuantitativa se usada cuando la arquitectura ha sido implantada, se puede emplear en simulaciones, modelos matemáticos, métricas, prototipos o experimentos.

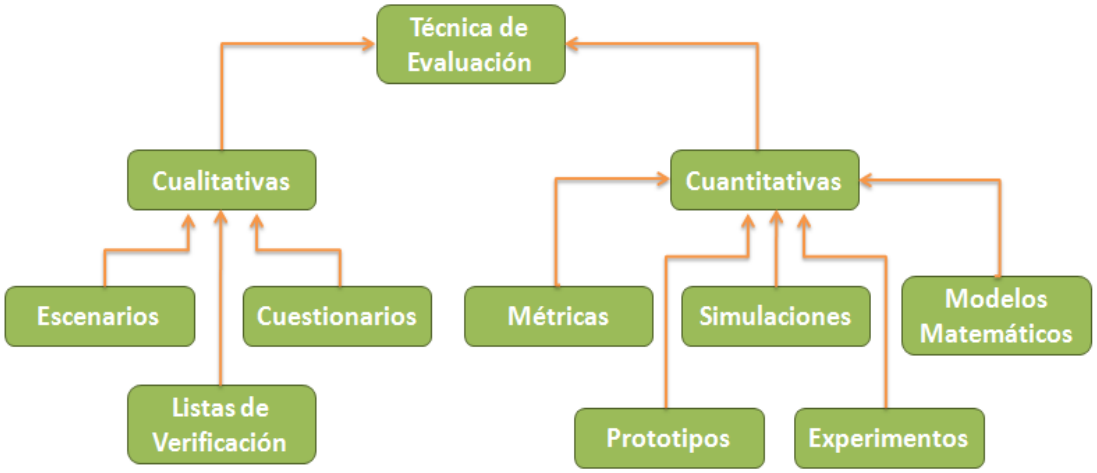


Figura 11. Clasificación de las técnicas de evaluación

Entre las técnicas de evaluación de la arquitectura de software se encuentran: evaluación basada en escenarios, evaluación basada en simulación, evaluación basada en modelos matemáticos y evaluación basada en experiencia.

| Técnica de Evaluación | Instrumento de Evaluación |
|---|---|
| Basada en Escenarios | ✓ Profiles<br>✓ Utility Tree |
| Basada en Simulación | ✓ Lenguajes de Descripción Arquitectónica (ADL)<br>✓ Modelos de colas |
| Basada en Modelos Matemáticos | ✓ Cadenas de Markov<br>✓ *Reliability Block Diagrams* |
| Basada en Experiencia | ✓ Intuición y experiencia<br>✓ Tradición<br>✓ Proyectos similares |

Figura 12: Instrumentos de evaluación asociados a sus respectivas técnicas de evaluación

**Técnicas de evaluación**

- Basada en Simulación: consiste en la implementación de componentes de la arquitectura centrado fundamentalmente en las interfaces y conexiones de estos. Realizando previamente un estudio de los principales ADL y herramientas de simulaciones. Su éxito está dado por la exactitud del perfil utilizado para evaluar el atributo de calidad y de la precisión con la que el sistema simula las condiciones del mundo real.

- Basada en Modelos Matemáticos: Es una alternativa a la simulación lo que permite utilizar los resultados de una como entrada del otro. La arquitectura es representada en forma de modelo ya que este no asume necesariamente que el sistema que intenta modelar consiste de componentes y conexiones. Su principal desventaja está dada por la inexistencia de modelos matemáticos apropiados a los atributos de calidad.

- Basada en Experiencia: El arquitecto en conjunto con el equipo de desarrollo aportan ideas basadas en factores subjetivos y la experiencia para evitar decisiones incorrectas, las cuales pueden llegar hacer aceptadas por seguir un razonamiento lógico.

- Basada en Escenarios: Es una descripción de la comunicación entre el sistema y el equipo del proyecto, el cual se basa en tres escenarios fundamentales: contexto, estímulo y respuesta. El estímulo describe lo que el usuario o integrantes del proyecto hacen para interactuar con el sistema. El contexto se refiere a qué sucede en el sistema al momento del estímulo y la respuesta describe cómo respondería el sistema ante el estímulo.

Entre las ventajas de su uso están:

- Son simples de crear y entender.
- Son poco costosos y no requieren mucho entrenamiento.
- Son efectivos.

Esta técnica está compuesta por dos instrumentos de evaluación: Árbol de Utilidad propuesto por Kazman (25) y los perfiles propuestos por Bosch (26).

**Árbol de Utilidad o Utility Tree:** es un esquema en forma de árbol que presenta los atributos de calidad de un sistema de software, refinados hasta el establecimiento de escenarios que especifican con suficiente detalle el nivel de prioridad de cada uno, los cuales son definidos por los involucrados en el desarrollo del sistema al momento de la construcción del árbol. Cada atributo de calidad perteneciente al árbol contiene una serie de escenarios relacionados y una escala de importancia y dificultad asociada.

**Perfiles o Profiles:** es un conjunto de escenarios, generalmente con alguna importancia relativa asociada a cada uno de ellos. El uso de perfiles permite hacer especificaciones más precisas del

requerimiento para un atributo de calidad tales como: desempeño *(performance)*, mantenibilidad *(maintainability)*, confiabilidad *(reliability)*, seguridad externa *(safety)* y seguridad interna *(security)***.**

## 3.4 Métodos de evaluación de la Arquitectura de Software

El objetivo de realizar evaluaciones a la arquitectura, es para analizar e identificar riesgos potenciales en su estructura y sus propiedades, que puedan afectar al software resultante y verificar que los requerimientos no funcionales estén presentes en la arquitectura.

