UNIVERSIDAD POLITÉCNICA SALESIANA
SEDE QUITO

CARRERA: INGENIERÍA DE SISTEMAS

Trabajo de titulación previo a la obtención del título de:
INGENIERA DE SISTEMAS

TEMA:
“ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA WEB, APLICANDO LA TECNOLOGÍA JAVA SERVER FACES (JSF), PARA LA GESTIÓN Y CONTROL DE TRANSPORTE TURÍSTICO DE LA COMPAÑÍA CHRISLAND SERVICE AND TOURING S.A.”

AUTORA:
VERÓNICA MARINA SANGUCHO CUEVA

DIRECTOR:
DANIEL GIOVANNY DÍAZ ORTIZ

Quito, febrero de 2015

DECLARATORIA DE RESPONSABILIDAD Y AUTORIZACIÓN DE USO DEL TRABAJO DE TITULACIÓN

Yo, autorizo a la Universidad Politécnica Salesiana la publicación total o parcial de este trabajo de titulación y su reproducción sin fines de lucro.

Además, declaro que los conceptos y análisis desarrollados y las conclusiones del presente trabajo son de exclusiva responsabilidad de la autora.

Quito, diciembre de 2014

Verónica Marina Sangucho Cueva

CC: 1718915349

DEDICATORIA

El presente trabajo de titulación se lo dedico con todo mi cariño y mi amor a las personas importantes en mi vida, que siempre estuvieron listas para brindarme su ayuda, para motivarme y darme la mano. A mi familia, mis padres, hermanos y hermana a ustedes por siempre en mi corazón.

AGRADECIMIENTO

A la Universidad Politécnica Salesiana, que a través del departamento de trabajos de titulación me han guiado en el camino para la culminación de mi carrera profesional.

Especial y sincero agradecimiento al Ing. Daniel Díaz, por su confianza al aceptar realizar este trabajo de titulación bajo su dirección, lo cual ha hecho posible su realización satisfactoria.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El presente trabajo de titulación describe de manera teórica y práctica el proceso de desarrollo e implementación del producto web, para la gestión y control del servicio de transporte turístico que brinda la compañía Chrisland Service And Touring.

Para el desarrollo del sistema se utilizaron las mejores tecnologías líderes en el mercado y de código abierto como son principalmente; el entorno de desarrollo integrado Eclipse Spring Tool Suite, el servidor de aplicaciones JBoss. Para el mapeo objeto relacional la solución Hibernate. Para la codificación con patrones estandarizados especialmente la inyección de dependencia, el patrón de diseño Objeto de Acceso a Datos (DAO) y Objeto de Transferencia de Datos (DTO) el framework Spring y para las interfaces gráficas de usuario la tecnología Java Server Faces en conjunto con el lenguaje básico de la World Wide Web en su versión 5 como es HTML5 y CSS3 como lenguaje de los estilos o apariencia de la página web.

Para brindar facilidad en su mantenimiento el sistema fue implementado en base a estándares. Para la nomenclatura en la base de datos, proyectos, paquetes, clases, interfaces, métodos y variables. En cuanto a la arquitectura del software se empleó el patrón Modelo Vista Controlador (MVC) el cual divide al sistema en tres capas que son el Modelo que se ocupa de la persistencia y recuperación de los datos, la Vista que es medio por el cual el usuario puede interactuar directamente con el sistema y el controlador que es la unión o enlace entre la Vista y el Modelo.

ABSTRACT

This paper describes theoretically and practically the development and implementation process of the Web product that will be used for management and control of the tourist transport service provided by the company Chrisland Service And Touring.

In order to develop the system we used the best leading technologies of the market and of the open source, such as the integrated development environment Eclipse Spring Tool Suite and the JBoss application server. For the object relational mapping, we use the Hibernate solution. For codifying standardized patterns, particularly the dependency inyection, we use the design pattern Data Access Object (DAO) and Data Transfer Object (DTO), the Spring framework and for the GUI Java Server Faces technology with the core language of the World Wide Web in its Version 5, such as HTML5 and CSS3, as style language or appearance of the web page.

In order to provide maintenance facilities, the system was implemented based on standards. For the nomenclature of projects, packages, types, interfaces, variable methods and the names in the database. With regard to the software architecture we use the pattern Model View Controller (MVC), which divides the system into three layers which are the model where the representation and direct connection to the database is located, the view that the user used in order to interact directly with the system and the controller that is the connection or link between the view and model.

INTRODUCCIÓN

"Chrisland Service and Touring S.A." es una empresa dedicada a la prestación de servicios de transporte turístico terrestre, sin embargo, la falta de un sistema que pueda estar en contacto con el cliente las 24 horas del día y los 7 días a la semana ha generado la pérdida de contrataciones y clientes.

Por esta razón, la empresa solicita una página web que permita mostrar los textos de información sobre la empresa en el idioma español e inglés con el fin de captar mayor mercado sobre todo el internacional, con un módulo que permita a los clientes realizar reservas en línea y un módulo que pueda registrar y mostrar a los vehículos que ofrecen para los viajes.

Debido a que el presente proyecto tiene un nivel de complejidad media-alta, el tamaño del equipo de desarrollo es pequeño, y no es de larga duración, se ha elegido la metodología ICONIX, la cual permite adaptabilidad a los cambios, para obtener sistema de buena calidad y que cumpla con los requerimientos de los usuarios.

Para garantizar la calidad del sistema final y que el mismo sea de fácil mantenimiento, expansión y que permita la reutilización de código, se utilizaron estándares y políticas de tanto en su arquitectura como en la nomenclatura definidas al inicio del proyecto, además, del uso de buenas prácticas de programación para la optimización del código fuente.

CAPÍTULO 1

PROBLEMA DE INVESTIGACIÓN Y OBJETIVOS

1.1 Diagnóstico de la situación

Chrisland Service and Touring S.A. es una empresa ecuatoriana legalmente constituida en figura de compañía limitada. La misma que cuenta con todos los permisos de operación otorgados por los organismos e instituciones respectivos para su normal funcionamiento. Actualmente se encuentran ubicada en la ciudad de Quito, sucursal principal Séptima Transversal lote 110 y General Rumiñahui.

Lleva más de 10 años de experiencia a nivel nacional en el servicio de transporte turístico institucional, empresarial y público en general. Brinda sus servicios con un staff de choferes profesionales capacitados en atención al cliente, rutas turísticas y relaciones humanas.

Los servicios que presta incluyen: traslados desde el aeropuerto o al lugar que sea solicitado, paseos y excursiones dentro del territorio ecuatoriano, atención a empresas, embajadas, agencias de viajes, colegios, fundaciones e instituciones en general. Los servicios se encuentran disponibles las 24 horas al día y los 365 días del año.

Desempeña sus labores en transporte turístico con diferentes tipos de unidades vehiculares de transporte terrestre, que tienen capacidades desde los 2 hasta los 45 pasajeros, es decir: de alta, mediana y baja capacidad como son: automóviles, vans, mini vans, buses, minibuses, busetas, furgonetas, todo terreno. Todas las unidades cuentan con el seguro del SOAT brindando así la seguridad necesaria.

La empresa tiene asociaciones y alianzas con proveedores del servicio de transporte turístico terrestre, que se encuentran en las principales ciudades del Ecuador como son; Quito, Guayaquil y Cuenca, esta alianza entre compañías les permite fortalecer la calidad del servicio al abordar la demanda que se presente; puesto que pueden contar con las unidades vehiculares propias de la compañía y también con las unidades vehiculares de sus asociados.

En la actualidad Chrisland Service And Touring realiza las reservaciones de los servicios de recorridos de transporte turístico terrestre a sus clientes vía telefónica y de forma manual. Sin embargo, debido a que no cuenta con un servicio de fácil y cómodo acceso y que pueda estar en contacto con el cliente las 24 horas del día y los 7 días a la semana ha generado la pérdida de contrataciones y clientes.

1.2 Objetivos

1.2.1 Objetivo general.

Analizar, diseñar e implementar un sistema web, que permita promocionar, gestionar y controlar el servicio de transporte turístico terrestre, que presta la compañía CHRISLAND SERVICE AND TOURING S.A. en el Ecuador.

1.2.2 Objetivos específicos.

- ✓ Analizar, diseñar e implementar el sistema con la metodología pesada-ligera de desarrollo ágil de software ICONIX.
- ✓ Desarrollar el sistema con las herramientas de entorno de desarrollo integrado con lenguaje de programación Java, JavaServer Faces (JSF) y PostgreSQL, con el manejo del patrón de diseño modelo vista controlador (MVC).
- ✓ Verificar que el código fuente de la aplicación sea documentado y óptimo a través del análisis estático del código Java con la herramienta PMD, Checkstyle y Javadoc con el objetivo de obtener un sistema que brinde facilidad en su mantenimiento.
- ✓ Habilitar una opción que permita mostrar los textos en el idioma inglés con el fin de captar mayor mercado sobre todo el internacional.
- ✓ Emplear en el desarrollo la estrategia de marketing digital como es y la vinculación de la página Web con redes sociales para ayudar al aumento de tráfico en la página.

1.3 Justificación del proyecto

La Organización Mundial del Turismo (OMT), tiene como objetivo promover el turismo al que define como: las actividades que realizan las personas durante sus viajes y estancias en lugares distintos al de su entorno habitual, por un período consecutivo de no más de un año, con fines de ocio, por negocios o por otros motivos (World Tourism Organization, 1995).

El turismo como actividad económica se conforma de los sectores que ofrecen bienes y servicios a los turistas para y durante su viaje, como son el sector hotelero, gastronómico, de transporte, agencias de viajes, entre otros. También representa una de las principales fuentes de ingresos económicos en el mundo. En el Ecuador, el turismo se encuentra ubicado en el tercer lugar de las fuentes que generan ingresos al país, de modo que puede constituirlo en una fuente de progreso consciente y sostenible del país.

Chrisland Service and Touring S.A. es una empresa turística constituida en figura de compañía dedicada a prestar servicios de transporte terrestre.

La empresa solicita una herramienta que permita potenciar la distribución directa de sus servicios a través de Internet. Con una página web que tenga la opción que permita mostrar los textos en el idioma ingles con el fin de captar mayor mercado sobre todo el internacional.

Los clientes interesados podrán registrar su reserva on-line de una manera fácil, rápida, intuitiva y cómoda. Con las especificaciones según sus preferencias en cuanto a las características del vehículo, número de personas, fecha del viaje, origen, destino y alguna observación adicional.

También desea gestionar la promoción de la flota con la que cuentan, a través del registro de los vehículos y sus características permitiéndole actualizar en cualquier momento la flota de vehículos que se muestra en Internet a través de su página web. El sistema permitirá el registro de los proveedores como son los miembros de la asociación y sus unidades vehiculares.

El sistema final permitirá agilitar la búsqueda y contratación del servicio de transporte, así como enriquecer la experiencia del viajero guiándolo en la excepcional diversidad cultural, natural y vivencial de nuestro país.

1.4 Herramientas tecnológicas

Para el desarrollo de la aplicación se va a utilizar principalmente las siguientes herramientas tecnológicas:

- **Tecnología Java Server Faces (JSF)**

Actualmente la tecnología y framework Java Server Faces es desarrollada por la Comunidad de Java. La cual crea estándares para el desarrollo de interfaces de usuario ricas en diseño en aplicaciones Java EE, fue creado con el objetivo de simplificar el desarrollo de aplicaciones web.

Java Server Faces está compuesto por un conjunto de APIs para la representación de componentes de interfaz de usuario, para el control de eventos, facilitar la internacionalización, validar datos de entrada, establecer reglas de navegación de las páginas.

El principal objetivo de esta tecnología es la facilidad en su uso, definiendo claramente la separación entre la capa lógica de negocio y la capa de presentación (Oracle Corporation).

- **Hibernate**

Hibernate es una solución para el mapeo objeto relacional o persistencia relacional para Java y .Net, es decir que se ocupa de la asignar las clases java a tablas de base de datos y a tipos de datos Java a tipos de datos SQL, con lo cual se crea una base de datos orientada a objetos virtual para utilizarlo dentro del lenguaje de programación.

También está encargado de proporcionar recuperación de datos a través de consultas. El objetivo principal de Hibernate es la de facilitar al desarrollador las tareas relacionadas con la persistencia de los datos, como son su almacenamiento y búsqueda (Red Hat, 2004).

- **Spring Framework**

Spring Framework es una solución que presta una infraestructura de apoyo para el desarrollo de aplicaciones empresariales Java. A pesar de que Java brinda un sin número de funciones para desarrollar aplicaciones, no cuenta con herramientas para organizar los todos los elementos.

De esta forma que deja esta tarea a los arquitectos y desarrolladores. Spring Framework soluciona este problema a través de la codificación de patrones de diseño formalizados (Pivotal Enterprise, 2004).

- **Herramientas de Revisión de Código Estático**

Las herramientas para la revisión de código estático son aquellas que sin ejecutar el código del software, lo analizan y ubican defectos de codificación. La ventaja de estas herramientas es que la aplicación no necesita estar integrada, ni ejecutarse para inspeccionar el código fuente, por lo que se puede garantizar el uso de buenas prácticas de programación en forma temprana.

- **PMD**

PMD es una herramienta que analiza código fuente de Java. Su objetivo es detectar errores comunes de programación como por ejemplo variables inutilizadas, creación de objetos innecesarios, código duplicado, entre otros. Compara el código fuente con las reglas preestablecidas, ubica los posibles errores y produce un informe (SourceForge, 2013).

- **Checkstyle**

Es una herramienta configurable que ayuda a los programadores a seguir un estándar de codificación, y ayuda a comprobar aspectos de forma del código fuente como son el número de líneas de código por clase entre otros. (SourceForge, 2012).

A continuación se presenta la tabla los recursos requeridos en cuanto a las herramientas tecnológicas en software necesarias para poder alcanzar los objetivos del proyecto y la consecución de la investigación.

Tabla 1. Herramientas en software para el desarrollo del producto

Lenguaje de Programación:	Java	"Tecnología que se usa para el desarrollo de aplicaciones que convierten a la web en un elemento más interesante y útil" (Oracle).
IDE (Entorno de desarrollo integrado):	Eclipse Spring Tool Suite	"Es ahora la plataforma por defecto para la construcción de aplicaciones de cliente enriquecido" (Eclipse).
Servidor de Aplicaciones:	Jboss	"Combina las tecnologías líderes en el mercado de código abierto, con capacidades de la empresa para proporcionar una única solución para los sitios web" (Red Hat, 2013)
Base de datos:	PostgreSQL	"PostgreSQL es un potente sistema gestor de bases de datos objeto-relacional de código abrierto" (PostgreSQL, 2013).
Desarrollo de Interfaces:	JSF	"JavaServer (TM) aborda a la tecnología que simplifica la creación de interfaces de usuario para aplicaciones JavaServer" (Java.net, 2013).
Lenguaje básico de la World Wide Web	HTML5	"Esta especificación define la quinta versión del lenguaje básico de la World Wide Web: el lenguaje de marcado de hipertexto (HTML). Se introducen nuevas características" (W3C,2013).
Controlador de Versiones:	VisualSVN Server	"VisualSVN Server le permite instalar y gestionar fácilmente un servidor de Subversion completamente funcional en la plataforma Windows" (VisualSVN Compay, 2013).
Modelado de Estándares de desarrollo:	Visio	"Crea diagramas profesionales para simplificar la información compleja con formas actualizadas" (Microsoft Corporation, 2013).
Plataforma de modelado visual:	Enterprise Architect	"Análisis UML y herramienta de diseño integral. Modelado de Rich para los negocios, software y sistemas. Ingeniería de código en más de 10 idiomas" (Sparx Systems, 2013).
Estilos:	CSS3	"es un mecanismo sencillo para añadir estilo (por ejemplo, fuentes, colores, espaciado) a los documentos web" (Bert Bos, 2013).
Framework:	Spring	"es una plataforma Java que proporciona un amplio soporte de infraestructura para el desarrollo de aplicaciones Java" (Pivotal, 2013).

Elaborado por: Verónica Sangucho

1.5 Procedimientos y recursos

Para el desarrollo del producto que se aplicó a una empresa de la vida real se procedió a utilizar dos metodologías. Una metodología de investigación para el levantamiento de información como es la investigación de campo y bibliográfica, y otra metodología para el desarrollo de software.

El levantamiento de información en la investigación de campo se llevó a cabo en la ciudad de Quito. En esta investigación de campo, bibliográfica y de desarrollo de software se cubrieron todas las etapas concernientes a este proyecto las cuales consistieron en:

- ✓ **Trabajo de campo:** Para determinar la funcionalidad del portal web propuesto se utilizó la entrevista como herramienta técnica de investigación durante las reuniones con los implicados. En el inicio del proyecto se realizarán entrevistas abiertas o cerradas como la herramienta para la obtención de los requerimientos iniciales del producto.

  Del mismo modo se realizarán reuniones con los implicados de manera constante durante todo el desarrollo del sistema. Las reuniones durante el desarrollo del software tienen el propósito de efectuar pruebas de aceptación del sistema aumentando por tanto las probabilidades de éxito del proyecto.

- ✓ **Implementación:** Consiste en la implementación de los requerimientos iniciales del sistema obtenidos a través de las entrevistas, y las mejoras o correcciones en los requerimientos del sistema que se descubran o se presenten durante el desarrollo del proyecto.
- ✓ **Pruebas al software:** Con el fin de verificar el buen funcionamiento de la aplicación. Las pruebas se realizaran por medio del análisis del código fuente estático PMD y CheckStyle. El propósito de realizar de manera permanente este análisis sobre el código fuente estático es mejorar la calidad el mismo.

  La herramienta PMD realiza una revisión del código con el fin de verificar que se están aplicando buenas prácticas de programación. De esta manera la herramienta PMD ayuda a obtener un código fuente óptimo.

La herramienta CheckStyle verifica que se cumplan estándares en cuanto al formato del código fuente como son: por ejemplo el ancho máximo de las filas, el número máximo de líneas de código por clase, entre otros, con el fin de establecer estándares que ayudan a facilitar la tarea de mantenimiento al sistema.

- ✓ **Metodología para el desarrollo de software**: Para tener éxito en la actual industria de desarrollo de software la competitividad exige una ágil adaptabilidad a los requerimientos de los usuarios y a los cambios que puedan surgir dentro de la línea de negocio.

  Por esta razón y debido a la naturaleza de este proyecto que es de corto plazo, se utilizará la metodología de procesos ágiles de desarrollo ICONIX, con el objetivo de obtener un sistema saludable y que cumpla con los requerimientos de los usuarios, tomándose como referencia el libro Agile Development with ICONIX Process.

  ICONIX es una metodología ágil de desarrollo de software, lo que quiere decir que durante el desarrollo del producto se van realizando pequeños ajustes en el software para que el mismo se adapte a los cambios o mejoras que puedan surgir en los requerimientos.

1.4.1 Justificación del uso de la metodología ICONIX.

En el proceso para elegir la metodología más adecuada a implementarse, se tomó en cuenta los factores de la naturaleza del presente producto de software a desarrollar, como son: el tiempo requerido para su implementación, el tamaño del proyecto, la disponibilidad de tiempo del cliente, la cantidad de documentación necesaria.

Tabla 2. Análisis comparativo de metodologías

	XP	RUP	ICONIX
Exigencia de documentación	Bajo nivel de exigencia en documentación, efectuándose el diseño mínimo para cumplir con los requerimientos actuales.	Trabajo pesado en documentación de diseño y planificación rígida.	Busca generar la documentación suficiente para el arranque del desarrollo.
Tipo de metodología	Ligera, ágil	Tradicional, pesada	Ágil Pesada-ligera
Tipo de proyecto	Es recomendable para proyectos de corto plazo.	Aún se implementan en proyectos de gran escala que no requieren resultados rápidos pero si procesos críticos.	Es adecuado para proyectos pequeños y medianos.
Participación del cliente	Requiere la participación a tiempo completo del cliente.	El cliente interactúa pero no es parte del equipo de desarrollo.	Promueve la interacción del cliente con el desarrollo del software, mostrándole versiones funcionales del sistema para que lo pueda evaluar.
Enfoque	Potenciar las relaciones interpersonales como clave para el éxito.	Enfoque disciplinado para asignar tareas y responsabilidades.	En ir de los casos de uso al código de forma fiable, en el menor número de pasos posible.
Tamaño del equipo	Pequeño	Medio o extenso	Pequeño o medio

Elaborado por: Verónica Sangucho

El presente proyecto no es de alta escala, el tamaño del equipo de desarrollo es pequeño, y no es de larga duración, y tiene un nivel de complejidad media por lo cual se utilizó la metodología ICONIX.

ICONIX se encuentra un punto medio entre la complejidad de RUP y la simplicidad de XP. Es decir la metodología ICONIX busca generar el análisis y documentación

necesaria y suficiente para el arranque del desarrollo del software. Por lo que la metodología ICONIX no abandona los beneficios del análisis y diseño iniciales.

La falta de un análisis inicial suficiente deriva en el surgimiento de futuros cambios o ajustes del sistema innecesarios ya que los mismos pueden ser prevenidos con un análisis preliminar al desarrollo del software. Por lo cual esta metodología ayuda a optimizar de este modo el uso de los recursos.

La metodología ICONIX tampoco se queda demasiado tiempo en el análisis inicial. Demasiado análisis en el inicio del proyecto puede llegar a ser un obstáculo para iniciar el desarrollo del producto y por tanto disminuir sus probabilidades de éxito del proyecto, o puede llevar a la comprensión de un problema que pudo haber cambiado con el tiempo y que por lo tanto está desactualizado.

CAPÍTULO 2

FUNDAMENTACIÓN TEÓRICA

2.1 Metodología ICONIX

La metodología ICONIX propone el uso de un subconjunto determinado de lo más importante y útil de UML. Saltándose por ejemplo los diagramas de colaboración y de estado por considerarlos redundantes. El subconjunto determinado de UML se conforma de:

- Los diagramas de clases, tanto en el análisis (dominio del problema) y diseño.
- Casos de uso (diagramas y textos).
- Los diagramas de secuencia.
- Diagramas de robustez.

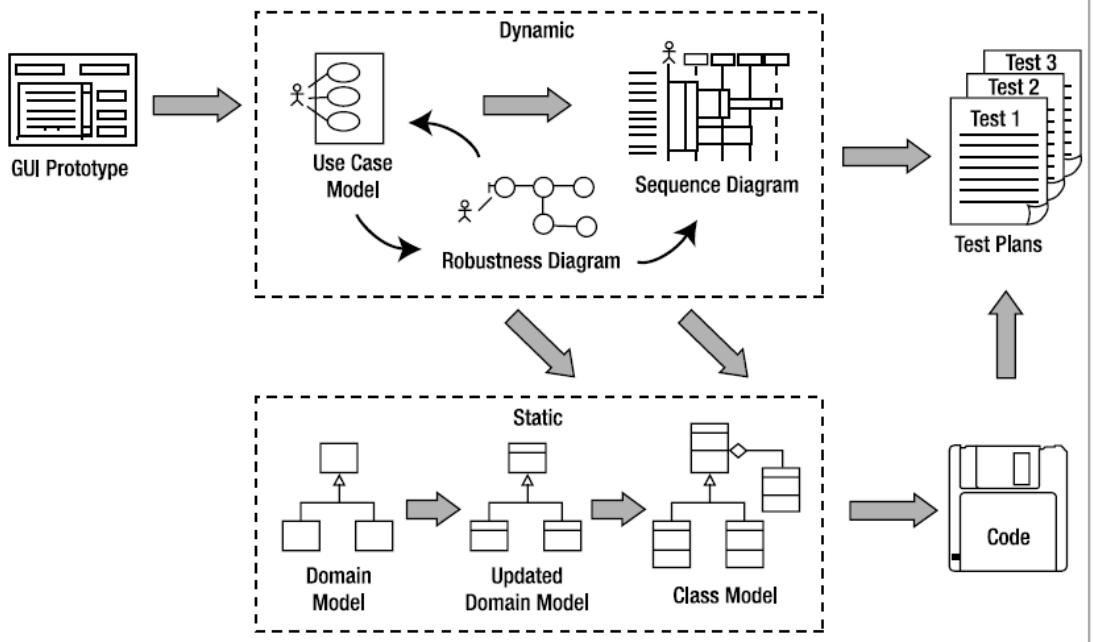
Esta metodología no abarca de manera directa y a detalle las cuestiones de alto nivel, es decir, la organización del equipo de trabajo. Un sistema que cumpla con los requerimientos es lo que trata el proceso ICONIX.

ICONIX difiere de las metodologías pesadas en que no exige mucha documentación, y se centra más bien en conseguir el código fuente tan pronto como sea posible.

En resumen, el proceso ICONIX describe cómo ir de los casos de uso al código de forma fiable, en el menor número de pasos posible. La figura 1 muestra cómo encajan las diferentes actividades en el proceso.

- ✓ Paso 1: Identifique sus objetos de dominio del mundo real (modelo de dominio).
- ✓ Paso 2: Definir los requisitos de comportamiento (casos de uso).
- ✓ Paso 3: Realizar un análisis de robustez para eliminar la ambigüedad de los casos de uso e identificar brechas en el modelo de dominio.
- ✓ Paso 4: Asignar el comportamiento de los objetos (diagramas de secuencia).
- ✓ Paso 5: Finalizar el modelo estático (diagrama de clases).
- ✓ Paso 6: Escribir / generar el código (código fuente).
- ✓ Paso 7: Pruebas de desempeño y aceptación de usuario. (Rosenberg D., Stephens M. & Collins-Cope M., 2005, páginas de la 1 a la 45).

Figura 1. Proceso ICONIX



Fuente: (Rosenberg D., Stephens M., & Collins-Cope M., 2005)

2.1.1 Fases de la metodología ICONIX.

La metodología ICONIX no impone ningún tipo de restricción en cómo está organizado un proyecto en cuanto a sus fases. Lo que la metodología si exige es que previo a la codificación del proyecto exista un análisis de los requisitos y un trabajo de diseño. Sin embargo, propone de manera general las siguientes fases:

- Modelado: análisis y diseño
- Planificación: inicialización
- Planificación: construcción

2.1.1.1 Modelado: análisis y diseño.

En esta fase se encuentra la parte principal de la metodología ICONIX que se divide en tres sub fases:

- **Análisis** esta fase consiste en identificar los objetos del mundo real con los diagramas de *modelo de dominio* y definir el comportamiento de los requerimientos con los diagramas de *casos de uso*.
- Una vez realizado el análisis el siguiente **paso es el intermedio entre** el **análisis** y el **diseño**, el cual radica en desambiguar he identificar vacíos en los casos de uso con los *diagramas de robustez*, para a continuación elaborar un diseño limpio
- **Diseño** en esta fase se ubica el comportamiento a nuestros objetos por medio de los *diagramas de secuencia* y se finaliza el modelo estático es decir los *diagramas de clases*, seguidamente se puede proceder a la codificación y pruebas.

2.1.1.2 Planificación: inicialización.

Esta es la fase en la que el equipo empieza tanto a *explorar los requerimientos, casos de uso* como a *familiarizarse con las tecnologías* que se utilizarán para el desarrollo del producto y se realiza *la elaboración del prototipo del sistema*. Esta fase se ejecuta una sola vez para todo el proyecto.

2.1.1.3 Planificación: construcción.

Esta fase se la realiza durante el resto del proyecto junto con el control de versiones. Esta fase consiste en pequeñas series de lanzamientos. El *mantenimiento* se considera parte de esta fase puesto que desde la primera versión que llega al cliente el producto entra en mantenimiento.

La planificación de la construcción consiste en determinar cuáles casos de uso que van a ir en que lanzamientos.

2.1.2 Características del desarrollo del proceso ICONIX.

La planificación del desarrollo del proceso ICONIX es incremental, iterativo he impulsado por modelos.

- Es **incremental,** puesto que cada nueva versión se construye progresivamente sobre la versión anterior
- Es **iterativo,** ya que cada versión se divide en pequeñas iteraciones.
- Es **impulsado por modelos,** ya que la planificación de desarrollo se basa en las tareas de ingeniería que resultan en los diagramas de los modelos.

2.1.3 El proceso ICONIX.

El proceso ICONIX describe un conjunto de pasos al estilo de un libro de recetas de cocina, como guía para ir de los casos de uso a la codificación. El proceso ICONIX evita también caer en procesos de análisis que impiden el arranque del desarrollo del producto, ya que el proceso ICONIX está en contra de los procesos que demandan grandes cantidades de estricta documentación detallada para poder pasar a la siguiente fase.

Durante el desarrollo del proceso es recomendable tener puntos de control que respaldan haber concluido con las tareas para poder pasar al siguiente paso y se presentan como los siguientes hitos.

- ✓ Hito 1: Revisión de los requerimientos
- ✓ Hito 2: Revisión del diseño preliminar
- ✓ Hito 3: Revisión detallada y crítica del diseño
- ✓ Hito 4: Entregable

ICONIX se centra en obtener a la codificación lo más pronto posible, sin perder los beneficios de un análisis y diseño inicial previo.

Por medio de 7 pasos los cuales constan de modelos que son las herramientas o técnicas y diagramas o esquemas los cuales son los instrumentos que resultan de aplicar las herramientas. Como son:

2.1.3.1 Paso 1: Identificación de los objetos de dominio del mundo real.

El primer paso consiste en el *modelado de dominio* y es de gran importancia para todo el proyecto, ya que es la primera fuente de inspiración para el diseño de los objetos software y por sus características que son:

- Forma una base sólida sobre la cual se levanta el resto del proyecto.
- Proporciona un vocabulario común para todo el equipo de desarrollo.
- A diferencia de otras metodologías el modelado de dominio se lo realiza antes de los casos de uso por que proporciona un vocabulario común, evitando así una nomenclatura ambigua como lo es el uso de varios términos para una misma cosa.
- En base al modelo de dominio se crea el diagrama de clases.
- Las clases en el modelo de dominio no muestran atributos ni métodos, puesto a que es un diagrama a nivel conceptual no a nivel de implementación, es decir, este tipo de clases no son software, sino más bien representan la abstracción de los conceptos concernientes al problema en cuestión.
- Es una representación de los objetos y las operaciones que forman el negocio.
- El primer modelo nunca es perfecto por lo que evoluciona en la misma medida en que avanza la comprensión del dominio del problema.

Las recomendaciones para elaborar el modelo de dominio son:

- Revisar la documentación relacionada como los *requerimientos*, glosarios, entrevistas, entre otros. Dentro de lo cual por lo general: Los sustantivos se vuelven en las clases y atributos.

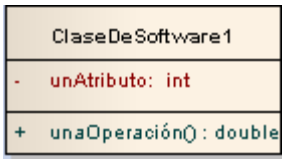
    - Los verbos y acciones se convierten en operaciones y asociaciones.
    - Las oraciones posesivas alertan de que una posible clase en realidad es un atributo.

- Determinar las relaciones de *generalización* y *agregación* entre los objetos de dominio del mundo real.
- En esta fase se recomienda elaborar la primera referencia visual del sistema, es decir, los prototipos.

Clase y objetos

Una clase es la representación de un conjunto de cosas u *objetos* que tienen las mismas características, son del mundo real y se distinguen de los demás objetos. Una clase tiene atributos y métodos. Como una persona, cuenta bancaria, casa, auto. Por ejemplo la clase Vehículo representa el conjunto de Vehículo 1, Vehículo 2, Vehículo n donde cada Vehículo n es un objeto.

Figura 2. Estructura de una clase



Elaborado por: Verónica Sangucho

Relaciones de generalización y agregación

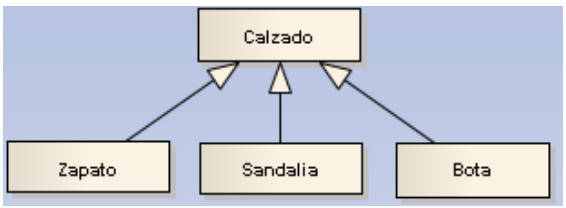
Dentro del diagrama de modelo de dominio las relaciones entre clases juegan un papel muy importante las cuales son:

- **Generalización**

La generalización se presenta cuando dos clases tienen partes en común, para lo cual se crea una tercera clase llamada clase *padre* que contiene las características en común de las clases *hijas* y donde las clases hijas heredan los atributos de la clase padre.

En la *especialización*, de una clase padre se crean clases hijas de las cuales se destacan sus diferencias.

Figura 3. Ejemplo de generalización



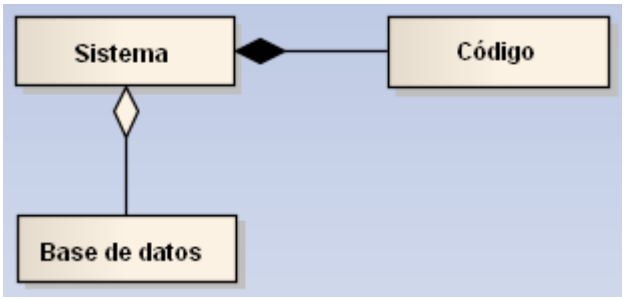
Elaborado por: Verónica Sangucho

- **Agregación**

Agregación o contención es una relación donde los objetos de una clase son compuestos por objetos de otra. Donde los rombos van en el objeto que posee las referencias.

Cuando la asociación es muy fuerte, es decir, ambos objetos dependen del otro para poder existir, como por ejemplo el objeto libro que está compuesto de los objetos capítulos, esta relación se denomina *composición* y se representa con un rombo negro.

Figura 4. Ejemplo de agregación



Elaborado por: Verónica Sangucho

Prototipos

Al igual que un arquitecto desarrolla la maquetación del edificio antes de su construcción, los arquitectos de información utilizan los diagramas y prototipos para presentar la organización, estructura, navegación y funcionamiento de la aplicación web antes del desarrollo.

Algo importante que hay que aclarar a los clientes es que los prototipos no contienen diseño gráfico, por lo que deben ser elaborados en escala de grises.

La importancia de realizar un prototipo del sistema antes de su desarrollo radica en que los involucrados se centran en el diseño de contenidos e interacción y no en el diseño visual, es mucho más comprensible para el cliente, y es un elemento de gran valor en el análisis, se modifica con facilidad y rapidez por lo que se evitan posteriores modificaciones que son costosas por estar en etapa de implementación (Carreras, 2012).

2.1.3.2 Paso 2: Definición del comportamiento de los requerimientos.

El segundo paso consiste en definir el comportamiento de los requerimientos con los **casos de uso** para el sistema.

Tipos de requerimientos

Los requerimientos son las capacidades que deben estar presentes en un sistema y para responder a una necesidad, resolver un problema o alcanzar un objetivo. Entre los requerimientos del sistema pueden dividirse en dos tipos los funcionales y los no funcionales

- **Funcionales:** Son aquellos que describen lo que el sistema **debe** hacer, están más apegados al modelo de negocio del cliente. Ejemplo: Sacar un reporte de clientes, registrar los datos del producto.
- **No funcionales:** Son aquellos que describen **cómo** el sistema debe responder. Ejemplo: Sacar el reporte de clientes en orden alfabético en menos de un segundo, el sistema debe ofrecer facilidad en su mantenimiento, el sistema debe desarrollarse bajo determinados estándares.

Relación entre los requerimientos y los casos de uso

Entre estos términos puede existir cierta ambigüedad, ya que se podría afirmar que un caso de uso es un requerimiento, para distinguirlos y aclarar la relación entre ambos a continuación se presentan sus características:

Casos de uso

- ✓ Cada caso de uso describe una unidad de comportamiento.
- ✓ Cada caso de uso puede satisfacer uno o más requerimientos funcionales.

Requerimientos

- ✓ Cada requerimiento describe una ley que gobierna un comportamiento.
- ✓ Cada requerimiento puede ser satisfecho por uno o más casos de uso (Rosenberg, Stephens & Collins-Cope, 2005, pág 43).

Casos de uso

Con el fin de especificar el comportamiento de un sistema la técnica "caso de uso" describe el conjunto pasos secuenciales que un actor ejecuta al interactuar con un sistema mientras utiliza alguno de sus servicios. Los casos de uso son utilizados para la captura de requerimientos de un nuevo sistema o para la actualización de un sistema existente.

Para lograr una mejor comprensión por parte del usuario se encuentra más eficaz nombrar a los casos de uso en tiempo presente y en voz activa. Por ejemplo en un una página web para una clínica un caso de uso se podría llamar "Registrar paciente", "Visualizar historial médico", "Buscar informe médico", "Generar informe".

Un caso debe describir sin ambigüedades una funcionalidad del sistema incluyendo el curso básico y los cursos alternativos, por esta razón un caso de uso es más detallado que un diagrama de caso de uso, sin embargo un caso de uso no debe describir aspectos de diseño o de implementación. El conjunto resultante del modelado de los casos de uso debe describir toda la funcionalidad del sistema.

Diagramas de casos de uso

Los diagramas de casos de uso muestran la relación entre los actores y los casos de uso en un sistema. Representan en forma gráfica los valores que el sistema debe producir para un actor como el generar un reporte o calcular un resultado. Los diagramas deben ser claros, simples y concisos.

Los elementos de un diagrama de casos de uso son:

- **Actores:** Son personas o entidades externas al sistema como otro sistema o una entidad abstracta como el tiempo, las cuales demandan una funcionalidad.
- **Casos de uso:** Es la funcionalidad que es demandada al sistema.
- **Relaciones:** Son las conexiones entre los elementos del modelo. Dentro de las cuales tenemos las relaciones "precedes" e "invoques". Donde la relación "*precedes*" denota que un caso de uso debe preceder y completarse antes que otro puesto que contiene alguna funcionalidad o información necesaria para el siguiente caso de uso formado así una secuencia lógica en el diagrama.

Y la relación "*invokes*" se utiliza cuando un caso de uso llama a otro caso de uso para poder completarse, de la misma forma que una función principal llama a una subfunción.

2.1.3.3 Paso 3: Eliminar la ambigüedad de los casos de uso.

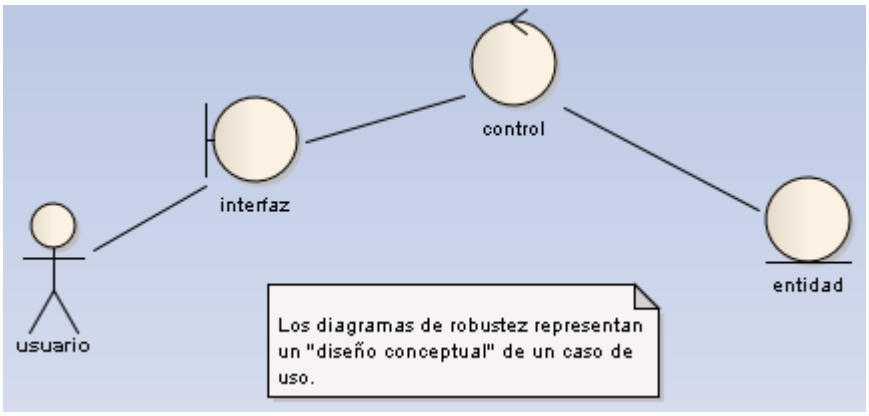
Este tercer paso sirve para pasar del análisis al diseño, consiste en identificar el conjunto de objetos que participan en un caso de uso, al realizar un análisis de robustez o solidez con el fin de eliminar ambigüedades o descubrir objetos aún no identificados, añadirlos al diagrama de clases y actualizar los casos de uso de ser necesario.

