UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Tarea 3: Diseño OO

Symdex

Cliente: Symbiose
Empresa: Blastosoft
Integrantes: Juan Arredondo
Flavio Ramos
Alberto Rodríguez
Matías Tobar

17 de Junio de 2019

1. Resumen ejecutivo

Symdex es una aplicación web donde se permiten dos usuarios. El administrador, el cual la aplicación le permite crear propuestas, editarlas, buscarlas y eliminarlas, como también asignar propuestas a un cliente, junto con poder ver las estadísticas de cierto cliente. Por ende, el objetivo general de Symdex para el usuario administrador es gestionar las propuestas que se le asignan a cada cliente, además ofrece la posibilidad de la indexación de estas.

Para el otro usuario de la aplicación el cual se le asigna el nombre de "usuario", es un trabajador de la empresa Symbiose, al cual se le asigna solamente la posibilidad de ver las propuestas técnicas de la aplicación, no tiene la posibilidad de gestionar nada, ni tampoco ver las propuestas económicas.

El siguiente escrito busca abarcar el desarrollo de la fase de Diseño OO de OMT++/UML y los resultados obtenidos de esta son el diseño arquitectural y el diseño detallado de la aplicación a implementar.

Tabla de contenidos

Índice de figuras

2. Diseño OO

El diseño OO (orientación a objetos) consiste en un enfoque de la ingeniería de software que modela un sistema como un grupo de objetos que interactúan entre sí, donde todo sistema de información requiere de artefactos o componentes (clases) para llevar a cabo tareas. Además es de gran importancia dentro de la ingeniería de software, ya que permite tener un buen .ªnálisis y diseño"para un mejor desarrollo, que conlleva a qué tan .esçalable"sea un sistema de información. Esta fase de diseño es realizada utilizando, el modelo de arquitectura 3+1 vistas propuesta por la metodología OMT++.

Para el caso de la tarea número tres, nos será de enorme importancia para la elaboración de los diagramas que son parte del proceso de creación de software ya que permite aplicar un conjunto de modelos utilizando una notación acordada como, por ejemplo, el lenguaje unificado de modelado (UML), dicho diagramas se describen a continuación:

2.1. Diseño Arquitectural

2.1.1. Vista lógica

Esta vista tiene como finalidad mostrar la descomposición lógica del sistema a implementar en abstracciones como por ejemplo subsistemas, módulos o grupos de objetos que están relacionados. En la notación utilizada en esta vista cada paquete o carpeta, representa una agrupación lógica de elementos. Para la aplicación que crearemos hemos dividido el sistema en las tres capas capas más importantes, donde se engloban distintos módulos y dependencias entre ellas. Estos subsistemas serían Front-end, Back-end y base de datos, lo cual se diagrama de la siguiente manera:

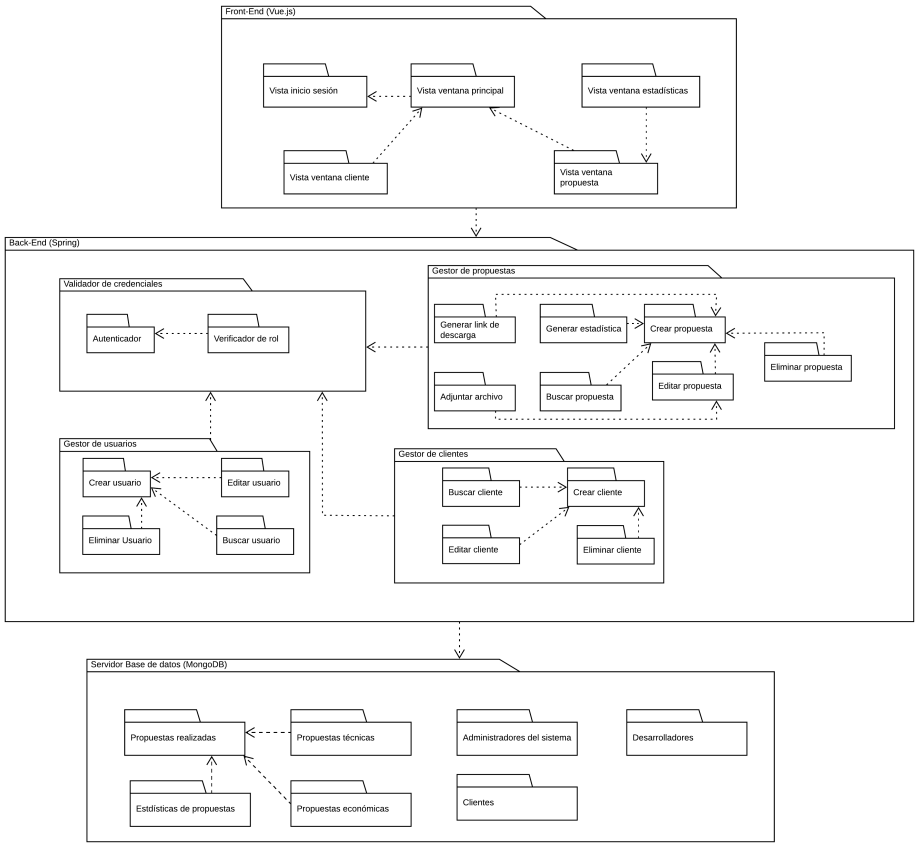

Figura 1: Vista Lógica (Paquetes).

2.1.1.1. Front-end

La primera capa y más dependiente de las demás que se ha identificado es el Front-End, la cual será implementada utilizando el framework Vue.js, aquí se puede apreciar una serie de módulos que serían las vistas de la interfaz gráfica con la que interactúa el usuario. Se puede apreciar que la vista que genera más dependencias es la de iniciar sesión, ya que si no se accede a ésta primeramente, no se podrá acceder a las demás. Siguiendo esa misma lógica, ocurre lo mismo con las vista de la ventana principal (lista de propuestas, clientes o usuario), donde desde aquella podemos acceder a datos más personalizados del cliente o la propuesta enviada.

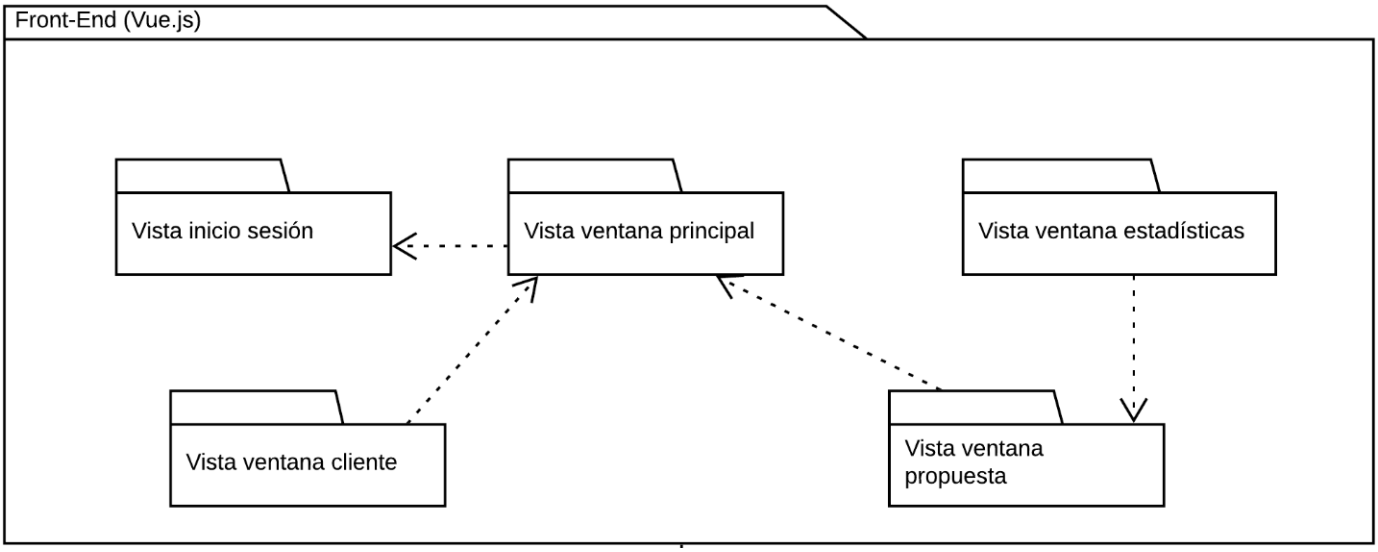

Figura 2: Fronted (Primera Capa).