A continuación se presentan un conjunto de métodos que son utilizados para evaluar la arquitectura de software según Rick Kasman (27):

- **Método de Análisis de la Arquitectura de Software (SAAM),** el método fue creado para el análisis de la modificabilidad de una arquitectura, pero en la práctica ha demostrado ser muy útil para evaluar distintos atributos de calidad, tales como 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. Su característica principal es la realización de un análisis que delimita la forma en que variarán los atributos de calidad, como resultado de algunas modificaciones futuras de la arquitectura. Este elemento ofrece una visión arquitectónica al equipo de desarrollo, hasta qué punto puede variar la arquitectura sin que afecte el nivel requerido de los atributos de calidad. Sin embargo, el comportamiento de un atributo de calidad puede afectar el desempeño de otros, por lo que no solamente se debe tener en cuenta la estructura de los componentes, sino también las relaciones que se establecen entre los mismos. Y es ahí donde este método, presenta su principal desventaja, ya que no valora la interrelación entre los distintos atributos.
- **Método de Análisis de Acuerdos de Arquitectura de Software (ATAM),** se centra en tres áreas fundamentales como: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM, explicado anteriormente. Revela la forma en que una arquitectura específica satisface ciertos atributos de calidad, y provee una visión de cómo dichos atributos interactúan con otros; estos son los tipos de *acuerdos* que se establecen entre ellos. Permite describir la forma en que el sistema puede responder a cambios e integrarse con otros sistemas.
- **Revisiones Activas para los Diseños Intermedios (ARID),** es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. ARID es un híbrido

entre Revisión Activa del Diseño (Active Design Review, ADR) y ATAM. Kazman proponen que tanto ADR como ATAM proveen características útiles para el problema de la evaluación de diseños preliminares, dado que ninguno por sí solo es conveniente. En el caso de ADR, los involucrados reciben documentación detallada y completan cuestionarios, cada uno por separado. En el caso de ATAM, está orientado a la evaluación de toda una arquitectura.

A continuación se presenta una comparación entre los distintos métodos de evaluación de la arquitectura estudiados.

| | ATAM | SAAM | ARID |
|---|---|---|---|
| **Atributos de calidad contemplados** | Modificabilidad<br>Seguridad<br>Desempeño<br>Confiabilidad | Modificabilidad<br>Funcionalidad | Conveniencia del diseño evaluado |
| **Objetos analizados** | Estilos arquitectónicos<br>Vistas arquitectónicas<br>Documentos<br>Flujos de Datos | Documentos<br>Vistas arquitectónicas | Especificación de los componentes |
| **Etapas del proyecto en las que se aplican** | Después del diseño de la arquitectura | Cuando la arquitectura cuenta con funcionalidad ubicada en los módulos | A lo largo del diseño de la arquitectura |
| **Enfoques utilizados** | Árbol de Utilidad y lluvia de ideas para articular los requerimientos de calidad.<br><br>Análisis arquitectónico que detecta puntos sensibles, puntos de balance y riesgo. | Lluvia de ideas para escenarios y articular los requerimientos de calidad.<br>Análisis de los escenarios para verificar funcionalidad o estimar costos de los cambios. | Revisiones de diseños, lluvia de ideas para obtener escenarios. |

Tabla 7. Comparación de los métodos de evaluación de la arquitectura. Fuente: Elaboración propia.

Hasta el momento se han presentado varios métodos de evaluación de arquitectura algunos más completos que otros, pero independientemente de esto todos tienen algo en común y es que utilizan la técnica de escenarios como vía de constatar en qué medida la arquitectura responde a los atributos de

calidad requeridos por el sistema. Se seleccionó el método ATAM ya que evalúa con más profundidad, en relación con otros métodos, cuestiones referentes a la arquitectura, como son: los atributos de calidad, analiza los estilos y patrones arquitectónicos, flujos de datos y documentos. Esta presente antes y después del desarrollo de software y se enfoca en el árbol de utilidad, lluvia de ideas y análisis de los resultados de los escenarios proponiendo nuevas estaciones de trabajo en caso que sea necesario.

### 3.5 Evaluación de la Arquitectura de la Plataforma

Para la evaluación de la arquitectura de la Plataforma de Servicios Bioinformáticos se tomaron en cuentas los atributos de calidad relacionados directamente con la arquitectura del software: seguridad, rendimiento, funcionalidad, eficiencia, usabilidad y mantenibilidad, definidos por la ISO/IEC 9126 (28). Además se decidió de las técnicas estudiadas seguir la línea de la técnica basada en escenarios con el instrumento de evaluación el Árbol de Utilidad, independientemente de que van implícitos en el método ATAM seleccionado para realizar la evaluación, el cual apoya a los involucrados con el proyecto a entender las consecuencias de las decisiones arquitectónicas respecto a los atributos de calidad del sistema.