El análisis de robustez radica en esbozar la primera aproximación al diseño del software para que un caso de uso pueda ser implementado.

Los tipos de objetos utilizados en un diagrama de robustez representan el patrón de diseño MVC los cuales son:

- **Objetos de interfaz:** Límite los cuales son utilizados por los usuarios para interactuar con el sistema y constituyen la vista o capa presentación.
- **Objetos de entidad:** Suelen ser los objetos del modelo de dominio y constituyen el modelo o capa objetos de negocio es decir la base de datos.
- **Objetos de control:** Son utilizados como el enlace entre los objetos de interfaz y los de entidad y constituyen el controlador del patrón de diseño MVC los cuales se ocupan de los eventos de la interfaz de usuario.

Figura 5. Notación en el diagrama de robustez



Elaborado por: Verónica Sangucho

2.1.3.4 Paso 4: Asignación del comportamiento de los objetos.

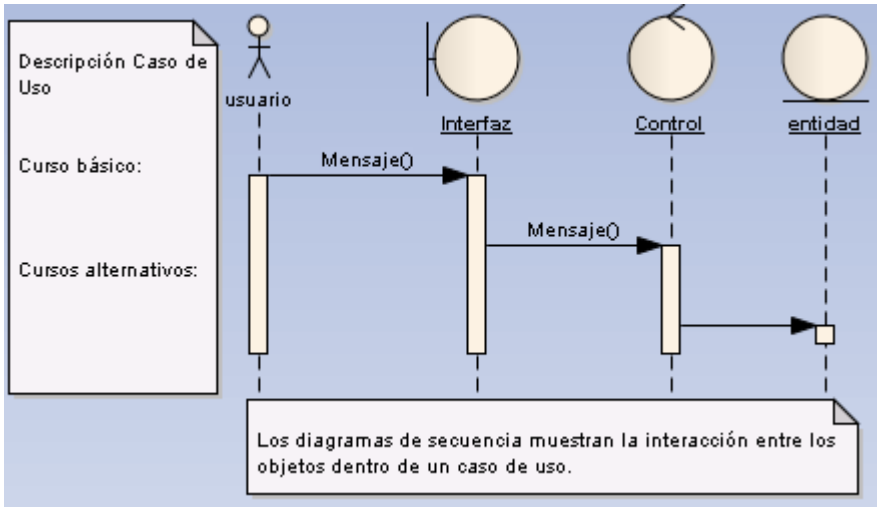
Este paso inicia la fase de diseño. La asignación del comportamiento de los objetos de una aplicación consiste en modelar la interacción que existe entre los objetos en el transcurso del tiempo. Mientras que en el paso anterior los casos de uso están a un nivel conceptual con la descripción de un escenario, en este paso se desarrollan los **diagramas de secuencia** los cuales contienen detalles de implementación de un caso.

La metodología ICONIX dice que es recomendable desarrollar los diagramas de secuencia para cada caso de uso con la descripción textual del caso a la izquierda y el diseño del diagrama de secuencia al lado derecho.

Un diagrama está formado de:

- **Objetos:** De entidad, control y límite con sus líneas de vida.
- **Mensajes:** Son de intercambio entre los objetos, con un rectángulo y flechas que pueden ser sincrónicos identificados por una flecha oscura, asíncronos denotados por una flecha en línea o un mensaje de retorno asíncrono de línea punteada.
- **Línea de vida:** Representa un actor individual en un diagrama de secuencia.
- **Tiempo:** El tiempo es representado en forma vertical en el diagrama he inicia en la parte superior.

Figura 6. Notación en el diagrama de secuencia



Elaborado por: Verónica Sangucho

2.1.3.5 Paso 5: Finalización del modelo estático.

Gracias a la identificación de los elementos del sistema por medio del análisis y la actualización de estos modelos con los elementos que surgieron para estructurar el software, con este paso el proceso ICONIX lleva al diseño a la parte de modelar la vista de los elementos estáticos esenciales de la aplicación con los **diagramas de clase**.

Los diagramas de clase son desarrollados en base a los diagramas de secuencia. La diferencia radica en que se modelan los elementos en forma independiente del transcurso del tiempo. Por esta razón se denomina al diagrama de clases como modelo estático de la estructura del sistema.

Los diagramas de clase son también una versión avanzada y más detallada del modelo conceptual de dominio puesto que muestran a las clases con sus métodos y atributos. ICONIX nos dice que el exceso de detalles en los diagramas de clases puede influir negativamente en la etapa de diseño.

2.1.3.6 Paso 6: Generación del código fuente.

ICONIX advierte que una vez definida la estructura general del software no significa que en adelante se debe realizar una tarea por completo mecánica, sino que el programador debe pensar en el diseño a nivel de código. También recomienda que los mismos diseñadores sean los programadores.

Este paso se conforma de los siguientes pasos:

- Escribir o generar el código fuente.
- Realizar pruebas unitarias y pruebas de integración.

2.1.3.7 Paso 7: Pruebas al sistema y pruebas de aceptación.

Para este paso ICONIX aconseja que el equipo de control de calidad sea independiente del equipo de programación, y que las pruebas de aceptación sean realizadas por el usuario final con los casos de uso.

2.2 Patrón de diseño Modelo Vista Controlador (MVC)

Dentro de las aplicaciones en las que no existe separación entre el código para el acceso a datos, el código para la lógica del modelo de negocio y código para la parte de presentación, es decir, contienen la mezcla de los tres, se pueden presentar problemas.

En dichas aplicaciones surgen dificultades al momento de su mantenimiento, debido a la alta interdependencia entre los componentes, ya que un cambio en cualquiera de las secciones afecta de manera directa a todo el proyecto. También causa problemas al momento de intentar reutilizar el código debido a que dependen de muchas otras clases.

El patrón de diseño modelo-vista-controlador afronta estos inconvenientes por medio de la separación del código fuente en tres capas, la capa de acceso a datos, la capa de lógica de negocio, y la capa de presentación de datos he interacción con el usuario (Sun Microsystems, 2002).

Figura 7. Representación del patrón Model-View-Controller

Elaborado por: Verónica Sangucho

2.3 Estándares para el desarrollo del software

Los presentes estándares y políticas tienen como objetivo mejorar la calidad de los resultados del desarrollo. Su definición en el inicio del proyecto tiene un alto nivel de incidencia en la fase de mantenimiento del sistema por las siguientes razones:

- Los lineamientos de codificación contribuyen en la legibilidad y comprensión del código fuente de una manera más rápida y mejor.
- El mantenimiento es una de las actividades más comunes en el ciclo de vida del sistema.
- La tarea de mantenimiento no siempre la realizan los mismos autores que desarrollaron el sistema.

Para lograr que el producto permita la reutilización de código, flexibilidad, y la incorporación de módulos en el futuro, en base al patrón de diseño MVC se siguieron los siguientes estándares para la estructura arquitectónica general del presente proyecto web con JSF.

2.3.1 Frameworks tecnológicos utilizados.

- **JSF 2.0** como framework para el desarrollo arquitectónico web de interfaces de usuario en aplicaciones web java y **Mojarra 2.0** como su implementación de referencia.
- **Richfaces** como librería principal de componentes de interfaz de usuario.
- **Primefaces** como librería de componentes alternativo.
- **Eclipse Spring Tool Suite** como IDE principal.
- **JBOSS 6.0** como servidor de la aplicación en desarrollo y **JBOSS 7** como servidor de la aplicación en producción.
- **Hibernate** como herramienta para el mapeo objeto relacional para Java.
- **Spring** como framework para el desarrollo del software.
- **PMD** como herramienta de revisión y mejora de código estático.
- **Checkstyle** como herramienta de análisis de código estático.
- **VisualSVN SERVER 2.6.4 Standard Edition, ECLIPSE TIGRIS**
- **Jars** las principales librerías para el funcionamiento del sistema son:

Tabla 3. Jars que usa el sistema

Librería	Descripción
commons-codec-1.6	Codificador para encriptar y desencriptar las contraseñas.
hibernate3	Contiene las clases necesarias para el uso de hibernate para la persistencia y recuperación de los datos.
log4j	Guarda los detalles cuando se producen errores o información en la aplicación en tiempo de ejecución, para depurar el código o el rastreo de fallas del sistema
org.springframework.core -3.1.0.RELEASE	Las librerías Spring brindan el soporte para la infraestructura del sistema como, la inyección de dependencias; al suministrar a una clase los objetos que necesite, evitando que sea la misma clase quien cree el objeto, con el fin de optimizar los recursos
postgresql-9.3-1100-jdbc41	Java Database Connectivity (JDBC), sirve para la conexión con la base de datos
cssparser-0.9.5	Contiene un conjunto de clases java usado para agregar información referente al estilo o apariencia de la página
javax.faces-2.1.10	Posee el conjunto de funciones y procedimientos o Application Programming Ingerface (API), de java server faces (JSF)
primefaces-4.0	Aquí se encuentran los fuentes o api de los componentes de Primefaces que ocupa la aplicación
junit-4.11	Es el conjunto de clases para realizar las pruebas de Caja Blanca del sistema en tiempo de ejecución

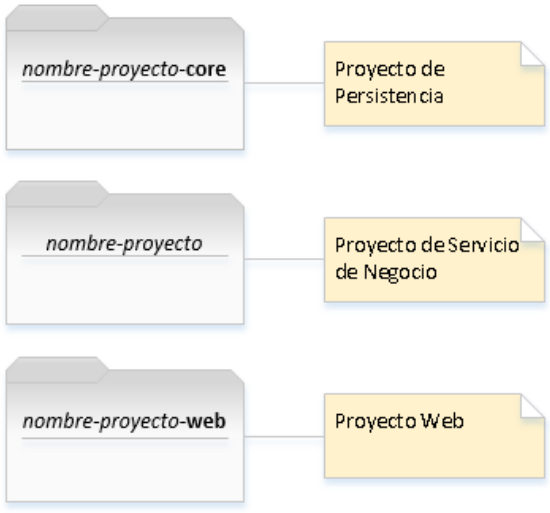
Elaborado por: Verónica Sangucho

En la siguiente sección se presenta el formato a seguir para la nomenclatura de los proyectos y como deben estar fundamentalmente organizados los paquetes y el código fuente del software

2.3.2 Convención para la nomenclatura de los proyectos.

Las convenciones para el nombrado de los proyectos Java tienen como objetivo brindar información sobre el tipo de proyecto que ha sido implementado. En la figura 8 se detalla los tipos de proyectos a implementarse para el presente producto y su formato.

Figura 8. Nomenclatura para los proyectos según su tipo



Elaborado por: Verónica Sangucho

Se debe tomar en cuenta que los nombres de los proyectos deben estar completamente en minúsculas.

2.3.3 Nomenclatura de los paquetes de los proyectos.

El formato de convención para la nomenclatura de los paquetes debe hacer referencia al sistema y su funcionalidad. Entonces el formato estándar de nomenclatura para los paquetes será:

{prefijo_pais}.{tipo_organización}.{id_organizacion}.{prefijo_sistema}.{(sub)modulo/funcionalidad}...

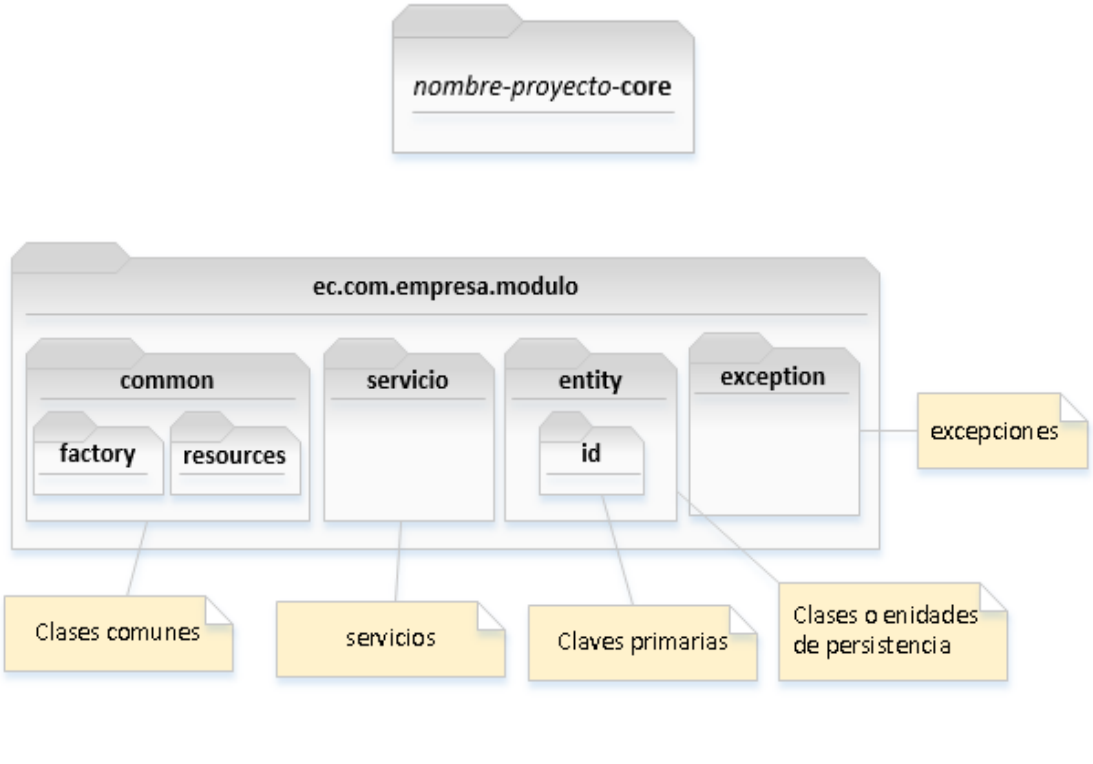
El nombre resultante del paquete dependerá tener; en el primer nivel el país y en el segundo nivel el tipo de organización que se esté implementando en el sistema, como puede ser : com, edu, gov, mil, net, org.

Ejemplo: ec.com.dtv.legalizaciones

2.3.4 Proyecto de persistencia.

La persistencia fue desarrollada con la tecnología hibernate con la siguiente nomenclatura.

Figura 9. Nomenclatura de los paquetes del proyecto persistencia



Elaborado por: Verónica Sangucho

Nomenclatura de las clases dentro del paquete: ec.com.empresa.modulo.entity

Este paquete contiene las clases de persistencia que representan las entidades de la base de datos. Se debe declarar a la clase con un nombre que describa la tabla que representa.

Por ejemplo la clase java de persistencia para la tabla TBL_CITAMEDICA.

La clase debe ser declarada de la siguiente manera:

```
public class CitaMedica implements Serializable{
```

Se debe tomar en cuenta que el nombre de las clases debe tener la primera letra en mayúscula.

Nomenclatura de las interfaces dentro del paquete:

ec.com.empresa.modulo.servicio

Las clases en este paquete contienen las interfaces de los servicios donde se declaran las firmas de los métodos. La declaración a la clase debe ser así:

I*NombreNegocio***Servicio**

Por ejemplo para los servicios para CitaMedica la declaración será:

```
interface ICitaMedicaServicio {
```

Nomenclatura de las clases dentro del paquete:

ec.com.empresa.modulo.common.factory

En este paquete se encuentran las clases que definen la interfaz de creación del tipo de objeto para la inyección de dependencia. La nomenclatura es la siguiente:

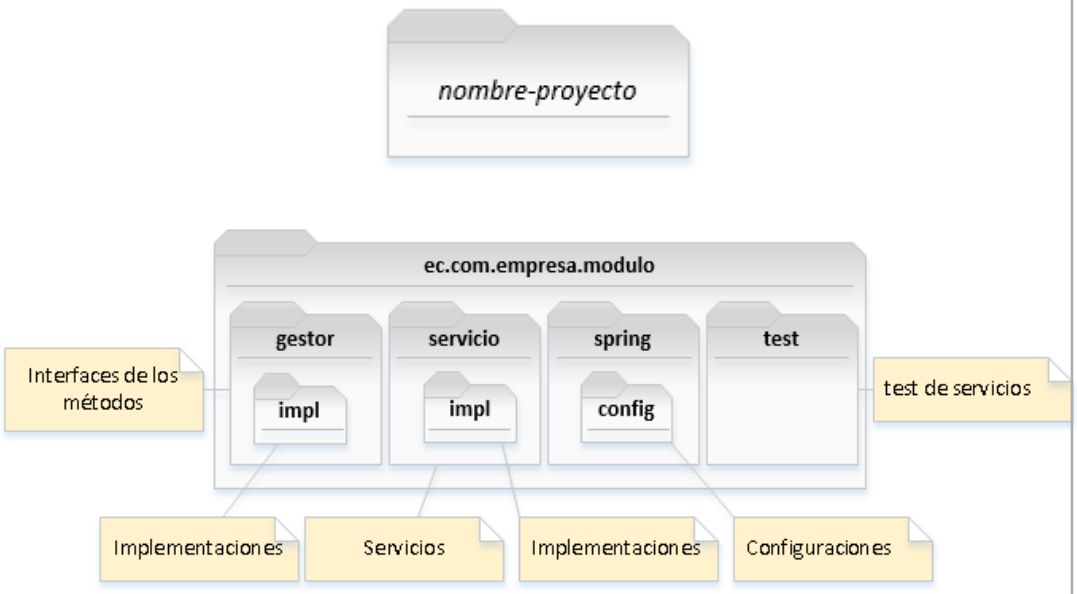
*NombreClase***Factory**

Por ejemplo la clase Factory para CitaMedica la declaración será:

```
public class CitaMedicaFactory {
```

2.3.5 Proyecto de servicios de negocio.

Figura 10. Nomenclatura de los paquetes el proyecto de servicio de negocio



Elaborado por: Verónica Sangucho

Nomenclatura de las interfaces dentro del paquete: ec.com.empresa.modulo.gestor

Aquí se ubican las interfaces o firmas de los métodos según su función. El formato para la declarar el nombre es:

*NombreInterface***Gestor**

Ejemplo:

```
interface CitaMedicaGestor {
```

Nomenclatura de las clases dentro del paquete: ec.com.empresa.modulo.gestor.impl

En este paquete están las implementaciones de los métodos declarados en el paquete anterior. El formato para nombrar las clases es:

*NombreInterface***GestorImpl**

Ejemplo: `public class CitaMedicaGestorImpl {`

Nomenclatura de las interfaces dentro del paquete:
ec.com.empresa.modulo.servicio

Estas son interfaces de los servicios y deben ser nombradas de acuerdo al negocio. El formato es:

*NombreInterface***Servicio**

Por ejemplo la interfaz para los servicios para CitaMedica será:

```
interface CitaMedicaServicio {
```

Nomenclatura de las clases dentro del paquete:
ec.com.empresa.modulo.servicio.impl

En estas clases se implementan los métodos de los servicios. El formato del nombre debe describir el negocio añadiendo la terminación "ServicioImpl" así:

*NombreClase***ServicioImpl**

Ejemplo:

```
public class CitaMedicaServicioImpl {
```

Nomenclatura de los archivos dentro del paquete:
ec.com.empresa.modulo.spring.config

Dentro de este paquete se encuentran archivos XML de configuraciones del spring. El formato para el nombre de los archivos es:

Negocio*ProyectoModulo***Beans**

Ejemplo:
AutorizacionesAdministracionBeans

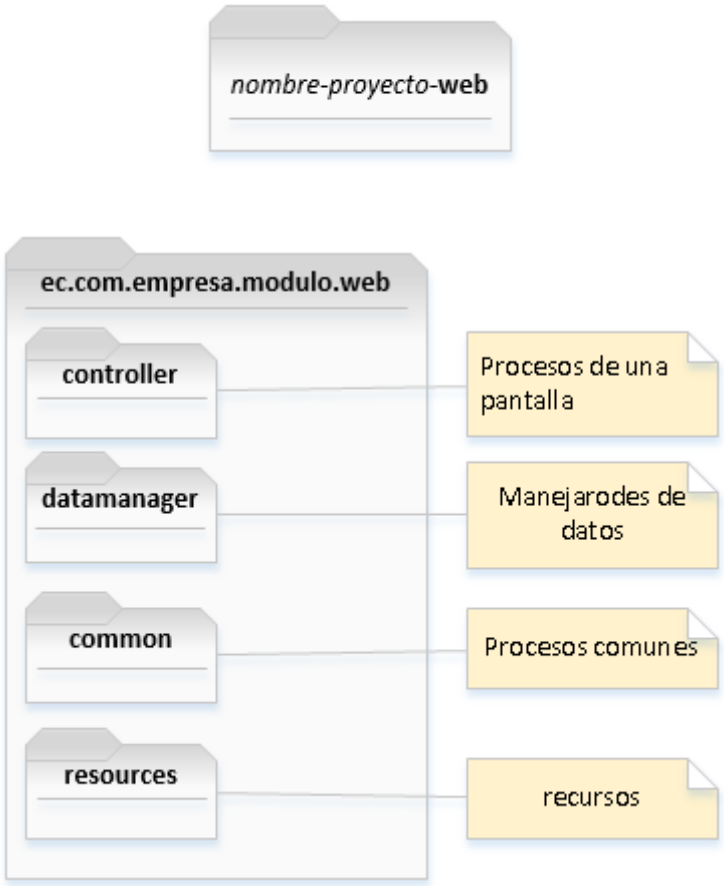
Nomenclatura de las clases dentro del paquete: ec.com.empresa.modulo.test

Las clases de este paquete permiten verificar el funcionamiento de los métodos. Generalmente este paquete no se entrega al cliente puesto que solo sirven para realizar pruebas. Su formato es: *NombreClase***FactoryTest**

2.3.6 Proyecto web.

El proyecto web fue desarrollado con la tecnología JSF y HTML5 y con la siguiente nomenclatura

Figura 11. Nomenclatura para los paquetes el proyecto web



Elaborado por: Verónica Sangucho

Nomenclatura de las clases dentro del paquete: ec.com.empresa.modulo.controller

Estas clases contienen la funcionalidad a las que el cliente accede directamente. El nombre debe ser descriptivo según su funcionalidad y deberá terminar con "Controller" así:

*NombreClase***Controller**

Por ejemplo:

```
public class BuscarCitaMedicaController extends Commons {
```

Nomenclatura de las clases dentro del paquete:

ec.com.empresa.modulo.datamanager

En las clases datamanager o manejadores de datos se encuentran las variables requeridas para la funcionalidad de la pantalla.

Su nombre debe ser descriptivo y terminado en "DataManager" así:

*NombreClase***DataManager**

Por ejemplo para la clase anterior su data manager será:

```
public class BuscarCitaMedicaDataManager extends Commons {
```

Nomenclatura de las clases dentro del paquete: ec.com.empresa.modulo.common

En esta carpeta se encuentran las clases que contienen funcionalidades genéricas para métodos que se utilizarán en varias pantallas.

El formato para la nomenclatura es:

*NombreClase***Controller**

Nomenclatura de las clases dentro del paquete:

ec.com.empresa.modulo.test

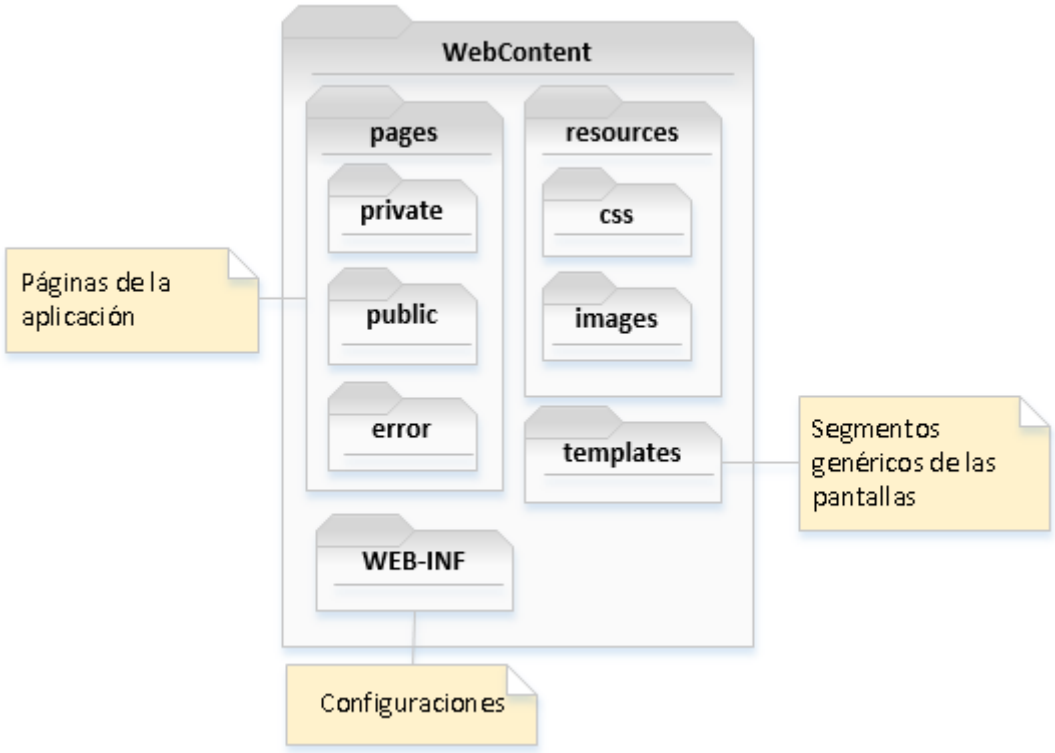
Las clases de este paquete sirven para realizar pruebas del funcionamiento de los métodos.

Su nombre debe ser descriptivo y su formato es:

*NombreClase***Test**

Por lo general este paquete es eliminado puesto que es una ayuda para el programador durante el desarrollo del producto pero no tiene ninguna funcionalidad al concluir el proyecto.

Figura 12. Nomenclatura para el contenido web según su funcionalidad



Elaborado por: Verónica Sangucho

Nomenclatura de las páginas de la carpeta: webContent/pages

Aquí se ubican las páginas a las cuales se puede ingresar con a la autenticación del usuario dentro de la carpeta *private*. El nombre debe estar minúsculas y con la primera letra de cada palabra interna en mayúsculas. Las páginas deberán estar organizadas dentro de subcarpetas según su funcionalidad, por ejemplo:

Figura 13. Ejemplo de la organización de las páginas xhtml

Elaborado por: Verónica Sangucho

Del mismo modo se pueden nombrar y organizar las páginas que se encuentran en la carpeta *public*, aquí se agrupan las páginas que no necesitan de la autenticación del usuario. Por ejemplo: pantallaGeneral.xhtml.

Nomenclatura de los archivos de la carpeta: webContent/resources

En está capeta se ubicarán los estilos e imágenes definidas y necesarias para la página web en las carpetas webContent/resources/css y webContent/resources/images respectivamente, los nombres deben ser descriptivos empezando con minúsculas y las palabras internas la primera letra con mayúscula.

Nomenclatura de los archivos de la carpeta: webContent/resources

En esta carpeta se guardan los archivos de recursos propiedades o porperties para la presentación de los mensajes o etiquetas de las pantallas. Los nombres de estos archivos según corresponda deberán ser:

- errors.properties
- labels.properties
- messages.properties
- faces.properties

En el archive faces.properties se especificarán formatos de fechas y zona horaria.

La declaración para las etiquetas o mensajes deberán estar en minúsculas y las palabras internas separadas con un punto y con el signo igual alineado, por ejemplo para las etiquetas de los botones dentro del archivo labels.properties deben estar declarados de la siguiente manera:

- `salir            = Salir`
- `aceptar          = Aceptar`
- `sistema          = Sistema`
- `cancelar.pedido  = Cancelar Pedido`

2.3.7 Estándares de la nomenclatura para la base de datos.

Para estandarizar la nomenclatura utilizada en el diseño de la base de datos se seguirán las siguientes normas:

Generalidades

Los nombres deben:

- Estar compuestos sólo de letras.
- Deben tener un largo máximo de 16 caracteres.
- Se puede utilizar underscore "_" para separar los sufijos.

Nombre de la base de datos

El nombre de la base de datos debe ser descriptivo y nombrado con letras en mayúsculas para que se diferencie de las bases de datos del sistema PostgreSQL que están nombradas en minúsculas.

Adicionalmente se pueden usar underscore "_" para separar el sufijo "BD".

Ejemplo:

*MARKETING*_DB

*AGENCIATURISMO*_DB

Nomenclatura de los objetos de la base de datos

Nombres de las tablas

El nombre de las tablas debe ser:

- Descriptivo y fácil de recordar
- En español
- En singular
- Sin espacios en blanco
- Deben tener el sufijo "_tbl"

- Se debe emplear el estándar “Pascal Case”, el cual dice que el nombre debe empezar con la primera letra mayúscula y las palabras subsecuentes con la primera letra en mayúscula.
- Si el nombre requiere ser compuesto solo la última palabra debe ir en plural. Por ejemplo: TipoEspecialidad_tbl es correcto mientras que Tipo**s**Especialidades_tbl **NO** es correcto.

Nombres de las columnas

Los nombres las columnas o campos deben ser:

- Descriptivos
- En español
- En singular
- No se debe nombrar de diferentes maneras a campos que representen lo mismo, por ejemplo: valor, precio, monto.
- Se debe usar el estándar “Camel Case” el cual indica que si el nombre del campo está formado por un conjunto de palabras, la primera letra de la primera palabra debe estar en minúsculas, y la primera letra de cada una de las palabras subsecuentes en mayúscula.
- No utilizar guion bajo para separar las palabras.
- Debe estar formado solo por letras.

Nombres de las claves primarias

Si el campo es clave primaria debe estar formado por el nombre de la tabla más el sufijo “_id”

2.4 Herramientas de revisión de código estático

2.4.1 PMD.

Programming Mistake Detector o PMD verifica que el código fuente siga un conjunto de reglas con el fin de lograr un código óptimo y que brinde facilidad en su mantenimiento.

En general PMD ubica en el código Java posibles problemas potenciales como:

- Posibles errores o bugs como sentencias *try* / *catch* / *finally* / *switch* vacías.
- Código muerto como *variables locales* / *parámetros* / *métodos privados* no utilizados.
- Optimización de código como *estructuras de bucles if inapropiadas* / mal uso de *String* / *StringBuffer* / *String Builder*.
- Demasiada complejidad de los métodos o expresiones como implementaciones con bucles *while / bucles if* innecesarios.
- Código duplicado sin ningún valor agregado.

Niveles de gravedad

Las violaciones de PMD tienen por defecto los siguientes niveles de gravedad:

Tabla 4. Niveles de gravedad de las violaciones de PMD

1. Error (alto)	
2. Error	
3. Advertencia (alta)	
4. Advertencia	
5. Información	

Elaborado por: Verónica Sangucho

Figura 14. Niveles de gravedad de las violaciones de PMD

Elaborado por: Verónica Sangucho

Estos niveles de gravedad sirven principalmente para entregar un producto totalmente libre de errores, con un número bajo de errores de niveles que se consideren admisibles.

Una vez instalado el pluning y configurada la herramienta PMD el tipo de reglas con las que va a trabajar son:

- **Basic** el conjunto de reglas básicas son la selección de buenas prácticas que deben ser seguidos por todos los programadores, como son:

  EmptyCatchBlock detecta casos en los que el manejo de excepciones solo captura el error pero no realiza acción alguna.

  Ejemplo:

```
public void doSomething() {
  try {
    FileInputStream fis = new FileInputStream("/tmp/bugger");
  } catch (IOException ioe) {
      // no está bien
  }
}
```

  UnnecessaryConversionTemporary

  Esta regla evita temporales innecesarios cuando se convierten datos de tipo primitivo a String.

  Ejemplo:

```
public String convert (int x) {

  / / Esto desperdicia un objeto
  . Cadena foo = new Integer (x) toString ();
  / / Esto es mejor
```

```
  Integer.toString return (x);
}
```

- **Braces** o llaves este conjunto de reglas aborda el tema del uso de las llaves en la estructura de los bucles en la codificación java, como son:

  IfElseStmtsMustUseBraces revisa que las sentencias *if else* estén con llaves.

  Ejemplo:

```
public void doSomething() {
   if (foo)
       x=x+1;
   else
       x=x-1;
 }
```

  ForLoopsMustUseBraces revisa que las sentencias *for* estén con llaves.

  Ejemplo:

```
public void foo() {
 for (int i=0; i<42;i++)
   foo();
}
```

- **Code Size** o tamaño del código abarca las reglas que regulan el tamaño del código, puesto que mientras más grande es el tamaño aumenta también la complejidad del código, como son:

  NPathComplexity evalúa la complejidad de un método dependiendo del número de caminos de ejecución y le asigna un valor. Si el valor sobrepasa los 200 es señal de tomar medidas para reducir la complejidad.

  ExcessiveMethodLength esta regla indica cuando un método es demasiado grande, y el cual debe tratarse de reducir.

  Ejemplo:

```
public class Foo {
 doSomething public void () {
  System.out.println ("Hola mundo");
  System.out.println ("Hola mundo");
  / / 98 copias omiten por razones de brevedad.
 }
}
```

- **Coupling** o acoplamiento estas reglas encuentran casos de inadecuado acoplamiento entre los objetos y los paquetes, como son:

CouplingBetweenObjects en base al análisis de la cantidad de atributos únicos, variables locales y tipos de retorno dentro de un objeto asigna un número. Si el número se encuentra sobre 100 es indicio un número muy alto de acoplamiento.
ExcessiveImports si existe un alto número de importaciones puede alertar sobre un alto grado de acoplamiento. El número de importaciones límite es determinado por el usuario.

- **Design** este conjunto de reglas abordan los temas de diseño cuestionable, como son:

  UseSingleton si una clase tiene solo métodos estáticos, se considera que es un Singleton. Esta regla no aplica a clases de tipo abstractas las cuales contienen las firmas de los métodos.

  Ejemplo:

```
public class MaybeASingleton {
 public static void foo () {}
 público bares static void () {}
}
```

  SimplifyBooleanReturns detecta declaraciones innecesarias if then cuando se devuelve un booleano.

  Ejemplo:

```
public class Foo {
  bar private int = 2;
  public boolean isBarEqualsTo (int x) {
    / / Este fragmento de código
    if (bar == x) {
     return true;
    } Else {
     return false;
    }
    / / Se puede sustituir por una simple
    / / Devuelve bar == x;
  }
}
```

- **Import Statement** estas reglas manejan las cuestiones relacionadas con los problemas que pueden surgir con las declaraciones *import* en una clase como son:

  DontImportJavaLang se debe evitar importar elementos del paquete "*java.lang*" porque estos se importan automáticamente.

Ejemplo:

```
// this is bad
import java.lang.String;
```

UnusedImports se debe evitar declarar imports inutilizados.

- **Logging** este conjunto de reglas analizan el uso del framework de logging, como son:

  MoreThanOneLogger se recomienda usar un solo logger por clase.

  Ejemplo:

```
clase Foo {
    Logger log = Logger.getLogger (Foo.class.getName ());
    / / Es muy raro ver a dos madereros en una clase
    / Información / log es multiplexado por niveles
    Logger log2 = Logger.getLogger (Foo.class.getName ());
}
```

  LoggerIsNotStaticFinal mayoritariamente el logger puede ser declarado estático y final.

  Ejemplo:

```
clase Foo {
    Logger log = Logger.getLogger (Foo.class.getName ());
    / / Es mejor declarar el registrador así:
    / / Static final Logger log = Logger.getLogger
(Foo.class.getName ());
}
```

- **Naming** contiene las reglas relativas a la nomenclatura de los elementos respecto de su tamaño, como son:

  ShortVariable advierte cuando se usan nombres de variables muy cortos.

  Ejemplo:

```
public class Something {
  private int q = 15; // violación
  public static void main( String as[] ) {  // violación
    int r = 20 + q; // violación
    for (int i = 0; i < 10; i++) {//no es violación en un for
      r += q;
    }
  }
}
```

  ShortMethodName muestra cuando se usan nombres de métodos muy cortos.

  Ejemplo:

```
public class ShortMethod {
  public void a( int i ) { // violación
  }
}
```

- **Optimization** gestionan las reglas pertinentes a optimizaciones referentes a las mejores prácticas que contribuyen en general a la mejora del rendimiento, como son:

  LocalVariableCouldBeFinal cuando una variable local que requiere un solo valor asignado es conveniente declararlo como *final*.

  Ejemplo:

```
Bar clase pública {
 public void foo () {
  String a = "a" / / si no se asignarán otros  valores, es
mejor hacer esto:
  final String b = "b";
 }
}
```

  MethodArgumentCouldBeFinal si el parámetro de entrada de un método nunca se le reasigna el valor dentro del método puede ser declarado como *final*.

  Ejemplo:

```
public void foo (param String) {
  / / Hacer cosas con parámetro nunca asignándole otro valor
  / / Es mejor: public void foo (final String param) {
```

- **Strict Exception** establece lineamientos sobre el manejo de excepciones, como son:

  AvoidThrowingNullPointerException no es recomendable lanzar NullPointerException porque normalmente es la máquina virtual quien lo genera y lanzarlo generaría confusión.

  Ejemplo:

```
public class Foo {
 void bar () {
  throw new NullPointerException ();
 }
}
```

  AvoidCatchingGenericException no se debe capturar excepciones genéricas en bloques *try-catch.*

Ejemplo:

```
package com.igate.primitive;
public class PrimitiveType {
  public void downCastPrimitiveType() {
    try {
      System.out.println(" i [" + i + "]");
    } catch(Exception e) {
      e.printStackTrace();
    } catch(RuntimeException e) {
      e.printStackTrace();
    } catch(NullPointerException e) {
      e.printStackTrace();
    }
  }
}
```

- **Unused Code** son el conjunto de pautas para determinar si existe código no utilizado en la aplicación, como son:

  UnusedPrivateField alerta cuando un atributo "private" es declarado y/o se le ha asignado un valor y no sea utilizado.

  Ejemplo:

```
public class Something {
  private static int FOO = 2; // Unused
  private int i = 5; // Unused
  private int j = 6;
  public int addOne() {
    return j++;
  }
}
```

  UnusedLocalVariable alerta cuando una variable es declarada y/o se le ha asignado un valor y no sea utilizado.

  Ejemplo:

```
public class Foo {
 public void doSomething() {
  int i = 5; // Unused
 }
}
```

2.4.2 CheckStyle.

CheckStyle es una herramienta cuyo principal objetivo es ayudar a los programadores en el proceso de desarrollo a obtener un código fuente que siga un mismo estilo de diseño o estándar de codificación en cuanto a su formato.

Viene con un archivo de configuración de ejemplo pero es configurable para que pueda adaptarse a las particularidades de cada proyecto.

El conjunto de controles disponible que maneja Checkstyle es muy completo y se muestran a continuación:

- **Comentarios Javadoc** su función es ayudar al mantenimiento puesto que por ejemplo revisa que todos los métodos tengan un comentario, como son:

  JavadocType verifica comentarios Javadoc para las definiciones de clases e interfaces.