2.1.1.2. Back-end

Posteriormente, la siguiente capa del sistema sería el Back-End, la cual será implementada utilizando el framework Spring, donde el acceso a datos de un software o cualquier dispositivo no es directamente accesible por los usuarios, además contiene la lógica de la aplicación que maneja dichos datos. Dicha lógica se ve expresada en cuatro subsistemas principales, con sus correspondientes módulos internos.

El módulo principal, similar al Front-end, con la diferencia que en este se implementa la lógica interna, sería el validador de credenciales, quien genera las dependencias de las demás funcionalidades, en que primeramente verifica el nombre y contraseña del usuario y luego verifica el rol que cumple en la plataforma, para restringir la cantidad de funcionalidades que puede utilizar dentro de la plataforma.

Luego, los demás subsistemas guardan la lógica de las propuestas y el manejo de perfiles, donde se puede apreciar que a grandes rasgos son el crud de funciones que deben ser implementadas en el programa, la mayoría de esas funcionalidades fueron descritas en la tarea número dos.

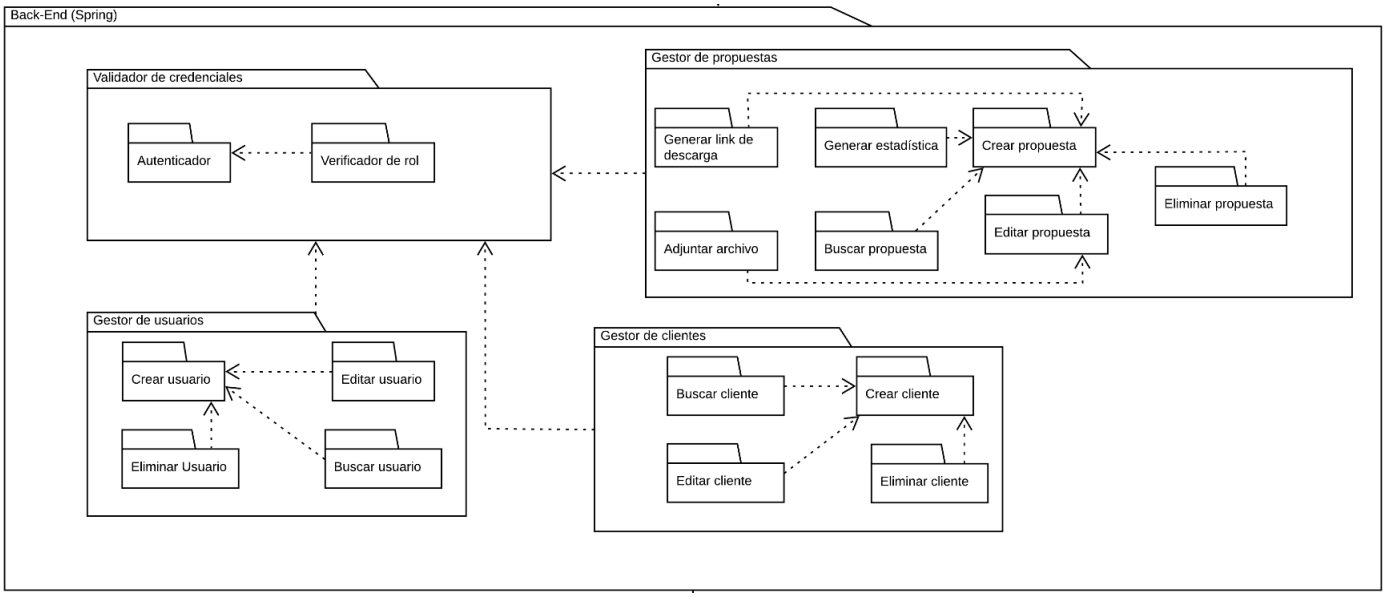

Figura 3: Backend (Segunda Capa).

2.1.1.3. Servidor de base de datos

La tercera capa sería el servidor de base de datos, donde se trabajará utilizando una base de datos no relacional (MongoDB), y se guardarán todos los datos asociados a las propuestas realizadas en el sistema, como los perfiles de las personas que interactúan con la plataforma. En el diagrama realizado, podemos apreciar los datos de los archivos que serán almacenados, mencionando las principales "entidades" del programa.

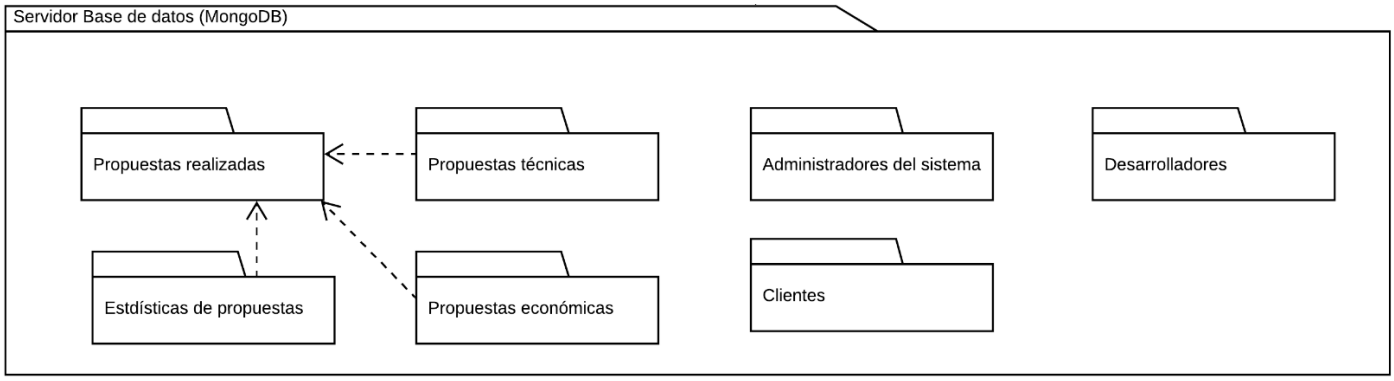

Figura 4: Servidor de Base de Datos (Tercera Capa).

2.1.2. Vista de desarrollo

Esta vista tiene como enfoque principal especificar los componentes visibles en el desarrollo, permitiendo así estimar el esfuerzo que se debe involucrar en desarrollo del sistema. La idea es que los componentes de esta vista puedan ser desarrollados como miniproductos independientes dentro de macroproducto que es el sistema.