El árbol de utilidad generado se toma como un plan para el resto de la evaluación que realiza el método. Indica además al equipo evaluador dónde ocupar su tiempo y dónde probar aproximaciones y riesgos arquitectónicos como se muestra en la tabla

| Árbol de Utilidad | | | |
|---|---|---|---|
| **Atributo de Calidad** | **Sub-característica** | **Escenario** | **Prioridad** |
| **Funcionalidad** | Seguridad | 1. Autenticar los usuarios para permitir el uso adecuado de la aplicación. | A |
| | | 2. Las funcionalidades de la aplicación se muestran de acuerdo al usuario que esté autenticado. | A |
| | | 3. Encriptar todo el tráfico de información. | A |
| | | 4. Protección contra ataques externos que puedan afectar la integridad de los datos. | A |
| | Interoperabilidad | 5. La aplicación posee módulos capaces de leer datos provenientes de otros sistemas. | M |
| | | 6. La aplicación posee módulos capaces de producir datos para otro sistema. | M |
| | | | |
| **Usabilidad** | Comprensibilidad, Aprendibilidad y Atractividad | 7. La aplicación debe tener la capacidad de ser atractivo, entendible para el usuario. | M |

| | | | |
|---|---|---|---|
| **Eficiencia** | Disponibilidad | 8. Búsqueda de información de forma rápida. | M |
| | | | |
| **Mantenibilidad** | Habilidad de cambio y escalabilidad | 9. Diseño con capacidad para la adición de nuevas funcionalidades y modificaciones futuras. | A |
| | | | |
| **Portabilidad** | Adaptabilidad | 10. Acceder a la aplicación desde cualquier parte del país. | A |
| | | 11. Usar cualquier navegador para acceder a la aplicación. | |
| **Leyenda: Prioridad Alta (A), Media (M), Baja (B)** | | | |

Tabla 8: Árbol de Utilidad. Fuente: Elaboración Propia

**Análisis de los escenarios:**

Se analizaron todos los escenarios teniendo en cuenta cómo reacciona la arquitectura del sistema. Se describe para cada uno el estímulo, el ambiente en el que se encuentra el sistema, la respuesta y una explicación de cómo responde la arquitectura a este evento así como el razonamiento del sistema ante las decisión arquitectónica correspondiente.

Los riesgos son decisiones arquitecturalmente importantes que no han sido tomadas o decisiones que han sido tomadas pero las consecuencias no han sido entendidas a plenitud.

Los puntos sensibles son parámetros en la arquitectura en los cuales la respuesta medible de algunos atributos de calidad es altamente correlacionada.

Un punto de intercambio (*tradeoff*) es descubierto en la arquitectura cuando un parámetro de construcción arquitectural es un anfitrión para al menos un punto sensible donde los puntos de calidad medibles son afectados indistintamente por cambio en el parámetro.

A continuación se muestran algunos escenarios analizados:

| **Escenario #: 3** | **Encriptar todo el tráfico de información.** | | | |
|---|---|---|---|---|
| **Atributo(s)** | Funcionalidad-Seguridad. | | | |
| **Ambiente** | Operación normal. | | | |
| **Estímulo** | Envío de la información a la UCI. | | | |
| **Respuesta** | Encripta la información y la envía a la UCI de forma segura. | | | |
| **Decisiones arquitectónicas** | **Riesgo** | **Punto de Sensibilidad** | **Punto de Intercambio** | **No Riesgo** |
| Protocolo seguro HTTPS | | S1 | | NR3 |

| | |
|---|---|
| **Razonamiento** | **NR3.** La seguridad de la información que es enviada a través de la red se garantiza con el uso del Protocolo HTTPS: versión segura del protocolo HTTP que implementa un canal de comunicación seguro basado en SSL (Secure Socket Layers) entre el navegador del cliente y el servidor HTTP.<br>**S1.** Aumenta la seguridad. |

Tabla 9. Análisis del escenario 3. Fuente: Elaboración Propia.

| **Escenario #: 4** | **Protección contra ataques externos que puedan afectar la integridad de los datos.** | | | |
|---|---|---|---|---|
| **Atributo(s)** | Funcionalidad-Seguridad. | | | |
| **Ambiente** | Operación normal. | | | |
| **Estímulo** | Agente externo intenta atacar la BD directamente o a través de la aplicación. | | | |
| **Respuesta** | No tiene efecto el intento de ataque. | | | |
| **Decisiones arquitectónicas** | **Riesgo** | **Punto de Sensibilidad** | **Punto de Intercambio** | **No Riesgo** |
| Framework Hibernate | | S2 | I1 | NR3 |
| SGBD PostgreSQL | | | | NR4 |
| En el despliegue el servidor Web es una ubicación física aparte del servidor de BD. | | S3 | I2 | NR5 |
| **Razonamiento** | La integridad de los datos almacenados en la BD es uno de los puntos más sensibles en la seguridad del sistema debido a la relevancia que tienen, como se ha expuesto con anterioridad, por tanto muchas son las decisiones arquitectónicas que responden a la protección de estos contra ataques externos:<br>**NR3.** Framework Hibernate: Las fallas de inyección, en particular la inyección SQL, son ataques comunes en aplicaciones Web. La inyección ocurre cuando los datos proporcionados por el usuario son enviados e interpretados como parte de una orden o consulta. Las | | | |

| | |
|---|---|
| | fallas de inyección permiten a un atacante crear, modificar o borrar cualquier información arbitraria disponible en la aplicación. El mecanismo clave para evitar inyecciones es el uso de APIs seguras, como consultas con parámetros de asignación rigurosa y bibliotecas de mapeo relacional de objetos (ORM) como Hibernate.<br>**S2.** Aumenta la seguridad.<br>**I1.** Disminuye la disponibilidad.<br>**NR4.** PostgreSQL: La seguridad de la base de datos está implementada en varios niveles y todos los ficheros almacenados en la base de datos están protegidos contra escritura por cualquier cuenta que no sea la del súper usuario de Postgre.<br>**NR5.** Despliegue: La distribución física del sistema en distintos servidores contribuye a la protección de los datos ya que no se encuentran en el mismo ordenador la aplicación y la BD, y el atacante probablemente desconoce cuál es la ubicación real del servidor de BD.<br>**S3.** Aumenta la seguridad.<br>**I2.** Disminuye la disponibilidad. |

Tabla 10. Análisis del escenario 4. Fuente: Elaboración Propia.