  JavadocStyle comprueba que los comentarios Javadoc estén bien estructurados. Ejemplo:

  Valida que las declaraciones Javadoc tengan descripción. Esto incluye las etiquetas como *@param* y *@return*.
- **Encabezados** comprueba las expresiones regulares para el inicio de las clases java, como son:

  Header determina si el archivo origen se inicia con un encabezado especificado. Ejemplo:

  ```
  línea 1: / / / / / / / / / / / / / / / / / / / / / / / / / / / /
  línea 2: / / checkstyle Descripción.
  línea 3: / / / / / / / / / / / / / / / / / / / / / / / / / / / /
  ```

  RegexpHeader cuando el encabezado requiere de detalles dinámicos como el año verifica que estos sean especificados.

  Ejemplo:

  ```
  línea 2:  / / checkstyle: $
  línea 3:  / / Comprueba la adhesión a un conjunto de reglas \
  línea 4:  / / Derechos de autor \ (C \) \ d \ d \ d \ d
  Línea 6:
  Línea 7:  paquetes
  Línea 9:
  línea 10: importaciones
  ```

- **Espacios en blanco** este conjunto de reglas tratan las políticas el uso de espacios en blanco en el código fuente, como son:

  GenericWhitespace inspecciona que exista espacios en blanco junto a los símbolos < >.

  Por ejemplo la forma correcta es:

```
Lista <Integer> x = new ArrayList <Integer> ();
List <List <Integer>> y = new ArrayList <List <Integer>> ();
```

  Por ejemplo la forma incorrecta es:

```
List <Integer> x = new ArrayList <Integer> ();
List <List <>> Integer y = new ArrayList <List <>> Integer ();
```

  WhitespaceAround inspecciona que existan espacios en blanco junto a los símbolos, como en el cuerpo de un constructor.

```
MyClass pública () {} / constructor / vacío
función public void () {} / / Método vacía
```

- **Modificadores**

  RedundantModifier detecta modificadores redundantes en los siguientes casos: En declaraciones de métodos en clases de tipo interfaz no se recomienda el uso de *public* porque resulta redundante ya que los métodos en una interfaz son de tipo públicas por defecto.

- **Codificación** contiene el conjunto de reglas especializadas de codificación como son:

  ArrayTrailingComma revisa que la inicialización de una matriz tenga una coma al final.

  Ejemplo de la forma correcta de declarar una matriz:

```
int [] a = new int []
{
    1,
    2,
    3,
};
```

  Ejemplo de la forma incorrecta de declarar una matriz:

```
return new int [] {0};
```

  EqualsAvoidNull revisa el uso de la asignación del valor *NULL* en métodos de comparación *equals.*

  Por ejemplo la forma incorrecta es:

```
String nullString = null;
nullString.equals("My_Sweet_String");
```

  La forma correcta es:

```
String nullString = null;
"My_Sweet_String".equals(nullString);
```

- **Miscelánea** contiene un conjunto variado de reglas para la codificación tales como:

  UncommentedMain el método *main* es muy utilizado en la etapa de depuración del código, pero una vez terminada esta etapa es frecuente olvidar borrar el método.

  TrailingComment esta regla requiere que los comentarios sean lo único en una línea. Especialmente para el caso de los comentarios que inician con / /.

  Se considera una mala práctica lo siguiente:

```
a = b + c; / / Algún comentario
d = e / f / / Otro comentario de esta línea
```

  Por estándares de diseño se considera conveniente que los comentarios no se encuentren dentro de la estructura del código.

2.5 Disposiciones legales a las empresas de transporte turístico terrestre

A continuación se presentan los artículos del reglamento de transporte terrestre turístico pertinentes puesto que guardan relación al producto desarrollado. La información fue recuperada del: **reglamento de transporte terrestre turístico, norma: decreto ejecutivo 830 publicado: registro oficial 252. Rafael Correa Delgado presidente constitucional de la república:**

"Art. 8. Tercerización del servicio de transporte.

Todas las actividades turísticas contempladas en el artículo 5 de la ley de turismo que no cuenten con transporte propio, deberán contratar únicamente los servicios de los prestadores de transporte terrestre turístico debidamente legalizado y autorizado para realizar este tipo de servicio, para lo cual necesariamente celebrarán el correspondiente contrato u obtendrán la correspondiente orden de trabajo, según lo dispuesto por el Ministerio de Turismo, salvo en las excepciones contempladas en el artículo 20 de este reglamento."

En relación con el anterior artículo el sistema facilita la contratación del servicio de transporte turístico terrestre de manera de directa es decir con vehículos propios de la compañía o subcontratando el servicio con vehículos de los asociados, por medio del módulo de registro de asociados y sus vehículos. Los asociados son compañías del mismo gremio es decir compañías prestadoras del servicio de transporte turístico terrestre.

"Art. 10. Facultad exclusiva de las agencias de viajes operadoras de turismo.

Únicamente las agencias de viajes operadoras de turismo definidas en los reglamentos pertinentes y en las normas técnicas establecidas por el Ministerio de Turismo se encuentran facultadas para elaborar, organizar, operar y vender, ya sea directamente al usuario o a través de los otros tipos de agencias de viaje, toda clase de servicios y paquetes turísticos dentro del territorio nacional para ser vendidos al interior o fuera del país. En consecuencia, los prestadores de servicios de transporte terrestre turístico, no podrán organizar ni prestar dichos servicios."

“Art. 19. Prohibición a los prestadores del servicio de transporte terrestre turístico.

Los prestadores de transporte terrestre turístico, estarán prohibidos de realizar transporte público regular de pasajeros o cualquier otra modalidad distinta a la que le corresponde. La violación a esta disposición se sancionará con la revocatoria o cancelación del permiso de operación. De comprobarse que los conductores y vehículos de transporte terrestre turístico, realizan el servicio de transporte público de pasajeros, serán sancionados de conformidad con lo que establece el Art. 90 literal o) de la ley de tránsito y transporte terrestres.”

Con referencia al artículo 10 y 19 del reglamento el presente sistema implementado para una compañía de transporte turístico se enfoca en facilitar la contratación exclusivamente de este servicio, por medio del módulo de reservas en línea, y ningún otro como por ejemplo de paquetes turísticos.

CAPÍTULO 3

ANÁLISIS Y DISEÑO

3.1 Análisis

3.1.1 Revisión de los requerimientos.

ICONIX dice que es un error común en la fase de análisis es intentar representar todo de todos los requerimientos en los casos de uso. Por esta razón se deben mantener los requerimientos no funcionales fuera de los casos de uso para ello se va a presentar los requerimientos separados en dos listas:

- **Requerimientos funcionales**

1. El sistema debe ser capaz de manejarse según los siguientes perfiles de usuarios:
   a. **Cliente** el sistema de permitirle visitar y navegar en la página web de la empresa y registrar su reserva on-line.
   b. **Administrador** para este perfil de usuario el sistema bebe poder registrar, la siguiente información: cédula de identidad, nombres, teléfono, correo electrónico, nombre de usuario y contraseña y debe existir un mecanismo de recuperación de contraseña.
2. La página inicial de la empresa debe mostrar:
   Inicio, Nosotros, Servicios prestados, Contáctenos, pre-reserva en-línea, Galería Flota
3. El usuario Cliente debe registrar su pre-reserva ingresando a la opción pre-reserva en línea y llenando la siguiente información:
   Nombre, correo electrónico, teléfono, fecha de salida, lugar de salida, lugar de destino, hora de salida, vehículo, observación.
   El sistema debe permitirle escoger el tipo de vehículo según las necesidades del cliente.
   Una vez llenada y enviada está información el sistema debe registrar la pre-reserva en una lista de pre-reservaciones registradas con el estado “registrada”, con la fecha y hora en la que se realizó la pre-reserva.

4. El sistema debe permitir al usuario Administrador buscar y actualizar una pre-reserva según su fecha de pre-reserva, nombre o estado.
5. El sistema permitirá al usuario Administrador gestionar el módulo para la flota de vehículos en la cual puede registrar, editar y eliminar, la flota de vehículos con la que cuenta la empresa.

   El sistema debe registrar la siguiente información del vehículo: foto, capacidad de pasajeros del vehículo y sus características.
6. El sistema debe permitir al usuario Administrador el registro los asociados y los vehículos con los que cuenta con los siguientes datos de contacto: Nombres, nombre de la empresa, teléfonos, dirección y los vehículos.

- **Requerimientos no funcionales**

1. Las pre-reservas deben ser mostradas en orden cronológico en forma descendente.
2. El sistema debe tener la opción de cambiarlo al idioma inglés a las páginas web a la pueden acceder los usuarios con el perfil Cliente.
3. El sistema debe presentar facilidad de mantenimiento por medio del uso de estándares en la nomenclatura dentro proyecto.
4. El sistema debe estar desarrollado con JSF.
5. El sistema debe estar en la nube
6. El sistema debe presentar facilidad de implementación de nuevos módulos por medio del uso del patrón de diseño MVC.
7. El sistema debe garantizar que la información sea administrada de forma segura y confidencial; para lo cual el usuario Administrador debe gestionar la información interna de la empresa previa a la autenticación por medio de su nombre de usuario y contraseña.
8. El sistema debe garantizar integridad de la información utilizada, es decir que la información manejada por el sistema debe ser completa, correcta y coherente según el modelo de negocio de Chrisland Service And Touring.

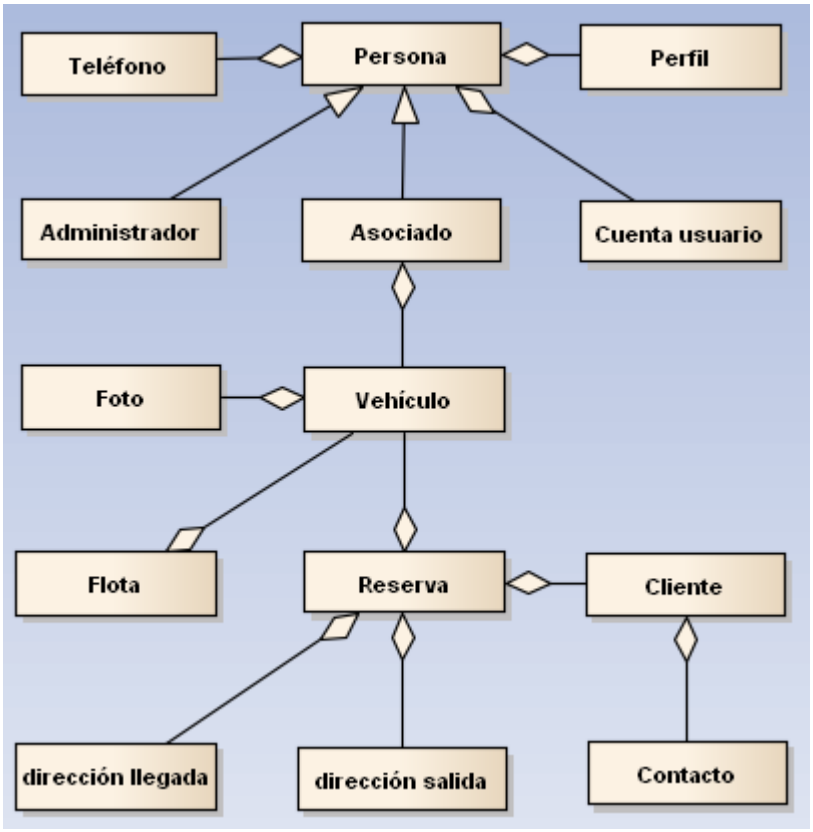
3.1.1.1 Identificar los objetos de dominio del mundo real.

El análisis para el de modelado de dominio conlleva identificar los conceptos de los objetos pertinentes en este caso para el sistema de transporte turístico.

Para lo cual la estrategia que se utilizó para descubrir las clases fue preguntarse qué información se requiere para la aplicación.

Para identificar las generalizaciones de analizaron los atributos que tienen en común las clases y para las agregaciones se buscaron las partes de las que están formadas las clases.

Figura 15. Modelo de dominio inicial para el sistema de transporte turístico



Elaborado por: Verónica Sangucho

El diagrama de modelo de dominio debe poder ser entendido y leído por cualquier persona como son el cliente o el equipo de desarrollo. Para lo cual las generalizaciones se leen "como un tipo de", y a las agregaciones y composiciones se las leen "es parte de".

3.1.1.2 Prototipos para el sistema de transporte turístico.

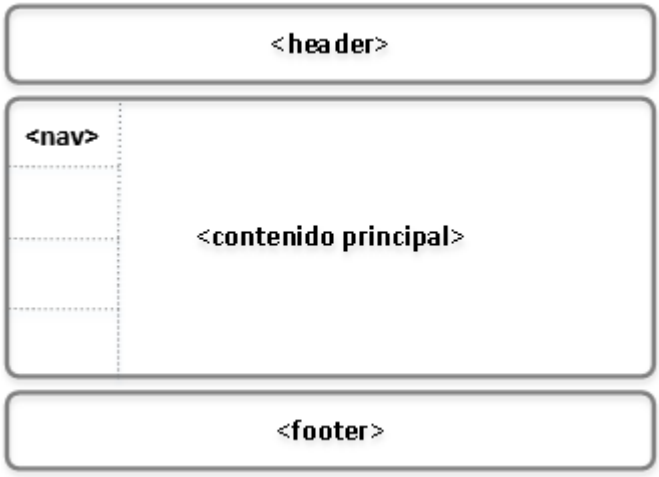
La estructura y diseño de los prototipos están pensados según la necesidad de la página web de subirse posteriormente a la nube o Internet. Por lo que los prototipos reflejan:

- Facilidad de acceso, altamente intuitivo, sencillo, debido a que el nicho de mercado en Internet contempla usuarios de todos los niveles de experticia en el manejo de Internet.
- El contenido principal se encuentra en la mitad superior de la página.
- Presenta también en su estructura el lugar en donde está localizada la vinculación con las redes sociales.
- Contempla la ubicación de la opción de internacionalización, es decir, para el cambio al idioma inglés.
- La página web no presenta música de fondo, debido a que en usuario podría resultar molesto y puede provocar que el cliente abandone el sitio web.
- Las nuevas etiquetas que brinda la tecnología HTML5 mejoran la estructura del contenido.

Los siguientes prototipos muestran una primera referencia gráfica general del funcionamiento del sistema.

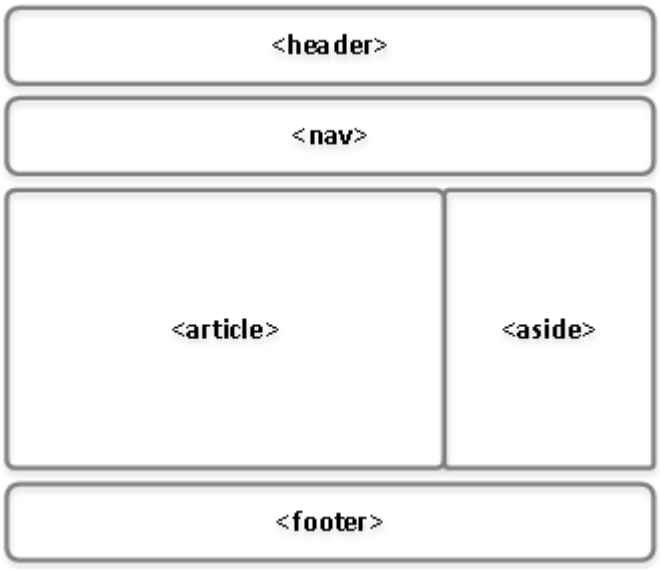
La aplicación utilizó dos estructuras bases, una para las páginas de tipo *private* a las cuales se puede acceder previo a la autenticación y otra para las páginas tipo *public* a las cuales se puede acceder sin autenticación.

Figura 16. Plantilla base para las páginas private del sistema



Elaborado por: Verónica Sangucho

Figura 17. Plantilla base para las páginas public del sistema

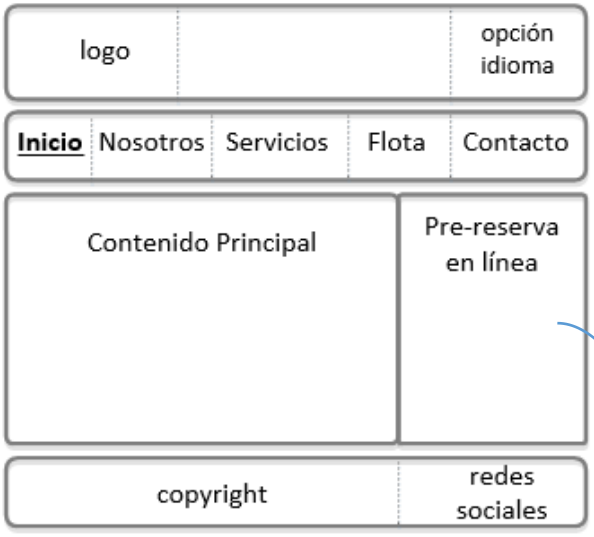


Elaborado por: Verónica Sangucho

Prototipos para las páginas public

A partir de la estructura base para satisfacer los requerimientos los elementos se distribuyeron de la siguiente manera:

Figura 18. Pantalla de tipo public Inicio

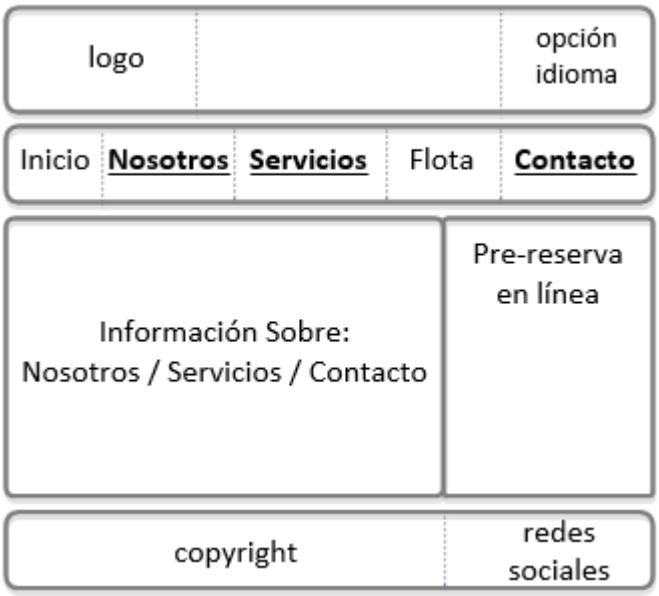


Elaborado por: Verónica Sangucho

Figura 19. Sección de pre-reserva

Elaborado por: Verónica Sangucho

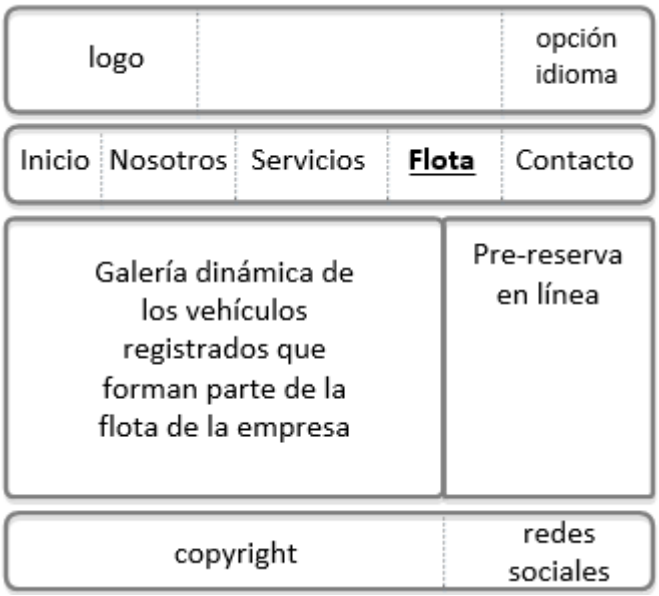
Figura 20. Pantalla de tipo public Nosotros



Elaborado por: Verónica Sangucho

En la figura anterior se puede apreciar que en la sección de navegación el usuario puede seleccionar la información de la empresa sobre la empresa, los servicios que brinda e información de contacto.

Figura 21. Pantalla de tipo public Flota.

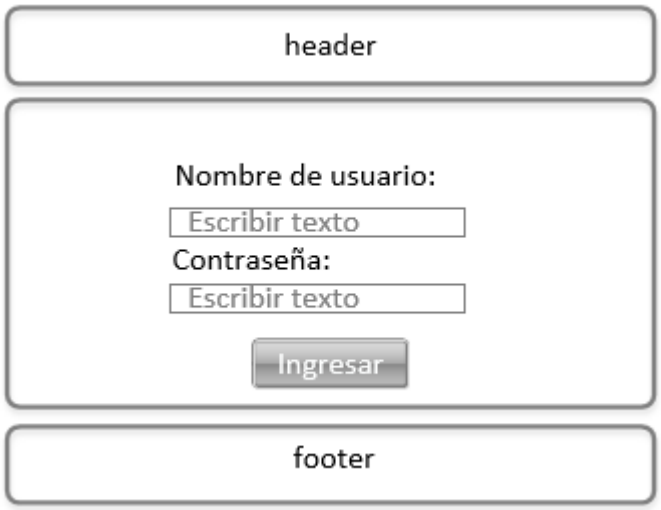


Elaborado por: Verónica Sangucho

Prototipos para las páginas private

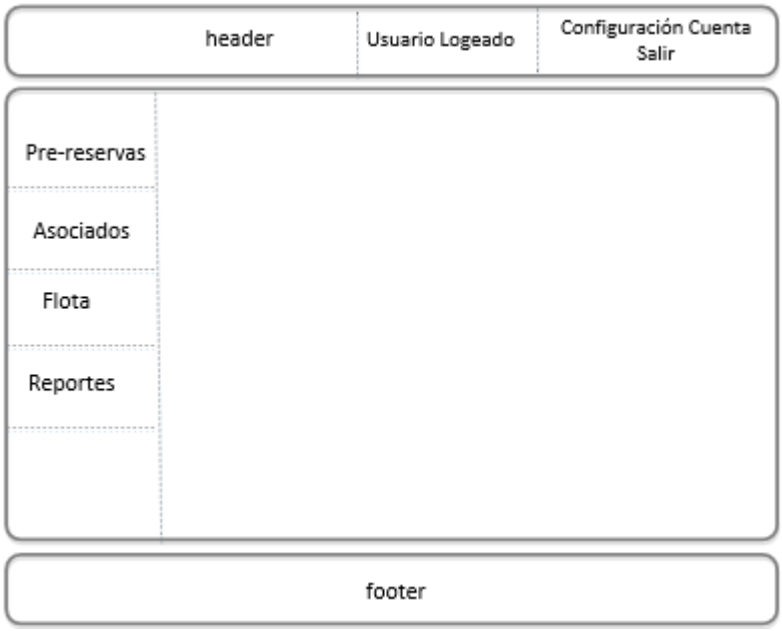
A partir de la estructura base se presenta la distribución de los elementos de la parte interna del sistema

Figura 22. Pantalla de tipo private ingreso



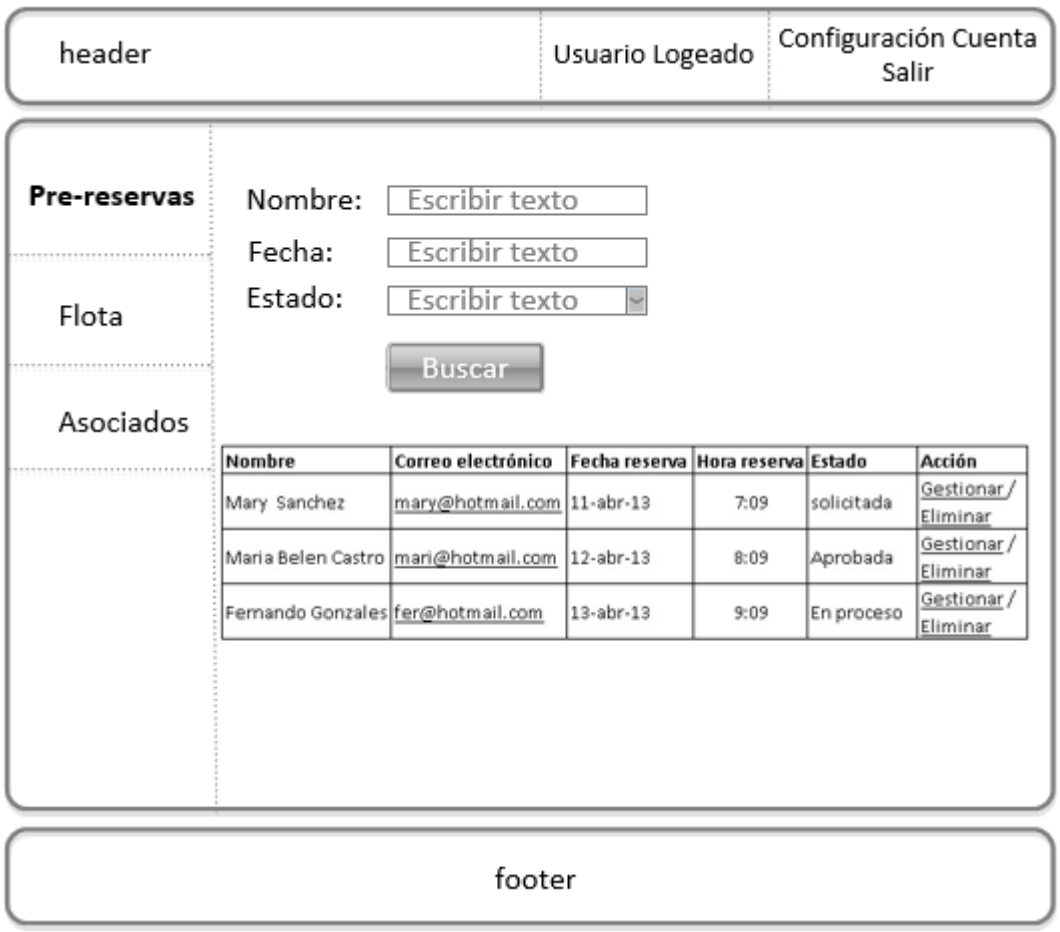
Elaborado por: Verónica Sangucho

Figura 23. Pantalla de tipo menú administrador



Elaborado por: Verónica Sangucho

Figura 24. Pantalla de tipo private pre-reservas

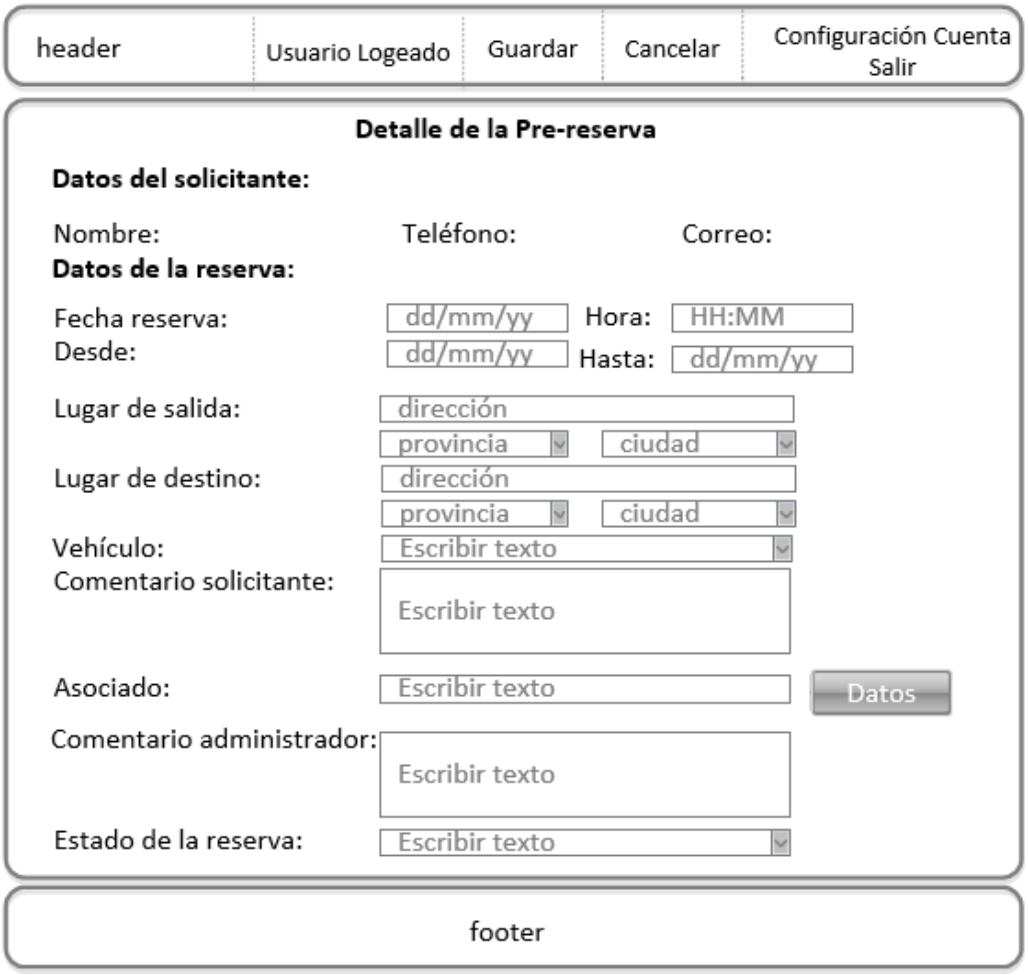


Elaborado por: Verónica Sangucho

Una vez realizado el proceso de autenticación el usuario administrador puede acceder a la opción pre-reservas, la cual se directamente cargará la tabla con las pre-reservas que fueron realizadas a través de la página web.

El usuario podrá buscar una pre-reserva por medio del nombre, fecha de pre-reserva o estado o cualquier combinación entre estas opciones, una vez encontrada la pre-reserva buscada el usuario podrá gestionarla.

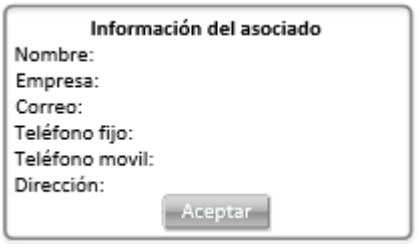
Figura 25. Pantalla de tipo private Gestionar pre-reserva



Elaborado por: Verónica Sangucho

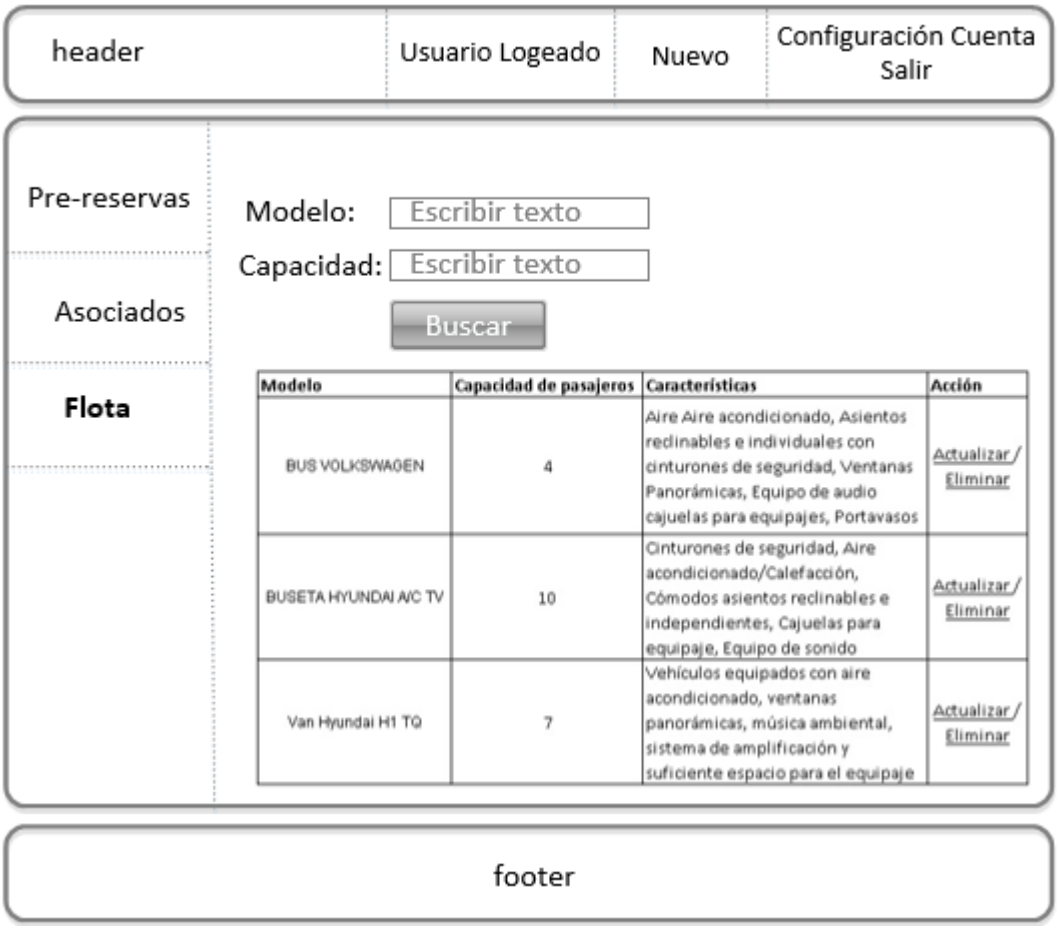
En la opción Gestionar se muestra los detalles de la pre-reserva, el usuario administrador puede actualizar los detalles de la pre-reserva excepto el nombre, teléfono, correo, fecha pre-reserva y comentario de solicitante. También puede agregar un comentario, cambiar el estado de la pre-reserva y guardar los cambios.