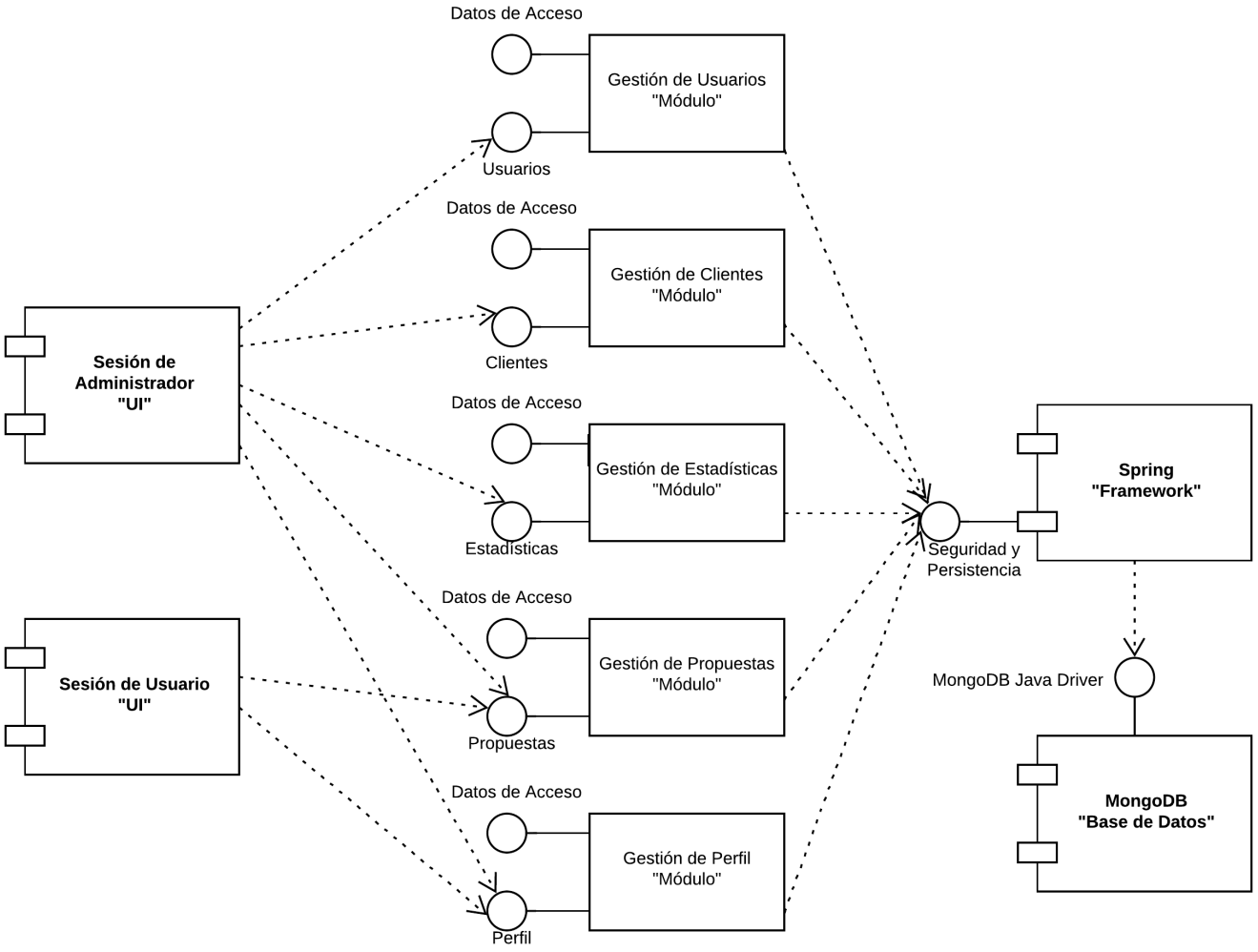

Figura 5: Vista de desarrollo (o de componentes).

Descripción de componentes:

- **Sesión de Administrador:** Corresponde al ingreso de un usuario con el rol de administrador de la plataforma con el fin de gestionar las propuestas y ver a cual cliente se le envió, junto con sus estadísticas. Por esto es capaz de gestionar los clientes y las estadísticas.
- **Sesión de Usuario:** Corresponde al ingreso de un usuario con el rol de trabajador de la empresa Symbiose con el propósito de solo poder ver las propuestas técnicas.
- **Gestión de Usuarios:** Módulo que engloba las operaciones que hace el administrador sobre los usuarios existentes en la aplicación.
- **Gestión de Clientes:** Este Módulo engloba las operaciones que hace el administrador sobre los clientes existentes.
- **Gestión de Estadísticas:** Módulo que engloba todo lo relacionado con las operaciones que hace el administrador con las estadísticas de los clientes.
- **Gestión de Propuestas:** Módulo que engloba las operaciones que puede realizar el administrador con las propuestas, así como vincularlas a algún cliente. Para el caso de un usuario el solo puede realizar la operación de ver las propuestas técnicas.
- **Gestión de Perfil:** Este Módulo engloba todas las posibles operaciones que pueden hacer los usuarios de la aplicación (sea administrador o usuario).

2.1.3. Vista de procesos

Esta vista tiene como propósito mostrar los procesos existentes dentro del sistema y la manera en la que estos se comportan cuando el sistema está en ejecución. Este diagrama le aporta más valor a los integradores de sistema y a los encargados de mantener todo en funcionamiento una vez que este está implementado.

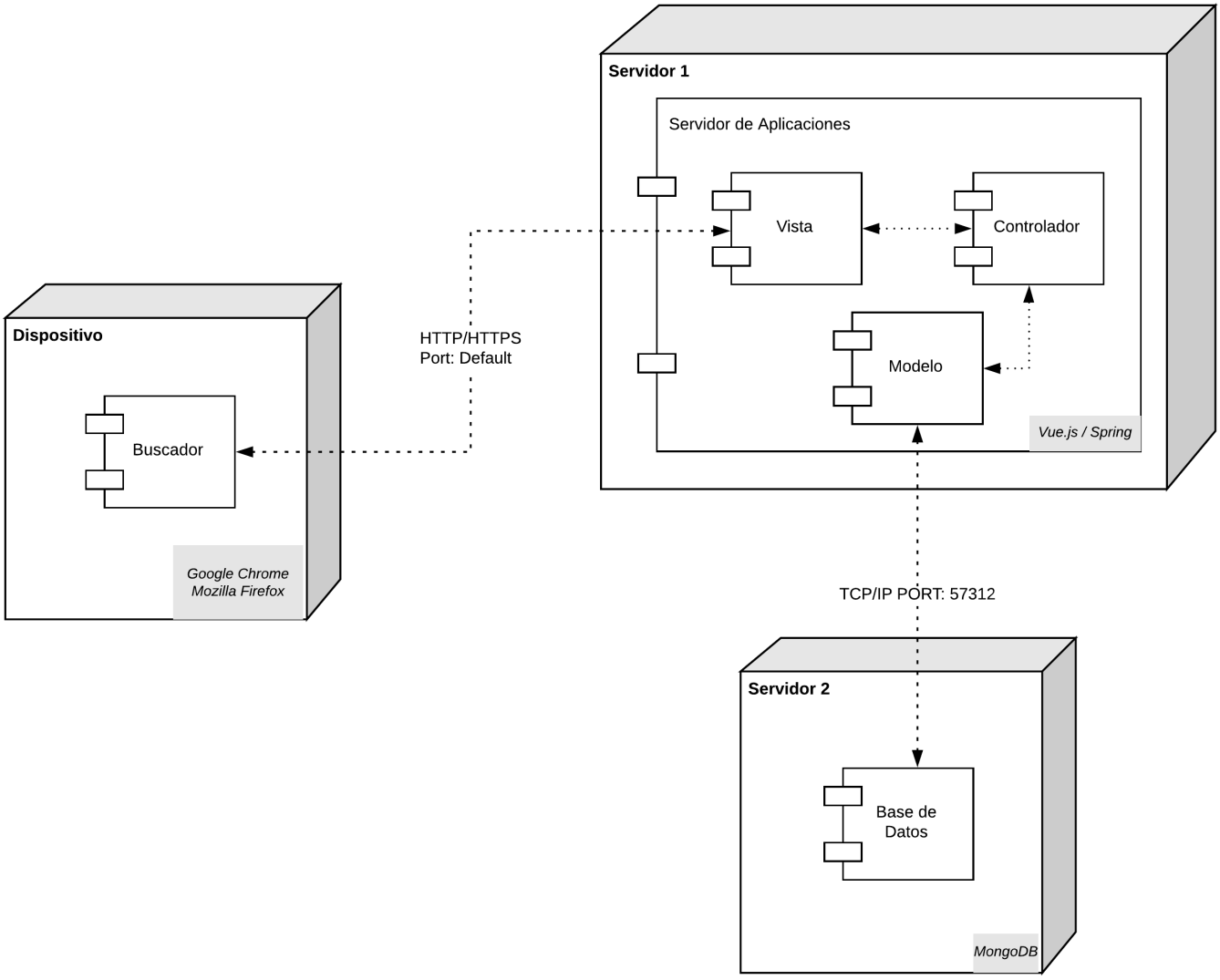

Figura 6: Vista de procesos(o despliegue).

2.2. Vista de comportamiento (colaboración entre componentes de desarrollo)

Esta vista tiene como objetivo especificar la colaboración de los componentes ejecutables de la vista de procesos y los componentes implementables de la vista de desarrollo, sin especificar los nombres de estos. Para esta vista se utilizan las operaciones definidas en la fase de análisis como punto de partida. En el caso actual se representan las operaciones o servicios que más valor le aportan al cliente, las cuales son:

1. Crear una propuesta (Ver Figura 7)
2. Indexar una propuesta (Ver Figura 8)
3. Visualizar estadísticas de propuesta (Ver Figura 9)

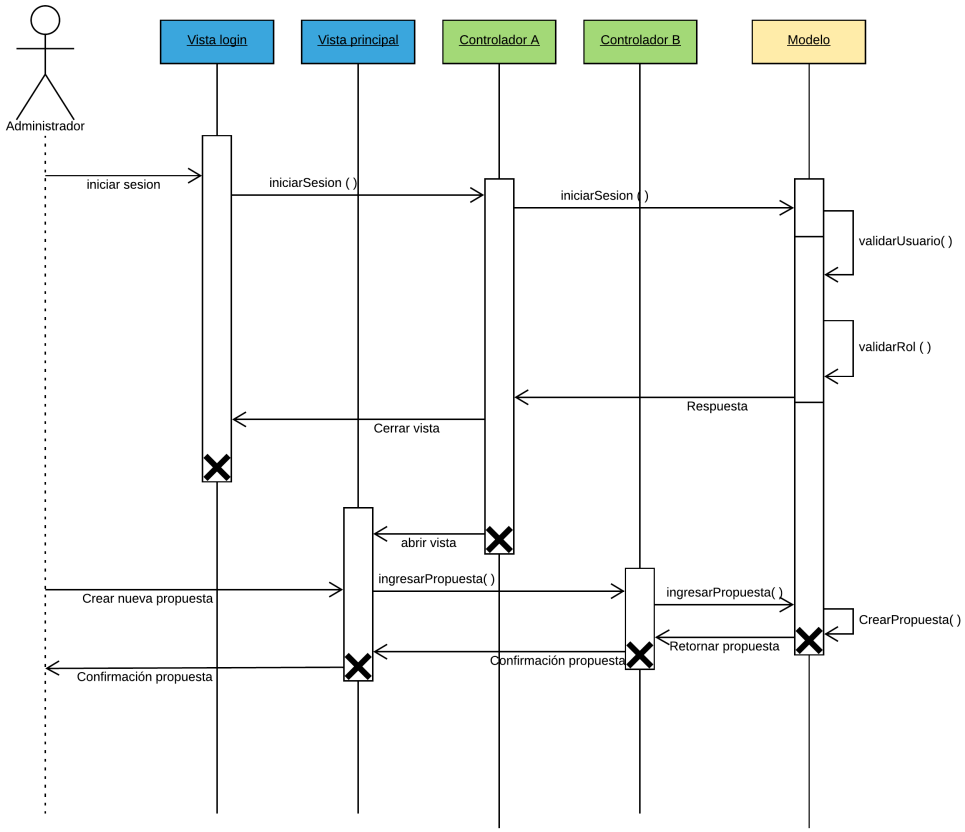

Figura 7: Vista de comportamiento: Crear una propuesta.

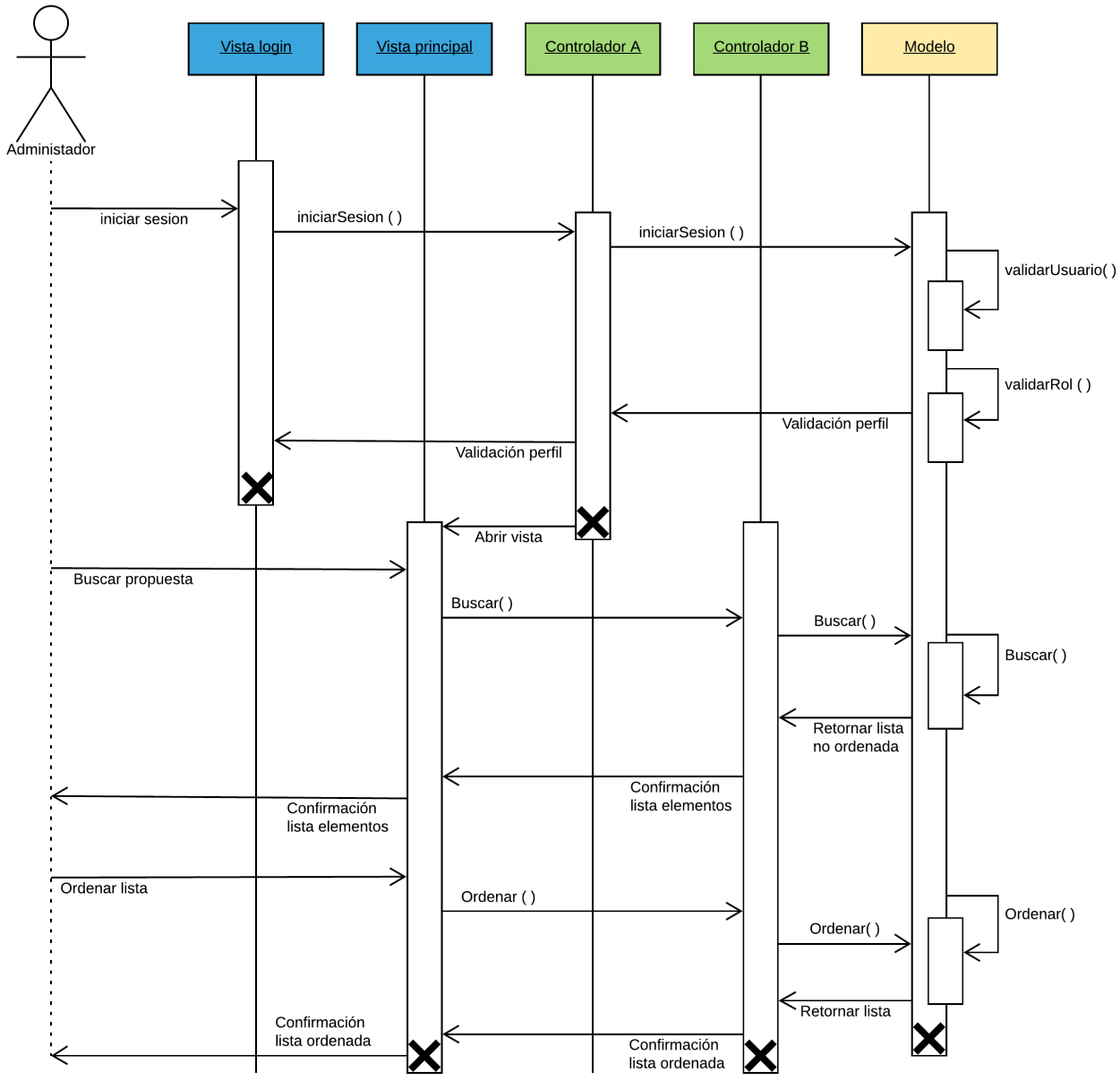

Figura 8: Vista de comportamiento: Indexar propuesta.

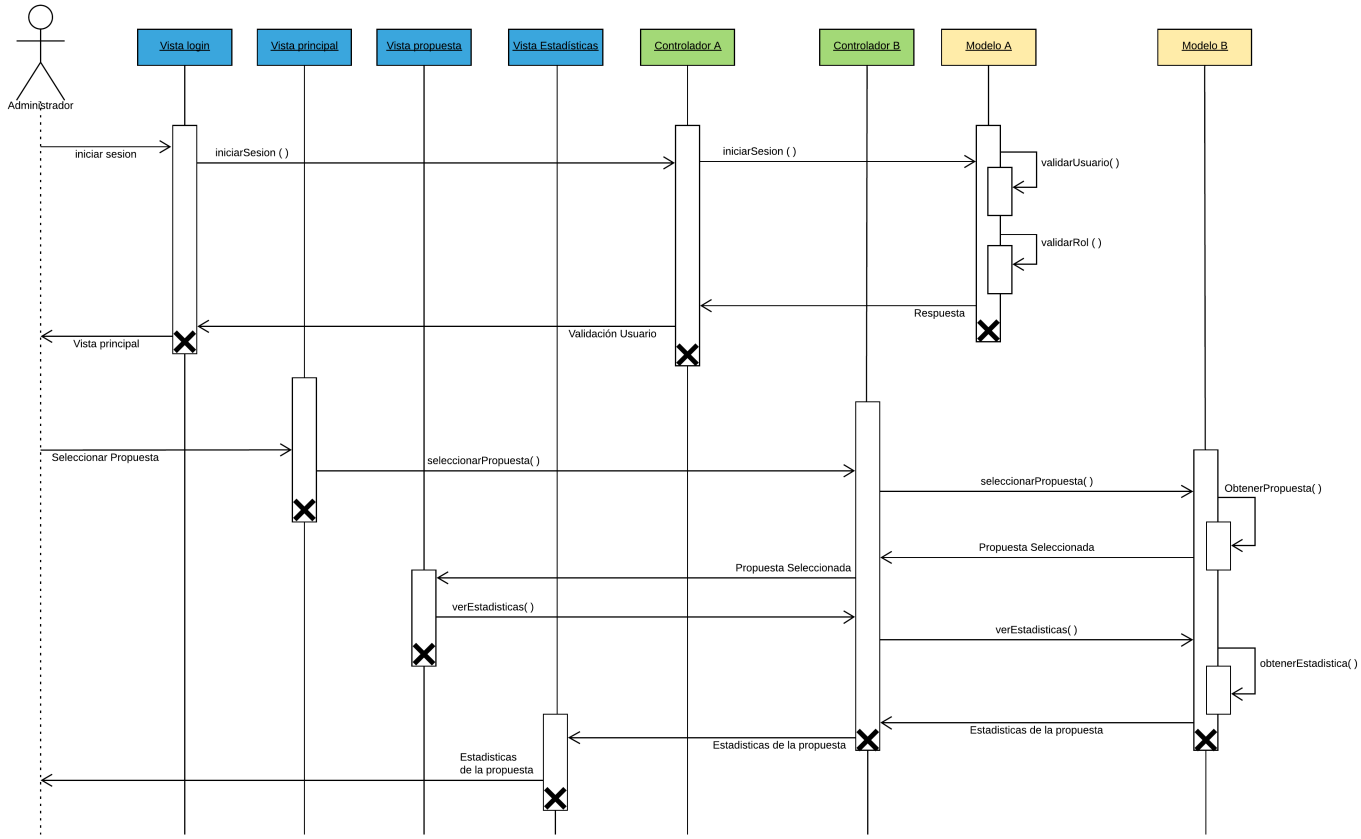

Figura 9: Vista de comportamiento: Revisar Estadísticas.

2.2.1. Interfaces

Una interfaz se utiliza para nombrar a la conexión funcional entre dos sistemas, programas, dispositivos o componentes de cualquier tipo, que proporciona una comunicación de distintos niveles permitiendo el intercambio de información. También es conocida como un contrato que permite la comunicación entre dos componentes, estableciendo los parámetros de entrada y salida. En el sistema se identifican las siguientes interfaces, utilizadas para cubrir las tres funcionalidades que más valor le aportan al cliente:

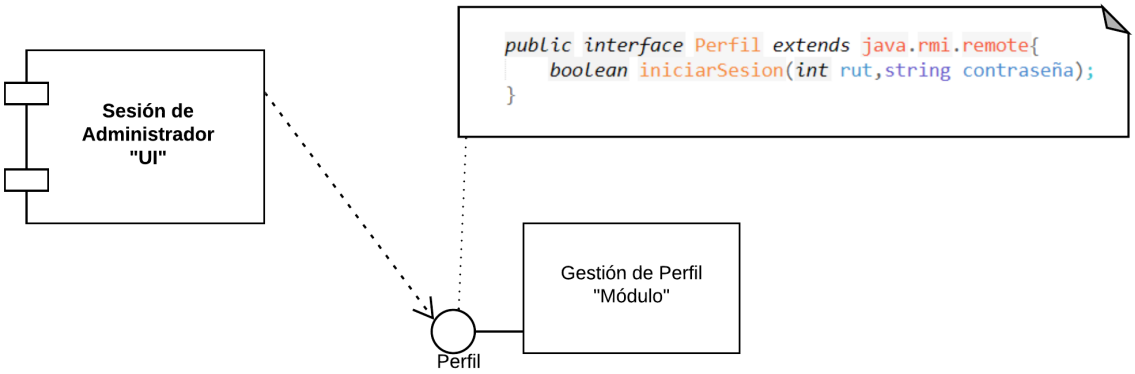

Figura 10: Interfaz de Perfil.

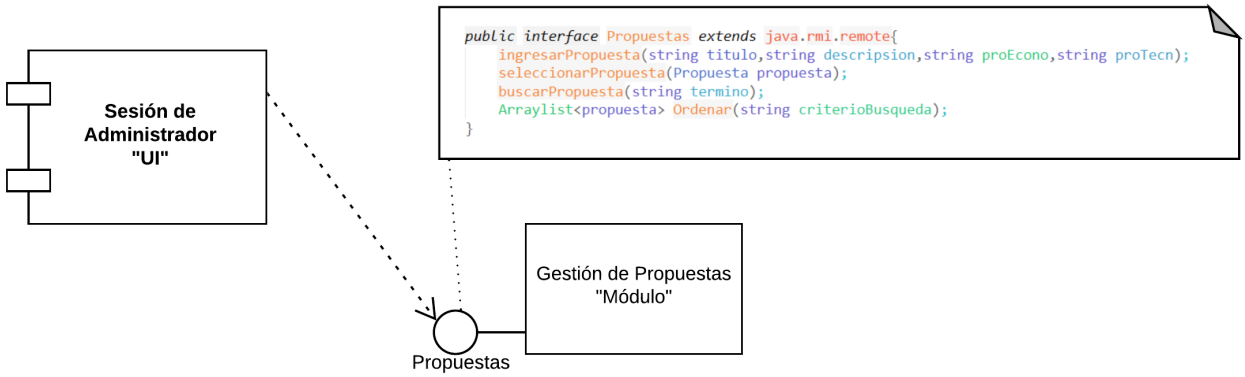

Figura 11: Interfaz de Propuestas.

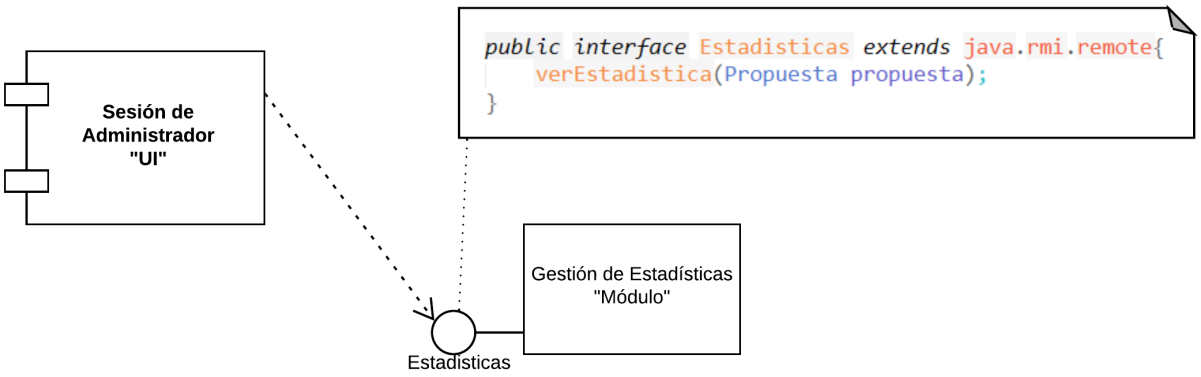

Figura 12: Interfaz de Estadísticas.

2.3. Diseño detallado

El diseño detallado tiene como finalidad presentar con aún mayor especificidad las definiciones e interacciones entre los distintos objetos que componen un sistema. Para cumplir con este propósito se utilizan dos diagramas según UML, el primero de ellos es el diagrama detallado de comportamiento o escenario, mientras que el segundo es un diagrama de clases detallado donde se especifican atributos y métodos de cada una de las clases que componen las tres capas de la arquitectura MVC++ del sistema.

2.3.1. Diagramas de secuencia

Los diagramas de comportamiento detallado muestran el flujo de actividad que sigue la operación a través de los diferentes módulos del sistema con el objetivo de proporcionar una visión más completa del funcionamiento de la aplicación. Estos diagramas son una versión más detallada de lo expuesto en la vista de comportamiento del diseño arquitectural, con la diferencia de que éstos muestran las funciones y módulos específicos utilizados por cada operación.

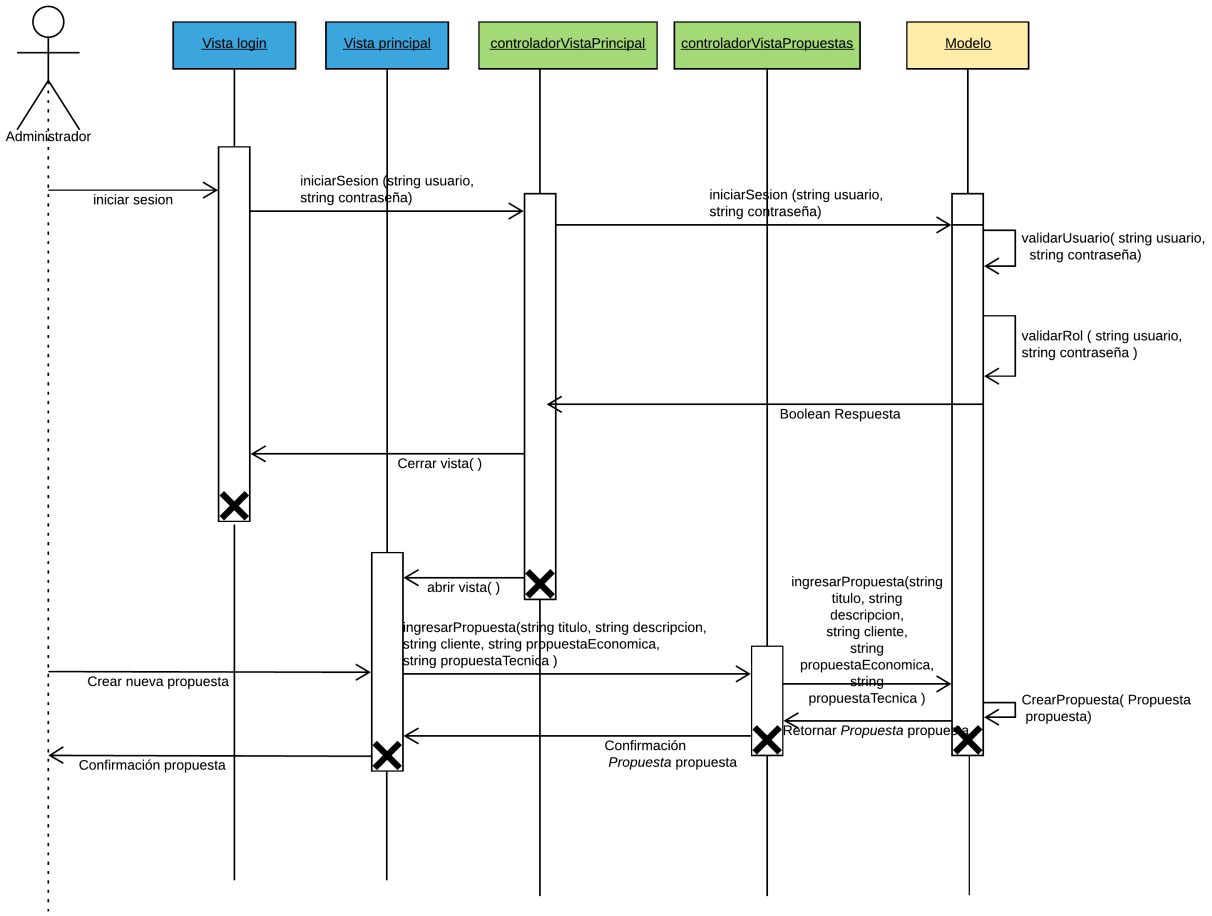

Figura 13: Comportamiento detallado: Crear una propuesta.

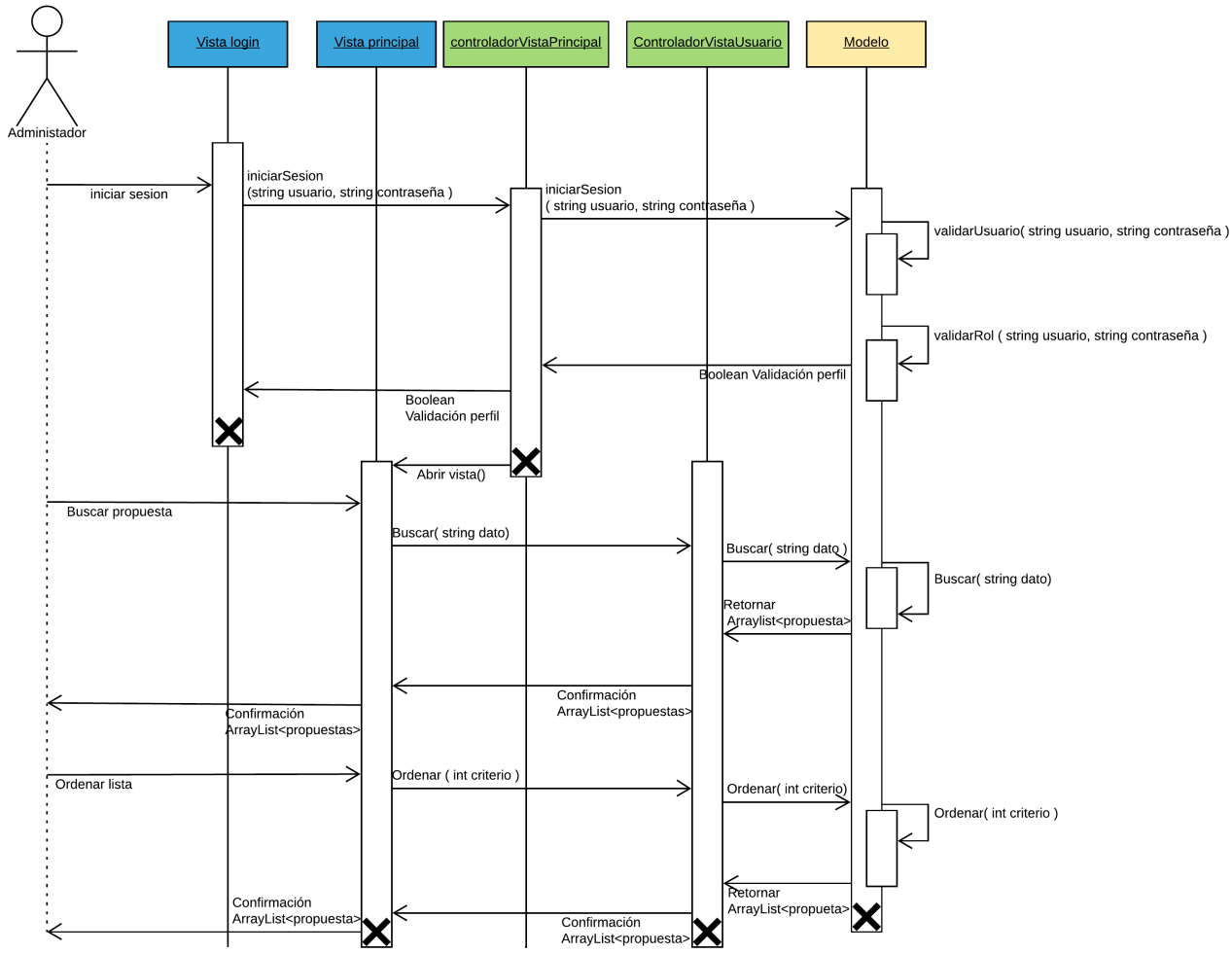

Figura 14: Comportamiento detallado: Indexar una propuesta.

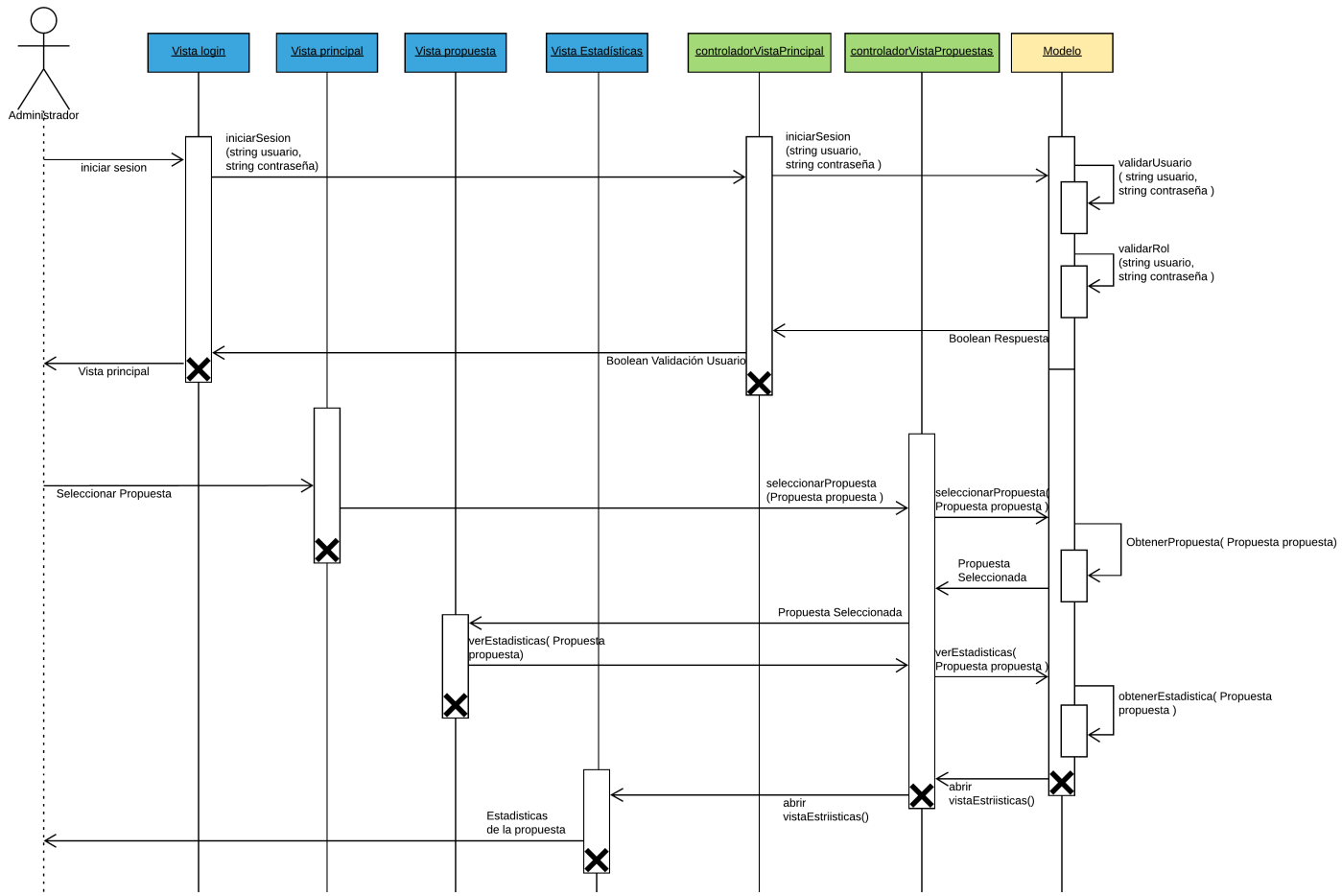

Figura 15: Comportamiento detallado: Revisar Estadística de un cliente.

2.3.2. Diagramas de clases detallado

El diagrama de clases detallado muestra las clases identificadas según un patrón de diseño MVC++, puesto que las partes de la vista no se relacionan directamente con el modelo. Se muestran tanto los atributos como los métodos pertenecientes a cada clase y cómo estas se relacionan entre sí.

En el diagrama se pueden ver los controladores, de los cuales existe uno por cada vista que ha de tener la aplicación web, además se puede ver cómo estos componentes se relacionan con los elementos del modelo, los cuales finalmente son los que relacionan la información contenida en la base de datos.

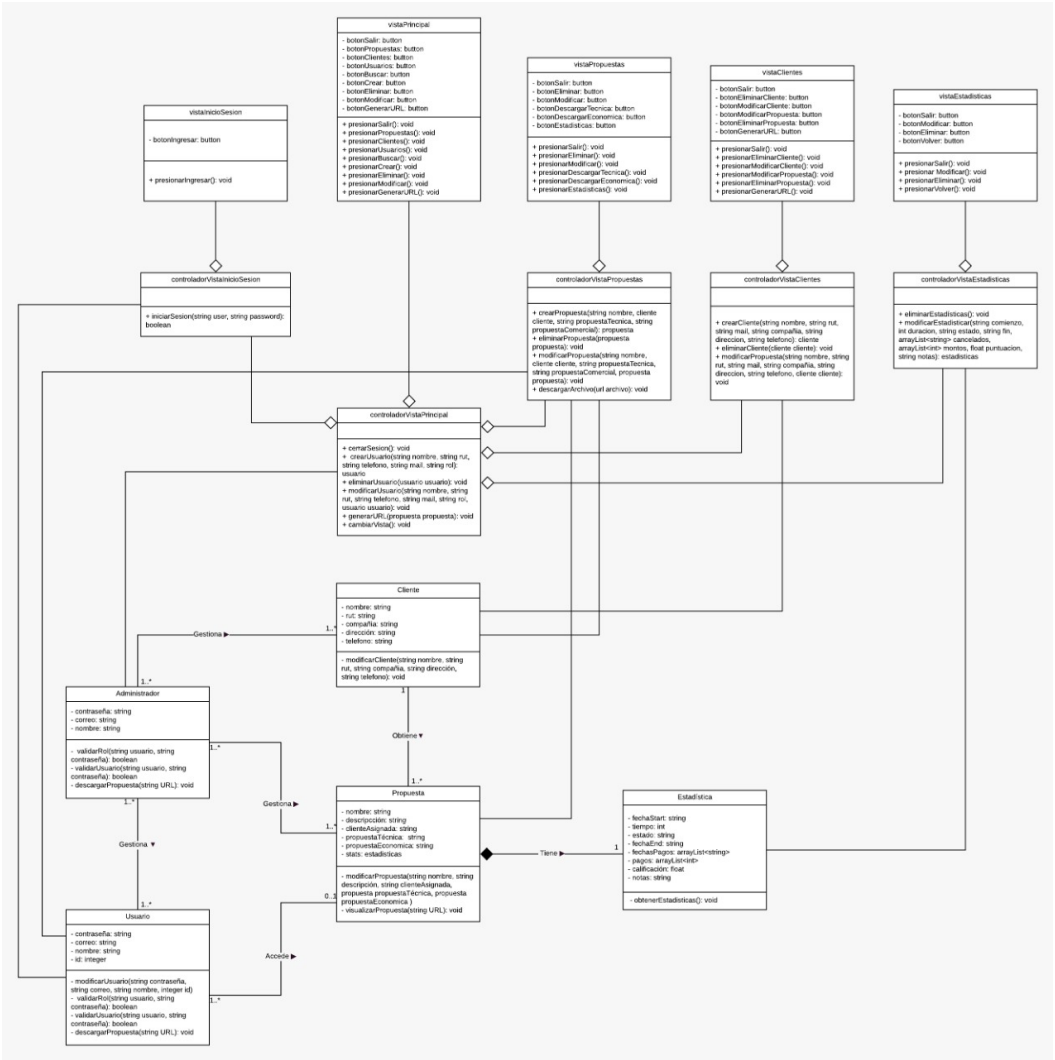

Figura 16: Diagrama de clases detallado.

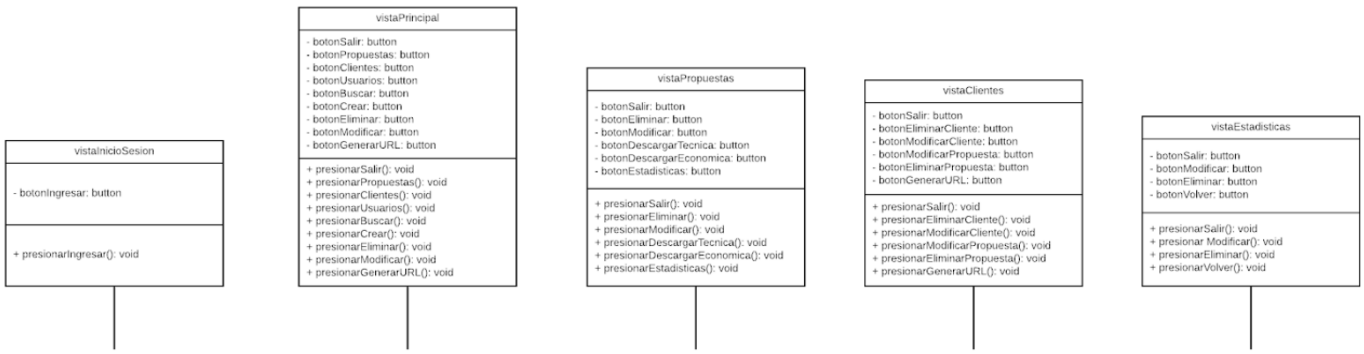

Figura 17: Diagrama de clases: Capa de Vista.

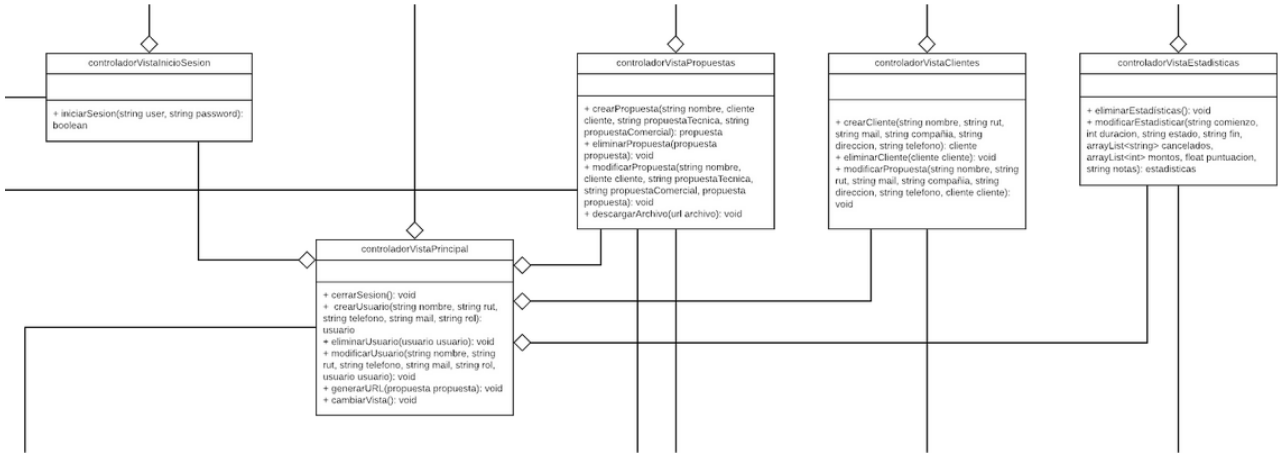

Figura 18: Diagrama de clases: Capa de Controladores.

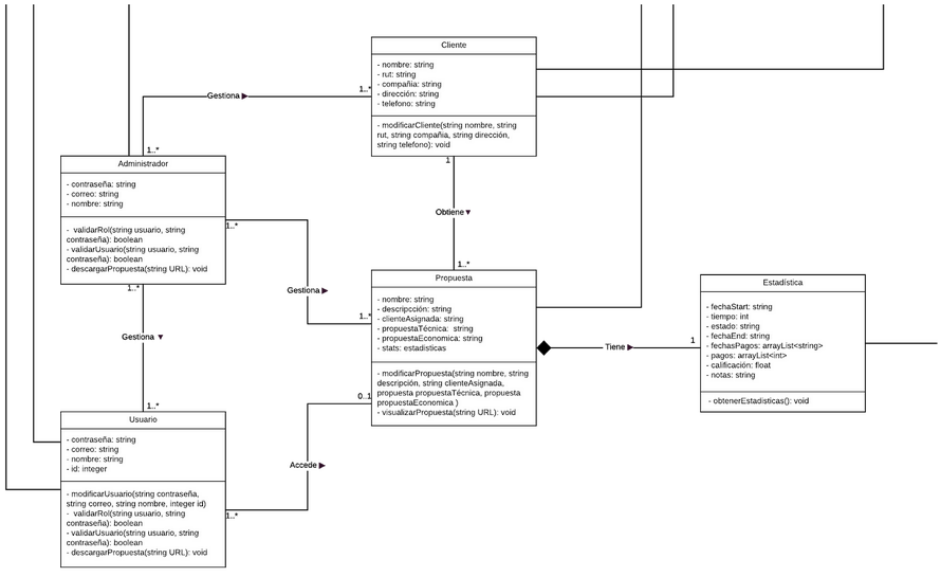

Figura 19: Diagrama de clases: Capa de Modelos.

3. Conclusiones

Luego de lo expuesto en el documento, podemos decir que se ha cumplido a cabalidad lo planteado para la tarea número tres de fundamentos de la ingeniería de software. Utilizando las herramientas que nos entrega el análisis OO, elaboramos una arquitectura de software consecuente con el programa que implementaremos, creando una serie de diagramas que nos permite visualizar el desarrollo del programa desde varias perspectivas, algunos de esos modelos fueron el diagrama de vista lógica, de proceso, desarrollo y modelos de comportamiento que nos ayudan a entender claramente el funcionamiento que tendrá el programa de nuestro cliente Symbiose.

Para la elaboración de estos diagramas se tomó como referencia los apuntes vistos en cátedra, donde luego de reflexionar y lograr entender mejor los diagramas, pudimos plantearnos la idea de como solucionar el problema. Primeramente identificamos los actores involucrados y las interacciones lógicas de las funcionalidades en el programa, añadiendo las dependencias se logró formar un sistema eficiente que represente de gran manera las funciones involucradas.

Además, luego de realizar los cambios planteados luego de la presentación, aplicamos el análisis MVC++, realizando pequeña modificaciones a los modelos para que tuvieran más consistencia con lo solicitado por el cliente. De aquello pudimos concluir que nuevamente resulta de vital importancia tener en consideración que los resultados obtenidos en las etapas anteriores dictan en gran medida el transcurso, o al menos los inicios, de cada nueva etapa en una metodología como lo es OMT++.

Es por ello que como conclusión general podemos decir que gran parte del entregable se logró realizar a plenitud, finalizando la etapa de diseño del proyecto y entrando de lleno en la implementación, donde con el análisis realizado por las metodologías aprendidas en cátedra, nos entrega una base para empezar a familiarizarnos con las tecnologías y comenzar a programar algunos aspectos del programa. Además, con el desarrollo de la tarea número tres logramos una buena experiencia sobre los pasos a seguir para crear un software de forma correcta, aumentando nuestras capacidades de resolución de problemas en la práctica.

4. Bibliografía

- Sommerville, I. (2005). Ingeniería del software, séptima edición. Disponible en https://ulagos.files.wordpress.com/2010/07/ian-sommerville-ingenieria-de-software-7-ed.pdf
- Diez, S. (2016). CASO PRÁCTICO: Definición de historias de usuario y criterios de aceptación (AC). Disponible en https://descubriendoagile.wordpress.com/2016/03/22
- Antillanca, H. (2003). ISO 5 y ISO 6. Disponible en www.udesantiagovirtual.cl