| **Escenario #: 8** | **Búsqueda de información de forma rápida.** | | | |
|---|---|---|---|---|
| **Atributo(s)** | Eficiencia-Desempeño. | | | |
| **Ambiente** | Operación normal. | | | |
| **Estímulo** | Múltiples usuarios conectados concurrentemente. | | | |
| **Respuesta** | El usuario selecciona los parámetros y realiza la búsqueda. | | | |
| **Decisiones arquitectónicas** | **Riesgo** | **Punto de Sensibilidad** | **Punto de Intercambio** | **No Riesgo** |
| SGBD PostgreSQL | | S7 | I3 | NR6 |
| Framework Hibernate | | S8 | | NR7 |
| **Razonamiento** | Las decisiones arquitectónicas que garantizan la | | | |

| | |
|---|---|
| | respuesta satisfactoria del sistema son:<br>**NR6.** El SGBD PostgreSQL, capaz de manejar las conexiones concurrentes de muchos usuarios de forma eficiente.<br>**S7.** Aumenta la eficiencia.<br>**I3.** Disminuye el rendimiento e integración de los datos, pueden llegar datos incorrectos.<br>**NR7.** El Framework Hibernate, que con su caché de dos niveles ofrece una gran flexibilidad y rendimiento ya que al almacenar las estructuras de objetos en memoria, evita volver a buscar dichos objetos en la base de datos ahorrando multitud de accesos, y por consiguiente tiempo. Además incluye una caché de consultas que da la posibilidad de obtener rápidamente resultados que ya habían sido consultados previamente.<br>**S8.** Aumenta el rendimiento |

Tabla 9. Análisis del escenario 8. Fuente: Elaboración Propia.

| **Escenario #: 9** | **Capacidad para la adición de nuevas funcionalidades y modificaciones futuras.** | | | |
|---|---|---|---|---|
| **Atributo(s)** | Mantenibilidad – Habilidad de cambio y escalabilidad. | | | |
| **Ambiente** | Operación normal. | | | |
| **Estímulo** | Necesidad de modificar o adicionar una funcionalidad a la aplicación. | | | |
| **Respuesta** | | | | |
| **Decisiones arquitectónicas** | **Riesgo** | **Punto de Sensibilidad** | **Punto de Intercambio** | **No Riesgo** |
| Patrón arquitectónico MVC | | S9 | | NR8 |
| Patrón de diseño Fachada | | | | NR9 |
| **Razonamiento** | **S9.** Estas decisiones arquitectónicas garantizan el alto grado de mantenibilidad y modificabilidad en la arquitectura propuesta ya que:<br>**NR8.** Mediante el patrón MVC se alcanza una mayor cohesión entre los elementos que se especializan en | | | |

| | |
|---|---|
| | sus funciones, y un bajo acoplamiento entre ellos, por tanto los cambios son fáciles de implementar.<br><br>**NR9.** Con el patrón Fachada se logra ocultar la complejidad de determinadas clases y disminuir el acoplamiento, lo que facilita el entendimiento de las clases por separado y evita que los cambios en unas afecten a otras. |

Tabla 10. Análisis del escenario 9. Fuente: Elaboración Propia.

**Toma de decisiones**

En el árbol de utilidad se analizaron un total de 11 escenarios clasificados según la prioridad que puede ser alta, media y baja, colocados por los stakeholders en el paquete de tormenta de ideas, lo que les da la oportunidad de refinar los escenarios y proponer nuevas estaciones de trabajo.

Se identificaron 2 riesgos en las tablas de los escenarios 5 y 11. Una vez identificados es necesario darle solución desde el punto de vista de los atributos de calidad analizados y dar paso a la construcción del sistema. A continuación se propone la estrategia para resolver los riesgos:

**R1.** Aumenta el riesgo de que los datos queden inconsistentes ante una eventual falla.

Ante una eventual falla, se pueden tomar varias acciones como contingencia. En ese caso, una de las maneras en las que puede fallar el sistema es:

- Caída de la conexión
- Falla eléctrica
- Ataque informático
- Problemas de almacenamiento

Para mitigar este riesgo se pueden tomar varias acciones, entre las que se incluyen:

- Llevar un registro de operaciones que indique el estado de cada acción de comunicación de la plataforma con otros sistemas lo que posibilitará que en caso de falla, se tenga una idea del último estado ocurrido y al restablecerse continuar desde ahí.
- Tener habilitadas UPS en los ordenadores que brindan los servicios.

- Implementar elementos de seguridad, tales como la seguridad para la transmisión de datos sobre canales no seguros así, mantener la seguridad de la aplicación
- Se cuenta con un sistema de almacenamiento distribuido para aumentar el espacio en disco.

**R2.** Desactivar el java script.
Este riesgo se puede evitar notificando al cliente, que para un uso eficaz de la aplicación se debe mantener activado el java script del navegador.

Como principales decisiones a tomar se aprobaron: definir políticas para la integración del sistema con otros componentes, así como mantener la seguridad e integridad de los datos que se manejan; desarrollar un proceso de auditoría y control al diseño arquitectónico del sistema y analizar las consecuencias que pueden arrojar cambios significativos en las decisiones arquitectónicas definidas.

## 3.6 Conclusiones del Capítulo

En este capítulo se analizó la influencia de las herramientas de descripción de la arquitectura, se aplicó el método de evaluación de arquitecturas ATAM y la técnica basada en escenarios con el instrumento árbol de utilidad, en aras de determinar cómo la propuesta arquitectónica cumple con los atributos de calidad propuestos por el modelo de negocio SaaS. Como resultado de este proceso se identificaron algunos riesgos, no riesgos, puntos sensibles y puntos intercambios presentes en la arquitectura del sistema, con lo que se demuestra que la arquitectura propuesta satisface las expectativas del objetivo trazado.