Figura 26. Pop Up de tipo private opción Datos de contacto del Asociado



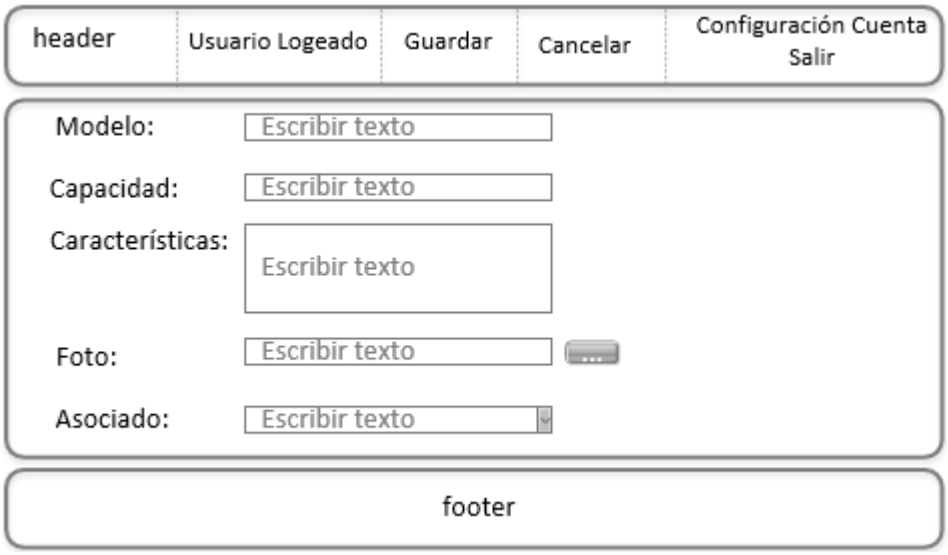
Elaborado por: Verónica Sangucho

Figura 27. Pantalla de tipo private Flota



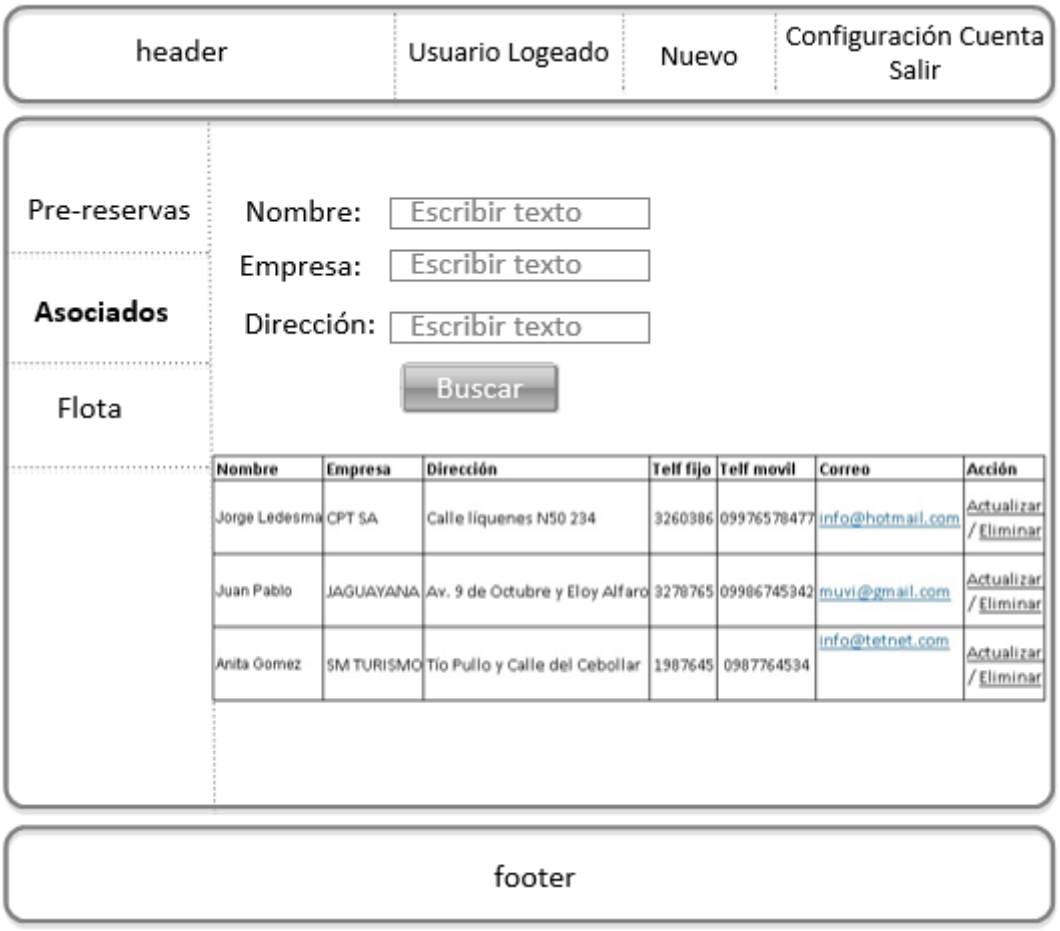
Elaborado por: Verónica Sangucho

Figura 28. Pantalla de tipo private Nuevo vehículo



Elaborado por: Verónica Sangucho

Figura 29. Pantalla de tipo private Asociados

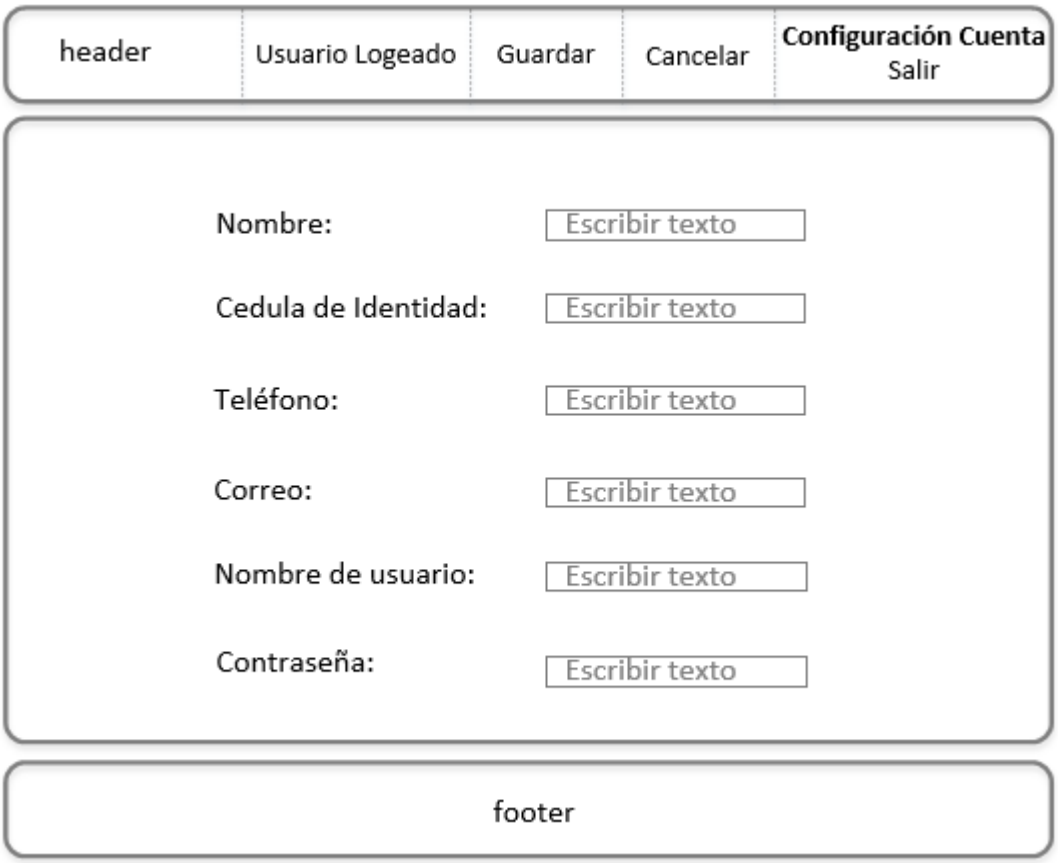


Elaborado por: Verónica Sangucho

Figura 30. Pantalla de tipo private Nuevo asociado

Elaborado por: Verónica Sangucho

Figura 31. Pantalla de tipo private Editar datos del administrador



Elaborado por: Verónica Sangucho

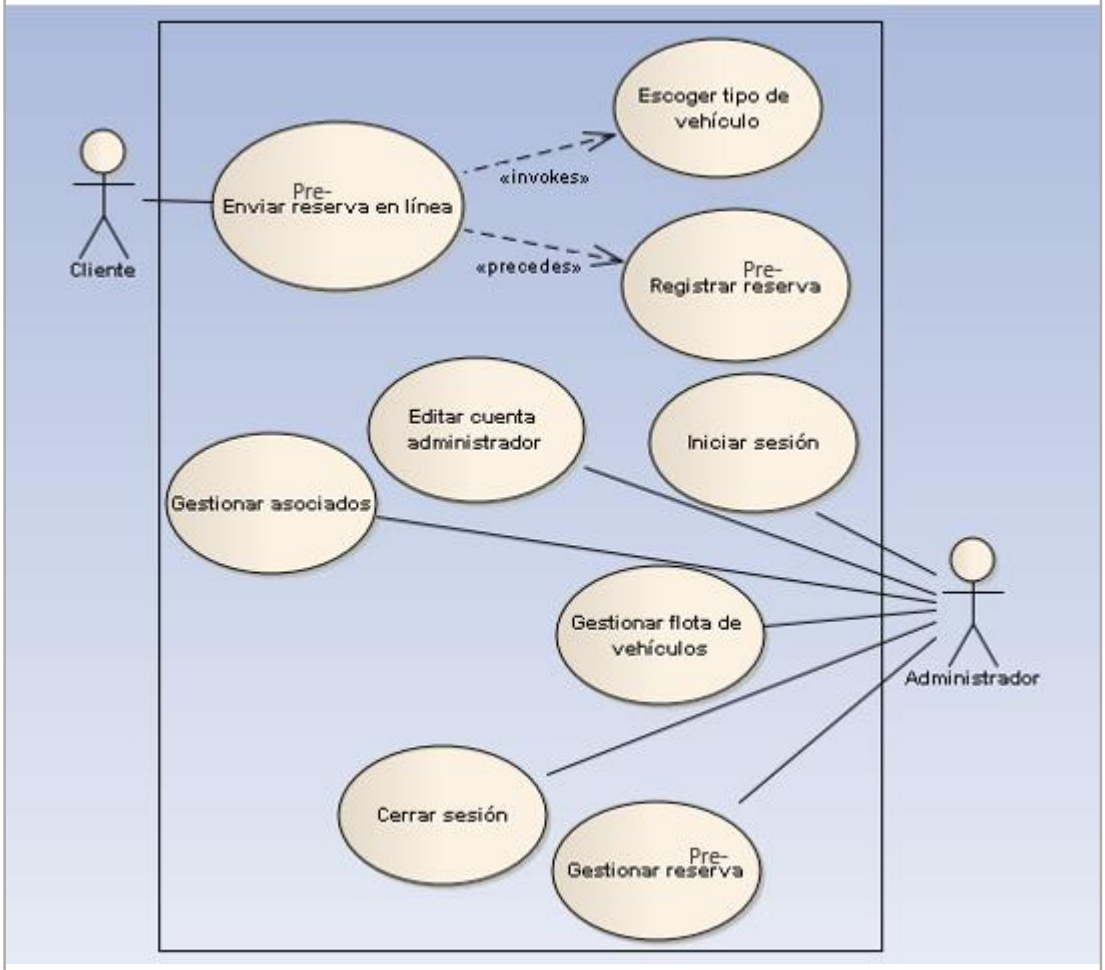
3.1.1.3 Definir el comportamiento de los requerimientos.

Este paso consiste en el **modelado de los casos de uso**. Con el fin de describir de forma clara los todos requerimientos funcionales del sistema, y como el sistema va a interactuar con el usuario.

Para lo cual Iconix propone seguir las siguientes etapas:

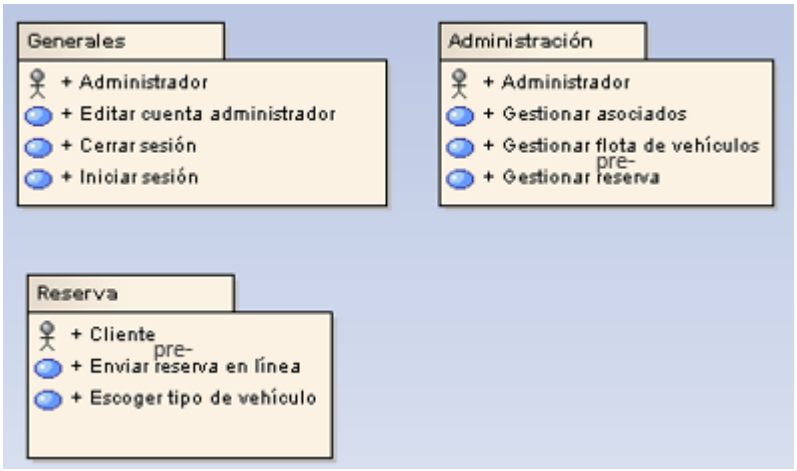
- ✓ Identificar todos los casos por medio de los diagramas de casos.
- ✓ Organizar los casos de uso en un diagrama de paquetes.
- ✓ Describirlos casos de uso textualmente de forma más detallada, incluyendo el curso básico y los cursos alternativos.

Figura 32. Modelo de casos de uso



Elaborado por: Verónica Sangucho

Figura 33. Diagrama de paquetes para organizar los casos de uso



Elaborado por: Verónica Sangucho

Una vez identificados y organizados los casos de uso, se describe textualmente cada uno de ellos.

Tabla 5. Descripción del caso de uso Iniciar sesión

Descripción de Caso de Uso		
Caso de Uso	Iniciar sesión	
Actor	Usuario Administrador	
Precondiciones	• El usuario administrador debe tener registrado un nombre de usuario y password. • El usuario administrador debe estar viendo la página de autenticación del sistema en el browser.	
Flujo de Eventos	**Básico**	El usuario administrador digita su nombre de usuario y contraseña y da clic en la opción ingresar. El sistema verifica que los datos ingresados sean válidos, si los datos existen, el sistema autentica al usuario en sesión.
	Alternativos	Si el nombre de usuario y la contraseña ingresados son incorrectos el sistema envía un mensaje "Datos inválidos comuníquese con el administrador del sistema".
Pos condiciones	• El sistema direcciona a la página Inicio para Administrador que contiene el menú principal, presenta la opción de "Salir" y muestra nombre del usuario administrador que se encuentra en la sesión.	

Elaborado por: Verónica Sangucho

Tabla 6. Descripción del caso de uso Cerrar sesión

Descripción de Caso de Uso		
Caso de Uso	Cerrar sesión	
Actor	Usuario Administrador	
Precondiciones	• El usuario administrador debe estar autenticado en sesión.	
Flujo de Eventos	**Básico**	El usuario administrador da clic en la opción "salir" de la página web. El sistema finaliza la sesión anulándola.
Pos condiciones	• El sistema direcciona al usuario a la página de autenticación	

Elaborado por: Verónica Sangucho

Tabla 7. Descripción del caso de uso Editar cuenta administrador

Descripción de Caso de Uso		
Caso de Uso	Editar cuenta administrador	
Actor	Usuario Administrador	
Precondiciones	• El usuario administrador debe estar autenticado en sesión por medio del browser.	
Flujo de Eventos	**Básico**	El usuario da clic en la opción "Configuración de la cuenta" del menú. El sistema direcciona a la página de configuración de la cuenta. El sistema carga los datos del usuario en sesión. El usuario edita sus datos y da clic en la opción "Guardar". El sistema guarda los cambios realizados por el usuario y muestra el mensaje de éxito.
	Alternativos	El usuario realiza cambios en sus datos que no desea guardar, por lo cual da clic en la opción "Cancelar". El sistema no guarda los cambios y direcciona al usuario a la página del menú principal de inicio para administrador. El usuario ingresa datos incorrectos o incompletos. El sistema envía un mensaje de los datos faltantes o incorrectos.
Pos condiciones	• El sistema y direcciona a la página del menú principal de inicio para administrador.	

Elaborado por: Verónica Sangucho

Tabla 8. Descripción del caso de uso Gestionar asociados

Descripción de Caso de Uso		
Caso de Uso	Gestionar asociados	
Actor	Usuario Administrador	
Precondiciones	• El usuario administrador debe estar autenticado en sesión.	
Flujo de Eventos	**Básico**	El usuario da clic en la opción "Asociados". El sistema lo direcciona a la página de gestión de asociados. El sistema lista en una tabla a los asociados y muestra la opción de nuevo asociado y las opciones de filtrado. El usuario ingresa los parámetros de filtro y da clic en Buscar. El sistema filtra la información según los parámetros. El usuario da clic en la opción "Actualizar" de un asociado. El sistema lo direcciona a la página "Edición de datos del asociado" y carga los datos del asociado. El usuario edita los datos y da clic en la opción "Guardar". El usuario también puede dar clic en la opción "nuevo". El sistema lo direcciona a la página de "Nuevo asociado". El usuario ingresa los datos y da clic en "Guardar". El sistema guarda los datos y envía un mensaje de éxito.
	Alternativos	El usuario realiza cambios en los datos de un asociado o crea un asociado que no desea guardar, por lo cual da clic en la opción "Cancelar". El sistema no guarda los cambios y direcciona al usuario a la página "Gestión de Asociados". El usuario ingresa datos inválidos o incompletos. El sistema valida los datos y si los datos son incorrectos envía un mensaje de datos inválidos o incompletos.
Pos condiciones	• Una vez guardados los datos y presentado el mensaje de éxito el sistema direcciona a la página de "Gestión de Asociados".	

Elaborado por: Verónica Sangucho.

Tabla 9. Descripción del caso de uso Gestionar flota de vehículos

Descripción de Caso de Uso		
Caso de Uso	Gestionar flota de vehículos	
Actor	Usuario Administrador	
Precondiciones	• El usuario administrador debe estar autenticado en sesión.	
Flujo de Eventos	**Básico**	El usuario da clic en la opción "Flota" del menú. El sistema lo direcciona a la página de "Gestión de la flota". El sistema lista los vehículos y los muestra en una tabla. El usuario ingresa los parámetros de filtrado y da clic en la opción "Buscar". El sistema lista los vehículos según los parámetros ingresados. El usuario escoge la opción de actualizar de un vehículo. El sistema direcciona a la página de Edición de datos del vehículo y carga los datos del vehículo seleccionado. El usuario edita los datos y da clic en la opción guardar. El sistema guarda los cambios y presenta el mensaje de éxito. El usuario también puede dar clic en la opción "Nuevo. El sistema direcciona a la página de Nuevo vehículo. El usuario ingresa los datos de un nuevo vehículo y da clic en "Guardar". El sistema guarda los datos y presenta el mensaje de éxito.
	Alternativos	El usuario realiza cambios en los datos de un vehículo o crea un vehículo que no desea guardar, por lo cual da clic en la opción "Cancelar". El sistema direcciona al usuario a la página de "Gestión de flota". El usuario ingresa datos inválidos o incompletos. El sistema presenta el mensaje datos inválidos o incompletos.
Pos condiciones	• El sistema guarda los cambios realizados por el usuario, presente el mensaje de éxito y direcciona a la página de "Gestión de la flota".	

Elaborado por: Verónica Sangucho

Tabla 10. Descripción del caso de uso Gestionar la pre-reserva

Descripción de Caso de Uso		
Caso de Uso	Gestionar pre-reserva	
Actor	Usuario Administrador	
Precondiciones	• El usuario administrador debe estar autenticado en sesión por medio del browser.	
Flujo de Eventos	**Básico**	El usuario da clic en la opción "pre-reservas" del menú. El sistema lo direcciona a la página de gestión de pre-reservas. El sistema lista las pre-reservas en una tabla. El usuario ingresa los parámetros de filtrado y da clic en el opción "Buscar". El sistema filtra las pre-reservas según los parámetros ingresados. El usuario da clic en la opción gestionar de una pre-reserva. El sistema direcciona a la pantalla gestionar pre-reserva y presenta los detalles de la pre-reserva. El usuario actualiza los datos de la pre-reserva y da clic en "Guardar". El sistema presenta el mensaje de éxito. El usuario también puede dar clic en la opción eliminar de una pre-reserva. El sistema presenta en pantalla el mensaje "Está seguro de eliminar la pre-reserva". El usuario da clic en aceptar. El sistema guarda los cambios.
	Alternativos	El usuario realiza cambios en los datos de una pre-reserva que no desea guardar, por lo cual da clic en la opción "Cancelar". El sistema no guarda los cambios y direcciona al usuario a la página "Gestión de pre-reservas". El usuario ingresa datos inválidos o incompletos. El sistema muestra el mensaje de datos inválidos o incompletos.
Pos condiciones	• El sistema guarda los cambios realizados por el usuario, presenta el mensaje de éxito y direcciona a la página de "Gestión de pre-reservas".	

Elaborado por: Verónica Sangucho

Tabla 11. Descripción del caso de uso Escoger tipo de vehículo

Descripción de Caso de Uso		
Caso de Uso	Escoger tipo de vehículo	
Actor	Usuario Cliente	
Precondiciones	• El usuario cliente debe tener acceso a Internet mediante el browser. • El usuario cliente debe estar viendo la página de Chrisland Service and Touring en el browser.	
Flujo de Eventos	**Básico**	En la sección de pre-reserva en línea el usuario cliente escoge el vehículo que desea pre-reservar primero según el número de pasajeros que requiere. El sistema llena el combo con los vehículos que tengan la capacidad requerida que el usuario escogió previamente. El usuario cliente puede escoger otra capacidad de pasajeros del vehículo, el sistema refresca nuevamente el combo de vehículos llenándolo con los que tengan la capacidad seleccionada. El usuario cliente da clic en alguna opción del combo de vehículos que desee pre-reservar.
Pos condiciones	• El usuario ha escogido el vehículo que requiere según su requerimiento.	

Elaborado por: Verónica Sangucho

Tabla 12. Descripción del caso de uso Enviar pre-reserva en línea

Descripción de Caso de Uso	
Caso de Uso	Enviar pre-reserva en línea
Actor	Usuario Cliente
Precondiciones	• El usuario cliente debe tener acceso a Internet mediante el browser. • El usuario cliente debe estar viendo la página de chrisland service and touring en el browser.

Elaborado por: Verónica Sangucho Continua…

Tabla 13. Descripción del caso de uso Enviar pre-reserva en línea (continuación…)

Flujo de Eventos	**Básico**	El usuario cliente ingresa los datos requeridos en la sección de pre-reserva en línea que presenta la pantalla principal del sistema, el sistema valida los datos que el usuario ingresa. El usuario cliente da clic en la opción enviar. El sistema verifica que los datos requeridos estén llenos y correctos y guarda los datos de la pre-reserva. Si los datos requeridos están completos el sistema guarda la pre-reserva y muestra un mensaje "pre-reserva registrada con éxito", el cliente da clic en aceptar.
	Alternativos	El usuario llena ingresa los datos y el sistema valida los datos. Si los datos requeridos para registrar una pre-reserva no están llenos o son inválidos el sistema presenta el mensaje datos requeridos faltantes o inválidos y señala los datos que faltan.
Pos condiciones	• El sistema registra los datos de la pre-reserva incluida la fecha y hora en que se realizó y direcciona a la página principal.	

Elaborado por: Verónica Sangucho

3.1.2 Revisión del diseño preliminar.

En el proceso ICONIX este es el último paso de la etapa de análisis y es intermedio entre en análisis (que consiste en el que) y el diseño (que consiste en el cómo).

3.1.2.1 Eliminar de la ambigüedad de los casos de uso.

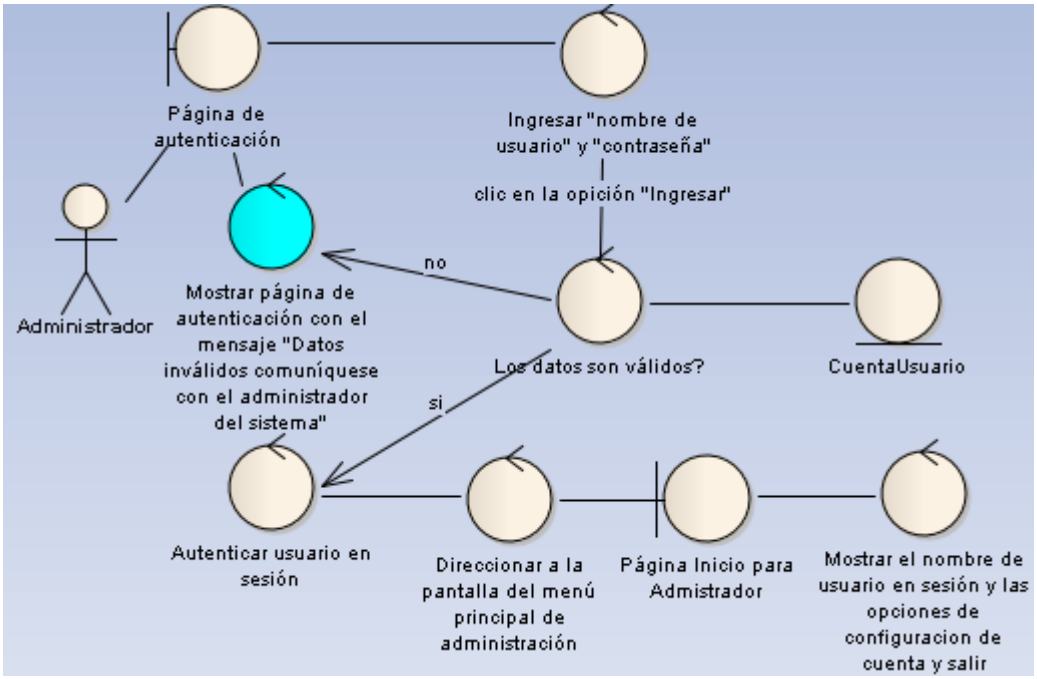
Para eliminar la ambigüedad de los casos de uso se realiza el análisis de robustez para lo cual hay que tomar en cuenta que:

- Los usuarios sólo pueden interactuar con los objetos de interfaz.
- Los objetos de interfaz sólo pueden interactuar con los controladores y los actores.
- Los objetos de entidad sólo pueden interactuar con los controladores.
- Los objetos controladores pueden interactuar con todos los objetos de interfaz, entidad y control.

A continuación se muestran los diagramas de robustez para cada caso de uso pertenecientes a esta etapa intermedia:

- **Diagrama de robustez del caso de uso:** Iniciar sesión

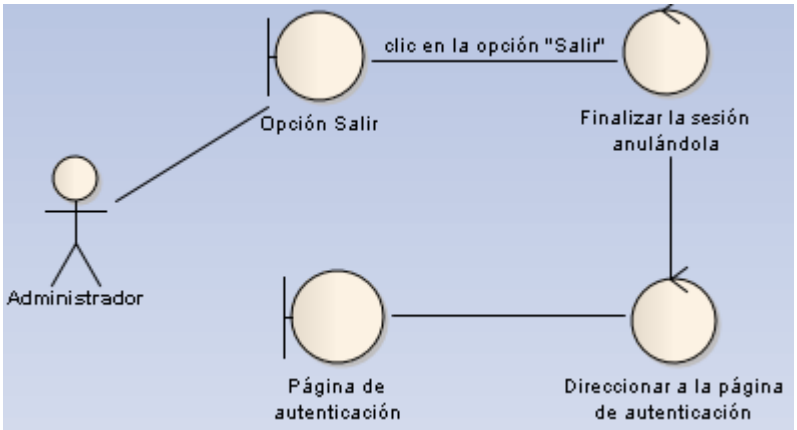
Figura 34. Diagrama de robustez para el caso de uso Iniciar sesión



Elaborado por: Verónica Sangucho

- **Diagrama de robustez del caso de uso:** Cerrar sesión

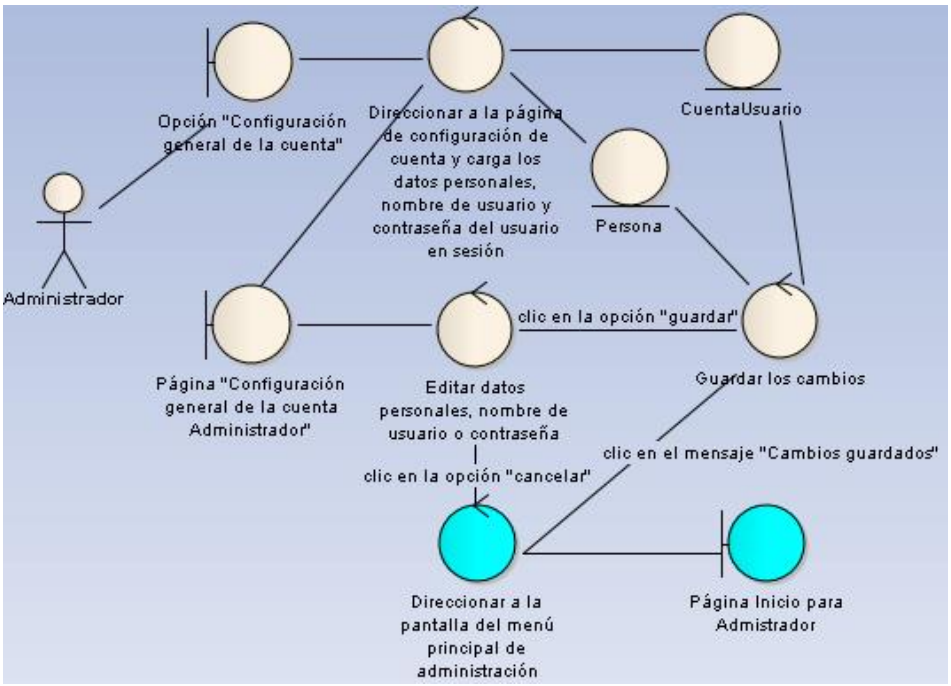
Figura 35. Diagrama de robustez para el caso de uso Cerrar sesión



Elaborado por: Verónica Sangucho

- **Diagrama de robustez del caso de uso:** Editar cuenta administrador

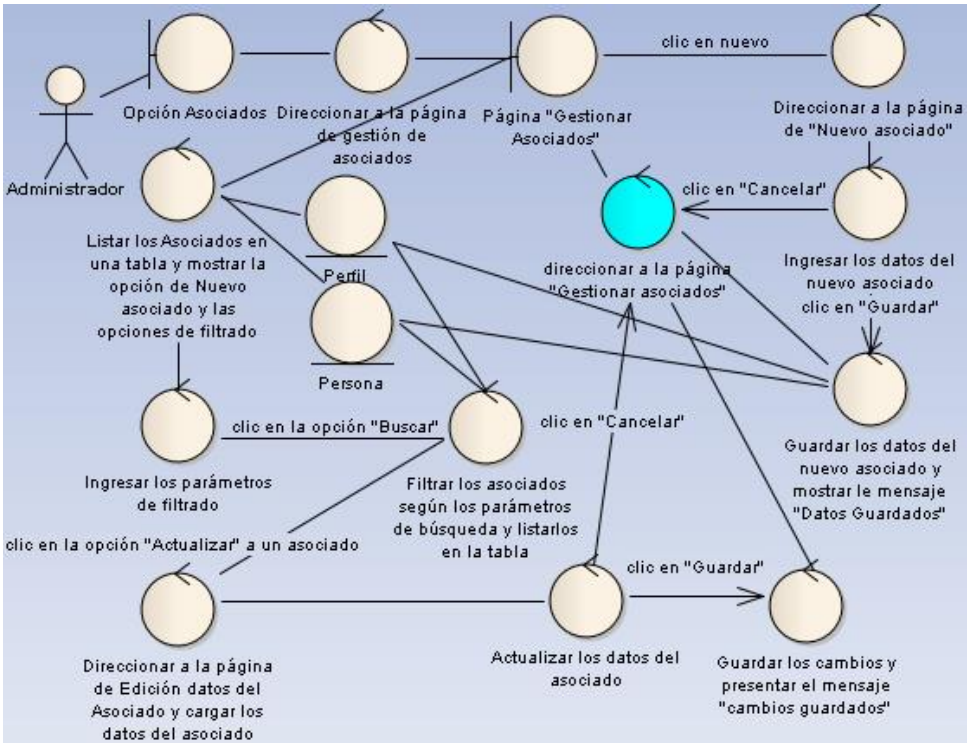
Figura 36. Diagrama de robustez para el caso de uso Editar cuenta administrador



Elaborado por: Verónica Sangucho

- **Diagrama de robustez del caso de uso:** Gestionar asociados

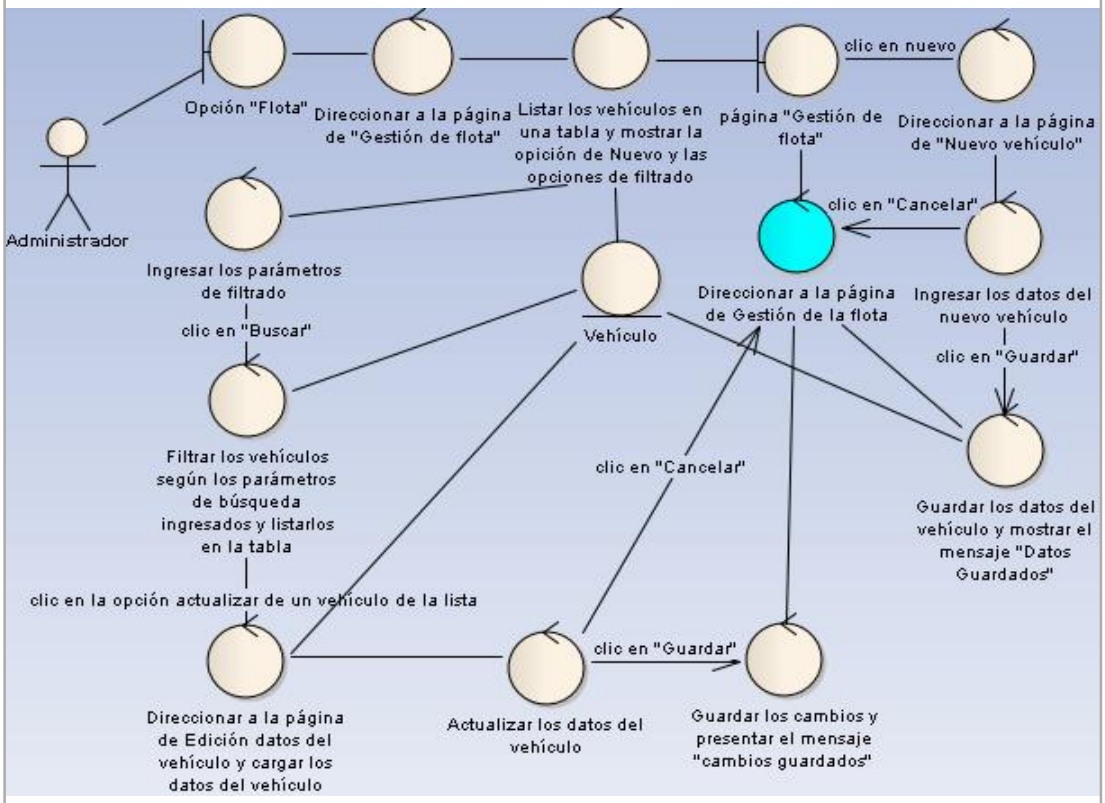
Figura 37. Diagrama de robustez para el caso de uso *Gestionar asociados*



Elaborado por: Verónica Sangucho

- **Diagrama de robustez del caso de uso:** Gestionar flota de vehículos

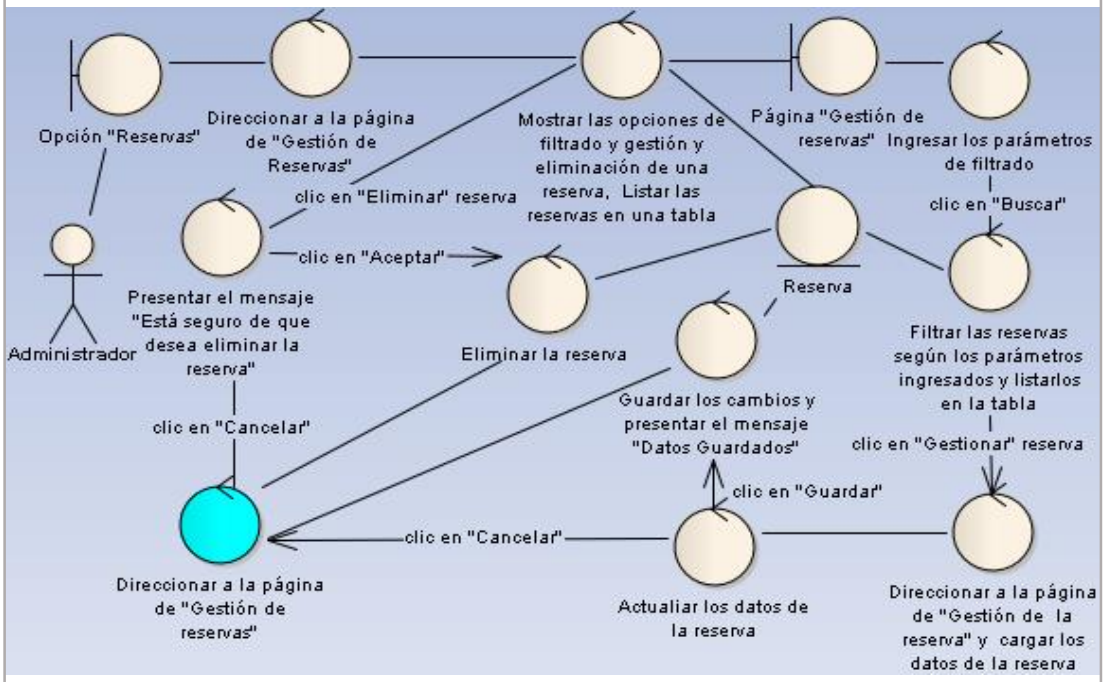
Figura 38. Diagrama de robustez para el caso de uso *Gestionar flota de vehículos*



Elaborado por: Verónica Sangucho

- **Diagrama de robustez del caso de uso:** Gestionar pre-reserva

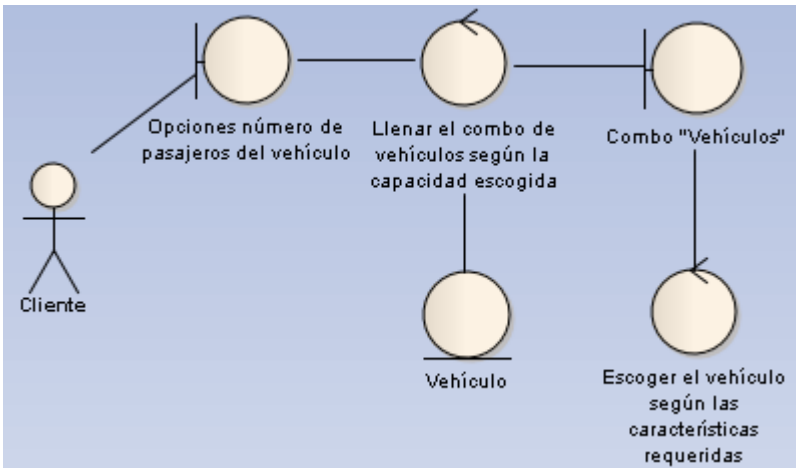
Figura 39. Diagrama de robustez para el caso de uso *Gestionar pre-reserva*



Elaborado por: Verónica Sangucho

- **Diagrama de robustez del caso de uso:** Escoger tipo de vehículo

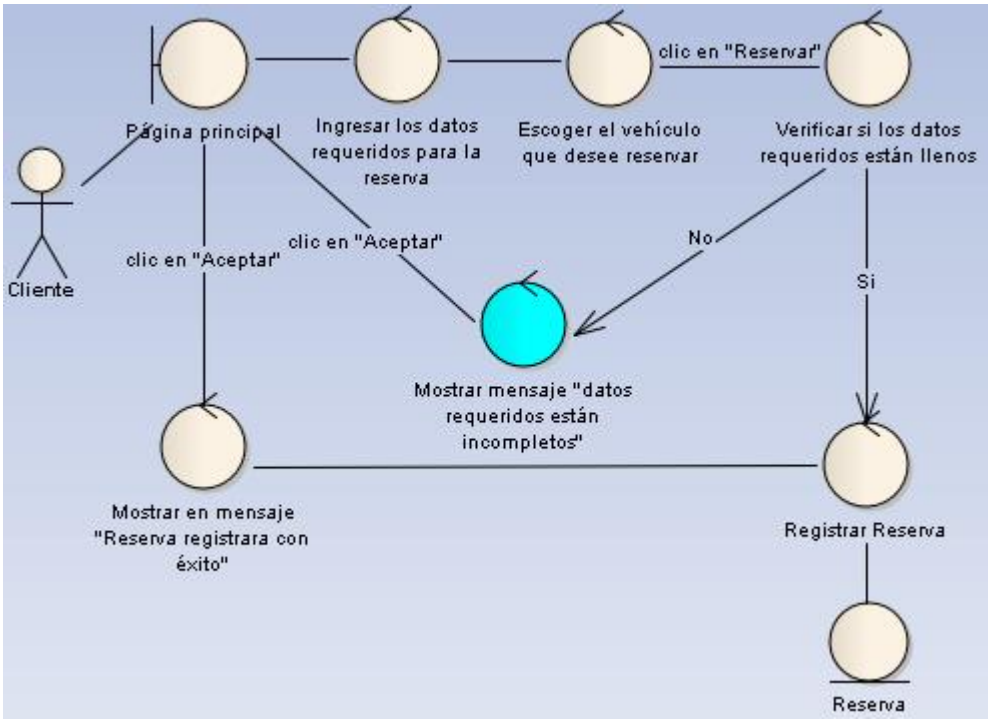
Figura 40. Diagrama de robustez para el caso de uso Escoger tipo de vehículo



Elaborado por: Verónica Sangucho

- **Diagrama de robustez del caso de uso:** Enviar pre-reserva en línea

Figura 41. Diagrama de robustez para el caso de uso Enviar pre-reserva en línea



Elaborado por: Verónica Sangucho

3.2 Diseño

La fase de diseño es el puente para ir del nivel conceptual del negocio al nivel que facilita llegar a la implementación del código por medio de los diagramas que contienen detalles de implementación de los casos de uso del proyecto.

3.2.1 Revisión crítica en detalle del diseño.

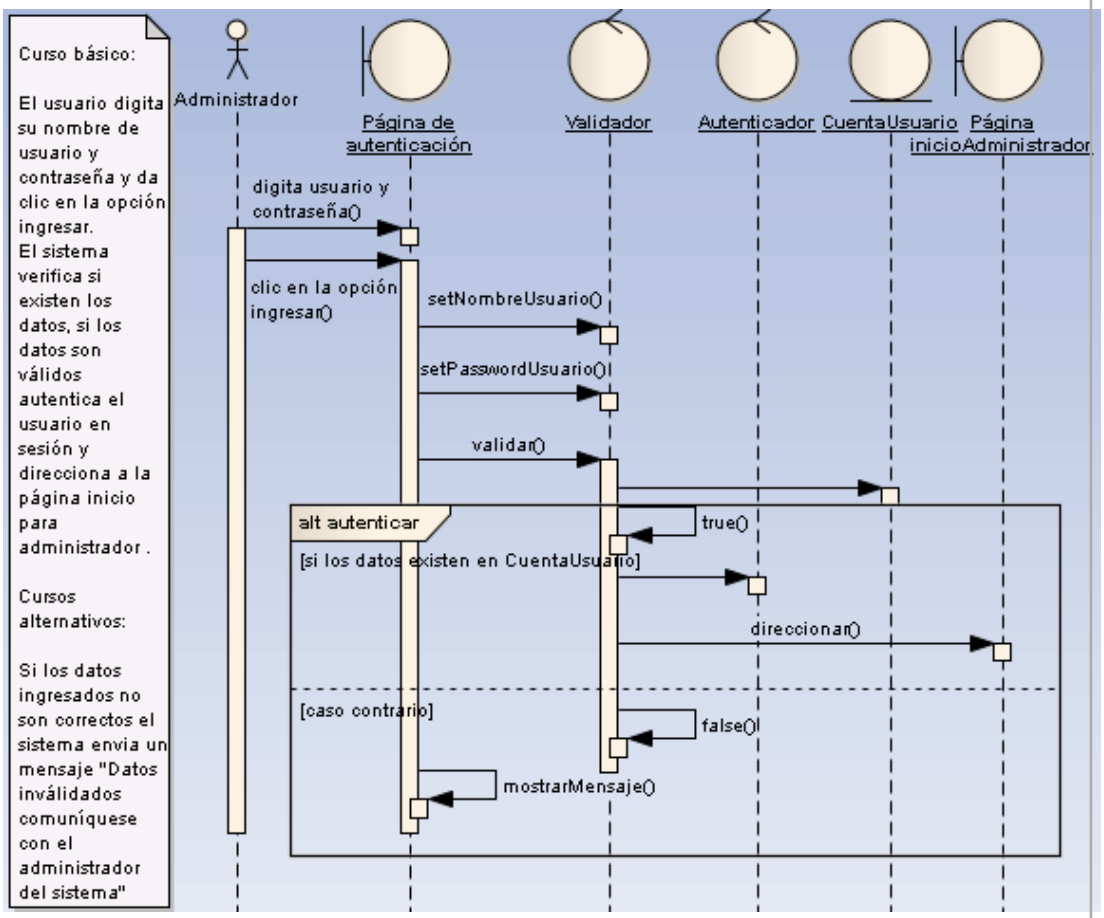
La revisión en detalle tiene como objetivo reconocer todos los elementos pertinentes para la creación del software.

3.2.1.1 Asignación del comportamiento de los objetos a las clases.

En base a los diagramas de robustez y los casos de uso a continuación de muestran los diagramas de secuencia de cada caso de uso:

- **Diagrama de secuencia del caso de uso:** Iniciar sesión

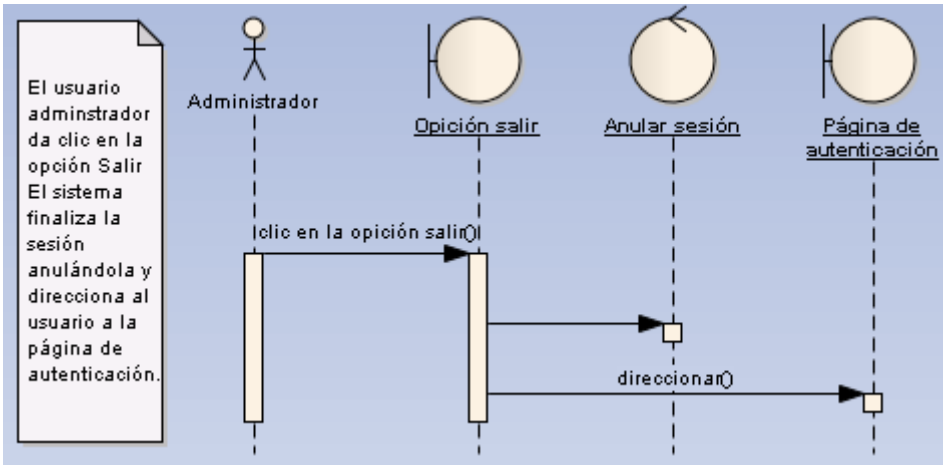
Figura 42. Diagrama de secuencia para el caso de uso Iniciar sesión



Elaborado por: Verónica Sangucho

- **Diagrama de secuencia del caso de uso:** Cerrar sesión

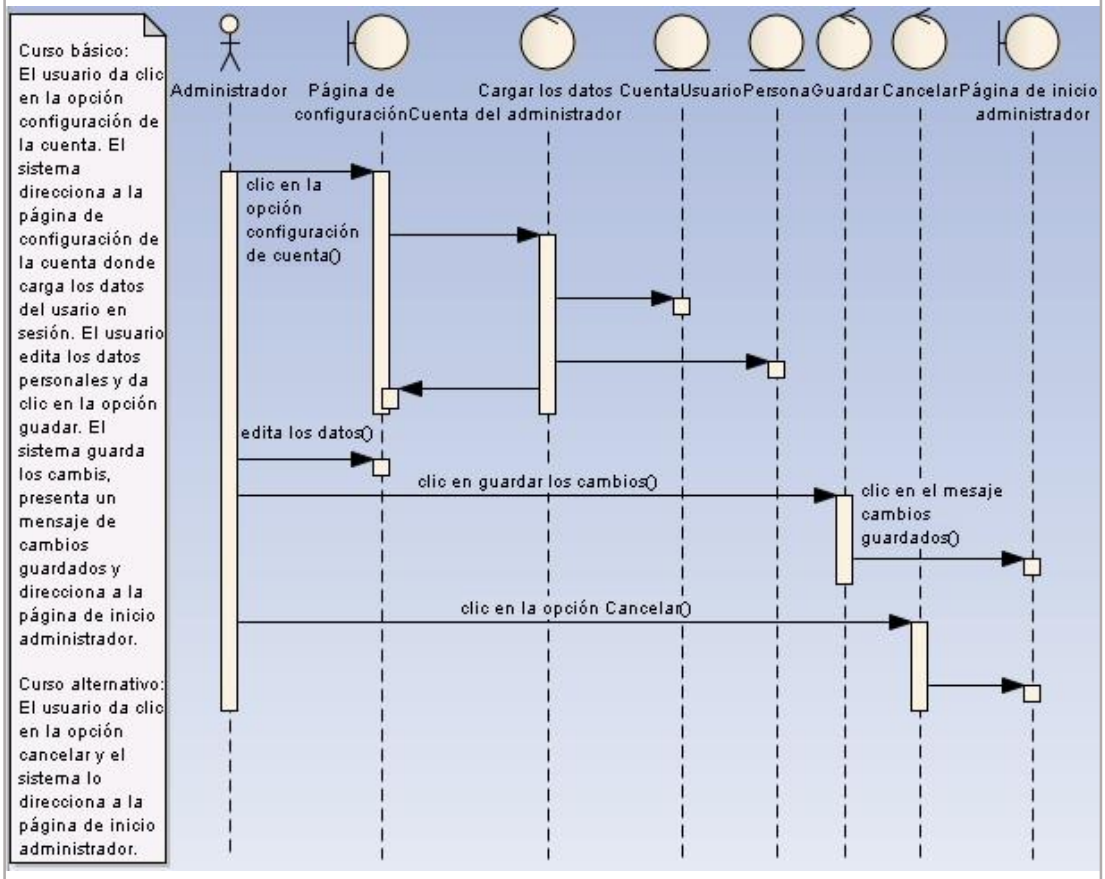
Figura 43. Diagrama de secuencia para el caso de uso Cerrar sesión



Elaborado por: Verónica Sangucho

- **Diagrama de secuencia del caso de uso:** Editar cuenta administrador

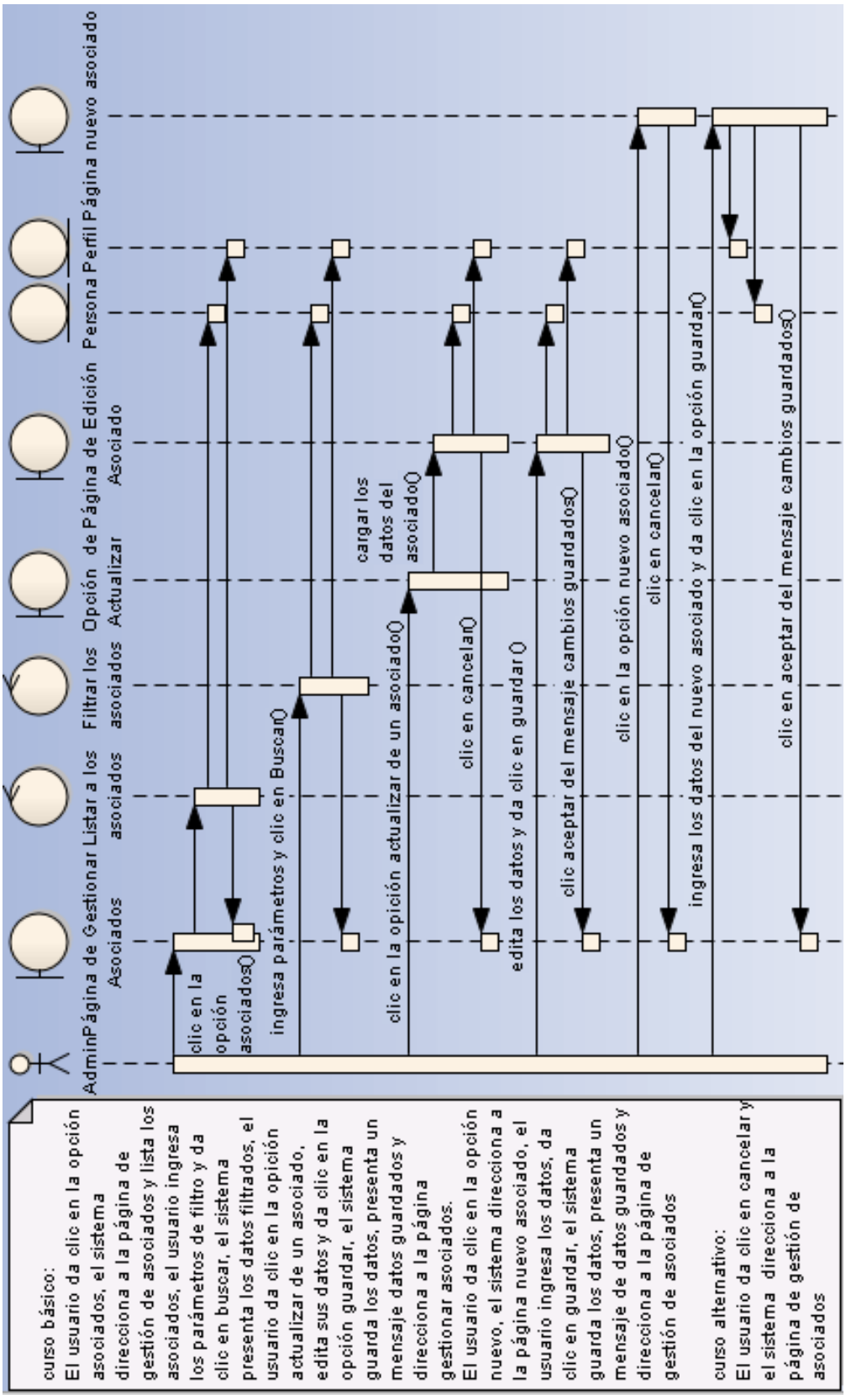
Figura 44. Diagrama de secuencia para el caso de uso *Editar cuenta administrador*



Elaborado por: Verónica Sangucho

- **Diagrama de secuencia del caso de uso:** Gestionar asociados

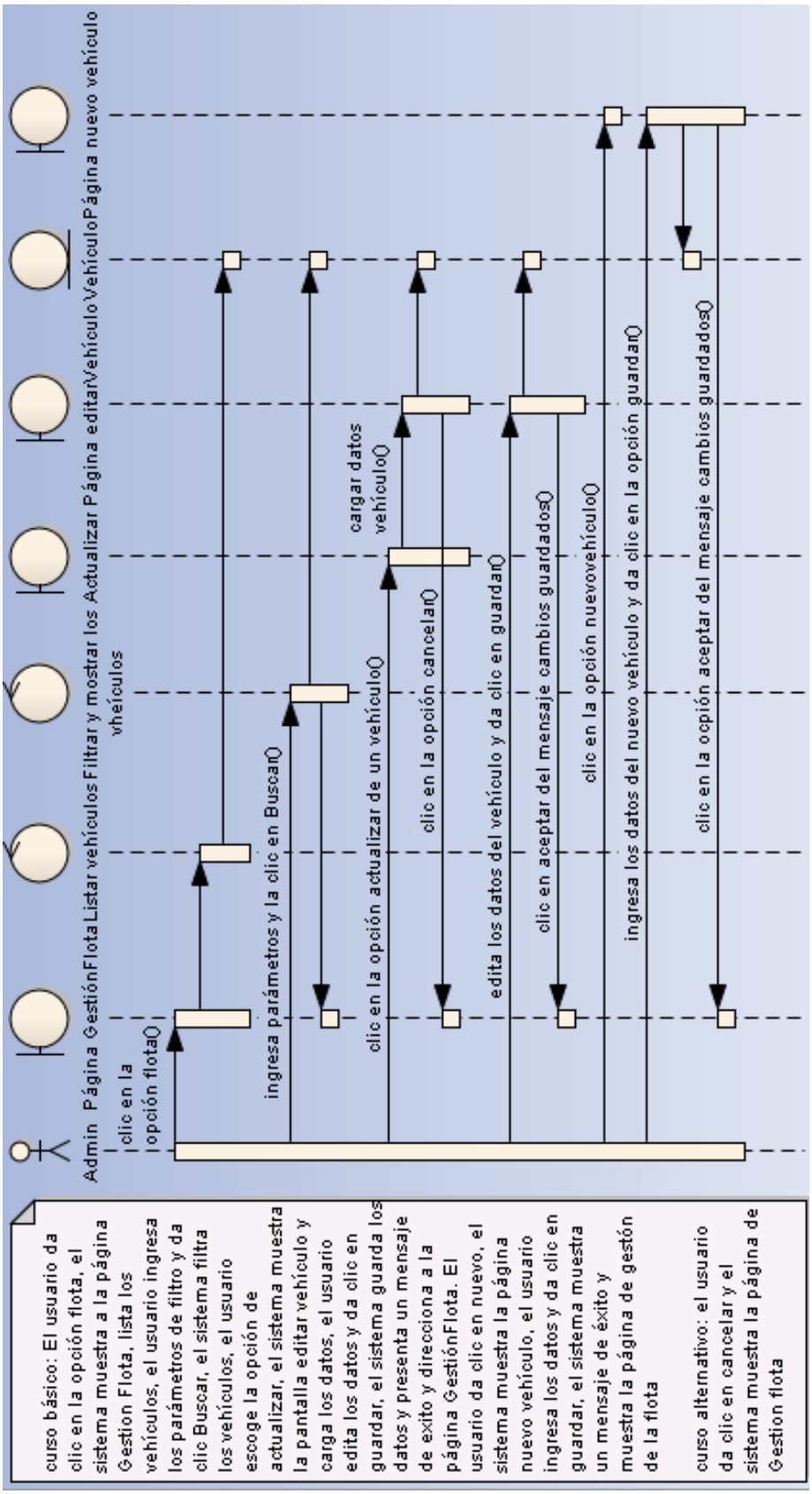
Figura 45. Diagrama de secuencia para el caso de uso *Gestionar asociados*



Elaborado por: Verónica Sangucho

- **Diagrama de secuencia del caso de uso:** Gestionar flota de vehículos

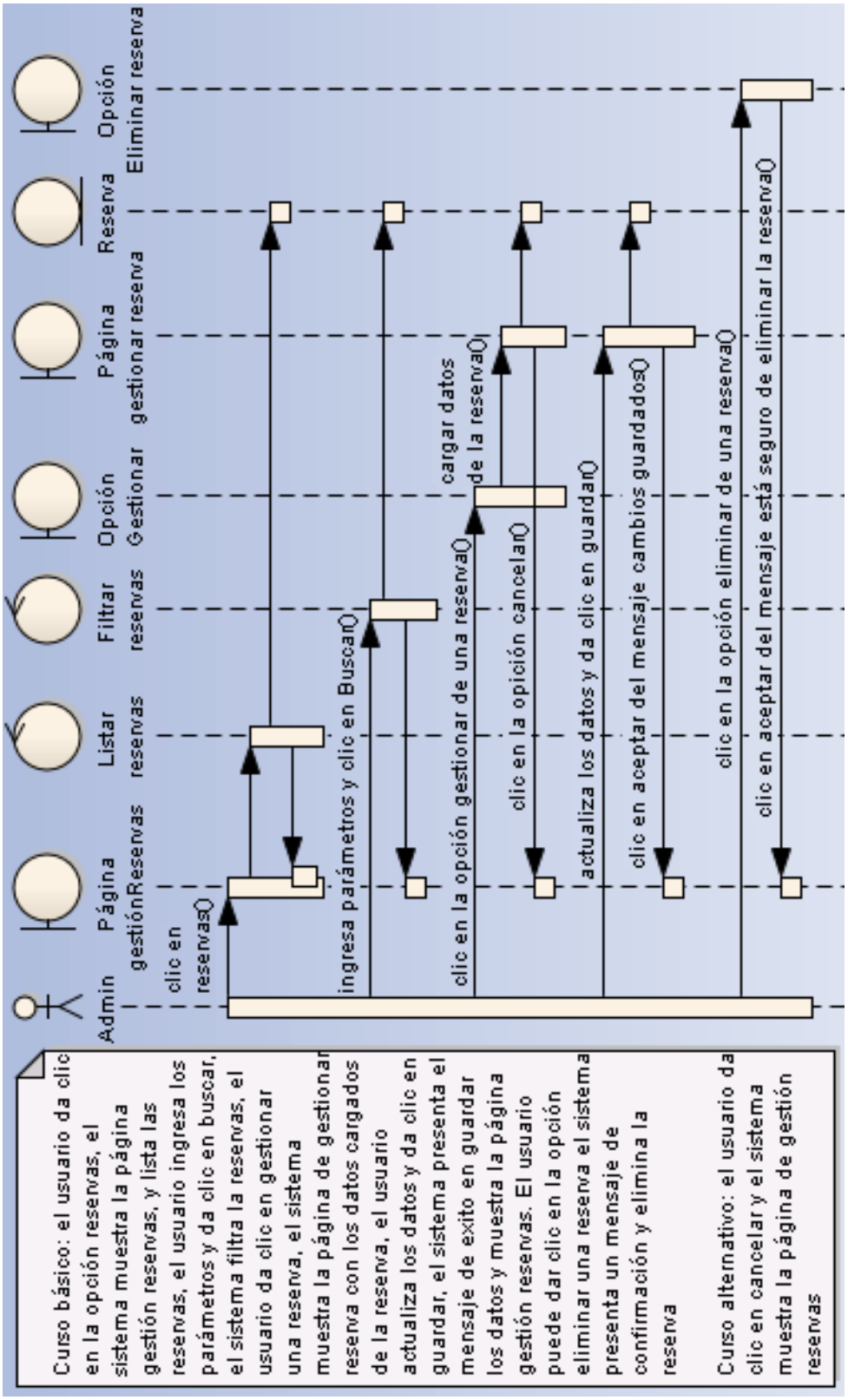
Figura 46. Diagrama de secuencia para el caso de uso *Gestionar flota de vehículos*



Elaborado por: Verónica Sangucho

- **Diagrama de secuencia del caso de uso:** Gestionar pre-reserva

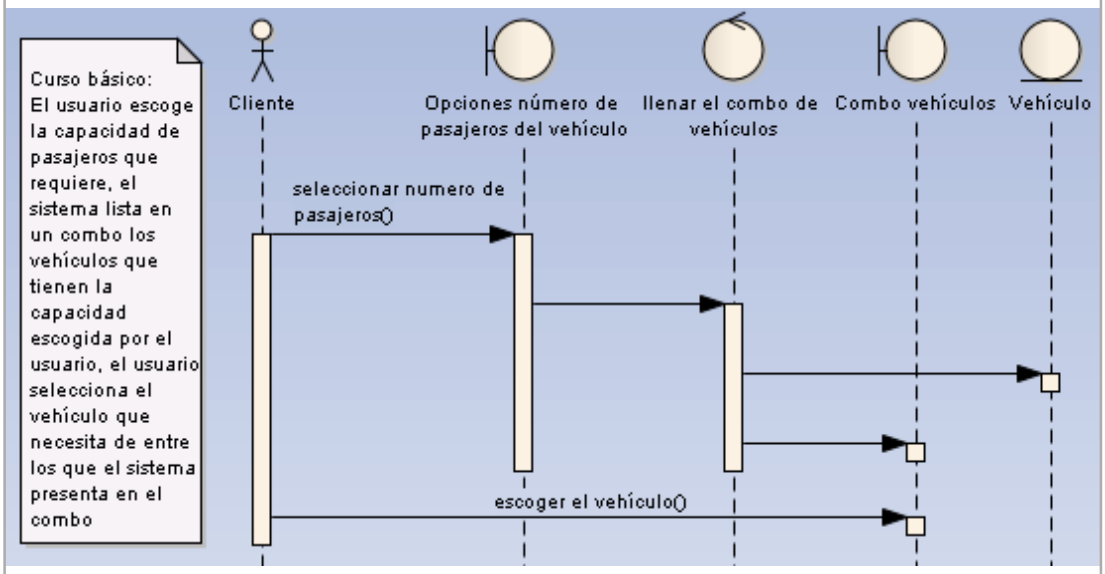
Figura 47. Diagrama de secuencia para el caso de uso Gestionar pre-reserva



Elaborado por: Verónica Sangucho

- **Diagrama de secuencia del caso de uso:** Escoger tipo de vehículo

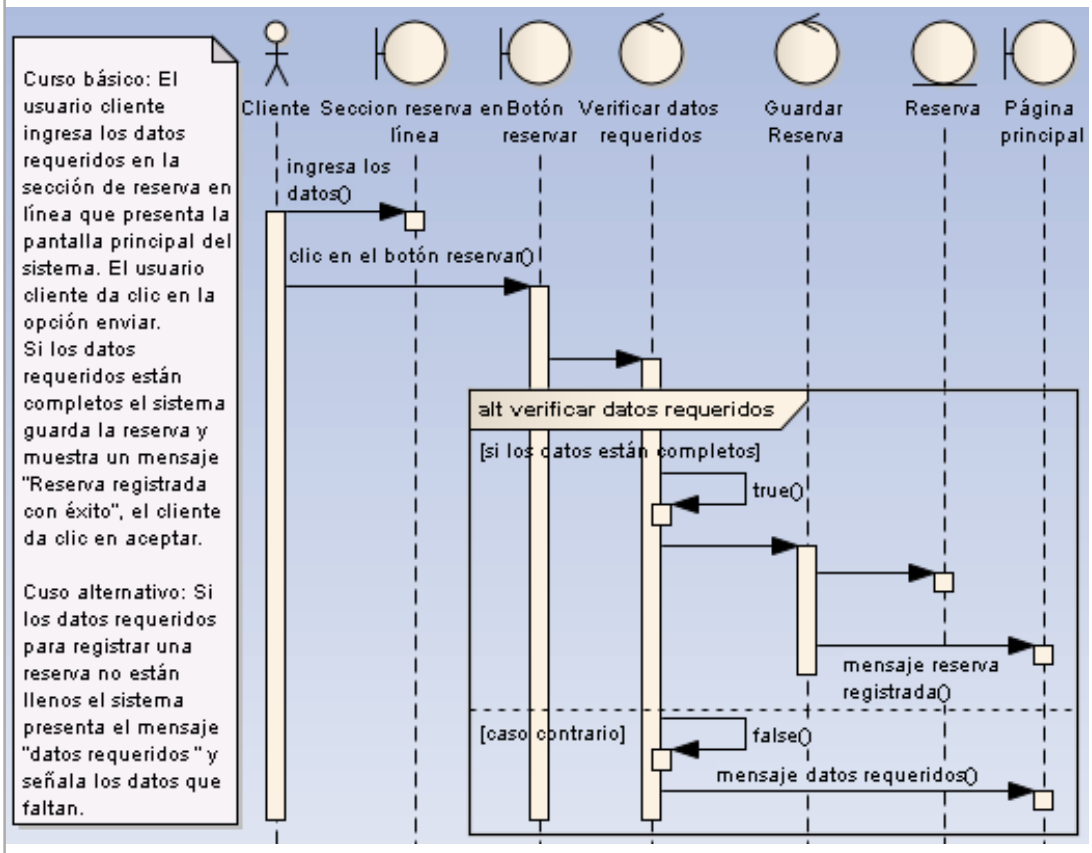
Figura 48. Diagrama de secuencia para el caso de uso Escoger tipo de vehículo



Elaborado por: Verónica Sangucho

- **Diagrama de secuencia del caso de uso:** Enviar pre-reserva en línea

Figura 49.Diagrama de secuencia para el caso de uso Enviar pre-reserva en línea



Elaborado por: Verónica Sangucho

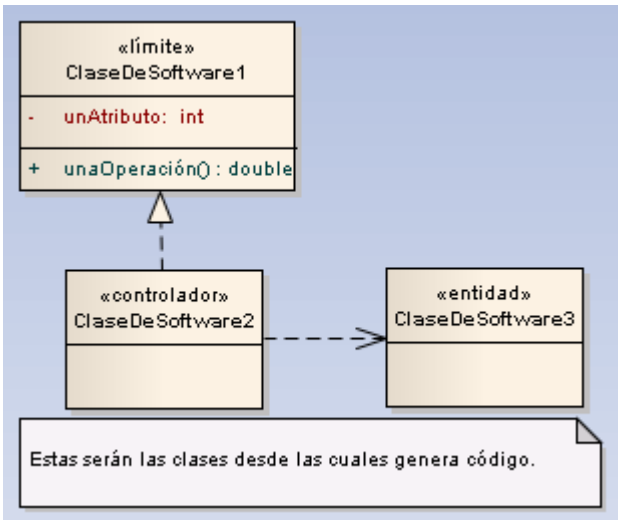
3.2.1.2 Finalizar el modelo estático.

Los diagramas de clase son los más importantes de la etapa de diseño. Estos diagramas muestran las clases que forman la estructura general de lo que se debe implementar.

Los tipos de clases que se utilizarán en el diagrama sus responsabilidades:

- **Clase <<límite>>.-** Borde, interfaz, en inglés boundary y en el patrón MVC la vista o view. Su función es captar peticiones y mostrar las respuestas.
- **Clase <<entidad>>.-** O en inglés entity y en el patrón MVC el modelo. Su función es modelar los datos para la gestión de la información como es la persistencia.
- **Clase <<control>>**.- O en inglés controller y en el patrón MVC el controller. Su función es obtener los datos del límite o/y entidad y procesar los datos realizando operaciones, cálculos, etc, y enviar las respuestas del sistema.

Figura 50. Tipos de clases



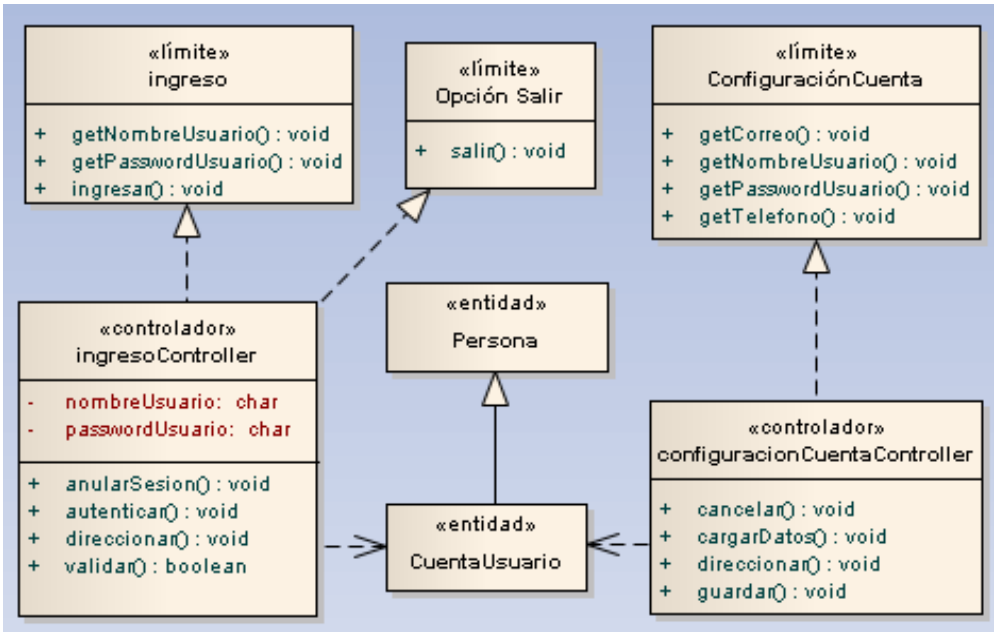
Elaborado por: Verónica Sangucho

En la anterior figura se ven las siguientes relaciones:

- Realización: se utiliza como relación de contrato o garantía, por ejemplo se pueden relacionar entre una interfaz y su clase controladora.
- Dependencia: es una relación de uso, cuando una clase utiliza otra.

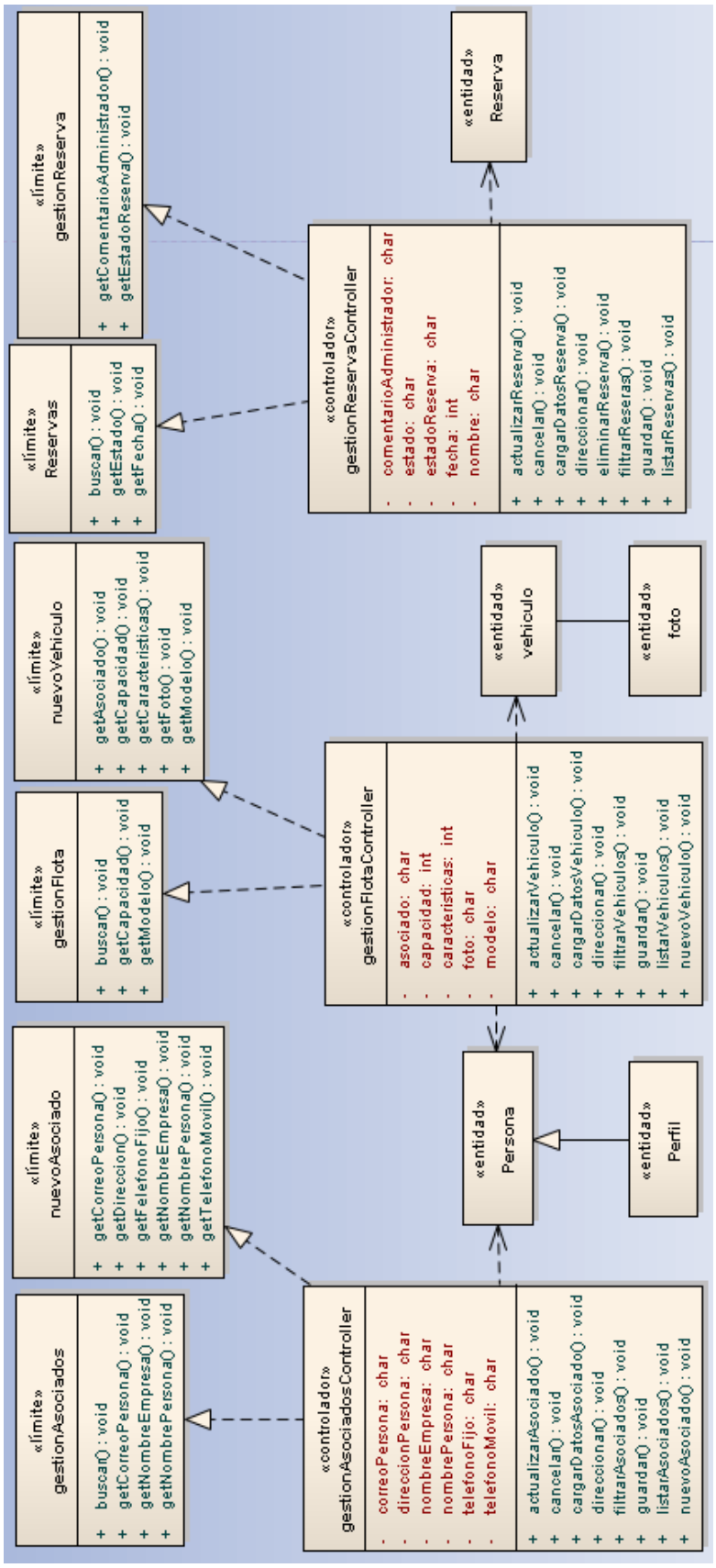
A continuación se presenta la estructura general del sistema a través de su diagrama de clases.

Figura 51.Diagrama de clases paquete General



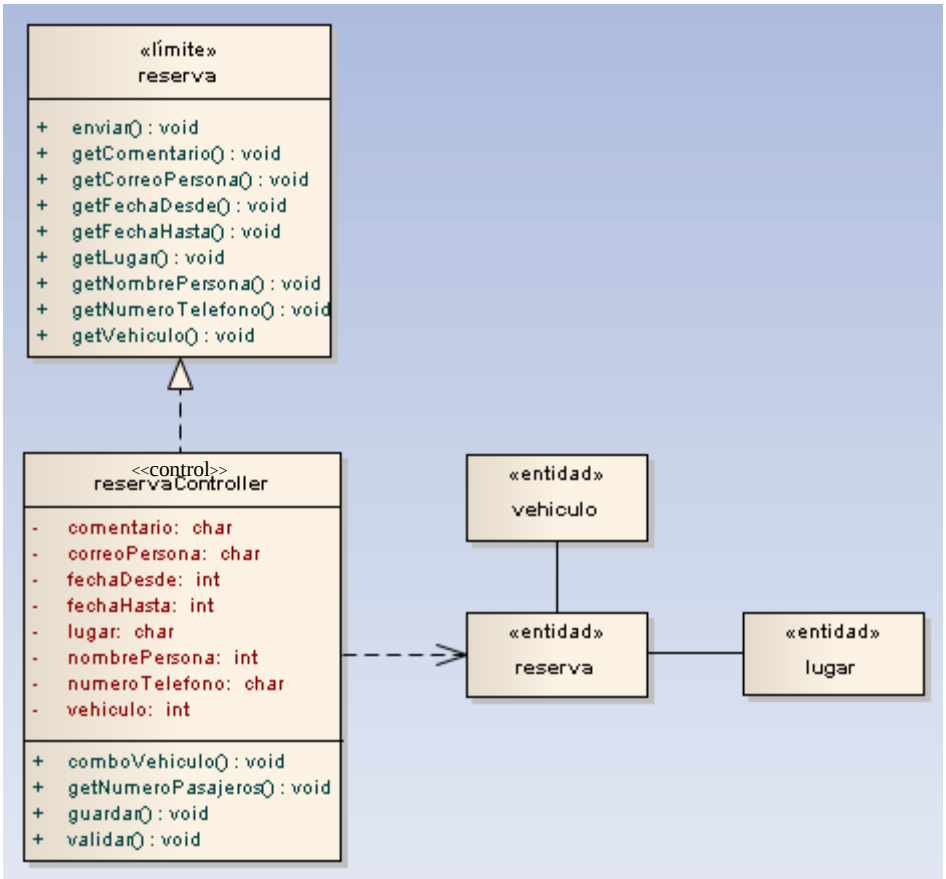
Elaborado por: Verónica Sangucho

Figura 52. Diagrama de clases paquete Administración



Elaborado por: Verónica Sangucho

Figura 53. Diagrama de clases paquete pre-reserva



Elaborado por: Verónica Sangucho

CAPÍTULO 4

IMPLEMENTACIÓN Y CONSTRUCCIÓN

4.1 Entregable de implementación

El presente capítulo muestra los resultados del desarrollo, según la aplicación del patrón de diseño MVC, el escaneo de código estático y el seguimiento de estándares fijados para el producto final.