# CONCLUSIONES GENERALES

Una vez culminado el trabajo es posible afirmar que se les dio cumplimiento a los objetivos trazados para el mismo:

- El análisis de los distintos modelos de negocios y de los elementos del macro entorno que influye en la Plataforma de Servicios Bioinformáticos, así como otras características del mercado permitió fundamentar y seleccionar el modelo de negocio SaaS para el software que se está desarrollando en el departamento de bioinformática.
- A partir características que demanda el modelo de negocio y requisitos funcionales presentes en los servicios seleccionados se describió la arquitectura a través de la representación y desarrollo de las vistas arquitectónicas, las cuales proporcionan una visión común y propuesta de solución desde diferentes perspectivas del sistema.
- La evaluación de la arquitectura con la aplicación del método ATAM, mostró que esta cumple con los atributos de calidad propuestos, basados en el modelo de negocio SaaS. La estrategia de mitigación permite darle respuesta los riesgos identificados en los distintos escenarios.

# RECOMENDACIONES

Después de haber alcanzado los objetivos que se trazaron al principio de este trabajo se proponen las siguientes recomendaciones:

- Implementar el modelo de negocio seleccionado y la arquitectura propuesta para posibilitar su implantación en la Plataforma de Servicios Bioinformáticos.

- Realizar un estudio de precios desde el área de Mercadotecnia para diseñar un precio que permita recuperar el costo de inversión de la plataforma y que a su vez pueda llegar a los diferentes sectores del mercado para el cual está enfocada dicha solución.
- Inclusión de nuevas tecnologías emergentes como las relacionadas con la computación en la nube o versiones superiores de los programas con los que trabaja el departamento de bioinformática que eleven la funcionalidad y calidad de los productos a desarrollar.
- Incluir en el portal de la plataforma una pasarela de pagos para potenciar la gestión empresarial, dadas las características del modelo de negocio adoptado.

REFERENCIA BIBLIOGRÁFICA


1. *Martínez, Pére Orlando, Agramonte Delgado Alina, Diaz León, Andry Daniel. UCIBioSoft: Portal web de servicios bioinformáticos.*

2. *Dotres, M. Carlos. (1997) "El sistema de Salud de Cuba: Retos y Logros". Conferencia. En: taller "La telemática y la universidad en el desarrollo de los sistemas locales de salud". Junio de 1997. Disponible en:* http://www.infomed.sld.cu/discursos/telem.html

3. Santos, Hernandez, Vismar. Estudio sobre la industria del Software a nivel mundial. Caracterización en América Latina y Cuba. 23 de junio del 2009.

4. Magretta, Joan. 2002. Why Business Models Matter. Harvard business review

5. Amit R, Zott C. 2001. Value Creation in E-Business. Value Creation in E-Business. Strategic Management Journal

6. Chesbrough H, Rosenbloom RS. 2002. The role of the business model in capturing value from innovation: evidence from Xerox Corporation's technology spin-off companies. Industrial & Corporate Change

7. Linder, J. and S. Cantrell, 2000. Changing Business Models: Surveying the Landscape, Accenture Institute for Strategic Change.

8. Osterwalder A., Pigneur Y., Tucci C. 2005. Clarifying Business Models: origins, Present, and Clarifying Business Models: origins, Present, and Future of the Concept. Communications of the Association for Information Systems.

9. Definición de Modelo de Negocio. 10 de Diciembre de 2010.] Disponible en: http://definicion.de/modelo-de-negocio/.

10. **Cusumano, Michael A**.EL NEGOCIO DEL SOFTWARE el: 12 de febrero de 2010

11. Rappa, Michael. 2005. Business Models on the Web. Disponible en http://www.grupoe.com/web/edu_modelos_negocios_internet.asp

12. Meriño Coronado, Eilen Marien. Quiala, Marlies Torres. Tesis: "Modelos de Negocio para comercializar productos del Grupo de Gestión de Información Biomédica en la Universidad de las Ciencias Informáticas". Junio 2010.

13. Gonçalves, V. and P. Ballon (2011). "Adding value to the network: Mobile operators' experiments with Software-as-a-Service and Platform-as-a-Service models." Telematics and Informatics **28**(1): 12-21.

14. Calvo, R. F. and A. d. T. d'Informàtica (2001). Glosario básico inglés-español para usuarios de Internet, Asociación de Técnicos de Informática (ATI).

15. Foster, I., Y. Zhao, et al. (2008). Cloud Computing and Grid Computing 360-Degree Compared. Grid Computing Environments Workshop, Ieee.

16. Disponible en: http://www.salesforce.com/saas/questions-about-saas/

*17.* **IEEE.** *Estándar 1471-2000.*

18. **Pressman, Roger S.** Ingeniería del Software: Un enfoque práctico 5ta Edición: s.l.: McGraw-Hill, 2002.

19. Philippe Kruchten. "The 4+1 View Model of Architecture". IEEE Software., Los Alamitos, CA, USA, Volume 12, Number 6, Pages 42-50, IEEE Computer Society Press, 1995.

20. Len Bass, P.C., Rick Kazman. Software Architecture in Practice. Second Edition ed. 2003: Addison-Wesley Professional.

21. Fundamentos de Definición de Arquitectura en RUP. Disponible en: http://www.ppt2txt.com/r/30e27e69/

22. Garlan, David y Shaw, Mary. An introduction to software architecture: s.n., 1996. CMU Software Engineering Institute Technical Report, CMU/SEI-94-TR-21, ESC-TR-94-21.

23. Fleitas, Raúl Alejandro Conil, Sabina, Yoelkys Rodríguez. "Arquitectura General del Proyecto Centro Rector de Universidad para Todos". Junio 2007.

24. **Reynoso, Carlos y Kicillof, Nicolás.** Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. Buenos Aires: s.n., 2004.

25. Kazman, R., Clements, P., Klein, M. (2001). Evaluating Software Architectures. Methods and case studies. Addison Wesley.

26. Bosch, J. (2000). Design & Use of Software Architectures. Addison-Wesley.

27. Kazman, R., Clements, P., Klein, M. (2001). Evaluating Software Architectures. Methods and case studies. Addison Wesley.

28. ISO/IEC. (1998). Information Technology – Software Product Quality – Part 1: Quality Model. Disponible en: http://www.usability.serco.com/trump/resources/standards.htm#9126-1

BIBLIOGRAFÍA


1. **Alba, Julio.** (167, feb-mar 2008). *¿Qué es SOA?* Revista Bit.

2. **Amit R, Zott C. 2001**. Value Creation in E-Business. Value Creation in E-Business. Strategic Management Journal.

3. *Apache Foundation, sitio oficial [en línea] http://www.apache.org.*

4. **Avila, Eladio.** *Propuesta de Arquitectura de Software para el desarrollo de portales web orientados al Ajedrez.* Universidad de las Ciencias Informáticas, La Habana, Cuba: s.n., 2009.

5. **Bianco, P., Kotermanski, R., Merson, P.** *Evaluating a Service-Oriented Architecture.* Carnegie Mellon SEI: s.n., 2007.

6. **Buyya, R., Broberg, J., Goscinski, A.** *Cloud Computing. Principles and Paradigms.* s.l.: John Wiley & Sons, 2011. 978 0 470 88799 8.

7. **Camacho, E., Cardeso, F., Nuñez, G.** *Arquitecturas de Software. Guía de Estudio.* s.n., 2004.

8. **Carrascoso, Y., Chaviano, E., Céspedes, A.** *Procedimiento para la Evaluación de Arquitecturas de Software Basadas en Componentes.* Universidad de las Ciencias Informáticas, La Habana, Cuba: s.n., 2008.

9. **Chesbrough H, Rosenbloom** RS. 2002. The role of the business model in capturing value from innovation: evidence from Xerox Corporation's technology spin-off companies. Industrial & Corporate Change

10. **Clements, Paul y Shaw, Mary.** *A field guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems.* 1996.

11. **Earl, Tomas**. *SOA , Principles of Service Design,* 2008.

12. **Eclipse y otros contribuidores** *OpenUp v1.5.0.1 2004, 2008*.

13. **Fowler, Martin y otros.** *Patterns of Enterprise Aplication Architecture:* s.l: Addison Wesley, 2002. 0-321-12742-0.

14. **Gamma, Erich y otros.** *Design Patterns. Elements of Reuseable Object-Oriented Software:* s.n., 1994.

15. **Linthicum, David S.** *Cloud Computing and SOA Convergence in Your Enterprise. A Step-by-Step Guide:* s.l.: Addison Wesley, 2010. 0 13 600922 0.

16. **Garlan, David y Shaw, Mary.** *An introduction to software architecture:* s.n., 1994. CMU Software Engineering Institute Technical Report, CMU/SEI-94-TR-21, ESC-TR-94-21.

17. **Hassan, Qusay F.** (en-feb 2011) *Demystifying Cloud Computing.* Revista Crosstalk. Mansoura University, Egypt.

*18.* **IEEE.** *Estándar 1471-2000.*

19. **Kabir, Mohammed J.** *La Biblia del Server Apache*.

20. **Kazman, R., Klein, M., Clements, P.** *ATAM: Method for Architecture Evaluation.* Carnegie Mellon SEI: s.n., 2000.

21. **Lazo, René y otros.** *Modelo de referencia para el desarrollo arquitectónico de sistemas de gestión. Proyecto ERP-Cuba.* Universidad de las Ciencias Informáticas, La Habana, Cuba: XVI Fórum de Ciencia y Técnica.

22. **Linder, J. and S. Cantrell**, 2000. Changing Business Models: Surveying the Landscape, Accenture Institute for Strategic Change.

*23. Lineamientos de la Política Económica y Social del Partido y la Revolución, 2011.*

**24. Fleitas, Raúl Alejandro Conill, Sabina. Yoelkys Rodríguez,** "Arquitectura General del Proyecto Centro Rector de la Universidad de Todos", Junio 2007.

25. **Magretta, Joan**. 2002. Why Business Models Matter. Harvard business review.

*26.* **Marks, Eric y Bell, Michael.** *Service-Oriented Architecture: a Planning and Implementation guide for Business and Technology: s.n., 2006.*

27. **Martínez, Pére Orlando, Agramonte Delgado Alina, Diaz León, Andry Daniel**. UCIBioSoft: Portal web de servicios bioinformáticos.

28. **McGovern, James y otros.** *Enterprise Service Oriented Architectures. Concepts, Challenges, Recommendations:* s.l: Springer, s.n., 2006.

29. **Meriño, Eilen y Quiala, Marlies.** *Modelos de Negocio para comercializar productos del Grupo de gestión de Información Biomédica en la Universidad de las Ciencias Informáticas.* Universidad de las Ciencias Informáticas, La Habana, Cuba: s.n., 2010.

30. *Microsoft Corporation, sitio oficial [en línea] http://www.microsoft.com.*

31. *Nagios, sitio oficial [en línea] http://www.nagios.org.*

32. *NetBeans IDE, sitio oficial [en línea] http://www.netbeans.org.*

33. **Osterwalder A., Pigneur Y., Tucci** C. 2005. Clarifying Business Models: origins, Present, and Clarifying Business Models: origins, Present, and Future of the Concept. Communications of the Association for Information Systems.

34. **Osterwalder, Alexander y Pigneur, Yves.** *Business model generation: a handbook for visionaries, game changers and challengers.* New Jersey: s.n., 2009. 9782839905800 2839905809 9780470876411 0470876417.

35. **Perry, Dewayne y Wolf, Alexander.** *Foundations for the study of software architecture: s.n.,* 2002.

36. *Portal de Producción UCI, Línea base de Arquitectura Proyectos UCI [en línea, citado el 10 de marzo del 2012] http://gespro.dgp.prod.uci.cu/projects/publico/wiki/Arquitectura_de_software.*

37. *PostgreSQL, sitio oficial [en línea] http://www.postgresql.org.*

38. **Pressman, Roger S.** *Ingeniería del Software: Un enfoque práctico 5ta Edición:* s.l.: McGraw-Hill, 2002. 8448132149.

39. **Reynoso, Carlos Billy.** *Introducción a la Arquitectura de Software.* Buenos Aires: s.n., 2004.

40. **Reynoso, Carlos y Kicillof, Nicolás.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* Buenos Aires: s.n., 2004.

41. **Reynoso, Carlos y Kicillof, Nicolás.** *Lenguajes de Descripción de Arquitectura (ADL).* Buenos Aires: s.n., 2004.

42. **Robert, Armando.** *Arquitectura de Software para el Sistema Integrado de Gestión Estadística 2.0 Nuragas.* Universidad de las Ciencias Informáticas, La Habana, Cuba: s.n., 2009.

43. **Santana, Marisel y Batista, Reinier.** *Propuesta de arquitectura Orientada a Servicios para alasMEDIGEN. Sistema Informático de Genética Médica.* Universidad de las Ciencias Informáticas, La Habana, Cuba: s.n., 2010.

44. **Santos Hernández, Vismar**. *Estudio sobre la industria del Software a nivel mundial. Caracterización en América Latina y Cuba, 2009.*

45. **Santos, Hernandez, Vismar.** Estudio sobre la industria del Software a nivel mundial. Caracterización en América Latina y Cuba. 23 de junio del 2009.

46. **Scalone, Fernanda.** *Estudio comparativo de los modelos y estándares de Calidad del Software.* Universidad Tecnológica Nacional, Facultad Regional Buenos Aires: s.n., 2006.

47. *Visual Paradigm, sitio oficial [en línea]* http://www.visual-paradigm.com.

ANEXOS

Anexo 1: Tabla de análisis de los escenarios.

| **Escenario #: 2** | **Las funcionalidades de la aplicación se muestran de acuerdo al usuario que esté autenticado.** | | | |
|---|---|---|---|---|
| **Atributo(s)** | Funcionalidad-Seguridad. | | | |
| **Ambiente** | Operación normal. | | | |
| **Estímulo** | Usuario entra al sistema. | | | |
| **Respuesta** | Solo se muestran las funcionalidades a las que tiene acceso según su tipo de usuario. | | | |
| **Decisiones arquitectónicas** | **Riesgo** | **Punto de Sensibilidad** | **Punto de Intercambio** | **No Riesgo** |
| Framework Spring | | | | NR2 |
| **Razonamiento** | **NR2.** Las funcionalidades del sistema deben mostrarse de acuerdo al usuario que este activo, garantizando de este modo que la aplicación sea utilizada en correspondencia con los privilegios permitidos a cada tipo de usuario, las disposiciones arquitectónicas que aseguran el cumplimiento de lo planteado anteriormente es el Framework Spring: este brinda las facilidad de implementar la seguridad haciendo uso del Framework Acegi Security. | | | |

| **Escenario #: 3** | **Encriptar todo el tráfico de información.** | | | |
|---|---|---|---|---|
| **Atributo(s)** | Funcionalidad-Seguridad. | | | |
| **Ambiente** | Operación normal. | | | |
| **Estímulo** | Envío de la información a la UCI. | | | |
| **Respuesta** | Encripta la información y la envía a la UCI de forma segura. | | | |
| **Decisiones arquitectónicas** | **Riesgo** | **Punto de Sensibilidad** | **Punto de Intercambio** | **No Riesgo** |
| Protocolo seguro HTTPS | | S1 | | NR3 |
| **Razonamiento** | **NR3.** La seguridad de la información que es enviada a | | | |

| | |
|---|---|
| | través de la red se garantiza con el uso del Protocolo HTTPS: versión segura del protocolo HTTP que implementa un canal de comunicación seguro basado en SSL (Secure Socket Layers) entre el navegador del cliente y el servidor HTTP.<br>**S1.** Aumenta la seguridad. |

| **Escenario #: 5** | **La aplicación posee módulos capaces de leer datos provenientes de otros sistemas.** | | | |
|---|---|---|---|---|
| **Atributo(s)** | Funcionalidad-Interoperabilidad. | | | |
| **Ambiente** | Operación normal. | | | |
| **Estímulo** | Subir datos | | | |
| **Respuesta** | La plataforma almacena los datos cargados. | | | |
| **Decisiones arquitectónicas** | **Riesgo** | **Punto de Sensibilidad** | **Punto de Intercambio** | **No Riesgo** |
| Módulo Servicios que implementa la captura de ficheros. | R1 | S4 | | |
| **Razonamiento** | **S4.** Aumentan los niveles de interacción con otros sistemas.<br>**R1.** Aumenta el riesgo de que los datos queden inconsistentes ante una eventual falla. | | | |

| **Escenario #: 6** | **La aplicación posee módulos capaces de producir datos para otro sistema.** | | | |
|---|---|---|---|---|
| **Atributo(s)** | Funcionalidad – Interoperabilidad. | | | |
| **Ambiente** | Operación normal. | | | |
| **Estímulo** | Descarga los datos | | | |
| **Respuesta** | El sistema permite la descarga ontología. | | | |
| **Decisiones arquitectónicas** | **Riesgo** | **Punto de Sensibilidad** | **Punto de Intercambio** | **No Riesgo** |

| | | | | |
|---|---|---|---|---|
| Módulo Servicios que implementa la captura de ficheros. | | S5 | | |
| **Razonamiento** | **S5.** Aumentan los niveles de interacción con otros sistemas. | | | |

| **Escenario #: 7** | **La aplicación debe tener la capacidad de ser atractivo, entendible para el usuario.** | | | |
|---|---|---|---|---|
| **Atributo(s)** | Usabilidad – Comprensibilidad y Atractividad. | | | |
| **Ambiente** | Operación normal. | | | |
| **Estímulo** | El usuario hace uso de las funcionalidades del sistema. | | | |
| **Respuesta** | No hay cambios en la implementación de dicha funcionalidad. | | | |
| **Decisiones arquitectónicas** | **Riesgo** | **Punto de Sensibilidad** | **Punto de Intercambio** | **No Riesgo** |
| Subsistema que permite la visualización de elementos de diseño Framework Spring | | S6 | | |
| **Razonamiento** | **S6.** Mejora la usabilidad. | | | |

| **Escenario #: 10** | **Acceder a la aplicación desde cualquier parte del país.** | | | |
|---|---|---|---|---|
| **Atributo(s)** | Portabilidad-Adaptabilidad. | | | |
| **Ambiente** | Operación normal. | | | |
| **Estímulo** | El usuario desea utilizar el sistema desde cualquier estado. | | | |
| **Respuesta** | La aplicación se encuentra disponible a través de la web. | | | |
| **Decisiones arquitectónicas** | **Riesgo** | **Punto de Sensibilidad** | **Punto de Intercambio** | **No Riesgo** |
| Tecnología Web | | S10 | | NR10 |
| **Razonamiento** | **NR10.** La decisión arquitectónica de utilizar tecnología Web se basa en la necesidad de garantizar el acceso al sistema desde cualquier geografía del país, y dada la | | | |

| | |
|---|---|
| | marcada diferencia tecnológica que existe, una aplicación Web garantiza el uso satisfactorio del sistema ya que no demanda muchos requerimientos de hardware y software de la PC en la que se trabaja, solo necesita estar conectado a la red y tener instalado algún navegador Web como por ejemplo Internet Explorer, Mozilla Firefox, Chrome, como ejemplo de la tecnología Web utilizada está JSP y el Framework Spring.<br>**S10.** Aumenta la portabilidad. |

| **Escenario #: 11** | **Usar Internet Explorer, Mozilla Firefox o Chrome, para acceder a la aplicación.** | | | |
|---|---|---|---|---|
| **Atributo(s)** | Portabilidad-Adaptabilidad. | | | |
| **Ambiente** | Operación normal. | | | |
| **Estímulo** | El usuario utiliza cualquier navegador web para acceder a la aplicación. | | | |
| **Respuesta** | La aplicación funciona correctamente sin ocurrir cambios en la interfaz de usuario. | | | |
| **Decisiones arquitectónicas** | **Riesgo** | **Punto de Sensibilidad** | **Punto de Intercambio** | **No Riesgo** |
| Java Script | R2 | S11 | | |
| **Razonamiento** | Las siguientes decisiones arquitectónicas garantizan el cumplimiento del uso del sistema independientemente del navegador Web que se utilice:<br>**R2.** Desactivar el java script. Usar el lenguaje de programación Java Script permite implementar funciones que permitan reconocer el navegador que se utiliza para acceder al sistema y en dependencia del mismo hacer las adaptaciones necesarias para que no haya cambios en la interfaz del usuario.<br>**S11**. Aumenta la portabilidad. | | | |

GLOSARIO DE TÉRMINOS

**BD:** Base de Datos.

**CASE:** (Computer Aided Software Engineering). Ingeniería de Software Asistida por Computación.

**CU:** Caso de Uso.

**DATEC:** Centro de Tecnologías de Gestión de Datos.

**Hardware:** Componentes físicos que constituyen las computadoras y demás dispositivos periféricos.

**HTTP:** (Hypertext Transfer Protocol) .Protocolo de transferencia de hipertexto usado en cada transacción de la World Wide Web.

**IDE:** Es un entorno de desarrollo integrado, programa compuesto por un conjunto de herramientas para un programador desde el que se pueden editar programas, compilarlos y depurarlos.

**JSP (Java Server Pages):** Es una tecnología orientada a crear páginas web con programación en Java.

**Lenguaje de Programacion:** Es un idioma artificial diseñado para expresar computaciones 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.

**Metodología de Desarrollo:** Marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo en sistemas de información.

**microRNA:** Moléculas pequeñas de RNA

**PostgreSQL:** Es un sistema de gestión de base de datos relacional orientada a objetos y libre.

**RNA:** Ácido Ribonucleico

**SNP:** Polimorfismos Nucleótido Simple.

**Software:** Equipamiento lógico o soporte lógico de una computadora digital; comprende el conjunto de los componentes lógicos necesarios que hacen posible la realización de tareas específicas, en contraposición a los componentes físicos, que son llamados hardware.

**URL: (***Uniform Resource Locator***).** Localizador de Recurso Uniforme, dirección global de documentos y de otros recursos en la World Wide Web.

**USB:** Universal Serial Bus, puerto que sirve para conectar periféricos a una computadora.