Estándares aplicados al sistema

Las convenciones aplicadas en la nomenclatura tanto en la base de datos y los proyectos que conforman estructura principal del sistema, según su funcionalidad quedó de la siguiente manera:

4.1.1 Diccionario de datos.

Tabla 14. Diccionario de datos

TABLA: TELEFONO_TBL					
Nombre	**Tipo de dato**	**Longitud**	**Clave primaria**	**Clave secundaria**	**Description**
telefono_id	INT4	4	TRUE	FALSE	id de teléfono
persona_id	INT4	4	FALSE	TRUE	id de la persona
numeroTelefono	TEXT		FALSE	FALSE	número de teléfono
TABLA: PERSONA_TBL					
persona_id	INT4	4	TRUE	FALSE	id de persona
ciudad_id	INT4	4	FALSE	TRUE	id de ciudad
provincia_id	INT4	4	FALSE	TRUE	id de provincia
perfil_id	INT4	4	FALSE	TRUE	id de la perfil
cedulaPersona	INT8	8	FALSE	FALSE	cédula de la persona

Elaborado por: Verónica Sangucho

Continua…

Tabla 15. Diccionario de datos (continuación…)

nombrePersona	TEXT		FALSE	FALSE	nombre de persona
apellidoPersona	TEXT		FALSE	FALSE	apellido de Persona
correoPersona	TEXT		FALSE	FALSE	correo de persona
nombreEmpresa	TEXT		FALSE	FALSE	nombre de empresa
estadoPersona	VARCHAR	1	FALSE	FALSE	estado de persona
direccionPersona	TEXT		FALSE	FALSE	dirección de persona
fechaCreacion	TIMESTAMP		FALSE	FALSE	fecha de la creación de persona
usuarioIdCreacion	INT4	4	FALSE	FALSE	id del usuario que creo la persona
fechaModificacion	TIMESTAMP		FALSE	FALSE	fecha de modificación de la persona
usuarioIdModificacion	INT4	4	FALSE	FALSE	id del usuario que modificó la persona
VEHICULO_TBL					
vehiculo_id	INT4	4	TRUE	FALSE	id de vehículo
persona_id	INT4	4	FALSE	TRUE	id de persona
modeloVehiculo	TEXT		FALSE	FALSE	modeo de vehículo
capacidadVehiculo	INT4	4	FALSE	FALSE	capacidad de vehículo
caracteristicasVehiculo	TEXT		FALSE	FALSE	características de vehículo
fechaCreacion	TIMESTAMP		FALSE	FALSE	fecha creación de vehículo
fechaModificacion	TIMESTAMP		FALSE	FALSE	fecha modificación de vehículo
usuarioIdModificacion	INT4	4	FALSE	FALSE	id del usuario que modificó el vehículo
usuarioIdCreacion	INT4	4	FALSE	FALSE	id del usuario que creó el vehículo
estadoVehiculo	VARCHAR	1	FALSE	FALSE	estado de vehículo
FOTO_TBL					
foto_id	INT4	4	TRUE	FALSE	id de foto
vehiculo_id	INT4	4	FALSE	TRUE	id de vehículo

Elaborado por: Verónica Sangucho

Continua…

Tabla 16. Diccionario de datos (continuación…)

urlFoto	TEXT		FALSE	FALSE	url de foto
RESERVA_TBL					
reserva_id	INT4	4	TRUE	FALSE	id de reserva
vehiculo_id	INT4	4	FALSE	TRUE	id de vehículo
fechaReserva	TIMESTAMP		FALSE	FALSE	fecha de reserva
fechaDesde	TIMESTAMP		FALSE	FALSE	fecha desde de reserva
fechaHasta	TIMESTAMP		FALSE	FALSE	fecha hasta de reserva
comentarioCliente	TEXT		FALSE	FALSE	comentario del cliente
comentarioAdministrador	TEXT		FALSE	FALSE	comentario del administrador
lugarSalida	TEXT		FALSE	FALSE	lugar salida de reserva
lugarLlegada	TEXT		FALSE	FALSE	lugar llegada de reserva
estadoReserva	VARCHAR	1	FALSE	FALSE	estado de reserva
usuarioIdModificacion	VARCHAR	1	FALSE	FALSE	id del usuario que modificó la reserva
fechaModificacion	TIMESTAMP		FALSE	FALSE	fecha de modificación de la reserva
nombreCliente	TEXT		FALSE	FALSE	nombre del cliente
telefonoCliente	TEXT		FALSE	FALSE	teléfono del cliente
correoCliente	TEXT		FALSE	FALSE	correo del cliente
precioCarrera	DECIMAL		FALSE	FALSE	precio de la carrera
PROVINCIA_TBL					
provincia_id	INT4	4	TRUE	FALSE	id de provincia
nombreProvincia	TEXT		FALSE	FALSE	nombre de provincia
LUGAR_TBL					
lugar_id	INT4	4	TRUE	FALSE	id de lugar
reserva_id	INT4	4	FALSE	TRUE	id de reserva

Elaborado por: Verónica Sangucho

Continua…

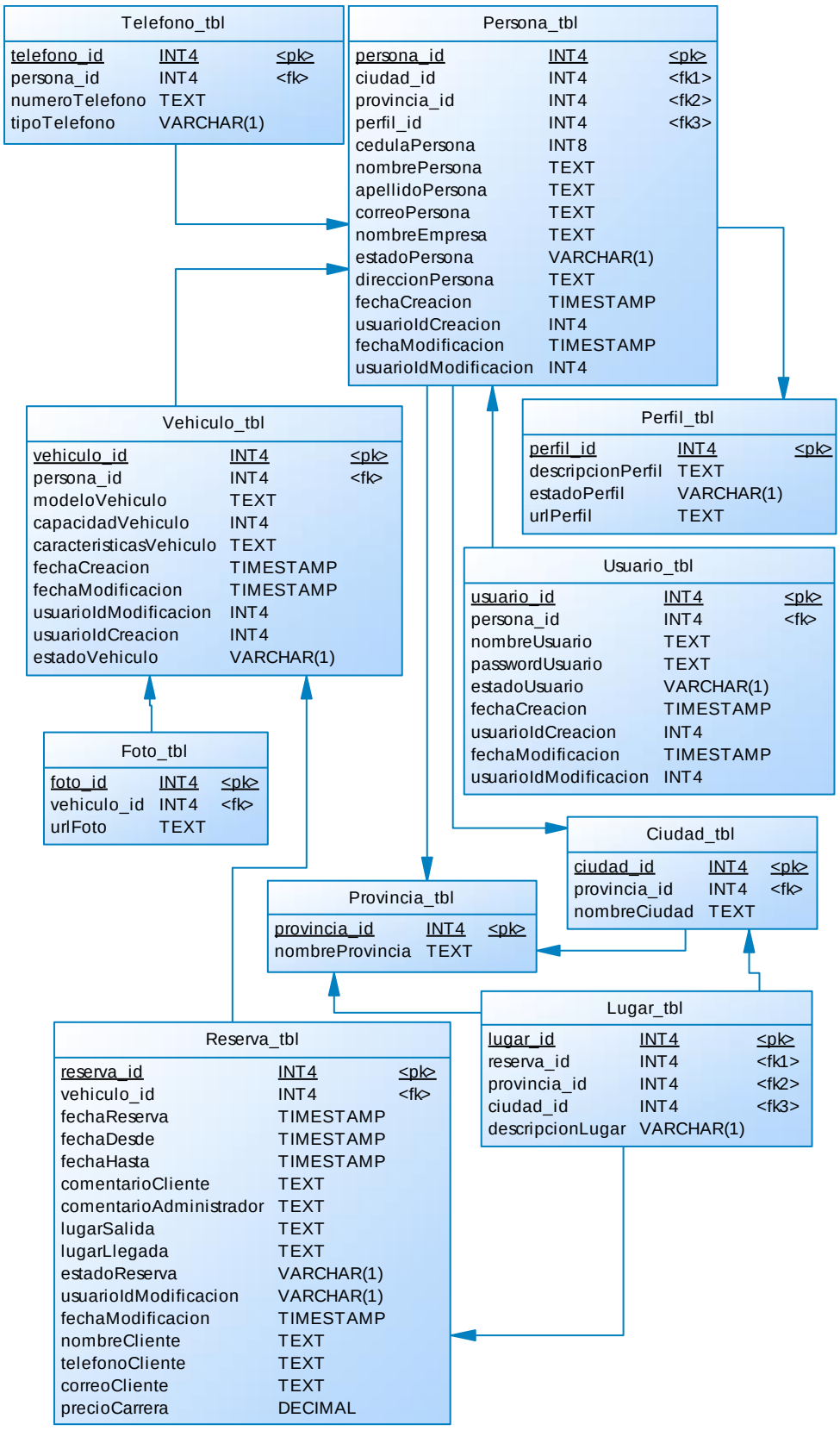
Tabla 17. Diccionario de datos (continuación…)

provincia_id	INT4	4	FALSE	TRUE	id de provincia
ciudad_id	INT4	4	FALSE	TRUE	id de ciudad
descripcionLugar	VARCHAR	1	FALSE	FALSE	descripción de lugar
CIUDAD_TBL					
ciudad_id	INT4	4	TRUE	FALSE	id de ciudad
provincia_id	INT4	4	FALSE	TRUE	id de provincia
nombreCiudad	TEXT		FALSE	FALSE	nombre de ciudad
USUARIO_TBL					
usuario_id	INT4	4	TRUE	FALSE	id de usuario
persona_id	INT4	4	FALSE	TRUE	id de persona
nombreUsuario	TEXT		FALSE	FALSE	nombreUsuario de usuario
passwordUsuario	TEXT		FALSE	FALSE	passwordUsuario de usuario
estadoUsuario	VARCHAR	1	FALSE	FALSE	estadoUsuario de usuario
fechaCreacion	TIMESTAMP		FALSE	FALSE	fechaCreacion de usuario
usuarioIdCreacion	INT4	4	FALSE	FALSE	id del usuario que creo el usuario
fechaModificacion	TIMESTAMP		FALSE	FALSE	fechaModificacion del usuario
usuarioIdModificacion	INT4	4	FALSE	FALSE	id del usuario que modificó el usuario
PERFIL_TBL					
perfil_id	INT4	4	TRUE	FALSE	id de perfil
descripcionPerfil	TEXT		FALSE	FALSE	descripción de perfil
estadoPerfil	VARCHAR	1	FALSE	FALSE	estado de perfil
urlPerfil	TEXT		FALSE	FALSE	url al que tiene acceso el perfil

Elaborado por: Verónica Sangucho.

4.1.2 Esquema de la base de datos.

Figura 54. Esquema de la base de datos



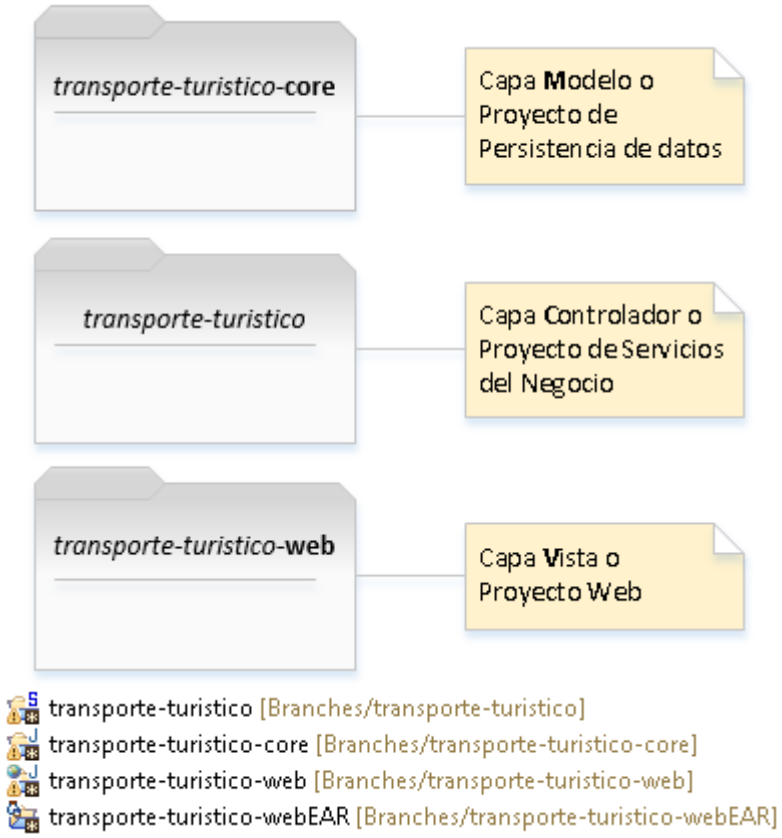
Elaborado por: Verónica Sangucho

4.1.3 Nomenclatura de los proyectos para el sistema.

Las convenciones para los nombres de los proyectos que conforman estructura principal de la arquitectura del sistema, según sus características y/o funcionalidades y que reflejan el patrón de diseño MVC quedaron de la siguiente manera:

- **transporte-turistico-core:** Es la Capa Modelo, se encarga de la persistencia y recuperación de los datos ya que en ella se encuentra mapeado las tablas a través de Objetos de Transferencia de Datos o DTOs (Data transfer Object). Es la capa más distante del usuario, por lo cual se lo nombró al proyecto con la terminación **core** ya que core en su traducción al español significa centro, interno o profundo.
- **transporte-turistico:** Es la capa Controlador en la que se encuentra implementado la lógica del negocio del transporte turístico de la empresa.
- **transporte-turistico-web:** Capa Vista, o proyecto web es el límite del proyecto y la interfaz con la que interactúa con el usuario final.

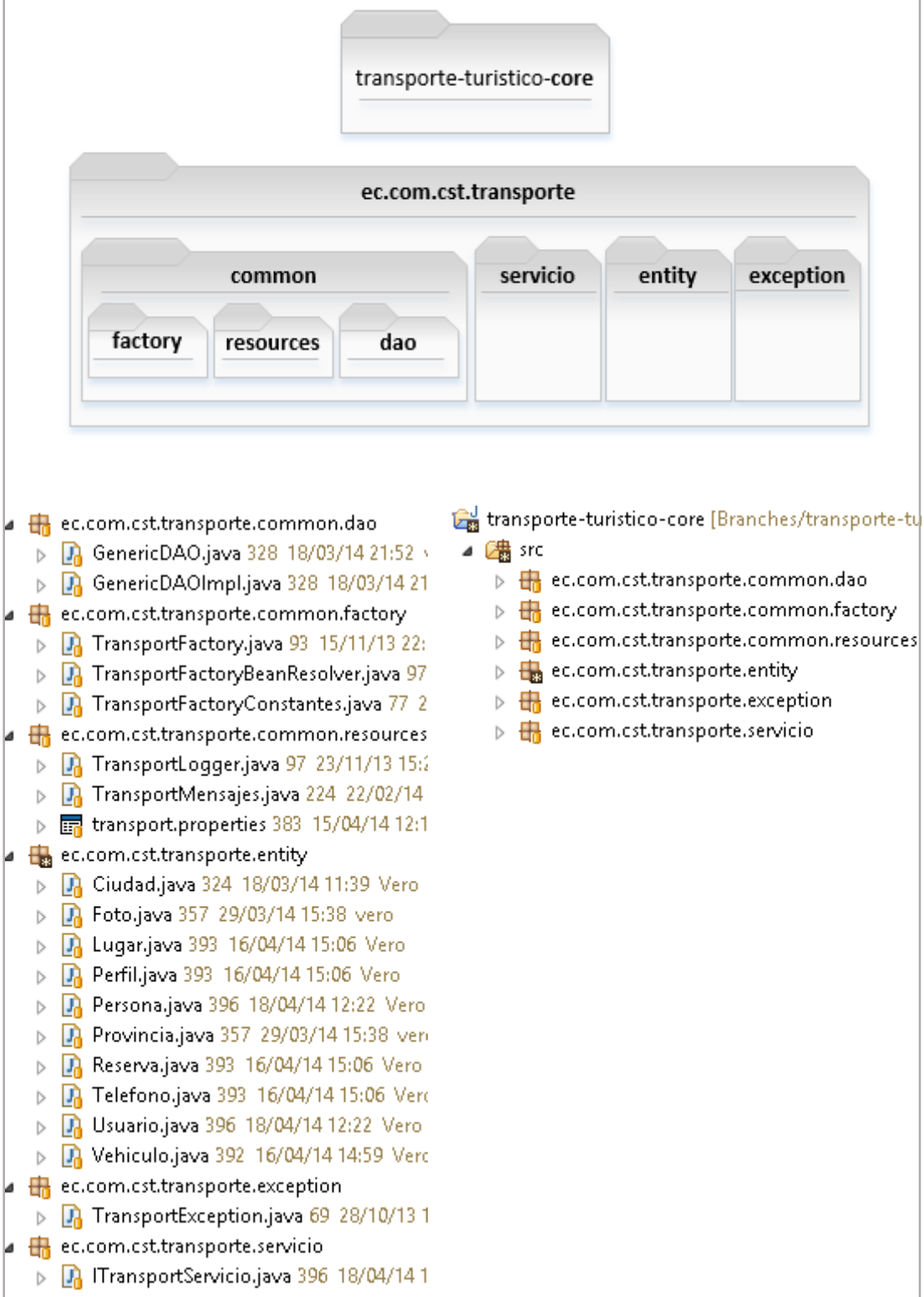
Figura 55. Nomenclatura en los proyectos del el presente sistema



Elaborado por: Verónica Sangucho

4.1.4 Nomenclatura de los paquetes del proyecto de persistencia.

Figura 56. Nomenclatura en los paquetes del proyecto persistencia del producto



Elaborado por: Verónica Sangucho.

Descripción de las clases dentro de los paquetes en el proyecto de persistencia:

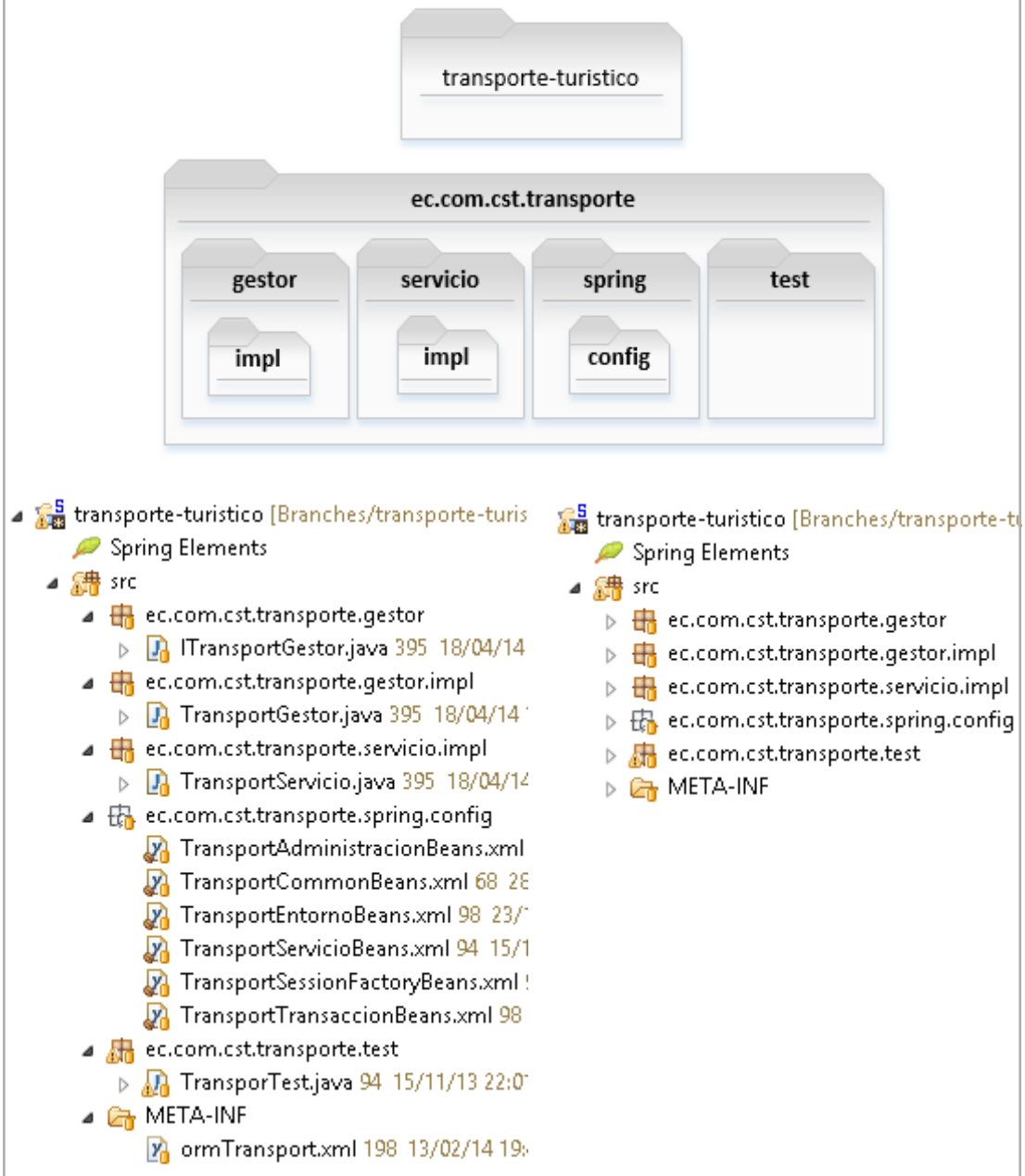
- **Paquete Dao.- GenericDAO, GenericDAOImpl:** un Dao o Data Acces Object, es un componente de software que facilita el acceso a datos, cada entidad necesita un DAO, pero como cada DAO tiene un comportamiento similar, crear, recuperar, actualizar, eliminar, creamos un DAO genérico que utiliza para ello hibernate a través de la extensión HibernateDaoSupport y lo inyectamos con ayuda de Spring utilizando la inyección de dependencia en la capa de servicios en la clase TransporGestor.
- **Paquete Factory:TransportFactory:** Crea el contenedor de Spring, el cual es el encargado del manejo del concepto de Inversión de control IoC o Inyección de dependencias DI, es decir, en vez de que sea el mismo objeto quien se encargue de instanciar a los otros objetos con los que necesita trabajar, el contenedor inyecta estas dependencias.
- **Paquete Factory: TransportFactoryConstantes, TransportFactoryBeanResolver:** Proveen el marco de configuraciones sobre los objetos y sus dependencias, y las rutas de los archivos xml de configuración del spring para el contenedor.
- **Paquete Resources: TransportFactory, TransportLogger:** Es un utilitario que produce logs o registros que son necesarios para saber lo que está sucediendo en la aplicación en tiempo de ejecución, tanto en el desarrollo como en producción:
- **Paquete Resources: TransportMensajes:** A través el utilitario ResourceBundle localiza y lee los mensajes o parámetros configurables de la aplicación ubicados en el archivo de propiedades **transport.properties** y transforma el mensaje en el tipo de dato deseado ya sea String, Long o integer.
- **Paquete Entity: Ciudad, Foto, Lugar, Perfil, Persona, Provincia, Reserva, Telefono, Usuario, Vehiculo:** estas clases representan el mapeo de la base de datos el sistema, cada clase representa una entidad o **DTO** o Data Transfer Object utilizadas para la transferencia de datos a la base de datos.
- **Paquete Exception: TransportException:** esta clase nos permite el manejo de excepciones no comprobadas, es decir, que no necesitan ser capturados en un bloque catch, por lo cual el programador al llamar el método no está obligado a

capturarlo ni a manejarlo, se utiliza en los casos en los que no se trata de una condición anormal que el programador cliente deba decidir conscientemente cómo manejar y lanzar una excepción comprobada.

- **Paquete Servicio: ITransportServicio:** Interfaz de los servicios que serán utilizados en los controllers de la parte web su implementación se encuentra en el proyecto de servicio del sistema transporte-turistico.

4.1.5 Nomenclatura de paquetes del proyecto de servicios del sistema.

Figura 57. Nomenclatura en los paquetes del proyecto de servicio del producto



Elaborado por: Verónica Sangucho.

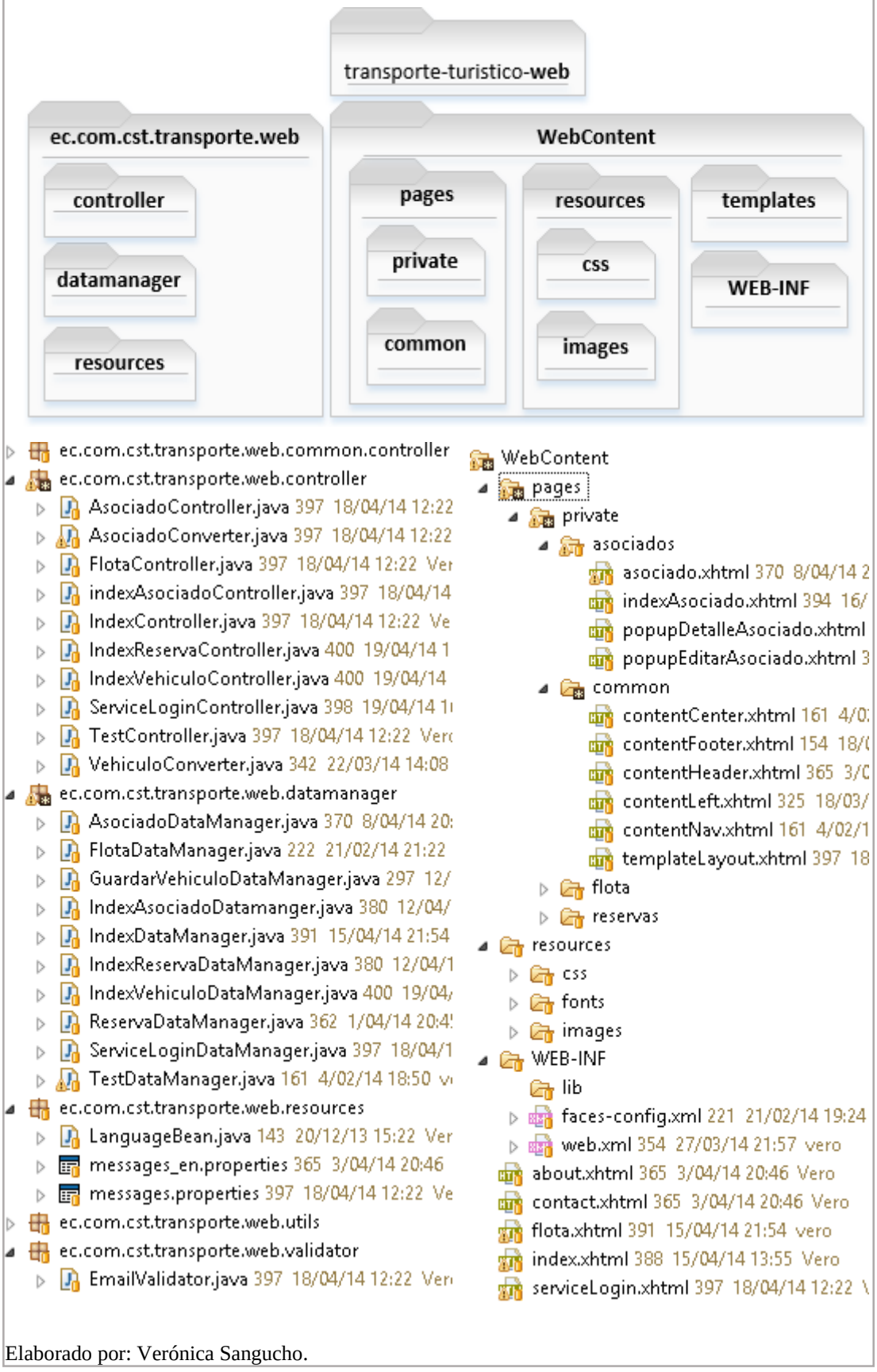
Descripción de las clases dentro de los paquetes en el proyecto de servicio:

- **Paquete Gestor: ITransportGestor, TransportGestor:** Interfaz e implementación del gestor de transacciones del spring, donde están los métodos transaccionales, insertar, actualizar, eliminar, buscar.

  Para dichos métodos utilizan *TransportException* que extiende de *RumetimeException* ya que provocará rollback en caso de alguna exception.
- **Paquete Servicio Impl: TransportServicio:** Implementación de los servicios que serán invocados directamente en los controlleres de la capa web.
- **Paquete Spring Config :**
- **TransportAdministracionBeans.xml:** Archivo de configuración del spring donde se encuentra especificado la clase que contiene los métodos transaccionales y la inyección del gerericDao que lo utiliza en los métodos.
- **TransportCommonBeans.xml:** Archivo de configuración que ubica la clase que contiene las dependencias dinámicas según lo especifique la clase *TransportFactoryBeanResolver*.
- **TransportEntornoBeans.xml:** Archivo de configuración del dataSourse para la conexión directa con la base de datos y la inyección de hibernate para los métodos transaccionales.
- **TrasnportServicoBeans.xml:** Archivo de configuración de la inyección de dependencia la interfaz *ITransportGestor* de la capa de servicio en *TransportServicio* de la capa de persstencia.
- **TransportSessionFactoryBeans.xml:** Archivo de configuración del *SessionFactory* responsable de abrir, cerrar y gestionar las sesiones de *Hibernate*, y la configuración para el mapeo de la base de datos para la clase *TransportFactoryBeanResolver*.
- **TransportTransaccionBeans.xml:** Archivo de configuración del procedimiento de transacciones del proyecto, de manera que realice las búsquedas en modo lectura y las demás tracciones en modo lectura/escritura.
- **Carpeta META-INF: ormTransport.xml:** Archivo de configuración que especifica las rutas de las entidades, utilizada en *TransportSessionFactoryBeans.*

4.1.6 Nomenclatura del proyecto web del producto.

Figura 58. Nomenclatura de los paquetes del proyecto web del producto



Elaborado por: Verónica Sangucho.

Descripción de las clases que se encuentran dentro de los paquetes en el proyecto de servicio:

- **Paquete Common Controller: CommonController:**
  Controlador común para inicializar los controladores de las pantallas a través del parámetro *form* que es enviado desde cada controlador.
- **Paquete Controller: AsociadoController, FlotaController , indexAsociadoController, IndexController, idexReservaController, IndexVehiculoController, ServiceLoginController:**
  Son las clases *bean* para la administración o gestión de las pantallas y los popups, para ello utilizan las anotaciones:
  @ManagedBean.- vincula la clase o *bean* al marco de trabajo *Java Server Faces*.
  @ViewScoped.- establece el alcance, ámbito de sesión o tiempo de vida de la clase o *bean*, en este el *bean* permanece siempre y cuando el usuario esté interactuando con la misma vista JSF.
  @ManagedProperty.- inyecta en la clase o un *bean* de respaldo o *Backing Bean* en este caso el *DataManager*.
  @PostConstruct.- indica a un método que debe ejecutarse después de la construcción del bean.
- **Paquete DataManager: AsociadoDataManager, FlotaDataManager, GuardarVehiculoDataManager, IndexAsociadoDatamanger, IndexDataManager, IndexReservaDataManager, IndexVehiculoDataManager, ReservaDataManager, ServiceLoginDataManager:**
  Son las clases *bean* de respaldo en el cual se declaran las variables que necesitamos que se mantengan en sesión para ello utiliza las siguientes anotaciones:
  @ManagedBean.- vincula la clase al marco de trabajo *Java Server Faces*.
  @SessionScoped.- establece el alcance, ámbito de sesión o tiempo de vida de la clase o *bean*, en este el *bean* permanece siempre y cuando viva la sesión http y se destruye cuando la sesión se invalida.
- **Paquete Resources: LanguageBean, messages_en, messages:**
  Clase y archivos de propiedades para la internacionalización de la página web.

- **Paquete Controller: AsociadoConverter:**

  Sirve para buscar un asociado en tiempo de ejecución. Mientras va ingresando los caracteres le ayuda al usuario a completar el texto que escribe, a través de la implementación de la interfaz *Converter* la cual puede transformar un objeto a un String y un String a un Objeto.
- **Paquete Util: TransportUtil:**

  Clase utilitario para la parte web, aquí se ubican los métodos utilizados en varias pantallas como el validador de cédula y la variable que contiene el número de provincias del Ecuador.
- **Carpeta pages private:**
- **indexAsociado.xhtml:**

  Página web para la búsqueda, edición y creación de un nuevo asociado.
- **popupDetalleAsociado.xhml:**

  Página web para el popup que puesta el detalle del asociado seleccionado.
- **popupEditarAsociado.xhml:**

  Págian web para el popup de edición de un asociado seleccionado.
- **templateLayout.xhml, contentCenter.xhml, contentFooter.xhml, contentHeader.xhml, contentLeft.xhml, contentNav.xhml:**

  Páginas template o plantilla y los elementos que contienen para todas las páginas web de parte de administración.
- **indexVehiculo.xhtml:**

  Página web para la búsqueda, edición y creación de un nuevo vehículo.
- **popupDetalleVehiculo.xhml:**

  Página web para el popup que muestra el detalle del vehículo.
- **popupEditarVehiculo.xhml:**

  Págian web para el popup de edición de un vehículo seleccionado.
- **indexReserva.xhml:**

  Página web para la búsqueda y gestión de una reserva realizada.
- **popupBuscarVehiculo.xhml:**

  Página web para el popup para buscar un vehículo.
- **popupGestionarReserva.xhml:**

  Página web para el popup para buscar un vehículo.
- **index, contact, about, services.xhml:**

  Páginas web de información para la página index.

4.1.7 Configuraciones en la capa modelo.

Dentro de la capa modelo o proyecto para la persistencia de los datos en el caso del presente sistema el proyecto *transporte-turistico-core*, las configuraciones de la infraestructura Spring se encuentran dentro del paquete *factory* (fábrica), así:

- **TransportFactory:**

```
/**
 * <b>
 *Crea el contenedor de Spring.
 * </b>
 * <p>[Author: Vero, Date: 23/04/2014]</p>
 *
 * @return instancia
*/
private TransportFactory() {
    try {
        context = new ClassPathXmlApplicationContext
                (TransportFactoryBeanResolver.getInstancia()
                .getTransportSpringContextPaths());
        } catch (Exception e) {
            throw new TransportException("error creando el contenedor Spring",e);
                }
        }
```

Esta clase crea el contenedor Spring dentro del cual ubica y gestiona los objetos e inyecta o incluye los respectivos objetos que necesita para trabajar.

- **TransportFactoryBeanResolver.-**

```
transportSpringContextPathList.add(
        TransportFactoryConstantes.CONTEXTPATH_SESSION_FACTORY);
transportSpringContextPathList.add(
        TransportFactoryConstantes.CONTEXTPATH_ADMINISTRACION);
transportSpringContextPathList.add(
        TransportFactoryConstantes.CONTEXTPATH_TRANSACCION);
```

La clase crea la lista de las variables con las rutas de los archivos de configuración donde se encuentran los objetos y sus dependencias

- **TransportFactoryConstantes.-**

```
public interface TransportFactoryConstantes {
    static final String CONTEXTPATH_SESSION_FACTORY =
TransportMensajes.getInstance().getMessageForKey(
                    "ec.com.cst.transporte.spring.config.sessionFactory");
    static final String CONTEXTPATH_ADMINISTRACION =
TransportMensajes.getInstance().getMessageForKey(
                    "ec.com.cst.transporte.spring.config.administracion");
```

En esta clase se encuentran las variables que contienen las rutas de los archivos de configuración para el contenedor Spring

4.1.8 Configuraciones en la capa controlador.

Dentro de la capa controlador o proyecto para la implementación del modelo de negocio en el caso del presente sistema el proyecto *Transporte-turistico*, las configuraciones de la infraestructura Spring se encuentran dentro del paquete *Sprinng.config*, así:

- **TransportAdministracionBeans.xml.-**

```
<bean id="sisTransportGestor"class= "ec.com.cst.transporte.gestor.impl.TransportGestor" >
        <property name="genericDAO" ref="genericDAO" />
</bean>
        <bean  id="genericDAO" class="ec.com.cst.transporte.common.dao.GenericDAOImpl">
                <property name="sessionFactory" ref="sessionFactory" />
</bean>
```

Así, se realiza la inyección o inclusión de los métodos transacciones del genericDaoImpl en TransportGestor.

- **TransportCommonBeans.xml.-**

```
<bean  id="sisTransportFactoryBeanResolver"
class="ec.com.cst.transporte.common.factory.TransportFactoryBeanResolver"  factory-
method="getInstancia"/>
```

Ubica la ruta de la clase que contiene todas las dependencias que necesita el sistema para el manejo de las inyecciones, es decir, la clase *TransportFactoryBeanResolver*.

- **TransportEntornoBeans.xml.-**

```
<property name="driverClassName">
        <value>org.postgresql.Driver</value>
</property>
<property name="url">
        <value>jdbc:postgresql://localhost:5432/CSTTRANSPORT</value>
</property>

<bean id="sisTransportTransactionManager"
                class="org.springframework.orm.hibernate3.HibernateTransactionManager">
                <property name="sessionFactory">
                        <ref bean="sessionFactory" />
                </property>
        </bean>
```

Configuración para la conexión directa con la base de datos e inyección de hibernate

- **TrasnportServicoBeans.xml.-**

```
<bean id="sisTransportServicio" class="ec.com.cst.transporte.servicio.impl.TransportServicio">
    <property name="transportGestor">
        <ref bean="sisTransportGestor"/>
    </property>
</bean>
```

Archivo de configuración de la inyección de dependencia la interfaz *ITransportGestor* de la capa de servicio en *TransportServicio* de la capa de persistencia de datos.

- **TransportSessionFactoryBeans.xml.-**

```
<property name="mappingResources" value="#{sisTransportFactoryBeanResolver.archivosOrm}"
/>
                <prop key="hibernate.dialect">org.hibernate.dialect.PostgreSQLDialect</prop>
                <prop key="hibernate.connection.autocommit">false</prop>
                <prop key="hibernate.connection.pool_size">20</prop>
```

Configura el *SessionFactory* responsable de gestionar las sesiones de *Hibernate*, y la configuración para el mapeo de la base de datos para la clase *TransportFactoryBeanResolver*.

- **TransportTransaccionBeans.xml.-**

```
<tx:advice id="txAdvice" transaction-manager="sisTransportTransactionManager">
    <tx:attributes>
            <!-- Las busquedas hacen acceso a la base con transacciones en modo lectura -->
                <tx:method name="*" read-only="true" rollback-for="Throwable"/>
            <!-- Todos los demas metodos son transaccionales en modo lectura/escritura -->
                <tx:method name="trans*" rollback-for="Throwable"/>
            </tx:attributes>
</tx:advice>
```

Archivo de configuración del procedimiento para el tratamiento de transacciones del proyecto, de manera que los métodos de búsqueda transaccionales que empiecen con trans se realicen en modo lectura y/o escritura.

- **ormTransport.xml.-**

```
<entity class="ec.com.cst.transporte.entity.Ciudad"/>

<entity class="ec.com.cst.transporte.entity.Foto"/>

<entity class="ec.com.cst.transporte.entity.Lugar" />

<entity class="ec.com.cst.transporte.entity.Perfil"/>

<entity class="ec.com.cst.transporte.entity.Persona"/>

<entity class="ec.com.cst.transporte.entity.Provincia"/>

<entity class="ec.com.cst.transporte.entity.Reserva"/>

<entity class="ec.com.cst.transporte.entity.Telefono"/>

<entity class="ec.com.cst.transporte.entity.Usuario"/>

<entity class="ec.com.cst.transporte.entity.Vehiculo"/>
```

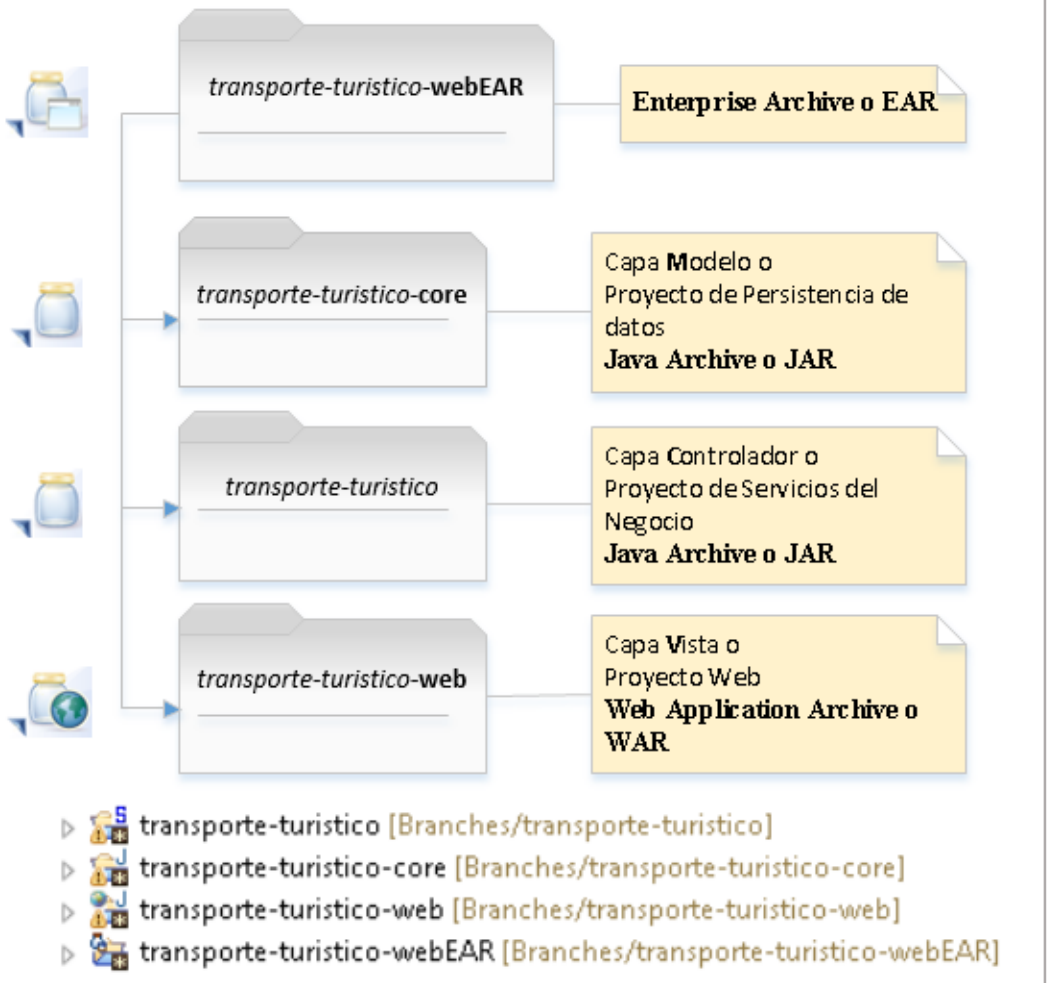
Archivo de configuración que especifica las rutas de las entidades, utilizada en *TransportSessionFactoryBeans.*

4.1.9 Empaquetamiento de los proyectos.

Para ensamblar la arquitectura del sistema, se empaquetaron o comprimieron a cada proyecto o capa que lo conforma del siguiente modo:

- El proyecto transporte-turistico-core y el proyecto transporte-turisco cada uno en archivos **Java Archive o JAR**.
- El proyecto transporte-turistico-web en un archivo **Web Application Archive o WAR** ya que permiten empaquetar en una sola unidad a aplicación web java.
- A continuación generamos un archivo **Enterprise Archive o EAR** que permite empaquetar los anteriores archivos dentro de este paquete.

Figura 59. Empaquetamiento de los proyectos del sistema



Elaborado por: Verónica Sangucho

Al empaquetar los proyectos se puede instalar el sistema para su despliegue al generarlos ubicándolos en el servidor de aplicaciones JBoss.

4.1.10 Generar / escribir el código.

Una vez levantada la arquitectura del sistema, se puede proceder a la generación de código fuente para poder darle ya la funcionalidad al sistema. El siguiente es un ejemplo de la generación del código.

- ✓ Se colocan las firmas de los métodos en la interfaz *ITransportGestor* del proyecto de servicios.

```
public List<Reserva> buscarReservas(Date desde, Date hasta, Reserva reserva) throws
TransportException;
```

- ✓ Las implementaciones de los métodos de los servicios se encuentran en *TransportGestor* del proyecto de servicios.

Figura 60. Implementación del método bucarReservas.

```
public List<Reserva> buscarReservas(Date desde, Date hasta, Reserva reserva)
        throws TransportException {
List<Reserva> listReservas = new ArrayList<Reserva>();
try {
        DetachedCriteria criteria = DetachedCriteria.forClass(Reserva.class);
        if(desde!=null && hasta!=null){
             Calendar c1 = Calendar.getInstance();
             c1.setTime(hasta);
             c1.set(c1.get(Calendar.YEAR), c1.get(Calendar.MONTH),(c1.get(Calendar.DATE))+1);
             hasta=c1.getTime();
             criteria.add(Restrictions.between("fechaReserva", desde, hasta));
        }
        if (reserva.getNombreCliente() != null && !reserva.getNombreCliente().isEmpty()) {
             reserva.setNombreCliente(reserva.getNombreCliente().toUpperCase());
             String nombreCliente="%".concat(reserva.getNombreCliente()).concat("%");
             criteria.add(Restrictions.like("nombreCliente",nombreCliente));
        }
        if (reserva.getCorreoCliente() != null && !reserva.getCorreoCliente().isEmpty()) {
             reserva.setCorreoCliente(reserva.getCorreoCliente().toLowerCase());
             criteria.add(Restrictions.eq("correoCliente",reserva.getCorreoCliente()));
        }
        if (reserva.getEstadoReserva() != null && !reserva.getEstadoReserva().isEmpty()) {
             criteria.add(Restrictions.eq("estadoReserva",
             reserva.getEstadoReserva()));
        }
        criteria.addOrder(Order.desc("fechaReserva"));
        listReservas = (List<Reserva>) genericDAO.findCriteria(criteria);
} catch (TransportException e) {
        TransportLogger.log.error("Error al buscarReservas ",e);
        throw new TransportException(e);
}
return listReservas;
```

Elaborado por: Verónica Sangucho

- ✓ Colocamos las firmas de los métodos con el prefijo trans en ITransportServicio el proyecto de persistencia.

List<Reserva> transBuscarReservas(Date desde, Date hasta, Reserva reserva) throws TransportException;

- ✓ Ubicamos las implementaciones en TransportServicio el proyecto de persistencia.

```
public List<Reserva> transBuscarReservas(Date desde, Date hasta,
        Reserva reserva) throws TransportException {
        return transportGestor.buscarReservas(desde, hasta, reserva);
}
```

- ✓ Para poder consumir el servicio en la parte web la llamamos desde el controler.

```
public void buscarReservas() {
        List<Reserva> listReservas = new ArrayList<Reserva>();
        listReservas =
        TransportFactory.getInstancia().getITransportServicio().
        transBuscarReservas(indexReservaDataManager.getFechaDesde(),
        indexReservaDataManager.getFechaHasta(),
        indexReservaDataManager.getBuscarReserva());
        indexReservaDataManager.setListReservas(listReservas);
}
```

- ✓ Desde la parte web llamamos al método implementado en el controller de la siguiente manera.

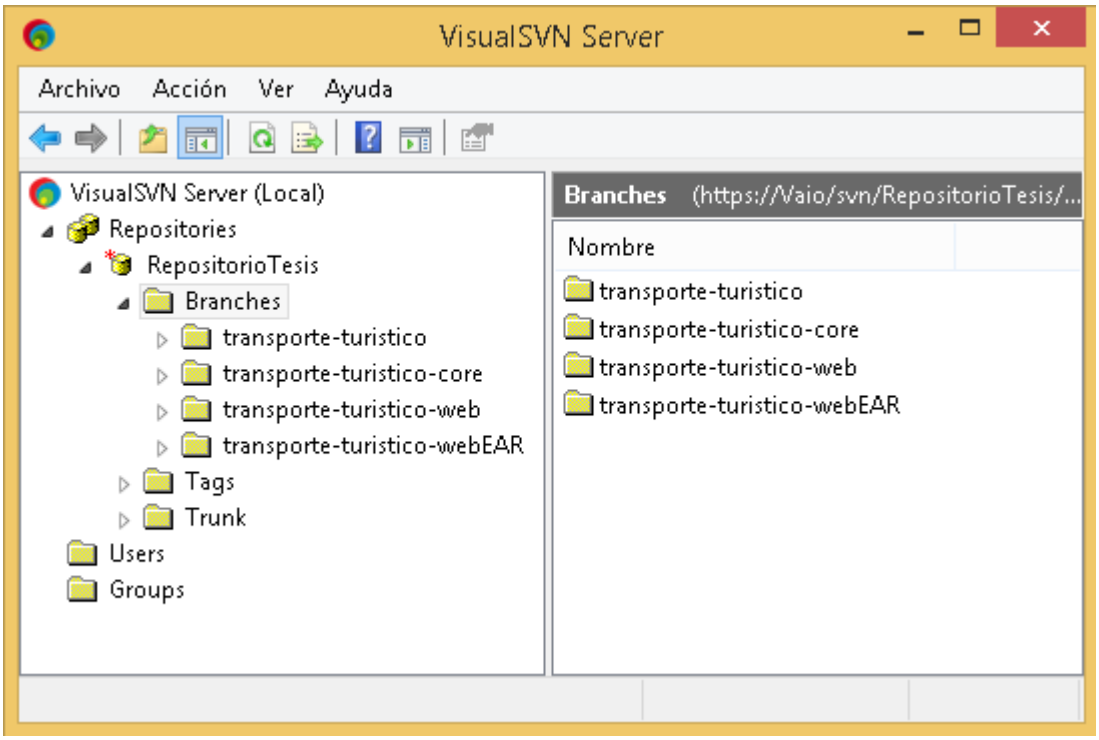
```
<p:commandButton value="#{msg['buscar']}"
        actionListener="#{indexReservaController.buscarReservas}"
        update="@form" partialSubmit="true" process="idBusqueda" />
```

Para los mensajes en pantalla se utiliza los archivos de propiedades en el proyecto web

```
empresa                         = Empresa
correo                          = Correo
buscar                          = Buscar
```

En la generación o escritura de código fuente es importante el control de versiona miento del sistema para lo cual se utilizó el sistema Visual SVN server.

Figura 61. Sistema en el sistema SVN.



Elaborado por: Verónica Sangucho.

4.2 Revisión del código fuente estático

- ✓ La herramienta Checstyle principalmente revisa el formato del código fuente así:

En todas las clases java el siguiente encabezado seguido de un salto de línea y el nombre del paquete este caso.

```
/*
 * Copyright 2013 CHRISLAND SERVICE AND TOURING - ECUADOR
 * Todos los derechos reservados
 */

package ec.com.cst.transporte.gestor;
```

Los comentarios en las firmas de los métodos en las interfaces en los proyectos de persistencia y de servicio, y en las implementaciones de los métodos en el proyecto web así:

Interfaz de un método:

```
/** * <b>
 * Guarda o actualiza un vehiculo.
 * </b>
 * <p>[Author: Verónica Sangucho, Date: 07/04/2014]</p>
 *
 * @param vehiculo a guardar o actualizar
 * @throws TransportException Exception
 */

public void guardarVehiculo(Vehiculo vehiculo) throws TransportException;
```

Las implementaciones de los métodos en el proyecto de presistencia y de servicio no deben llevar comentarios:

```
/* (non-Javadoc)
        * @see
ec.com.cst.transporte.gestor.ITransportGestor#buscarVehiculo.long)
* busca el vehiculo con todos los datos del asociado
 */
@Override
public Vehiculo buscarVehiculo(Long vehiculoId) throws TransportException {
```

Las implementaciones de los métodos en el proyecto web deben llevar comentarios de los metodos y sus parámetros de esta forma:

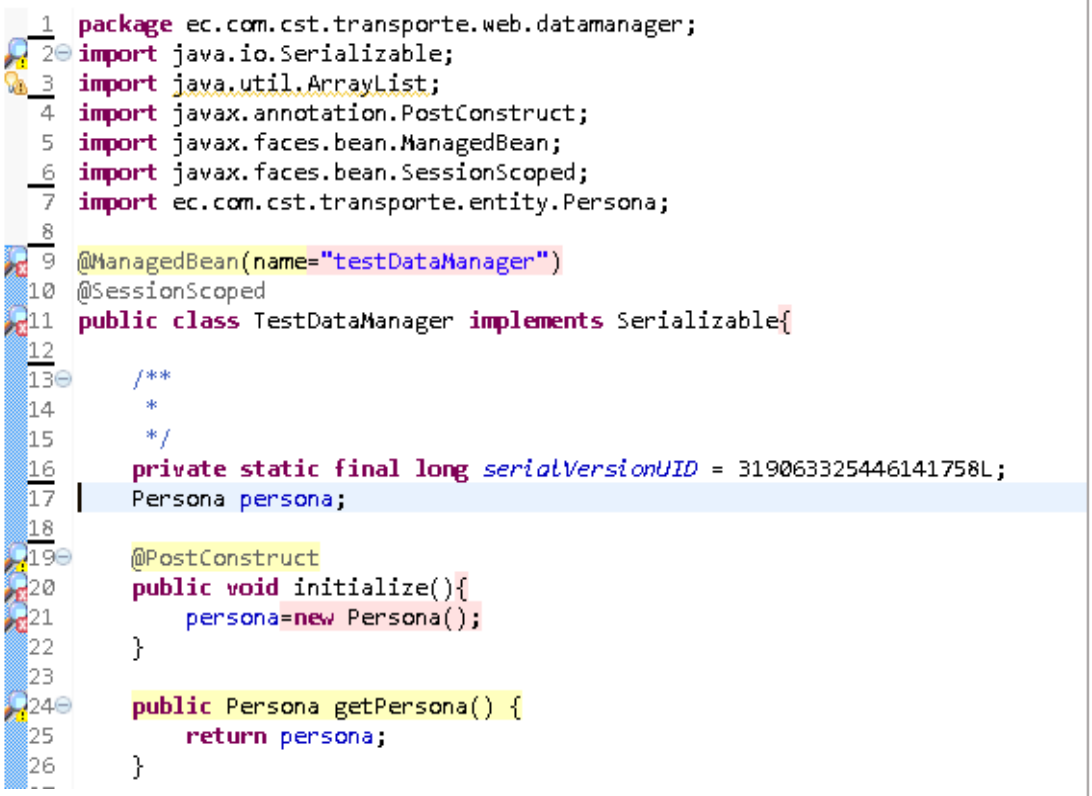
```
/**
 * <b>
 *  muestra el detalle de la reserva.
 * </b>
 * <p>[Author: Verónica Sangucho, Date: 10/04/2014]</p>
 *
 * @param reserva seleccionada
 */
public void seleccionarDetalleReserva(Reserva reserva) {
    buscarDetalleReserva(reserva);
    ndexReservaDataManager.getSelectedReserva().setIsDetail(Boolean.TRUE);
    indexReservaDataManager.getSelectedReserva().setIsEdit(Boolean.FALSE);
}
```

Examina que los signos =, {estén precedido de un espacio y seguido de un espacio así:
persona = new Persona();
public void initialize() {

Revisa el ancho máximo que puede tener una línea en las clases java en este caso no puede superar los 120 caracteres de ancho.

Cuando una clase viola estas las reglas de checstyle esta herramienta muestra los errores de la siguiente manera:

Figura 62. Errores localizados checstyle.



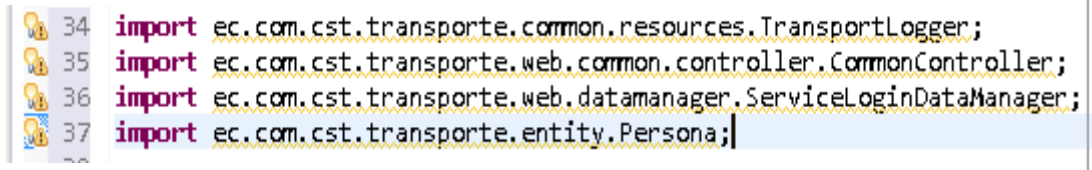
```
1  package ec.com.cst.transporte.web.datamanager;
2  import java.io.Serializable;
3  import java.util.ArrayList;
4  import javax.annotation.PostConstruct;
5  import javax.faces.bean.ManagedBean;
6  import javax.faces.bean.SessionScoped;
7  import ec.com.cst.transporte.entity.Persona;
8
9  @ManagedBean(name="testDataManager")
10 @SessionScoped
11 public class TestDataManager implements Serializable{
12
13     /**
14      *
15      */
16     private static final long serialVersionUID = 319063325446141758L;
17     Persona persona;
18
19     @PostConstruct
20     public void initialize(){
21         persona=new Persona();
22     }
23
24     public Persona getPersona() {
25         return persona;
26     }
```

Elaborado por: Verónica Sangucho.

- ✓ La herramienta PMD principalmente revisa lo óptimo del código fuente así:

Imports inutilizados:

Figura 63. Error imports inutilizados localizado por el PMD.

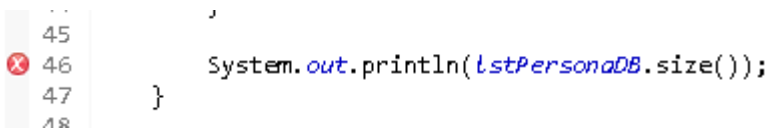


```
34 import ec.com.cst.transporte.common.resources.TransportLogger;
35 import ec.com.cst.transporte.web.common.controller.CommonController;
36 import ec.com.cst.transporte.web.datamanager.ServiceLoginDataManager;
37 import ec.com.cst.transporte.entity.Persona;
```

Elaborado por: Verónica Sangucho.

Utilización de system.out.println:

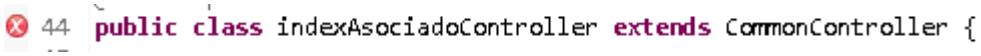
Figura 64. Error system utilizado localizado por el PMD.



Elaborado por: Verónica Sangucho.

Revisa que las clases java empiecen con una letra mayúscula

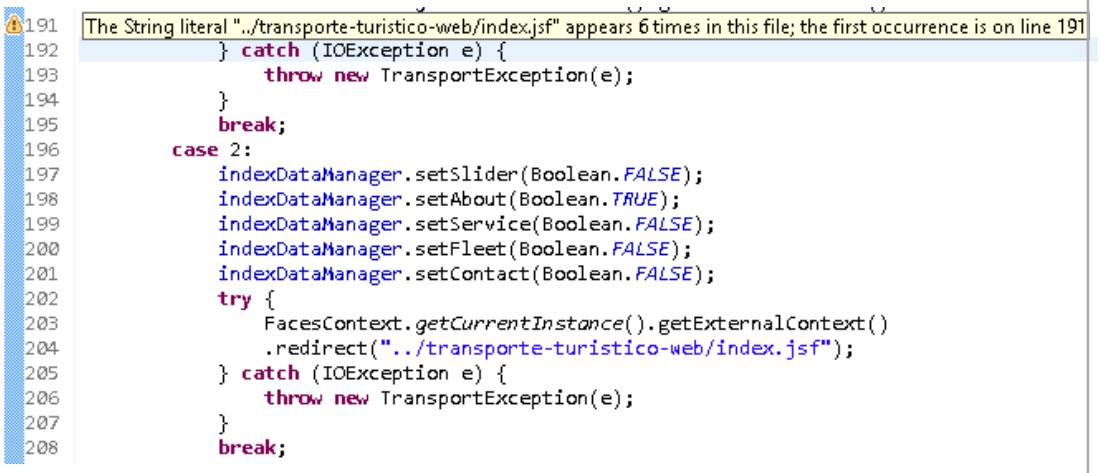
Figura 65. Error clase con minúscula localizado por PMD.



Elaborado por: Verónica Sangucho.

Detecta código repetido:

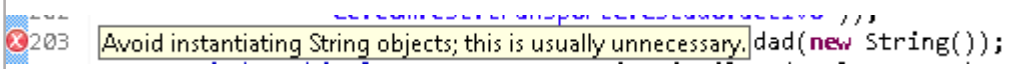
Figura 66. Error código repetido localizado por PMD.



Elaborado por: Verónica Sangucho.

Detecta código innecesario:

Figura 67. Error código innecesario localizado por PMD.



Elaborado por: Verónica Sangucho.

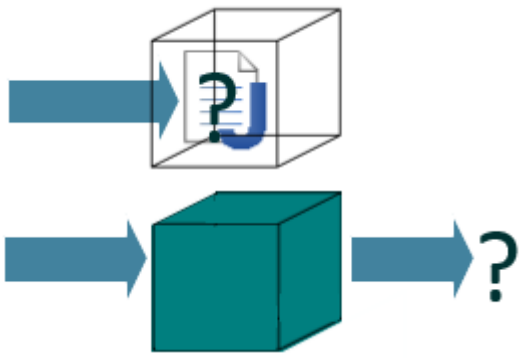
4.3 Pruebas al sistema

A continuación se describen los procedimientos con los cuales se verificó y reveló la información sobre la calidad del presente producto software.

Estos procedimientos se los puede clasificar según se pueda no observar el funcionamiento y/o estructura interna del sistema, en:

- Pruebas de caja blanca o caja de cristal
- Pruebas de caja negra

Figura 68. Pruebas de Caja Blanca y Caja Negra



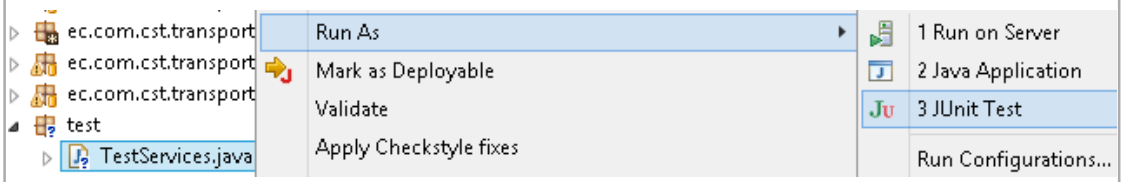
Elaborado por: Verónica Sangucho.

4.3.1 Pruebas de Caja Blanca.

O pruebas de Caja de Cristal, denominadas así debido a que se enfocan y se aprecian los detalles procedimentales de las funciones internas del sistema, por lo cual están ligados al código fuente.

Para realizar las pruebas de Caja Blanca se usó el plunning Junit de eclipse sobre el código fuente, como se puede apreciar a continuación.

Figura 69. Uso de JUnit para pruebas de Caja Blanca



Elaborado por: Verónica Sangucho.

```
@Test
public void testServicioBuscarProvincias(){
        List<Provincia>
        respuesta=TransportFactory.getInstancia().getITransportServicio().transBuscarProvincias();
        System.out.println("fin del test exitoso, numero de provincias : "+respuesta.size());
}
```

Resultado de la prueba:

Hibernate: select this_.PROVINCIA_ID as PROVINCIA1_5_0_, this_.NOMBREPROVINCIA as NOMBREPR2_5_0_ from PROVINCIA_TBL this_ order by this_.NOMBREPROVINCIA asc
fin del test exitoso, numero de provincias : 23

```
@Test
public void testServicioGuardarAsociado(){
        Persona per =new Persona();
        per.setNombrePersona("TEST");
        per.setApellidoPersona("apellidoPersonaTest");
        per.setNombreEmpresa("nombreEmpresaTest");
        per.setCorreoPersona("correoPersonaTest");
        per.setProvinciaId(2L);
        per.setCiudadId(2L);
        per.setPerfilId(1L);
        TransportFactory.getInstancia().getITransportServicio().transGuardarAsociado(per);
        System.out.println("fin del test exitoso, id de la persona creada: "+per.getPersonaId());

}
```

Resultado de la prueba:

Hibernate: select nextval ('PERSONA_ID_SEQ')
Hibernate: insert into PERSONA_TBL (APELLIDOPERSONA, CEDULAPERSONA, CIUDAD_ID, CORREOPERSONA, DIRECCIONPERSONA, ESTADOPERSONA, FECHACREACION, FECHAMODIFICACION, NOMBREEMPRESA, NOMBREPERSONA, PERFIL_ID, PROVINCIA_ID, USUARIOIDCREACION, USUARIOIDMODIFICACION, PERSONA_ID) values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
fin del test exitoso, id de la persona creada: 27

```
@Test
public void testServicioBuscarTelefonos(){
        List<Telefono> listTelefonos = new ArrayList<Telefono>();
        listTelefonos =
TransportFactory.getInstancia().getITransportServicio().transBuscarTelefonos(6L);
System.out.println("fin del test exitoso, numero de telefonos encontrados por id persona:
"+listTelefonos.size());
}
```

Resultado de la prueba:

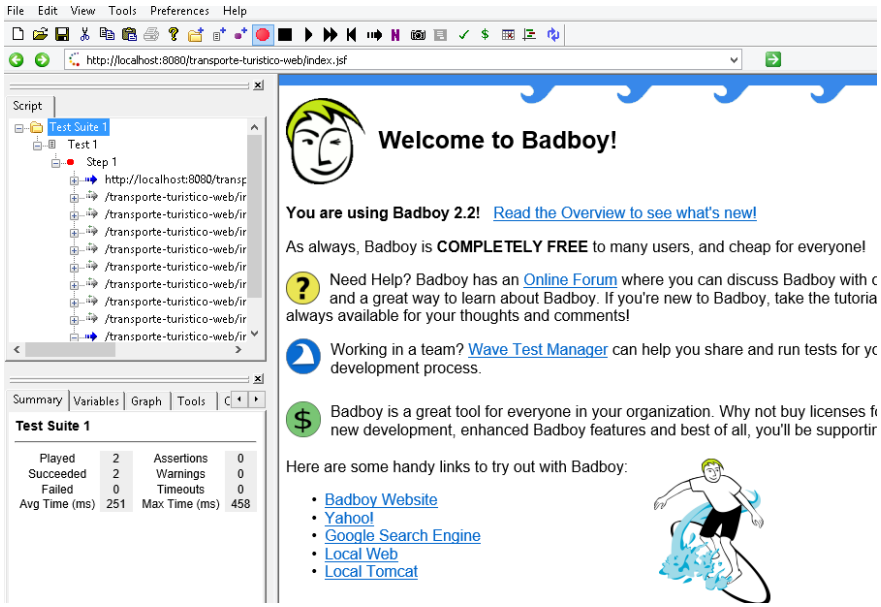
Hibernate: select this_.TELEFONO_ID as TELEFONO1_7_0_, this_.NUMEROTELEFONO as NUMEROTE2_7_0_, this_.PERSONA_ID as PERSONA3_7_0_, this_.TIPOTELEFONO as TIPOTELE4_7_0_ from TELEFONO_TBL this_ where (this_.PERSONA_ID=?)
fin del test exitoso, numero de telefonos encontrados por id persona: 2

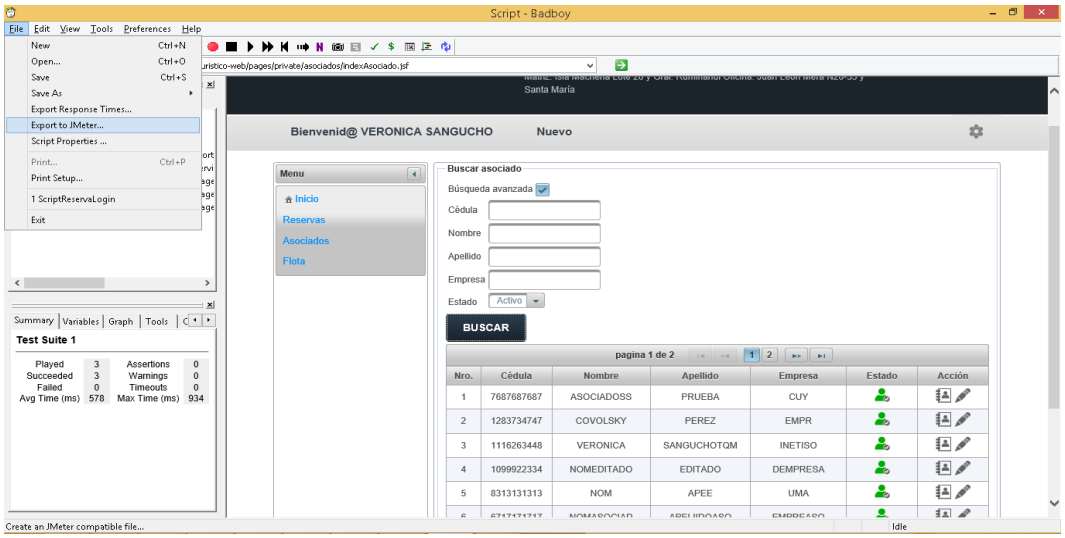
4.3.2 Pruebas de Caja Negra.

Las pruebas de caja negra se las realiza sobre la capa superficial o límite del sistema, es decir, su interfaz gráfica, estas se encuentran enfocadas en el comportamiento del sistema, sin tomar en cuenta como se ejecuta internamente.

Para esta finalidad se utilizó el software Jmeter para verificar el rendimiento del sistema a través de pruebas de rendimiento, emulando una gran concurrencia sobre el servidor de aplicaciones, junto con la herramienta badboy al cual permitió la interacción con la interfaz gráfica del sistema como se muestra en las figuras 70 y 71.

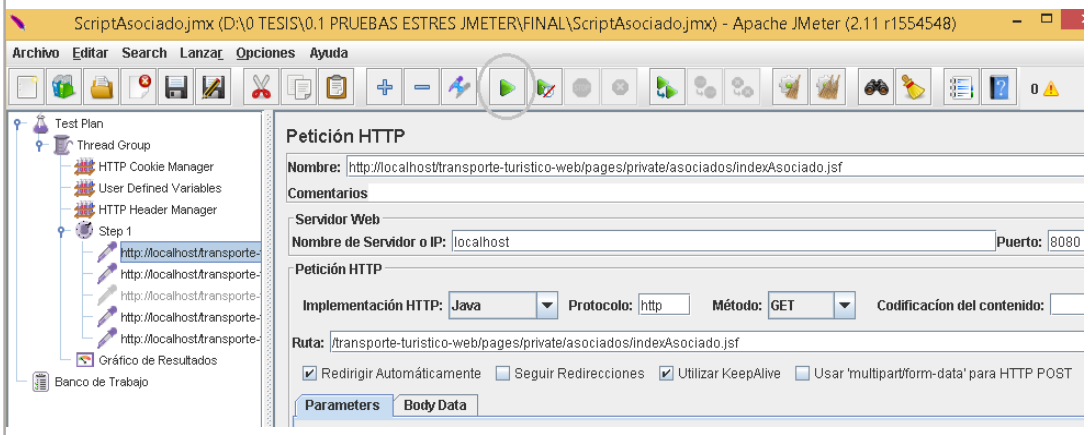
Figura 70. Software Badboy





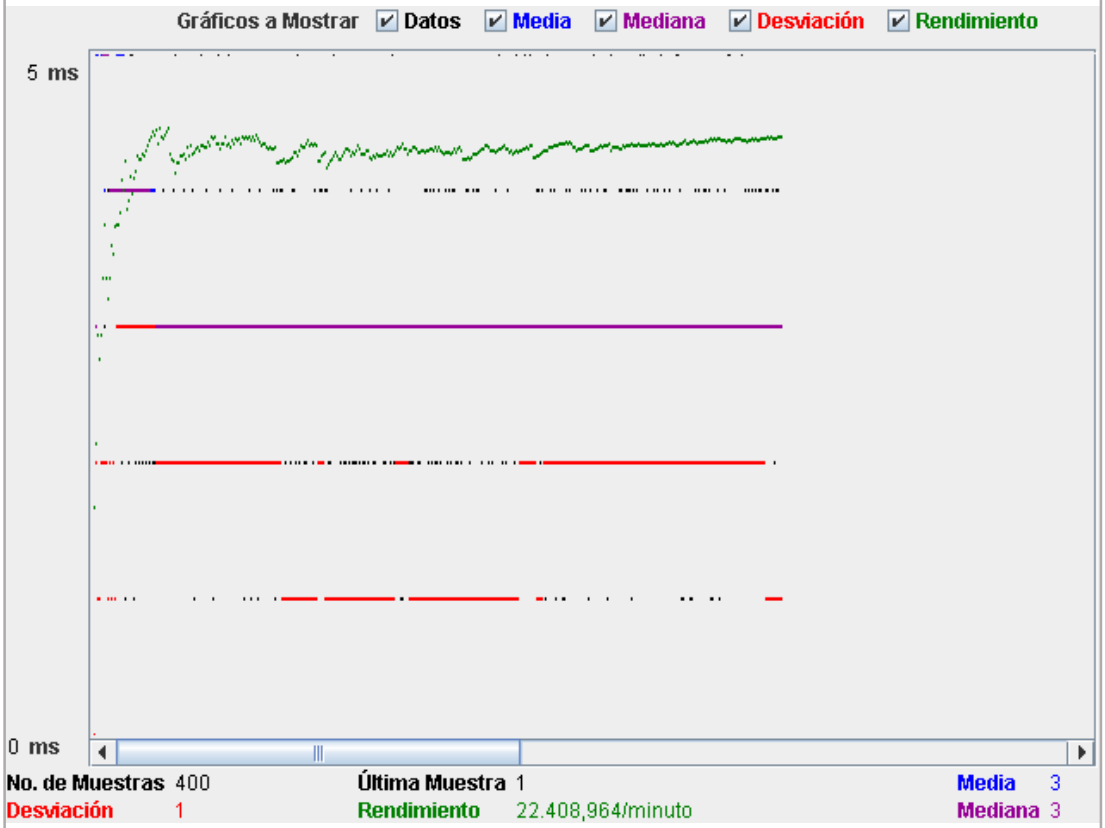
Elaborado por: Verónica Sangucho.

Figura 71. Software Jmeter



Elaborado por: Verónica Sangucho.

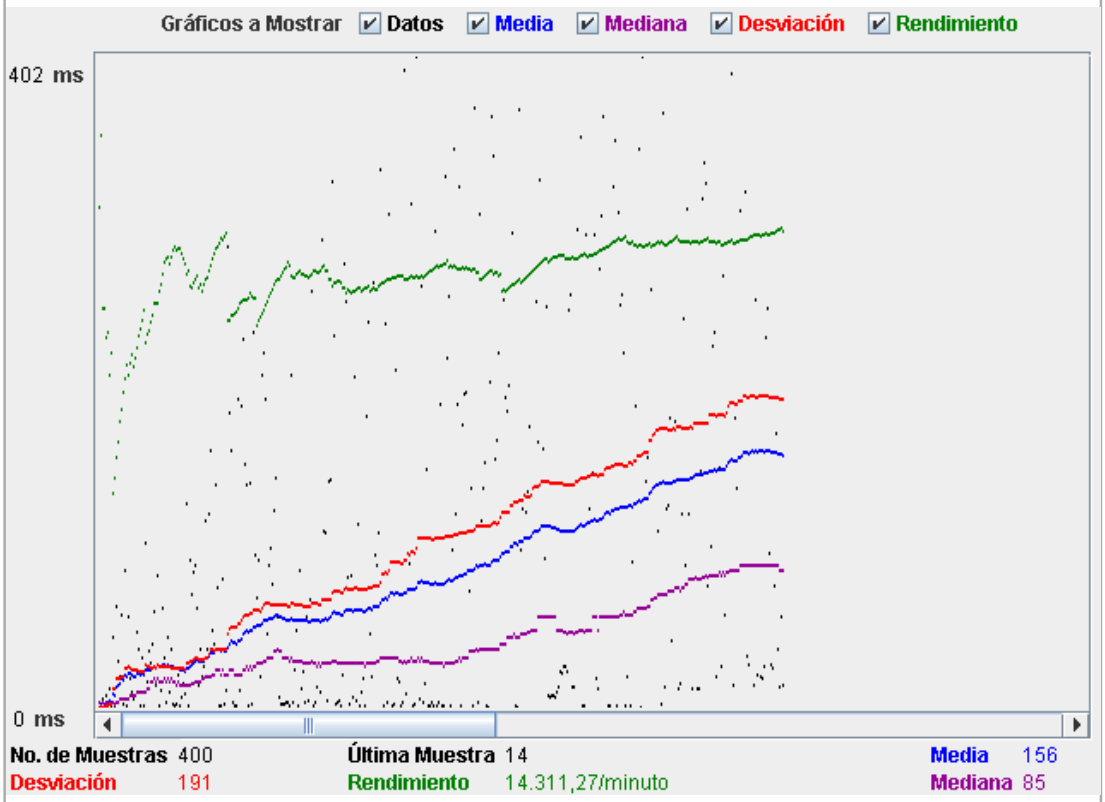
Figura 72. Resultado prueba de rendimiento Login



Elaborado por: Verónica Sangucho.

En la figura 72, la muestra representa a los 400 usuarios que simulan la acción de iniciar sesión y que de manera simultánea realizan las conexiones con el servidor de aplicaciones. Donde el rendimiento no supera los 5 milisegundos, por lo que las pruebas de estrés realizadas en el Login del sistema dieron resultados efectivos.

Figura 73. Resultado prueba de rendimiento módulo pre-reserva

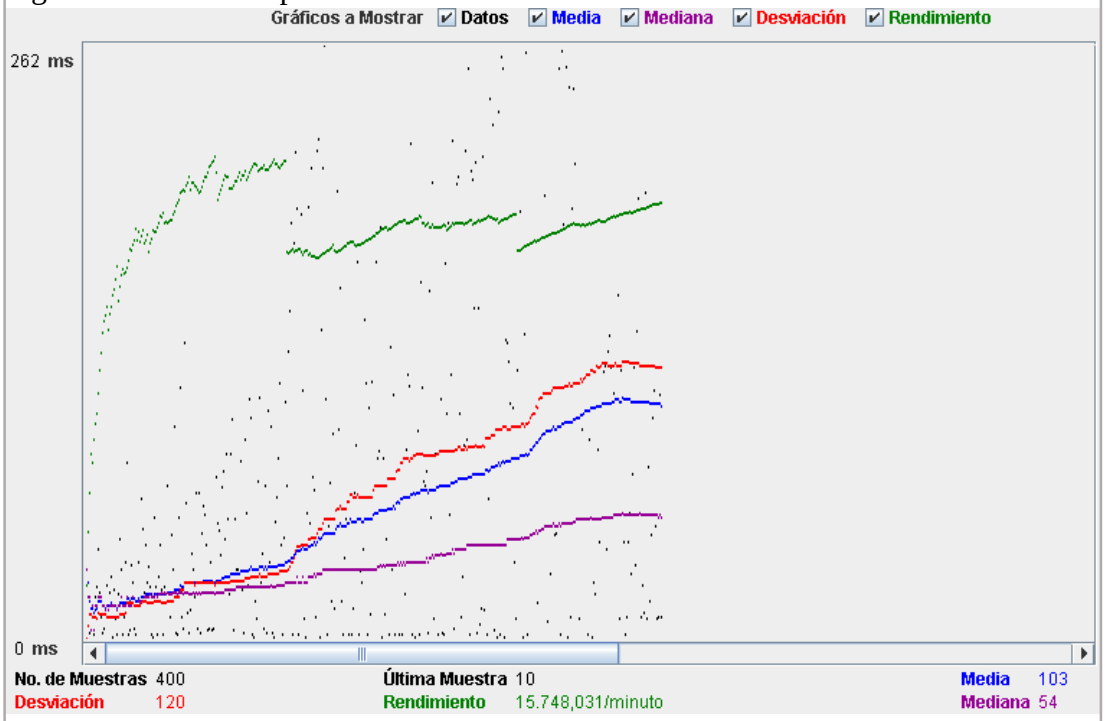


Elaborado por: Verónica Sangucho.

En la figura 73, se utiliza la misma muestra de la 72 y se producen las mismas curvas, con la diferencia de que en este caso se simula la acción de buscar las pre-reservas realizadas por los clientes. Se puede ver que todas las curvas tienen un sentido creciente, esto se debe a que los procesos de las peticiones realizados en el módulo de pre-reservas son más complejas que las del módulo del Login.

El máximo rango en el eje las ordenadas es de cuatrocientos dos milisegundos por tanto se vuelve a comprobar que el sistema soporta de forma efectiva las pruebas de rendimiento en el módulo pre-reserva.

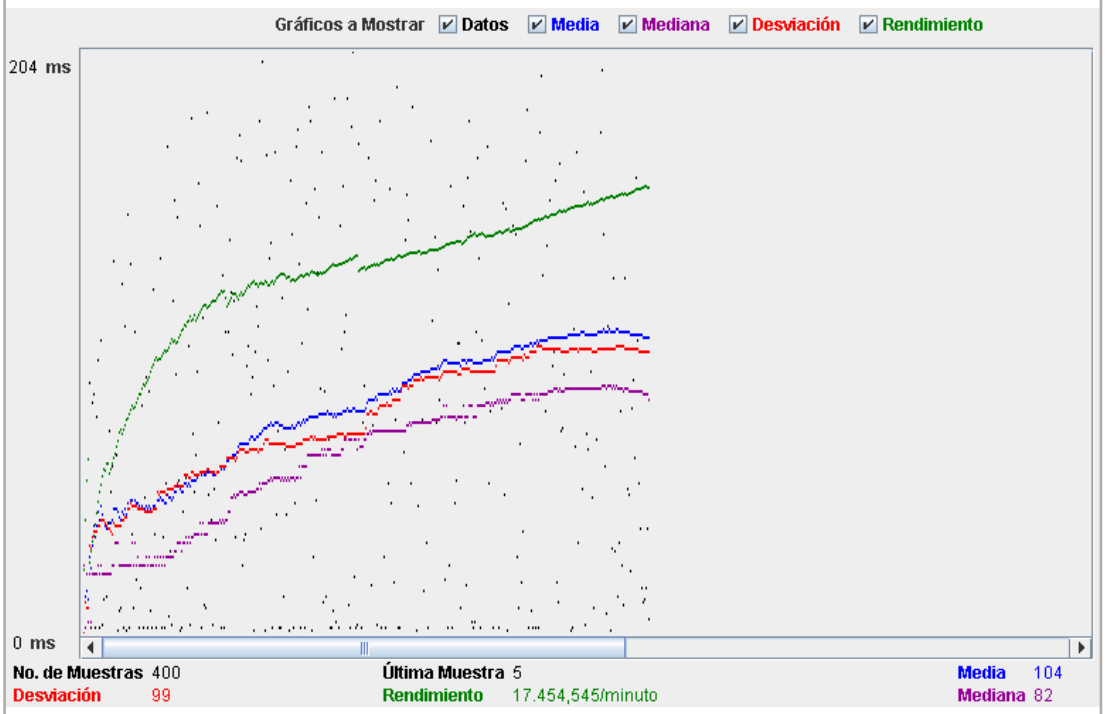
Figura 74. Resultado prueba de rendimiento módulo asociado



Elaborado por: Verónica Sangucho.

En la figura 74, se utiliza la misma muestra de la figura 73, en este caso la acción realizada por los usuarios es la de buscar los asociados que fueron registrados en el sistema. Se producen curvas de sentido creciente similares a las de la figura 73, excepto en la curva de color verde que se divide en tres periodos crecientes, esto se debe a que los procesos de las peticiones realizados en el módulo de asociados son similares a los procesos que en el módulo de pre-reservas. Además, el máximo rango en el eje las ordenadas es de 262 ms este valor es menor que el de 402ms en el módulo de pre-reservas, puesto que los procesos requieren de menos parámetros de búsqueda que en el módulo de pre-reservas.

Figura 75. Resultado prueba de rendimiento módulo flota

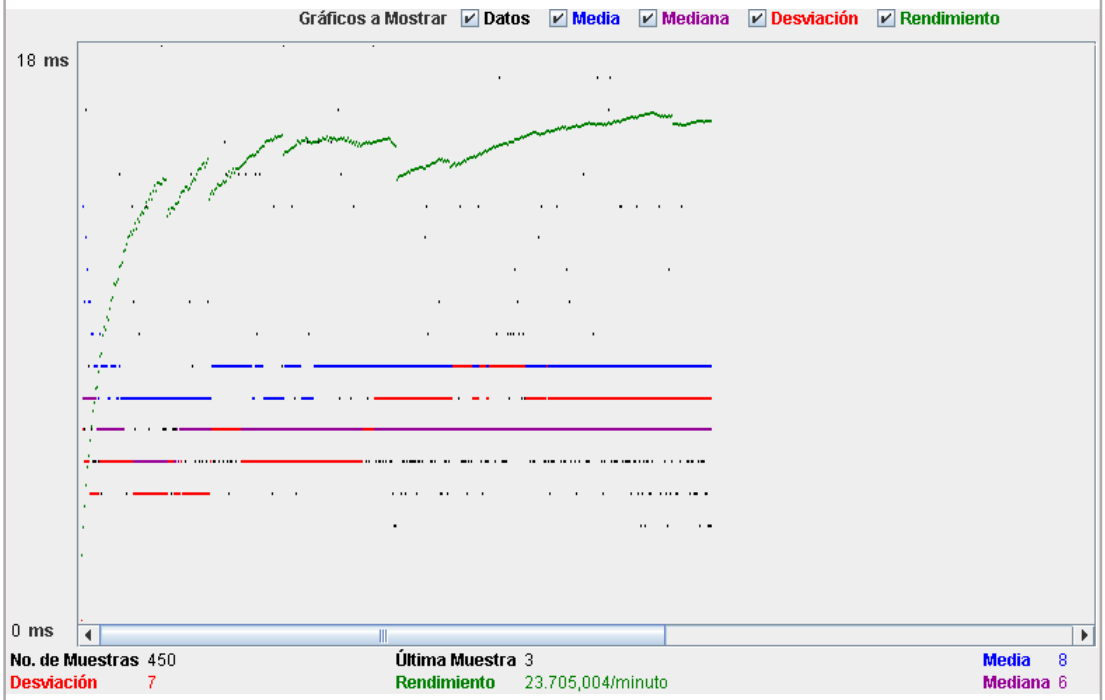


Elaborado por: Verónica Sangucho.

En la figura 75, los usuarios simulan la acción de buscar los vehículos registrados en el sistema y el análisis es semejante al de la figura 74 por lo que la explicación se concentrará en la curva verde de rendimiento.

La curva de color verde con un sentido un creciente más pronunciado representa que el rendimiento está por encima de la media con un máximo de 204 ms menor que 262ms del módulo asociados, debido a que los procesos requieren de menos parámetros de búsqueda y sus funciones son menos complejas que en el módulo de asociados.

Figura 76. Resultado prueba de rendimiento Index



Elaborado por: Verónica Sangucho.

En la figura 76, la muestra de 400 usuarios simula el ingreso simultáneo de los clientes a la página principal del sistema. El máximo rango en el eje de las ordenadas es de 18 milisegundos debido a que este módulo requiere de procesos de consulta sencillos como la carga de los vehículos en la sección pre-reserva en línea, este resultado es relevante en tanto en cuanto es la vía de comunicación directa del sistema con el cliente.

4.4 El sistema en internet

El sistema final fue subido a internet en JBoss 7 a través del portal en internet de OpenShift (https://www.openshift.com/) de RedHat, el cual es una plataforma como servicio (PaaS) para aplicaciones de clase empresarial en la nube. OpenShift ofrece una consola web para poder crear y administrar la aplicación a través de líneas de comando.

La plataforma OpenShift es de código abierto, soporta una gama de tecnologías como los leguajes de programación como Java, Ruby, Php, servidores JBoss6, JBoss7, Tomcat, bases de datos MySQL, Postgrest, para el despliegue de aplicaciones.

CONCLUSIONES

- La implementación del sistema web para la gestión y el control del servicio de transporte turístico terrestre, de la compañía Chrisland Service And Touring, se desarrolló bajo el patrón de diseño de software modelo vista controlador (mvc); esta tecnología permitió la configuración de una arquitectura de N capas (N = 3), lo cual hace que el sistema sea portable, escalable y seguro, además permitió el uso de herramientas software que se encuentran disponibles en Internet mediante esta forma de trabajo se optimizaron los costos.
- El uso de la metodología ICONIX permitió al proyecto centrarse en obtener la codificación en el menor número de pasos posible y de forma fiable de tal modo que no se perdieron los beneficios del análisis y diseño inicial, así se generó un producto final según el modelo de negocio actual.
- El Framework Spring es una herramienta muy útil, ya que ayudó a organizar la gran cantidad de elementos, instrucciones y funciones que se generaron al desarrollar un sistema con lenguaje de programación Java, de tal forma que la unión de estas dos tecnologías contribuyeron a generar un código mejor organizado.
- La tecnología JSF junto con PrimeFaces, html5 y css brindaron facilidad y estética para la creación de la capa vista, presentación o interfaz gráfica del sistema.
- El uso en conjunto de los programas Badboy y JMeter facilitó el desarrollo de las pruebas de rendimiento del sistema en tiempo de ejecución, ya que por un lado el software Badboy capturó la navegación de un solo usuario sobre la interfaz gráfica, y por su parte la herramienta JMeter simuló a 400 usuarios realizando la acción capturada, generando este número de peticiones al servidor de aplicaciones de manera concurrente.
- Los resultados de las pruebas de rendimiento del sistema son buenos ya que los límites de tiempo no superaron el rango de los milisegundos sin llegar siquiera a un segundo, aún con 400 usuarios realizando en forma simultánea peticiones al servidor de aplicaciones. Por lo tanto des puede decir que la herramienta PMD si contribuye a optimizar el código fuente.
- El uso de la herramienta CheckStyle sirve para la comprensión de los métodos de manera más rápida y mejor a través de los estándares en los comentarios de

los métodos en el código fuente, por lo cual es de gran ayuda en la etapa de mantenimiento del sistema.

- La herramienta Visual SVN Server resultó ser fácil e intuitiva para el control de las versiones del sistema, y útil en la generación o escritura del código fuente ya en la fase de desarrollo frecuentemente resulta práctico revertir alguna modificación que haya perjudicado alguna parte del sistema.
- OpenShift es una buena opción como plataforma para aplicaciones de tipo empresarial en la nube ya que el precio por su servicio va desde gratis hasta planes Premium que son pagados según el tiempo de suscripción que sea solicitado, por ejemplo el costo por mes es de $20, soporta las tecnologías usadas en el sistema como son el lenguaje de programación Java y la base de datos Postgresql, para el presente proyecto se utilizó la versión gratuita, para lo cual se encontró suficiente documentación en el portal para el despliegue de la aplicación en esta plataforma.
- El uso del servidor de aplicaciones Jb oss en su versión 7, es ventajoso porque tiene reducido su tiempo de arranque con respecto a las versiones anteriores, es decir, bajo de un par de minutos a pocos segundos, ya que solo inicia los servicios que son críticos para su funcionamiento.

RECOMENDACIONES

- Es importante seguir las reglas del PMD y CheckStyle desde el inicio y analizar el código periódicamente durante su desarrollo para no tener que refactorizar en la culminación del proyecto ya que esto ahorra tiempo.
- En proyectos que se van a subir a la web es recomendable guardar las imágenes en bytes o lo que es lo mismo en ceros y unos, ya que su carga es mucho más rápida que guardándola en formato de imagen.
- Es recomendable la configuración del spring para la inyección de dependencias, ya que se suministran objetos de una clase a otra en lugar de que la misma clase lo cree cada vez que lo necesite, optimizando el uso de recursos.
- Cada objeto de transferencia de datos o DTO que representa una tabla de la base de datos que necesita un objeto de acceso a datos o un DAO, pero como cada Dao tiene un comportamiento similar, es aconsejable crear un Dao genérico para todos que contenga las funciones de crear, recuperar, actualizar y eliminar registros para cualquier DTO.
- Para la creación de las interfaces gráficas, el uso de PrimeFaces es ventajoso ya que cuenta con el mayor número de componentes y temas disponibles, permite la combinación con otros componentes como RichFaces y facilita la personalización de su apariencia por medio de la sobre-escritura de estilos en sus componentes.

LISTA DE REFERENCIAS

Para el presente trabajo de titulación se utilizó como guía los siguientes libros y referencias a manera de bibliografía

- Bert Bos (2013). What is CSS?. Recuperado el 20 de noviembre del 2013 de http://www.w3.org/Style/CSS/.
- Eclipse, Documentación ¿Estás listo para Eclipse?, Recuperado el 24 de junio del 2013 de http://www.eclipse.org/juno/.
- Java.net (2013), Java.net the Source for Java Technology Collaboration", Recuperado el 24 de junio del 2013 de https://javaserverfaces.java.net/.
- Microsoft Corporation (2013). Visio Professional 2013. Recuperado el 16 de agosto del 2013 de http://office.microsoft.com/es-es/visio/.
- Olga Carreras Montoto (2012). Prototipado. Recuperado el 8 de septiembre del 2013 de http://www.youtube.com/watch?v=vJrEbO50kxA.
- Oracle, Documentación "¿Qué es Java?", Recuperado el 24 de junio del 2013 de http://www.java.com/es/download/whatis_java.jsp.
- Orale Company. Descripción de la tecnología JavaServer Faces. Recuperado el 21 de junio del 2013 de http://www.oracle.com/technetwork/java/javaee/overview-140548.html.
- Pivotal (2013). Introduction to Spring Framework. Recuperado el 28 de noviembre del 2013 de http://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/overview.html.
- Pivotal Enterprise (2004). Spring Java Application Framework. Recuperado el 24 de junio del 2013 de http://static.springsource.org/spring/docs/3.2.x/spring-framework-reference/html/overview.html.
- PostgreSQL. Acerca de PostgreSQL. Recuperado el 24 de junio del 2013 de http://www.postgresql.org/about/.
- Red Hat Company (2004). Hibernate Getting Started Guide. Recuperado el 21 de junio del 2012 de http://docs.jboss.org/hibernate/orm/4.2/quickstart/en-US/html/pr01.html#d5e41.
- RedHat Company. JBoss Web Server. Recuperado el 24 de junio del 2013 de http://www.jboss.org/products.

- Rosenberg D., Stephens M., & Collins-Cope M. (2005), "Agile Development with ICONIX Process. People, Process, and Pragmatism", Library of Congress Cataloging-in-Publication Data.
- SourceForge (2012). Checkstyle Documentation. Recuperado el 24 de junio del 2013 de http://checkstyle.sourceforge.net/.
- SourceForge (2013). Bienvenido a PMD. Recuperado el 24 de junio del 2013 de http://pmd.sourceforge.net/pmd-5.0.4/.
- Sparx Systems (2013). Plataforma de Modelado Visual. Recuperado el 05 de Septiembre del 2013 de http://www.sparxsystems.com/products/ea/.
- Sun Microsystems (2002). Modelo-Vista-Controlador. Recuperado el 21 de junio del 2013 de http://www.oracle.com/technetwork/java/mvc-140477.html.
- VisualSVN Company (2013). VISUAL SVN SERVIDOR / / Servidor Subversion para Windows. Recuperado el 16 de agosto del 2013 de http://www.visualsvn.com/server/.
- W3C (2013). Vocabulario y API asociadas para HTML y XHTML. Recuperado el 30 de octubre del 2013 de http://www.w3.org/TR/html5/.
- World Tourism Organization (1995). COLLECTION OF TOURISM EXPENDITURE STATISTICS. Recuperado el 25 de junio del 2013 de http://pub.unwto.org/WebRoot/Store/Shops/Infoshop/Products/1034/1034-1.pdf.

ANEXOS

Anexo 1. Descripción del sistema

A continuación se muestra el sistema y sus funcionalidades.

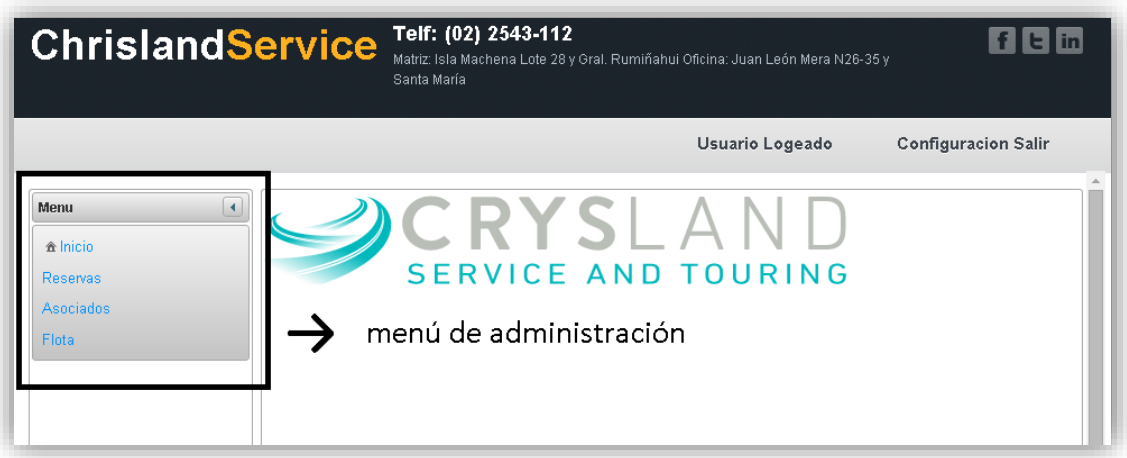
Ingreso al sistema

Ingresar el nombre de usuario y contraseña del administrador y dar click en GO.

Pantalla inicio de sesión

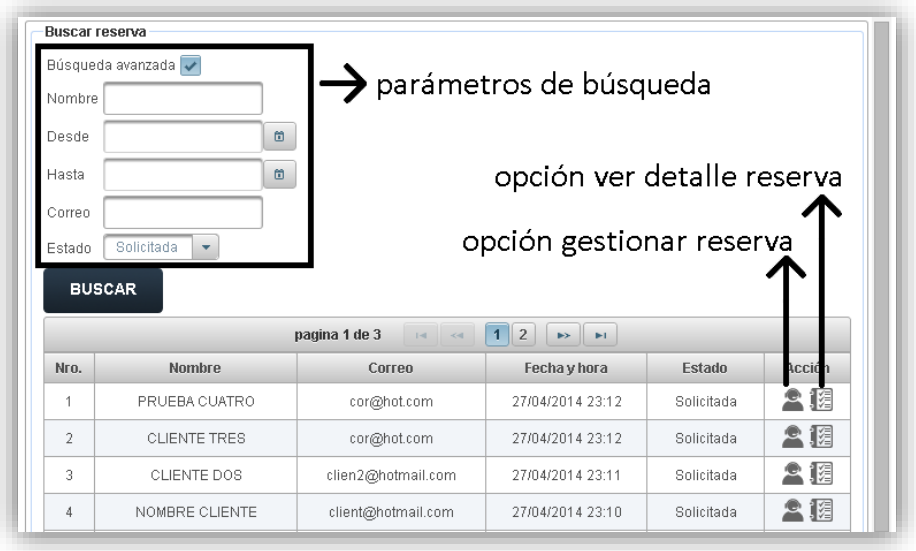
Menú administrador

Una vez en el sistema se muestra las opciones del menú principal de administrador.



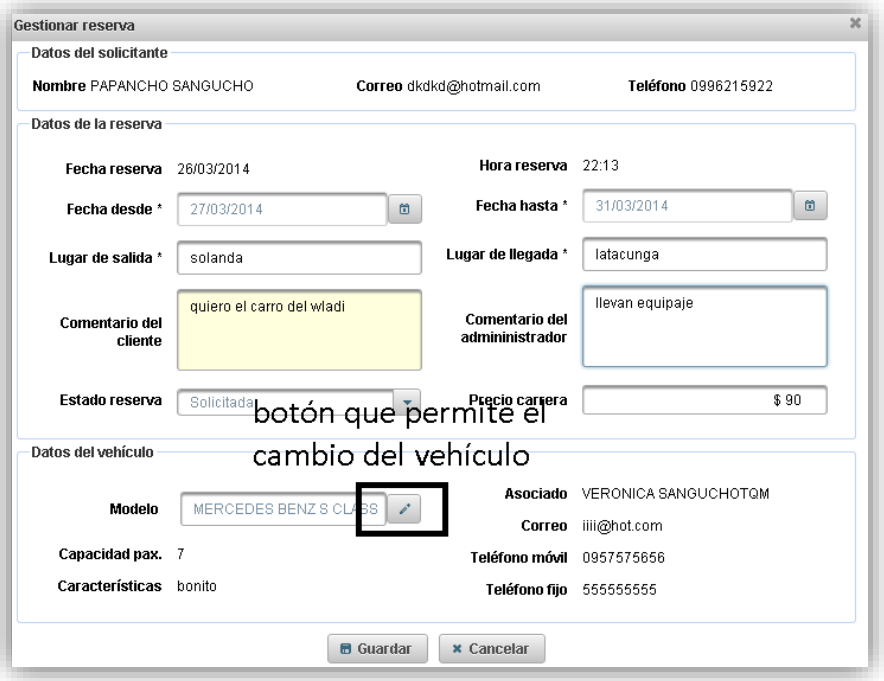
Opción pre-reservas

Muestra las pre-reservas de viajes realizadas por Internet a través de la página principal del sistema. Esta opción permite gestionar una pre-reserva o ver los detalles de una pre-reserva.



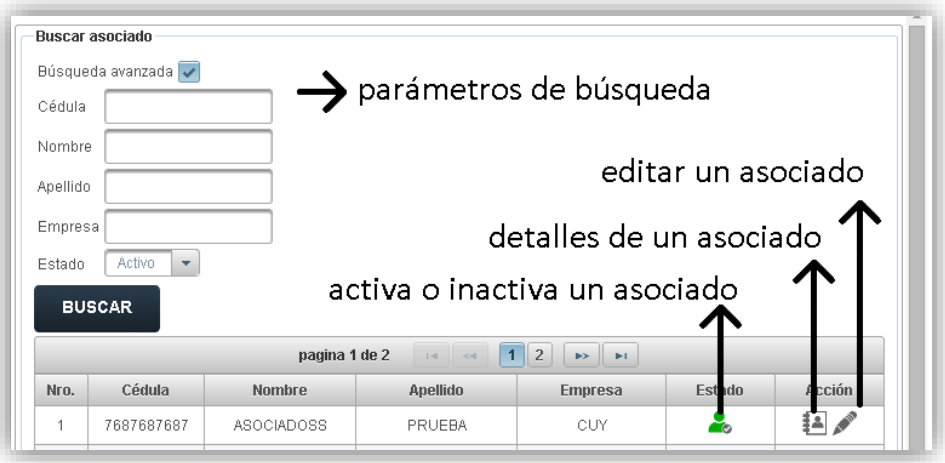
Opción gestionar pre-reserva

Muestra los datos detalles del cliente para ponerse en contacto con el mismo



Opción asociados

Permite la búsqueda, edición, inactivación y registro de un asociado.

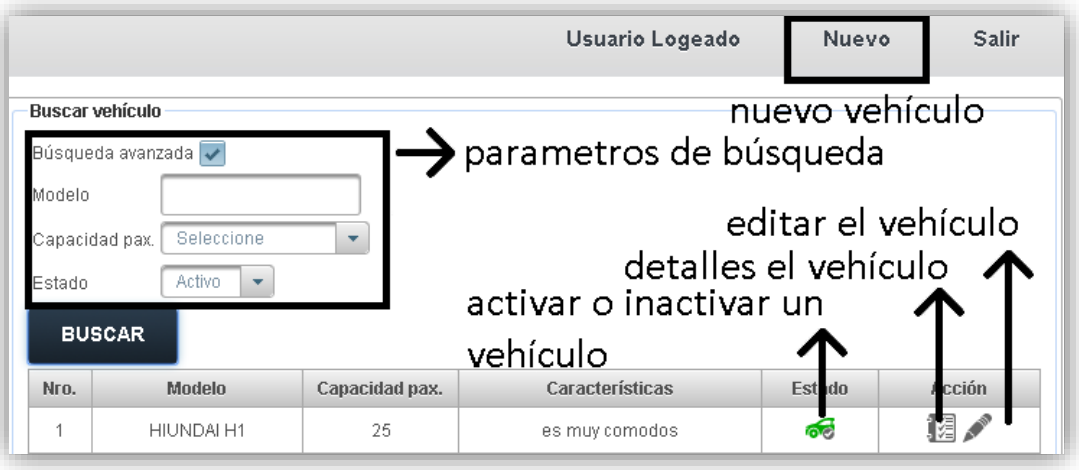


Opción editar y ver detalles del asociado

Permite actualizar o corregir los datos de un asociado

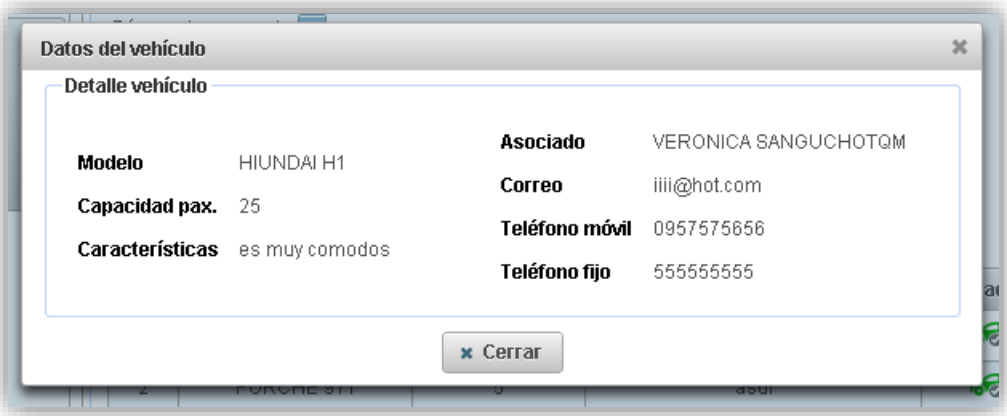
Opción flota

Permite la búsqueda, edición, inactivación y registro de un vehículo.



Opción editar y ver detalles del vehículo

Permite actualizar o corregir los datos de un vehículo.



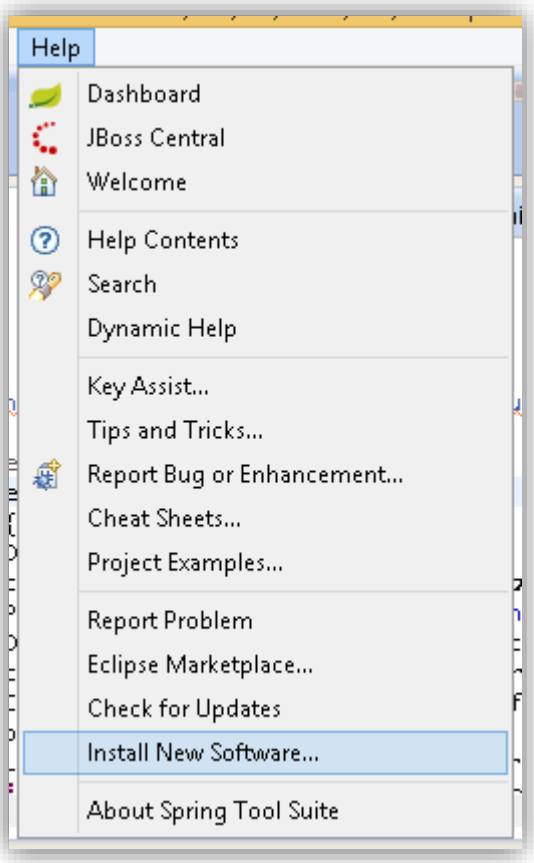
Pantalla index principal

Información de la empresa

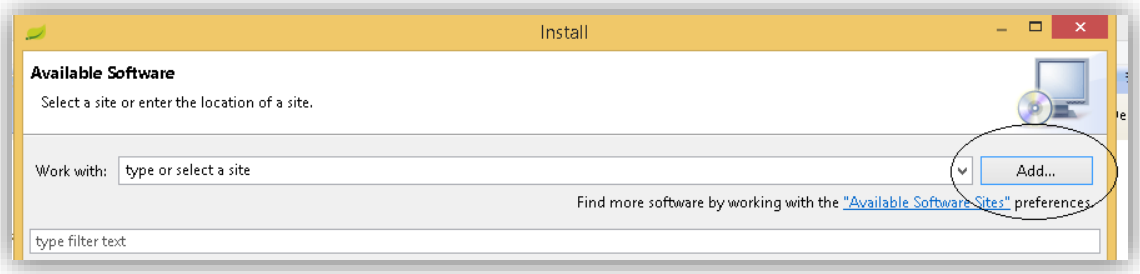
Anexo 2. Instalación de las herramientas de revisión de código estático

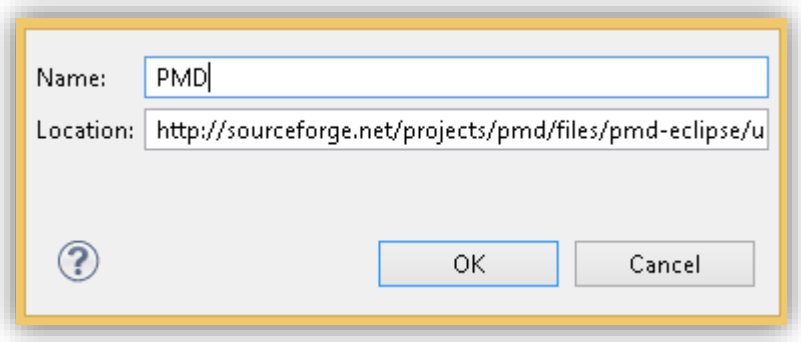
PMD (Programming Mistake Detector)

Para instalar el plugin de PMD en Eclipse, ingresamos en la opción “Help” -> “Install New Software”

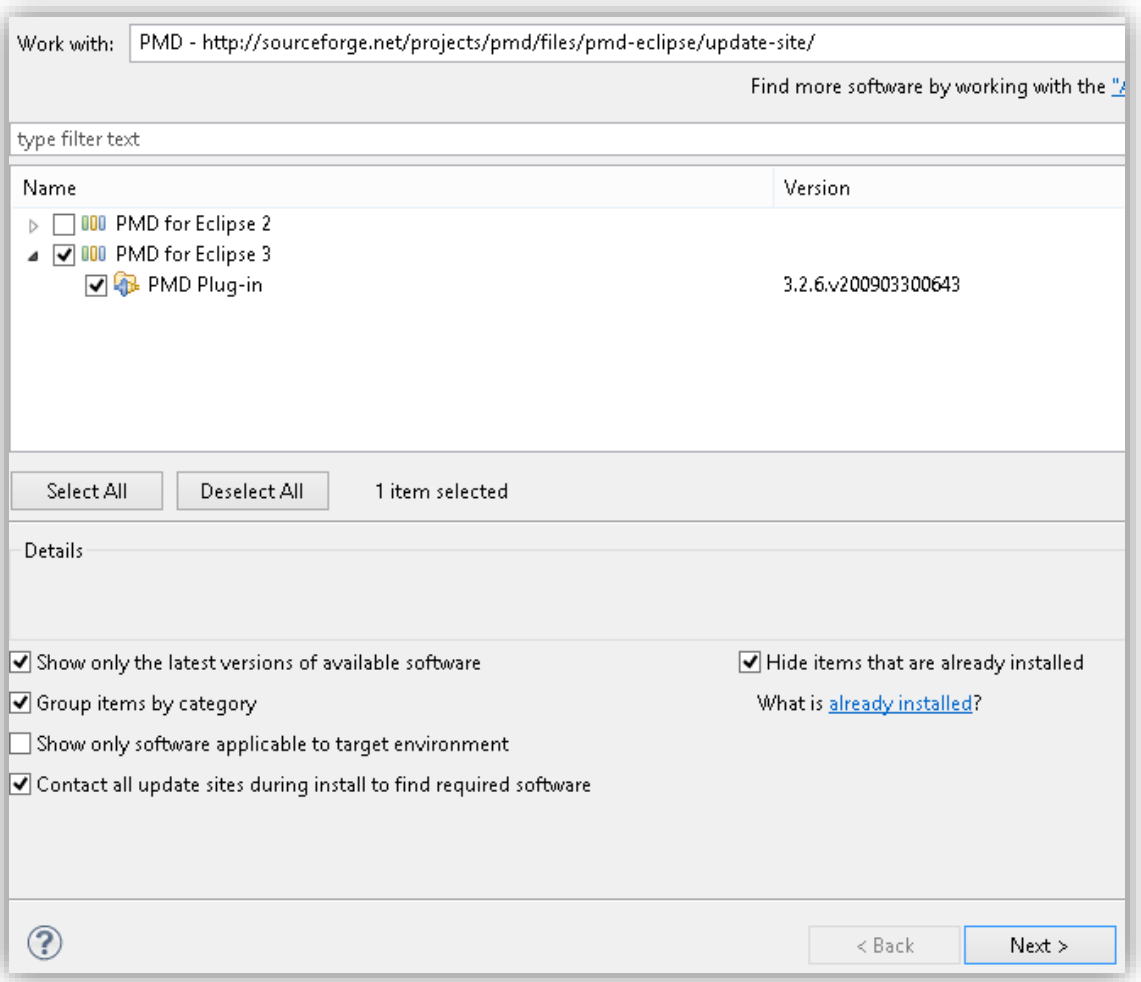


Configuramos la ubicación del sitio web oficial de descarga de Pmd en la opción “Add”: http://sourceforge.net/projects/pmd/files/pmd-eclipse/update-site/

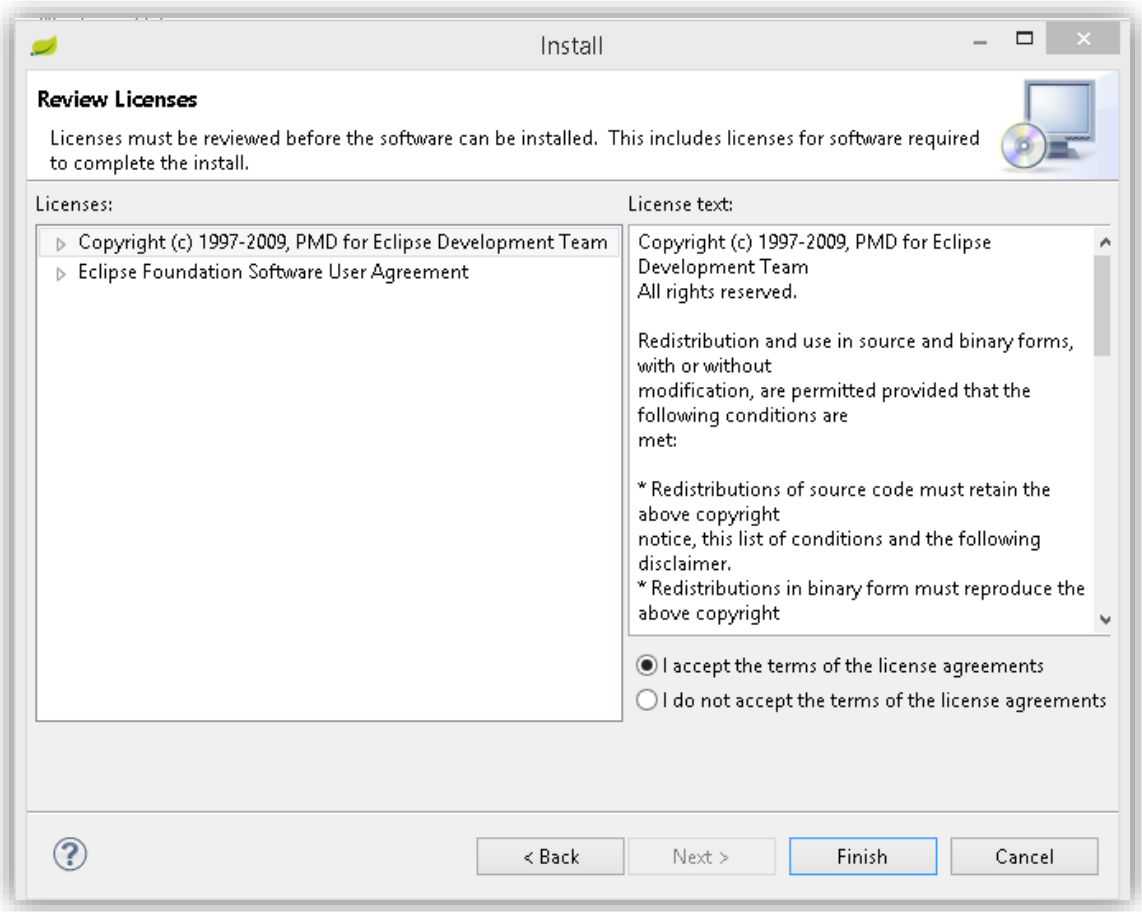




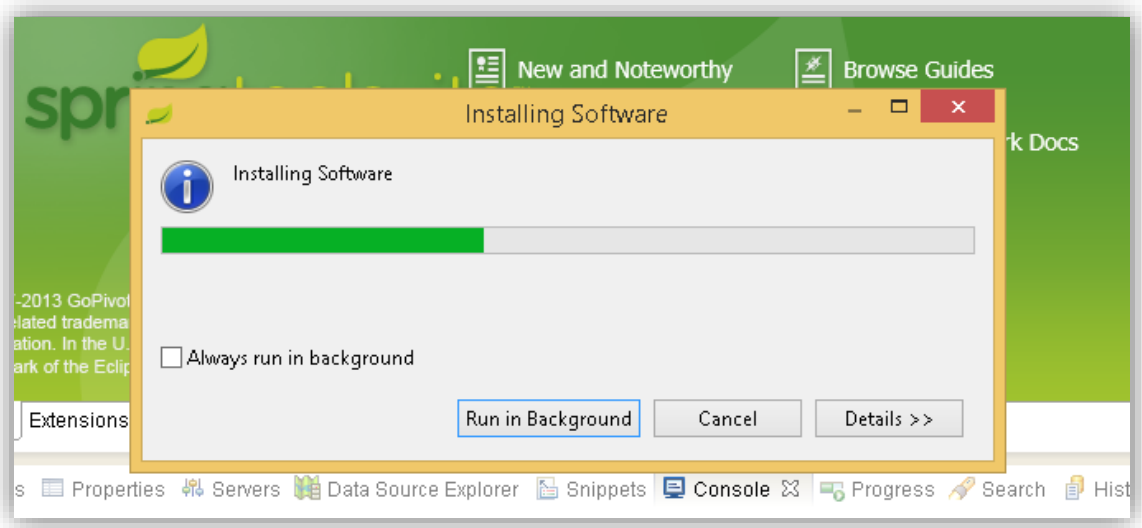
Escogemos la opción el plug-in para Eclipse 3 y se debe dar click en Next.



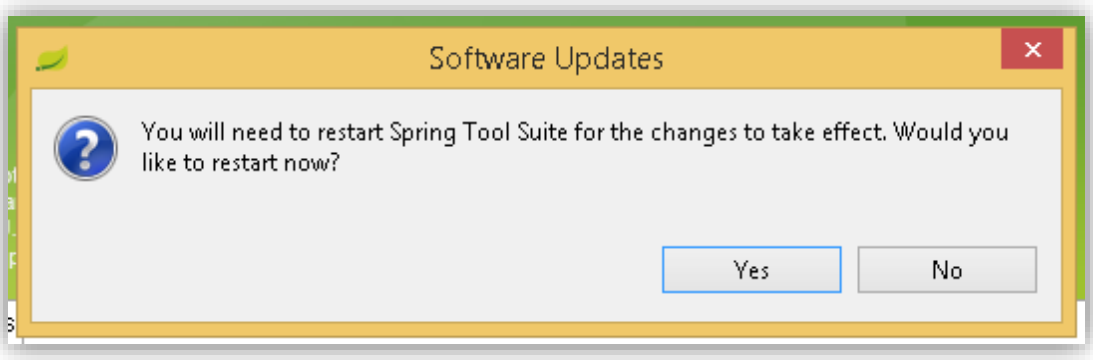
Se aceptan los términos de la licencia



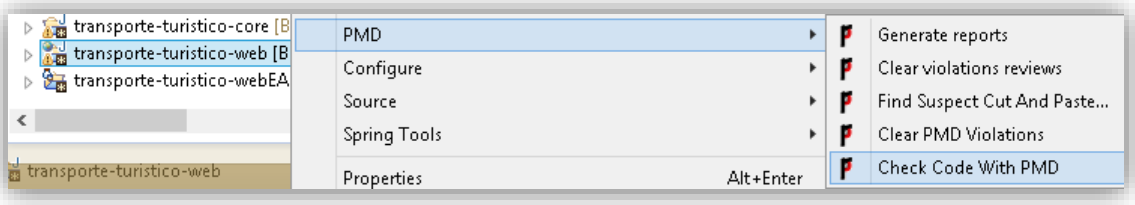
Al presionar en la tecla Finish el pluning será descargado de Internet



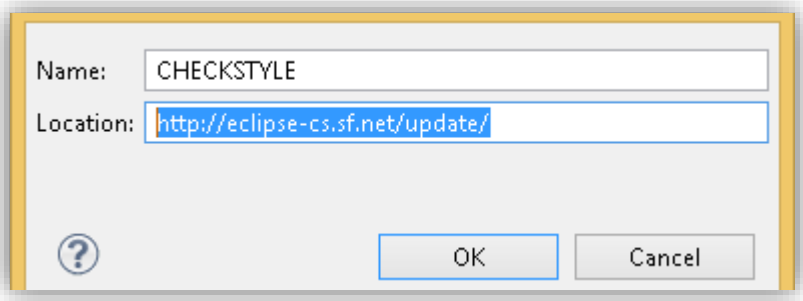
En cuanto finaliza la descarga se debe reiniciar el programa para utilizarlo.



Para poder analizar el código; se debe dar click derecho sobre el proyecto que se desea analizar, dirigirse a la opción PMD y elegir "Check code with PMD" así:



Al instalar el pluning de **Checstyle** seguimos los mismos pasos, con la página de descarga oficial que es: http://eclipse-cs.sf.net/update/



El uso de Checkstyle es similar al de PMD así: