Portada // TO DO //

Agradecimientos // TO DO //

Resumen // TO DO //

Palabras Clave // TO DO //

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

INTRODUCCIÓN

1.1. Contextualización

Durante las últimas décadas, el turismo ha experimentado un continuo crecimiento que lo ha llegado a convertirse en uno de los sectores económicos más importantes. Por su parte, el éxito de la telefonía móvil y el continuo uso de smartphones en la sociedad, promulga el desarrollo de las aplicaciones móviles, otro sector que continúa en pleno crecimiento.

El desarrollo de está aplicación está englobado entre los dos sectores anteriormente comentados.

Se pretende presentar a los usuarios finales una aplicación móvil, fácilmente accesible, que permita ofrecer un servicio que ayude a planificar rutas turísticas en un viaje determinado. A parte de la planificación de rutas, el usuario podrá descubrir diferentes lugares del mundo así como explorar los eventos que haya disponibles en el lugar donde realice dicho viaje.

1.2. Objetivos

1.3. Estructura de la memoria

Capítulo 2

FUNDAMENTOS TECNOLÓGICOS

2.1. Lenguajes utilizados

2.1.1. Java

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos. Fue diseñado para tener tan pocas dependencias de implementación como fuera posible tal que permitiera a los desarrolladores escribir el programa una vez y ejecutarlo en cualquier dispositivo sin necesidad de recompilarlo.

Fue originalmente desarrollado por James Gosling, de Sun Microsystems, la cual fue adquirida por la compañía Oracle.

Puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente y su sintaxis deriva en gran medida de lenguajes como C y C++, pero con menos utilidades de bajo nivel.

2.1.2. JavaScript

JavaScript es un lenguaje de programación interpretado, dialecto del estándar ECMAScript, orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Se utiliza principalmente del lado del cliente, implementado como parte de un navegador web, permitiendo mejoras en la interfaz de usuario y páginas web dinámi-

cas.

JavaScript se diseñó con una sintaxis similar a C, aunque adopta nombres y convenciones del lenguaje de programación Java. Sin embargo, Java y JavaScript tienen semánticas y propósitos diferentes.

2.1.3. TypeScript

TypeScript es un lenguaje de programación libre y de código abierto desarrollado y mantenido por Microsoft.

Este lenguaje es un superset del ya conocido JavaScript y que está pensado para grandes proyectos, los cuáles a través de un compilador de TypeScript se traducen a código JavaScript original.

Un aspecto característico de TypeScript es su sistema de tipos. Permite a los desarrolladores definir variables y funciones tipadas sin perder la esencia de JavaScript gracias a una representación estática de los tipos dinámicos. Definir tipos durante el diseño, nos ayudará a evitar errores en tiempo de ejecución.

2.1.4. HyperText Markup Language

HyperText Markup Language, conocido comúnmente por sus siglas, HTML, es un lenguaje de marcado cuya finalidad es la elaboración de páginas web. Define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web.

Es un estándar a cargo del World Wide Web Consortium (W3C), organización dedicada a la estandarización de casi todas las tecnologías ligadas a la web.

2.1.5. Cascading StyleSheets

Las hojas de estilo en cascada (o CSS, por sus siglas en inglés) es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado. Especifica como se mostrarán por pantalla los denominados elementos HTML.

Junto con HTML y JavaScript, CSS es una tecnología usada por muchos sitios web para crear páginas web visualmente atractivas, interfaces de usuario de aplicaciones web y GUIs para muchas aplicaciones móviles.

Su principal objeto es mantener la separación del contenido del documento de su forma de presentación. Con las hojas de estilo se puede prescindir del uso de formatos de estilo dentro de la propia página HTML, de manera que se pueda modificar el estilo de toda una web modificando un único archivo CSS.

2.2. Frameworks, librerías y técnicas de desarrollo

2.2.1. Spring

Spring es un framework cuya finalidad es facilitar el desarrollo de aplicaciones desarrolladas en Java. Es de código abierto y la primera versión fue elaborada por Rod Johnson. A pesar de que no impone ningún modelo de programación en particular, este framework se ha vuelto popular en la comunidad al ser considerado una alternativa, sustituto, e incluso un complemento a varias APIs de Java EE.

Spring está compuesto de diversos módulos que se pueden agregar a nuestras aplicaciones, permitiendo a los desarrolladores agregar sólo los módulos que vayan usar. El único módulo necesario para trabajar con Spring es el Spring Core puesto que es el que contiene la DI (Inyección de Dependencias) y la configuración de uso de objetos Java.

Spring MVC

Spring MVC es un framework de aplicaciones web basado en el patrón MVC(model-view-controller) y que alberga todas las ventajas del framework de Spring.

- Separación clara de roles. Cada objeto controlador, validador, formulario, de modelo pueden ser realizados por objetos especializados.
- Configuración potente y directa. Capacidad de configuración que permite una

fácil referencia a través de contextos, como por ejemplo, desde controladores web a objetos de negocio.

- Adaptabilidad, flexibilidad y no intrusividad. Definir los métodos de cualquier controlador utilizando las anotaciones de parámetros (como @RequestParam, @RequestHeader, @PathVariable, y más).

- Código de negocio reutilizable. No existe necesidad de duplicación.

Spring MVC es, como otros frameworks MVC, basado en solicitud (request-driven). Están diseñados en torno a un Servlet central que sirve las solicitudes a los controladores y ofrece unas funcionalidades que facilitan el desarrollo de las aplicaciones web. Sin embargo, el DispatcherServlet de Spring es más que eso, está completamente integrado con el contenedor Spring IoC y permite hacer uso de las características y funcionalidades de Spring.

Spring Security

Spring Security ofrece exhaustivos servicios de seguridad para las aplicaciones empresariales basadas en Java EE. Las dos principales áreas en las que se enfoca Spring Security son la Autenticación y la Autorización, probablemente, los dos temas más relevantes en la seguridad de las aplicaciones.

- Autenticación es el proceso por el que se determina que uno es el que dice ser.

- Autorización hace referencia al proceso de determinar qué acción o acciones puede realizar en la aplicación.

A nivel de autenticación, Spring Security soporta un amplio rango de modelos de autenticación. La mayoría de estos modelos de autenticación son proporcionados por terceros, o desarrollados por los organismos estándar pertinentes, como Internet Engineering Task Force. A mayores, Spring Security provee su propio conjunto de mecanismos de autenticación y soporta integración de autenticación con diferentes tecnologías

2.2.2. Java Persistence API

Java Persistence API, comúnmente conocida por sus siglas JPA, es la API que describe la gestión de datos relacionales en aplicaciones que utilicen Java. La primera especificación fue lanzada en mayo de 2006 como parte del trabajo del JSR 220.

JPA en sí mismo es solo una especificación, no un producto. Son un conjunto de interfaces que requieren una implementación. Existen implementaciones de JPA de código abierto y comerciales, y cualquier servidor de aplicaciones JAVA EE 5 debe proporcionar soporte para su uso.

El objetivo de esta API es no perder las ventajas de la orientación a objetos al interactuar con una base de datos y permitir usar objetos regulares, comúnmente conocidos como POJOs.

2.2.3. JAX-RS

JAX-RS es la API de Java para la elaboración de servicios web RESTful que brinda soporte en la creación de servicios web de acuerdo con el patrón arquitectónico REST. Desde la versión 1.1, JAX-RS es una parte oficial de Java EE 6. Una característica notable de ser parte oficial de Java EE es que no es necesaria ninguna configuración para comenzar a utilizar JAX-RS.

Esta API utiliza anotaciones, introducidas en Java SE 5, para simplificar el desarrollo y la implementación de clientes y recursos web.

De la misma forma que sucedía con JPA, JAX-RS no es más que una especificación, necesita un producto que la implemente. Jersey y RESTEasy son implementaciones de JAX-RS,

2.2.4. Thymeleaf

Thymeleaf es una librería Java que implementa un motor de plantillas válidas para entornos web cómo independientes. Es un software de código abierto creado originalmente por un ingeniero de software español llamado Daniel Fernández. No está hecho ni respaldado por ningún software de ninguna compañía y se ofrece al público de manera totalmente gratuita, tanto en formato binario como en código

fuente, bajo licencia Apache.

Su objetivo principal es la creación de plantillas de una manera elegante y con un código bien formateado.

Thymeleaf ofrece una buena integración con Spring MVC a través de su dialecto SpringStandard, pero esta integración con Spring es completamente opcional y el dialecto estándar está destinado a usarse sin Spring.

2.2.5. jQuery

jQuery es una librería multiplataforma de JavaScript, creada incialmente por John Resig. Es un software libre y de código abierto, y posee un doble licenciamiento bajo la licencia MIT y la licencia Pública General de GNU, permitiendo su uso en proyectos libres y privados.

Su objetivo es la realización de funcionalidades basadas en JavaScript de forma rápida y sencilla. Permite realizar recorridos y manipulaciones de documentos HTML, manejar eventos, animaciones y usar AJAX mucho más simple con una API fácil de usar que funciona en multitud de navegadores.

2.2.6. Bootstrap

Bootstrap es un kit de herramientas de código abierto para desarrollo web junto a HTML, CSS y JS.

Originalmente creado por un diseñador y desarrollador de Twitter, Bootstrap es uno de los frameworks front-end y proyectos de código abierto más populares en el mundo. Antes de ser un framework de código abierto, Bootstrap era conocido como Twitter Blueprint.

Bootstrap incluye plantillas de diseño basadas en HTML y CSS para tipografías, formularios, botones, tablas,... así como complementos de JavaScript opcionales. También brinda la capacidad de crear fácilmente diseños receptivos.

2.2.7. AJAX

AJAX es una técnica de desarrollo web para crear aplicaciones interactivas. Es una tecnología asíncrona, en el sentido que los datos adicionales solicitados al servidor, se cargan en segundo plano sin inferir con la visualización ni el comportamiento de la página.

La funciones de llamada AJAX se efectúan, normalmente, bajo el lenguaje de programación JavaScript mientras que el acceso a los datos se realiza mediante el objeto JavaScript XMLHttpRequest.

AJAX no constituye una tecnología en sí misma, sino que es un término que engloba a un grupo de éstas que trabajan conjuntamente.

- XHTML (o HTML) y CSS para el diseño que acompaña a la información.
- Document Object Model (DOM) accedido con un lenguaje de scripting por parte del usuario, generalmente, JavaScript.
- El objeto XMLHttpRequest para intercambiar datos de forma asíncrona con el servidor web.
- XML es el formato usado generalmente para la transferencia de datos solicitados al servidor.

2.3. Herramientas de desarrollo

2.3.1. Eclipse

Eclipse es una plataforma software compuesto por un conjunto de herramientas de programación de código abierto multiplataforma.

Fue desarrollado originalmente por IBM como el sucesor de su familiar de herramientas VisualAge. Ahora, está siendo desarrollado por la Fundación Eclipse, una organización independiente, sin ánimo de lucro, que fomenta una comunidad de código abierto y un conjunto de productos complementarios.

2.3.2. Maven

Maven es una herramienta de software para la gestión y construcción de proyectos Java creada por Jason van Zyl en 2002 que tiene un modelo de configuración de construcción basado en formato XML.

Maven utiliza un Project Object Model, conocido como POM, para describir el proyecto software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos.

Una de sus características clave son su disponibilidad para usarse en la red puesto que el motor incluido en su núcleo puede dinámicamente descargar plugins de un repositorio.

2.3.3. Git

Git es un sistema de control de versiones distribuido de código abierto y gratuito diseñado para manejar todo, desde proyectos pequeños a muy grandes, con velocidad y eficiencia.

Originalmente fue diseñado como un motor de sistema de control de versiones de bajo nivel sobre el cual otros podían codificar interfaces frontales. Desde entonces hasta ahora, el núcleo del proyecto Git se ha vuelto un sistema de control de versiones completo, utilizable en forma directa.

Git es fácil de aprender y ofrece un rendimiento increíblemente rápido. Su principal objetivo es llevar el registro de cambios en archivos y coordinar el trabajo que varias personas realizan sobre archivos compartidos.

2.3.4. Oracle SQL Developer

Oracle SQL Developer es un entorno de desarrollo integrado y gratuito que simplifica el desarrollo y la administración de las bases de datos de Oracle, tanto de implementaciones tradicionales como en la nube.

Este software admite productos de Oracle y una grana variedad de complementos de terceros que los usuarios pueden implementar para conectarse a bases de datos

que no sean de Oracle.

SQL Developer ofrece un desarrollo completo de extremo a extremo de sus aplicaciones PL / SQL, una hoja de trabajo para ejecutar consultas y scripts, una consola DBA para administración, una interfaz de informes y una solución completa de modelado de datos.

2.4. Sistema de gestión de bases de datos

2.4.1. Oracle Database

Oracle Database es un sistema de gestión de base de datos de tipo objeto-relacional, desarrollado por Oracle Corporation. Es la base de datos más popular para el procesamiento de transacciones el línea (OLTP) y almacenes de datos (Data warehousing).

La tecnología Oracle se encuentra prácticamente en todas las industrias alrededor del mundo y es el proveedor mundial líder de software para administración de información.

El producto Oracle para el sistema de base de datos cuenta con 7 ediciones diferentes de las cuales, una es completamente gratuita.

2.5. Servidor de aplicaciones

2.5.1. Apache Tomcat

Apache Tomcat funciona como un contenedor de servlets desarrollado por Apache Software Foundation. Es un una implementación de código abierto de las tecnologías Java Servlet, JavaServer Pages, Java Expression Language y Java WebSocket. Se publica bajo la versión 2 de Apache License.

Tomcat no es un servidor de aplicaciones, pero puede funcionar como servidor web por sí mismo. Actualmente es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.

Está escrito en Java, de manera que puede funcionar en cualquier sistema operativo que disponga de la máquina virtual Java.

2.6. Otros

2.6.1. Ionic

Ionic es un completo SDK de código abierto diseñado para el desarrollo de aplicaciones móviles híbridas. La versión original fue lanzada en 2013 y construida sobre AngularJS y Apache Cordova. Las versiones más recientes, conocidas como Ionic 3, están construidas sobre Angular.

Proviene de herramientas y servicios para desarrollar aplicaciones móviles híbridas usando tecnologías web (CSS, HTML,...).

Ionic proviene todas la funcionalidades que se puede encontrar en los SDK de desarrollo de aplicaciones nativas. Las aplicaciones construidas se pueden personalizar en función del uso final, Android o iOS, por ejemplo.

Además del SDK, Ionic también brinda servicios que los desarrolladores pueden usar para habilitar funciones, como la tecnología push, test A/B, construcciones automáticas, etc...

Ionic también proporciona una poderosa interfaz de línea de comandos, lo que permite poder comenzar a desarrollar un proyecto con un simple comando. También permite a los desarrolladores agregar complementos de Cordova.

Capítulo 3

METODOLOGÍA

El Proceso Unificado es un marco de desarrollo software caracterizado por estar dirigido por casos de uso, centrado en la arquitectura y por ser iterativo e incremental.

3.1. Características

Para el desarrollo de este proyecto se ha escogido una metodología basada en el Proceso Unificado. Esta metodología nos permite obtener un producto de alta calidad gracias a su carácter iterativo e incremental. De manera que, en cada ciclo o iteración, el producto es revisado y mejorado.

3.1.1. Dirigido por casos de uso

Un caso de uso es un fragmento de funcionalidad del sistema que proporciona un resultado de valor a un usuario. Los casos de uso modelan los requerimientos funcionales del sistema y todos juntos constituyen el modelo de casos de uso.

Los casos de uso también guían el proceso de desarrollo (diseño, implementación y prueba). Basándose en los casos de uso, los desarrolladores crean una serie de modelos de diseño e implementación que llevan a cabo. De este modo, los casos de uso no son solo una herramienta para especificar los requerimientos e iniciar el proceso de desarrollo, sino que también dirigen su diseño, implementación, pruebas, etc...

3.1.2. Centrado en la arquitectura

La arquitectura de un sistema software se describe mediante diferentes vistas del sistema en construcción. Se define arquitectura como el conjunto de decisiones significativas acerca de la organización de un sistema software, la selección de los elementos estructurales a partir de los cuales se compone el sistema y su composición.

Los casos de uso y la arquitectura están profundamente relacionados. Los casos de uso deben encajar en la arquitectura, y esta a su vez, debe permitir el desarrollo de todos los casos de uso requeridos, actualmente y a futuro.

De tal forma que, el arquitecto software desarrolla la forma o arquitectura a partir de la comprensión de un conjunto reducido de casos de uso fundamentales o críticos.

3.1.3. Iterativo e incremental

Se divide el desarrollo de un proyecto software en partes más pequeñas o mini proyectos. Cada mini proyecto es una iteración que supono un incremento. Las iteraciones hacen referencia a pasos en el flujo de trabajo, y los incrementos, a crecimientos en el producto.

Las iteraciones deben estar controladas de manera que se deben seleccionar y ejecutar de forma planificada.

En cada iteración los desarrolladores identifican y especifican los casos de uso relevantes y crean un diseño utilizando la arquitectura seleccionada para implementar dichos casos de uso. Si la iteración cumple sus objetivos, se continúa con la próxima, si no, deben revisarse las decisiones previas y probar un nuevo enfoque.

3.2. Ciclo de vida

El Proceso Unificado se repite a lo largo de una serie de ciclos que constituyen la vida de un sistema.

Cada ciclo consta de cuatro fases: **Inicio**, **Elaboración**, **Construcción** y **Transición**. Cada fase se divide en iteraciones, en las cuales se desarrolla, secuencialemte, una serie de disciplinas o flujos de trabajo como por ejemplo: *Análisis, Diseño, Imple-*

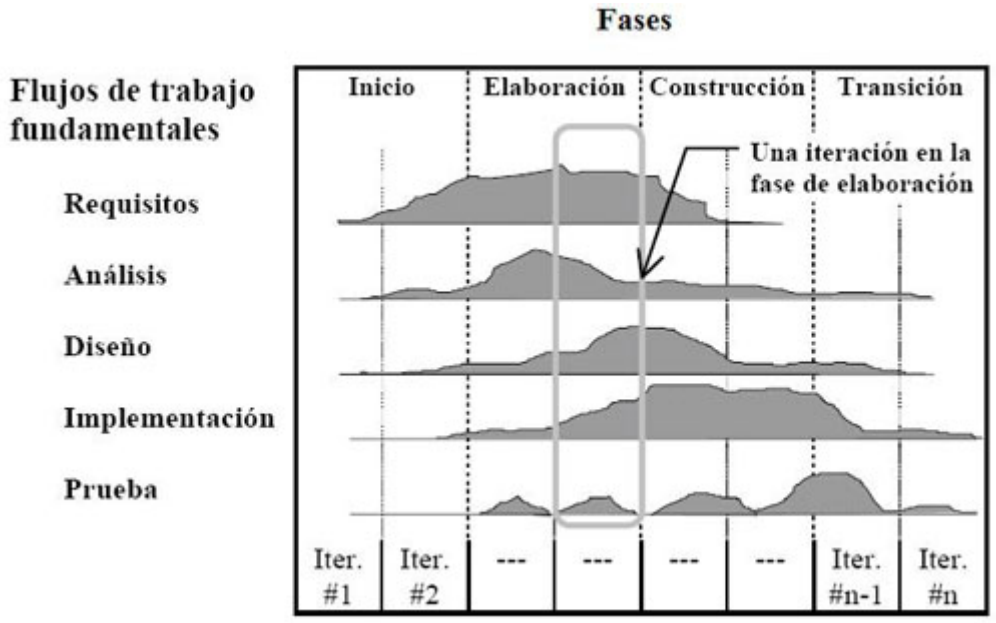

Figura 3.1: Ciclo de vida - Proceso Unificado de Desarrollo

mentación y Pruebas.

En la figura podemos observar que el ciclo de vida del Proceso Unificado presenta dos dimensiones:

- Un eje horizontal que representa el aspecto dinámico del proceso conforme se va desarrollando. Expresado en términos de *Fases, Iteraciones e Hitos.*
- Un eje vertical que representa el aspecto estático del proceso mediante las disciplinas o flujos de trabajo.

3.2.1. Fases

3.2.2. Fase de inicio

Durante la fase de inicio se desarrolla una descripción del producto final y se presenta el análisis de negocio.

El objetivo de esta fase es ayudar al equipo de proyecto a decidir cuales son los verdaderos objetivos del proyecto.

3.2.3. Fase de elaboración

En esta fase se especifican la mayoría de los casos de uso del producto y se diseña la arquitectura del mismo. Se establece una firme comprensión del problema a solucionar.

3.2.4. Fase de construcción

Durante la fase de construcción se elabora el producto, de tal manera que la línea base de la arquitectura avanza hasta convertirse en el sistema completo. Al final de esta fase se obtienen todos los casos de uso implementados aunque pueden incluir defectos.

3.2.5. Fase de transición

La fase de transición hace referencia al período donde el software ya debe estar listo para ser instalado, probado y utilizado. Las iteraciones en esta fase continúan agregando características al software.

3.3. Iteraciones

3.3.1. Iteración 1: Análisis de requisitos y diseño

En esta primera iteración se determinará la planificación del proyecto, se realizará la recogida de requesitos y el modelado de casos de uso. También se especificará el diseño del sistema y se elaborará el diagrama de clases de la aplicación.

3.3.2. Iteración 2: Base inicial del proyecto

La segunda iteración implicará el comienzo del desarrollo del proyecto. Se preparán los entornos necesarios, se realizará una pequeña formación en las tecnologías que se emplearán y se preparará y configurará la base datos. Al hacer uso de APIs externas, en esta iteración se configurarán para su posterior uso.

3.3.3. Iteración 3: Usuarios

La iteración 3, tiene como objetivo la implementación de las funcionalidades relacionadas con los usuarios de la aplicación. Se realizará el análisis y diseño, así como la implementación de la persistencia y lógica de negocio para la gestión de los usuarios. Finalmente, se realizarán las pruebas necesarias.

3.3.4. Iteración 4: Rutas

En esta iteración se realizará el análisis y diseño, la implementación de la persistencia y la lógica de negocio, y las pruebas necesarias para la gestión de las rutas en la aplicación.

3.3.5. Iteración 5: Eventos

Análogamente, en la iteración número 5, se realizará el mismo ciclo de acciones. Análisis y diseño, implementación de la persistencia y lógica de negocio, y pruebas, esta vez, para la gestión de eventos.

3.3.6. Iteración 6: Servicios externos

En esta iteración se llevará a cabo la gestión de las funcionalidades y datos obtenidos de las fuentes externas. Se realizará el análisis y diseño para la implementación de los servicios externos necesarios, se realizarán las modificaciones necesarias para el correcto funcionamiento de la librería Java de Foursquare, así como la implementación y pruebas necesarias para dichos servicios.

3.3.7. Iteración 7: Lugares y categorías

Se realizará análisis, diseño, implementación y pruebas para la gestión de lugares y categorías, cuyos datos serán obtenidos por los servicios definidos e implementados en la iteración anterior.

3.3.8. Iteración 8: Capa servicios

Para la iteración 8 se llevará a cabo el desarrollo de la capa de servicios del modelo. Se realizará el análisis, diseño, implementación y pruebas.

3.3.9. Iteración 9: Autenticación y Autorización

En este iteración se llevará a cabo el proceso de autenticación y autorización de los usuarios. Se realizará el análisis y diseño necesarios, así como la implementación y pruebas necesarias.

3.3.10. Iteración 10: Cliente móvil

De igual forma, en esta iteración, se realizará anáisis, diseño, implementación y pruebas para la elaboración del cliente móvil. Se realizará también una formación en las tecnologías utilizadas en su creación, la implementación de las pantallas y controladores, y de los módulos de acceso de a los servicios previamente implementados.

3.3.11. Iteración 11: Cliente web

Iteración cuyo objetivo es el análisis y desarrollo del cliente web. De igual forma que para en cliente móvil, se realizará el análisis y diseño necesario, la implementación del módulo de acceso a los servicios y las pantallas y controladores. Posteriormente, se realizarán las pruebas necesarias.

3.3.12. Iteración 12: Panel de administración

En esta iteración se realizará una ampliación del cliente web con la creación de un panel de administración de la aplicación. Se realizará análisis y diseño, junto a la implementación y pruebas.

3.3.13. Iteración 13: Cierre

En la última iteración se realizará la evaluación final del sistema y su revisión.

Capítulo 4

PLANIFICACIÓN Y COSTES

4.1. Planificación

4.1.1. División de las tareas

Para las iteraciones definidas anteriormente se establecerá una división en tareas, sobre las que se realizará la planificación necesaria para obtener la estimación del esfuerzo requerido en cada iteración.

A continuación, se mostrará las diferentes tareas que componen cada una de las iteraciones junto con los valores del tiempo necesario para su realización (en horas).

Iteración 1: Análisis de requisitos y diseño

		Recursos		
Tarea	Tiempo	Jefe Proyecto	Analista	Programador
Planificación	*8*	*2*	*6*	-
Modelado de casos de uso	*12*	*4*	*8*	-
Diseño de la arquitectura	*8*	*2*	*6*	-
Diseño diagrama de clases	*8*	*2*	*6*	-
Total	**36**	**10**	**26**	**0**

Figura 4.1: Diagrama división tareas - iteración 1

Iteración 2: Base inicial proyecto

		Recursos		
Tarea	Tiempo	Jefe Proyecto	Analista	Programador
Preparación del entorno	*6*	-	-	*6*
Formación en las tecnologías	*8*	-	-	*8*
Configuración APIs externas	*4*	-	-	*4*
Creación Base de Datos	*6*	-	-	*6*
Total	24	0	0	24

Figura 4.2: Diagrama división tareas - iteración 2

Iteración 3: Usuarios

		Recursos		
Tarea	Tiempo	Jefe Proyecto	Analista	Programador
Análisis y Diseño	*6*	*2*	*4*	-
Implementación persistencia	*4*	-	-	*4*
Implementación lógica de negocio	*12*	-	-	*12*
Pruebas	*6*	-	-	*6*
Total	**28**	**2**	**4**	**22**

Figura 4.3: Diagrama división tareas - iteración 3

Iteración 4: Rutas

		Recursos		
Tarea	Tiempo	Jefe Proyecto	Analista	Programador
Análisis y Diseño	*10*	*2*	*8*	-
Implementación persistencia	*14*	-	-	*14*
Implementación lógica de negocio	*24*	-	-	*24*
Pruebas	*8*	-	-	*8*
Total	**56**	**2**	**8**	**46**

Figura 4.4: Diagrama tareas iteración 4

Iteración 5: Eventos

		Recursos		
Tarea	Tiempo	Jefe Proyecto	Analista	Programador
Análisis y Diseño	*8*	*2*	*6*	-
Implementación persistencia	*12*	-	-	*12*
Implementación lógica de negocio	*24*	-	-	*24*
Pruebas	*8*	-	-	*8*
Total	**52**	**2**	**6**	**44**

Figura 4.5: Diagrama división tareas - iteración 5

Iteración 6: Servicios externos

		Recursos		
Tarea	Tiempo	Jefe Proyecto	Analista	Programador
Análisis y Diseño	*4*	*2*	*2*	-
Modificación Librería	*8*	-	-	*8*
Implementación lógica de negocio	*12*	-	-	*12*
Pruebas	*4*	-	-	*4*
Total	**28**	**2**	**2**	**24**

Figura 4.6: Diagrama división tareas - iteración 6

Iteración 7: Lugares y categorías

		Recursos		
Tarea	Tiempo	Jefe Proyecto	Analista	Programador
Análisis y Diseño	*6*	*2*	*4*	-
Implementación persistencia	*8*	-	-	*8*
Implementación lógica negocio	*14*	-	-	*14*
Pruebas	*4*	-	-	*4*
Total	**32**	**2**	**4**	**26**

Figura 4.7: Diagrama tareas iteración 7

Iteración 8: Capa servicios

		Recursos		
Tarea	Tiempo	Jefe Proyecto	Analista	Programador
Análisis y Diseño	*8*	*2*	*6*	-
Implementación servicios	*42*	-	-	*42*
Pruebas	*16*	-	-	*16*
Total	**66**	**2**	**6**	**58**

Figura 4.8: Diagrama división tareas - iteración 8

Iteración 9: Autenticación y autorización

		Recursos		
Tarea	Tiempo	Jefe Proyecto	Analista	Programador
Análisis y Diseño	*10*	*2*	*8*	-
Implementación autenticación	*26*	-	-	*26*
Implementación autorización	*22*	-	-	*22*
Pruebas	*12*	-	-	*12*
Total	**70**	**2**	**8**	**60**

Figura 4.9: Diagrama división tareas - iteración 9

Iteración 10: Cliente móvil

		Recursos		
Tarea	Tiempo	Jefe Proyecto	Analista	Programador
Análisis y Diseño	*16*	*4*	*12*	-
Formación Ionic - Angular	*8*	-	-	*8*
Implementación acceso servicios	*14*	-	-	*14*
Implementación vista	*86*	-	-	*86*
Pruebas	*20*	-	-	*20*
Total	**144**	**4**	**12**	**128**

Figura 4.10: Diagrama división tareas - iteración 10

Iteración 11: Cliente web

		Recursos		
Tarea	Tiempo	Jefe Proyecto	Analista	Programador
Análisis y Diseño	*12*	*4*	*8*	-
Implementación acceso servicios	*6*	-	-	*6*
Implementación vista	*32*	-	-	*32*
Pruebas	*10*	-	-	*10*
Total	**60**	**4**	**8**	**48**

Figura 4.11: Diagrama división tareas - iteración 11

Iteración 12: Panel de administración

		Recursos		
Tarea	Tiempo	Jefe Proyecto	Analista	Programador
Análisis y Diseño	*12*	*4*	*8*	-
Implementación acceso servicios	*6*	-	-	*6*
Implementación vista	*28*	-	-	*28*
Pruebas	*8*	-	-	*8*
Total	**54**	**4**	**8**	**42**

Figura 4.12: Diagrama división tareas - iteración 12

Iteración 13: Cierre

		Recursos		
Tarea	Tiempo	Jefe Proyecto	Analista	Programador
Evaluación y pruebas final	*16*	-	-	*16*
Revisión	*24*	*4*	*12*	*8*
Total	**40**	**4**	**12**	**24**

Figura 4.13: Diagrama división tareas - iteración 13

Planificación global

		Recursos		
Iteración	Tiempo	Jefe Proyecto	Analista	Programador
Iteración 1 : Análisis y Diseño	*36*	*10*	*26*	*0*
Iteración 2 : Base inicial Proyecto	*24*	*-*	*-*	*24*
Iteración 3 : Usuarios	*28*	*2*	*4*	*22*
Iteración 4 : Rutas	*56*	*2*	*8*	*46*
Iteración 5 : Eventos	*52*	*2*	*6*	*44*
Iteración 6 : Servicios externos	*28*	*2*	*2*	*24*
Iteración 7 : Lugares y categorías	*32*	*2*	*4*	*26*
Iteración 8 : Capa Servicios	*66*	*2*	*6*	*58*
Iteración 9 : Autenticación	*70*	*2*	*8*	*60*
Iteración 10 : Cliente móvil	*144*	*4*	*12*	*128*
Iteración 11 : Cliente web	*60*	*4*	*8*	*48*
Iteración 12 : Panel administración	*54*	*4*	*8*	*42*
Iteración 13: Cierre	*38*	*4*	*6*	*28*
Total	**622**	**38**	**92**	**492**

Figura 4.14: Diagrama división tareas - global

4.1.2. Planificación de las tareas

En el siguiente diagrama se podrá observar la planificación total de proyecto así como la establecida en cada iteración.

	Moc de	Nombre de tarea	Duración	Comienzo	Fin	Prec
1		**Proyecto**	**86,5 días**	**lun 08/01/18**	**mar 08/05/18**	
2		**Iteración 1 : Análisis y Diseño**	**6 días**	**lun 08/01/18**	**lun 15/01/18**	
7		**Iteración 2 : Base Inicial**	**3,25 días**	**mar 16/01/18**	**vie 19/01/18**	**2**
12		**Iteración 3 : Usuarios**	**3,5 días**	**vie 19/01/18**	**mié 24/01/18**	**7**
17		**Iteración 4 : Rutas**	**7 días**	**mié 24/01/18**	**vie 02/02/18**	**12**
22		**Iteración 5 : Eventos**	**6,5 días**	**vie 02/02/18**	**mar 13/02/18**	**17**
27		**Iteración 6 : Servicios Externos**	**3 días**	**mar 13/02/18**	**vie 16/02/18**	**22**
32		**Iteración 7 : Categorías y Lugares**	**4 días**	**vie 16/02/18**	**jue 22/02/18**	**27**
37		**Iteración 8 : Capa Servicios**	**8,25 días**	**jue 22/02/18**	**mar 06/03/18**	**32**
41		**Iteración 9 : Autenticación y Autorización**	**8,75 días**	**mar 06/03/18**	**lun 19/03/18**	**37**
46		**Iteración 10 : Cliente Móvil**	**17 días**	**lun 19/03/18**	**mié 11/04/18**	**41**
52			**7,5 días**	**mié 11/04/18**	**vie 20/04/18**	**46**
57		**Iteración 11 : Cliente Web**	**6,75 días**	**vie 20/04/18**	**mar 01/05/18**	**52**
62		**Iteración 13 : Cierre**	**5 días**	**mar 01/05/18**	**mar 08/05/18**	**57**

Figura 4.15: Diagrama planificación - global

Observando el diagrama se puede observar la duración, la fecha de comienzo y de fin para el proyecto entero. Se muestra también la división y planificación del mismo, en las diferentes iteraciones definidas previamente. Para cada iteración, también se puede observar la planificación realizada.

En los siguiente diagramas se mostrará la planificación de las tareas para cada una de las iteraciones, indicando para cada una de ellas, el nombre, la fecha de inicio y fin, la duración, las tareas predecesoras y los recursos asignados encargados de realizar dicha tarea.

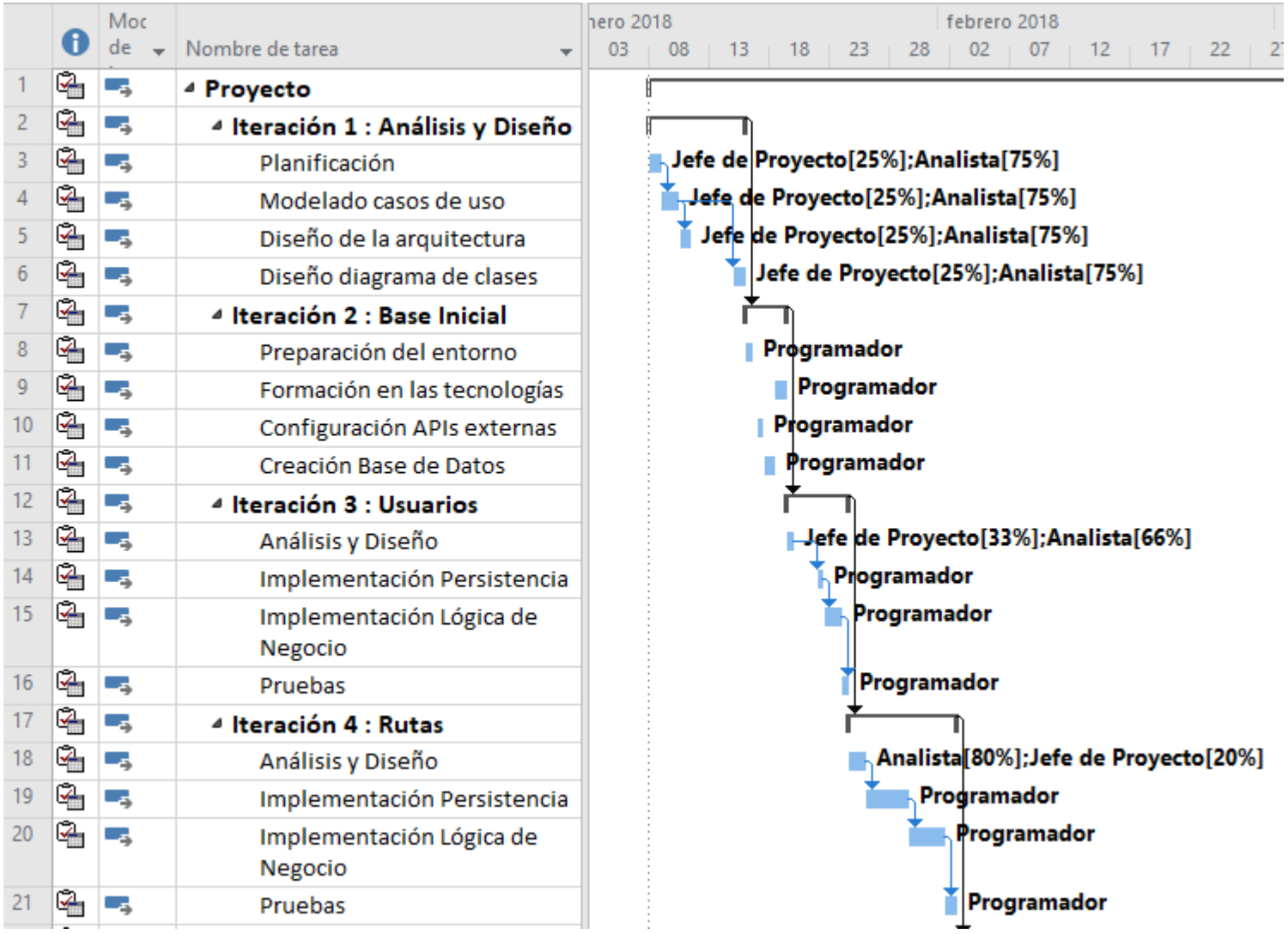

Figura 4.16: Diagrama planificación - iteraciones 1-4

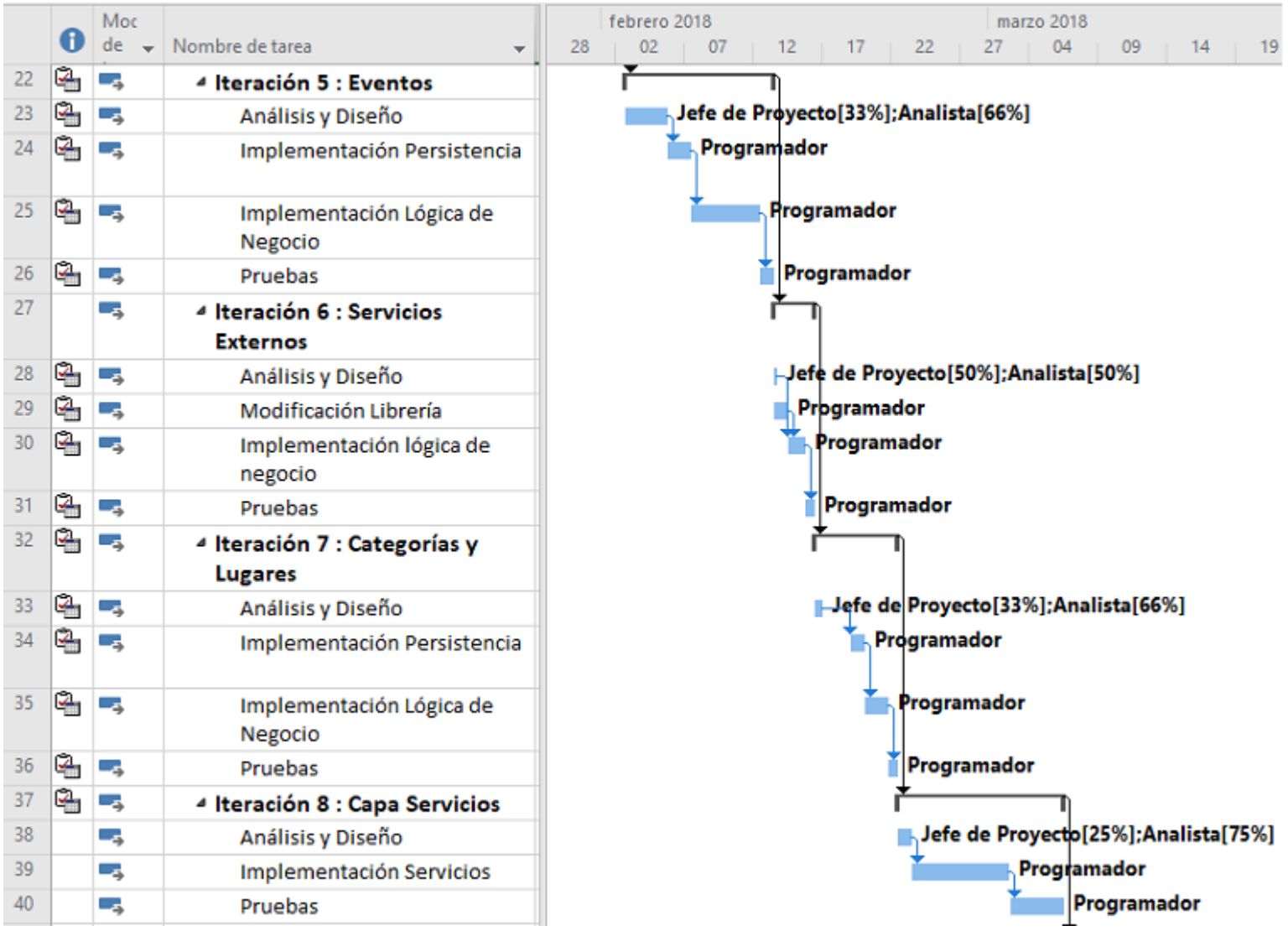

Figura 4.17: Diagrama Gantt - iteraciones 5-8

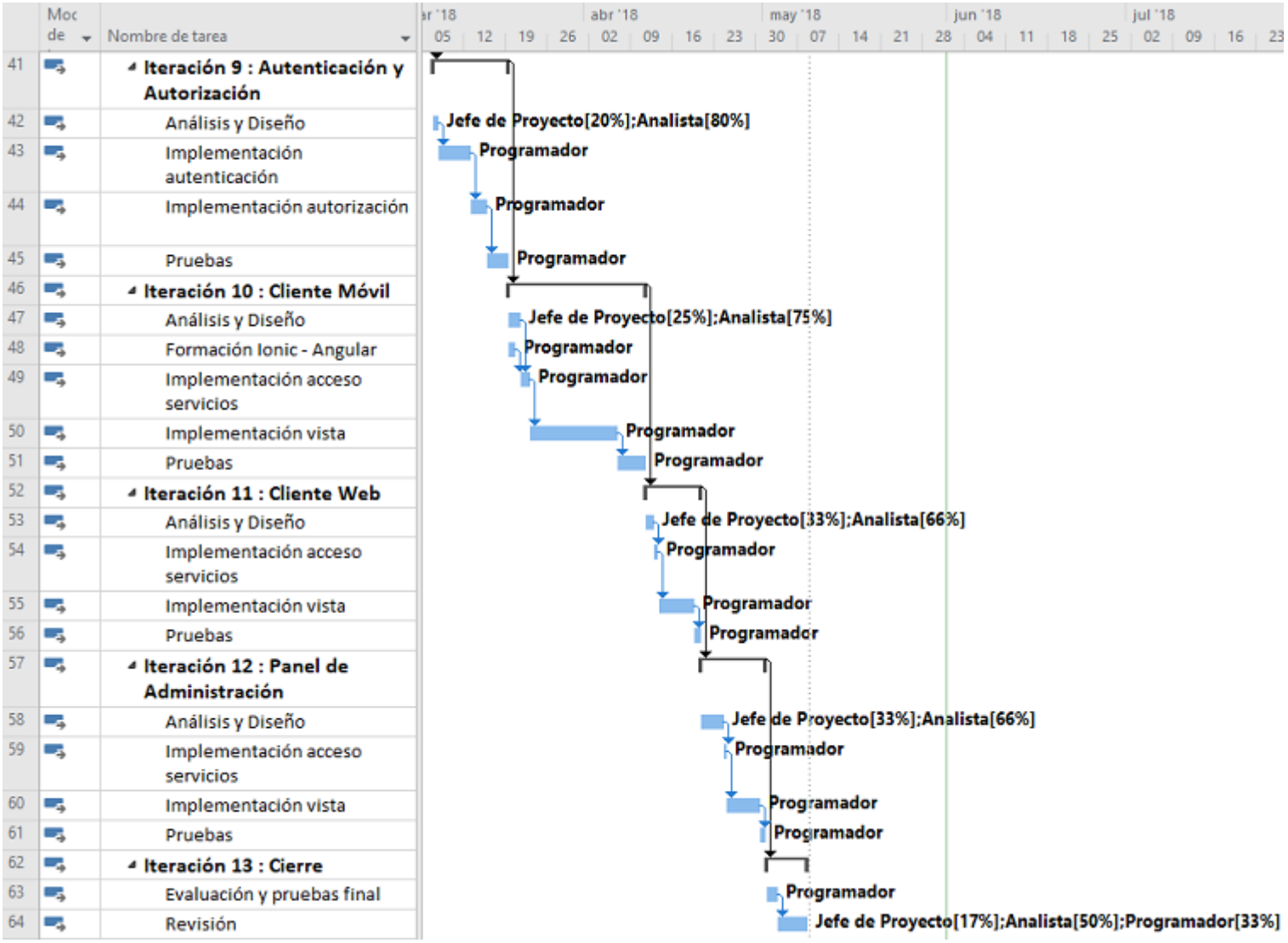

Figura 4.18: Diagrama Gantt - iteraciones 9-13

4.2. Evaluación de costes

4.2.1. Recursos Humanos

En la planificación del proyecto se han distinguido diferentes perfiles profesionales involucrados en la realización de las diferentes tareas. Conociendo el número total de horas trabajadas por cada recurso y su coste por hora, podemos calcular el coste total, de todos los recursos humanos, en el proyecto.

Recurso	**Coste/hora**	**Horas Trabajadas**	**Coste Total**
Jefe de Proyecto	20 €/h	38	760 €
Analista	18 €/h	92	1.656 €
Programador	16 €/h	492	7.872 €
Total			**10.288 €**

Tabla 4.1: Coste recursos humanos

4.2.2. Costes Hardware

En cuanto al coste hardware se contabilizará únicamente los equipos utilizados para el desarrollo del proyecto. Una vez que la aplicación se lleve a la etapa de producción, se deberán contabilizar los gastos que supondrían mantener los servidores y equipos necesarios para su correcto funcionamiento.

En el desarrollo se ha empleado un ordenador personal del autor, con un coste aproximado de 1.300 euros.

4.2.3. Costes Software

Las principales herramientas software utilizadas en el desarrollo del proyecto son gratuitas. Únicamente, destacar el uso de una versión para estudiantes gratuita del software MagicDraw, y el uso de las versiones estándar de las APIs de Foursquare y Google que implican una serie de límites y restricciones, asumibles en el desarrollo y pruebas de la aplicación.

Al igual que sucedía con los costes hardware, cuando la aplicación se encuentre en producción, será necesario asumir el coste de las versiones premium de las APIs mencionadas.

4.2.4. Costes totales

Como resultado de cada uno de las estimaciones de costes correspondientes a los recursos humanos, hardware y software; podemos estimar el coste total del desarrollo del proyecto.

Recurso	**Coste Total**
Recursos Humanos	10.288 €
Hardware	1.300 €
Software	0 €
Total	**11.588 €**

Tabla 4.2: Coste recursos humanos

Finalmente, se obtiene un coste total de 11.588 €.

Capítulo 5

ANÁLISIS

En este apartado se explicará, detalladamente, el análisis realizado.

5.1. Análisis de requerimientos

5.1.1. Requerimientos funcionales

A continuación se muestran los requerimientos funcionales del sistema, clasificados en distintas áreas.

Acceso a la aplicación

- El sistema ofrecerá la posibilidad de que un usuario se registre en la aplicación.
- El sistema ofrecerá la posibilidad de que el usuario se identifique en el sistema. Los usuarios deben ingresar al sistema con nombre de usuario y contraseña.

Cliente Móvil

- El sistema ofrecerá la posibilidad de crear nuevas rutas.
- El usuario podrá consultar las rutas, propias y de otros usuarios, según ciertos criterios de búsqueda.

- El sistema permitirá a los usuarios autorizados eliminar las rutas propias que deseen.
- El sistema permitirá a los usuarios autorizados consultar los detalles de las rutas.
- Para las rutas, el sistema permitirá:
  - Establecer las fechas de inicio y fin.
  - Consultar, asignar y desasignar a la ruta, los eventos disponibles en esas fechas.
  - Consultar el itinerario por días.
  - Modificar la hora de comienzo establecida para cada día.
  - Consultar, añadir y eliminar lugares de interés a cada día de la ruta.
  - Editar el modo de viaje a realizar entre diferentes lugares.
  - Mostrar el itinerario, por días y en total, en el mapa.
  - Habilitar y deshabilitar el sistema de geolocalización para conocer la ruta hecha en tiempo real.
  - Consultar y comparar, el itinerario definido con el obtenido a tiempo real.
  - Editar los permisos de la ruta.
- El cliente deberá mantener una sesión iniciada, cada vez que el usuario abra la aplicación.

Cliente Web

- El usuario podrá consultar las rutas existentes, propias y de otros usuarios, según ciertos criterios de búsqueda.
- El sistema permitirá a los usuarios autorizados consultar los detalles de las rutas.
- El sistema permitirá a los usuarios autorizados marcar las rutas propias como privadas, con el fin de no compartirlas con los demás usuarios.
- El sistema solo ofrecerá la posibilidad de consulta sobre los detalles de una ruta, permitiendo ver el itinerario, si tiene datos en tiempo real guardados, etc...

- El sistema permitirá a los usuarios autorizados eliminar las rutas propias que deseen.

Cliente Administración Web

- El sistema solo permitirá acceso a usuarios con permisos de administración.
- El sistema permitirá a los usuarios con dichos permisos, dar de alto nuevos usuarios.
- El sistema permitirá las altas, bajas, modificaciones y consultas de las entidades del sistema.
    - El sistema ofrecerá la posibilidad de crear, eliminar, modificar y consultar datos de usuarios.
    - El sistema ofrecerá la posibilidad de crear, eliminar, modificar y consultar datos de rutas.
    - El sistema ofrecerá la posibilidad de crear, eliminar, modificar y consultar datos de lugares.
    - El sistema ofrecerá la posibilidad de crear, eliminar, modificar y consultar datos de categorías.
    - El sistema ofrecerá la posibilidad de crear, eliminar, modificar y consultar datos de eventos.
- El sistema permitirá la existencia de usuarios con capacidades para la administración y gestión exclusivamente de los eventos. Permitiendo así, altas, bajas, modificaciones y consultas de los mismos en el sistema.

Seguridad

- El sistema ofrecerá la posibilidad de que el usuario modifique sus datos de acceso al sistema.
- El sistema solo permitirá acciones correctamente autenticadas, exceptuando las de acceso a la aplicación.

- Los usuarios de la aplicación solo podrán modificar los datos para los que tengan autorización. Un usuario no podrá modificar la información de los recursos de los que no es propietario.

5.1.2. Requerimientos no funcionales

Rendimiento

En condiciones normales, el tiempo de respuesta a las peticiones realizadas no deberá superar el máximo de 6 segundos.

Disponibilidad

La proporción de tiempo que el sistema debe estar en condiciones funcionales deberá ser del 99 %.

Portabilidad

El sistema diseñado y sus componentes deben ser capaces de ser transferidos entre plataformas GNU/Linux y Windows ofreciendo facilidad de adaptación e instalación.

Facilidad de Uso

La capacidad del software para ser comprendido, aprendido y utilizado de forma amigable por el usuario es un requerimiento esencial. Por ello, se debe ofrecer al usuario una interfaz sencilla y atractiva, con un manual de uso que describa el funcionamiento y uso del sistema final.

Seguridad

El sistema solo permitirá el acceso autorizado a la información y el intercambio de esta por la red será exclusivamente mediante el uso del protocolo encriptado https.

5.2. Modelo de casos de uso

5.2.1. Actores del sistema

Analizando los requerimientos funcionales del sistema, se detectan cuatro tipos de actores, que demandan una determinada funcionalidad en el sistema. Estos cuatro actores son, el cliente móvil, el cliente web, el administrador y el gestor de eventos.

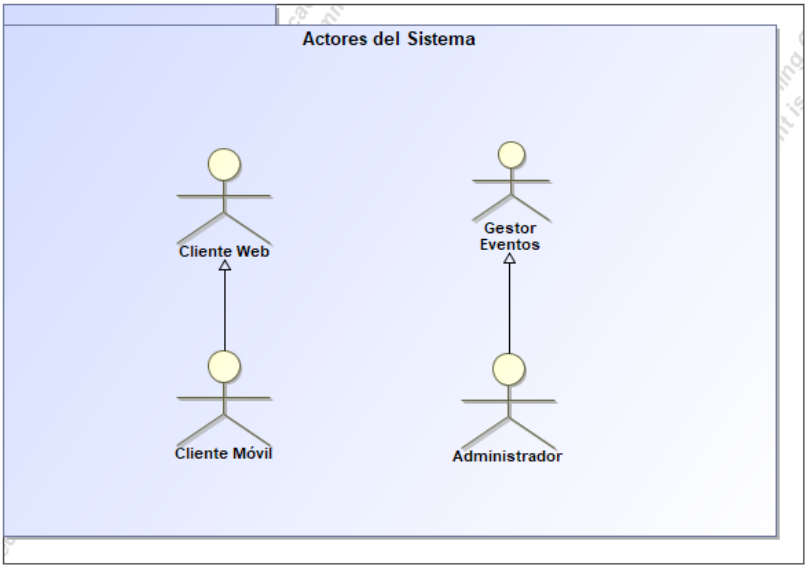

Figura 5.1: Diagrama casos de uso - Actores.

5.2.2. Diagrama de casos de uso

Tras conocer los requerimientos funcionales del sistema y reconocer las necesidades del sistema, se ha optado por diseñar un sistema general compuesto por los diferentes subsistemas del mismo.

Diagrama sistema general

Dicho sistema, estará compuesto por tres grupos de subsistemas: subsistema de acceso, subsistema de administración y subsistema de aplicación.

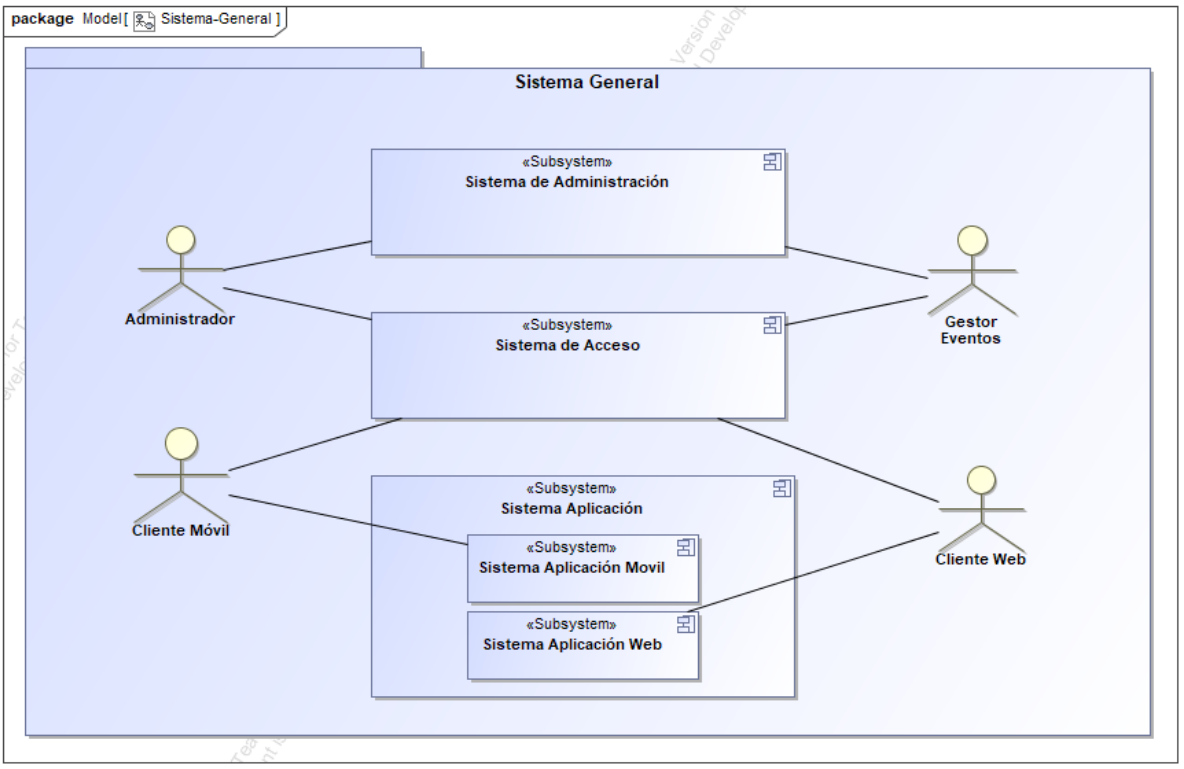

Figura 5.2: Diagrama casos de uso - Sistema general.

Diagrama sistema de acceso

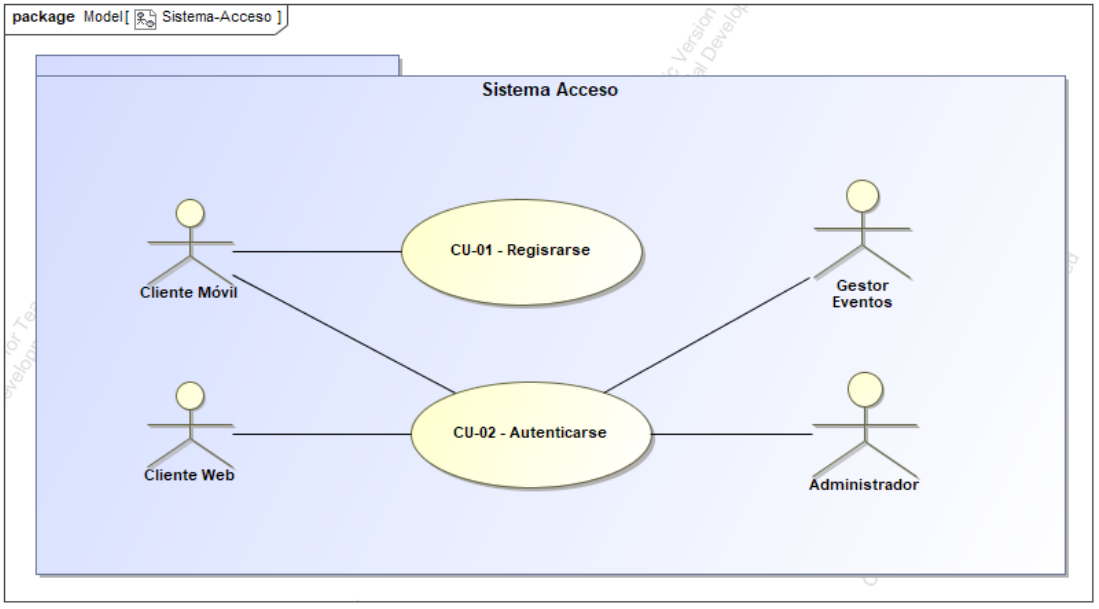

Figura 5.3: Diagrama casos de uso - Sistema de acceso.

Diagrama sistema aplicación móvil

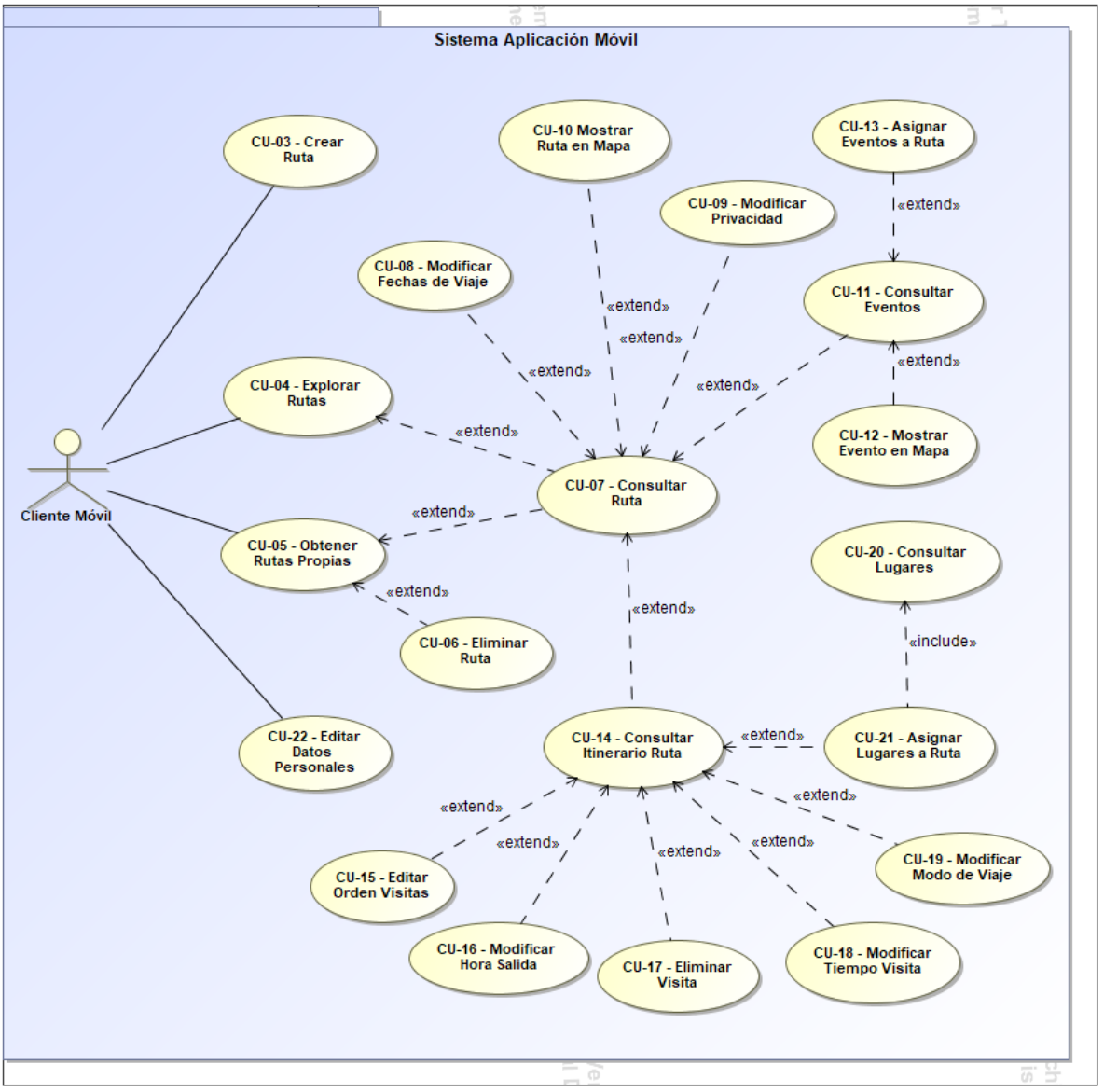

Figura 5.4: Diagrama casos de uso - Sistema aplicación móvil.

Diagrama sistema aplicación web

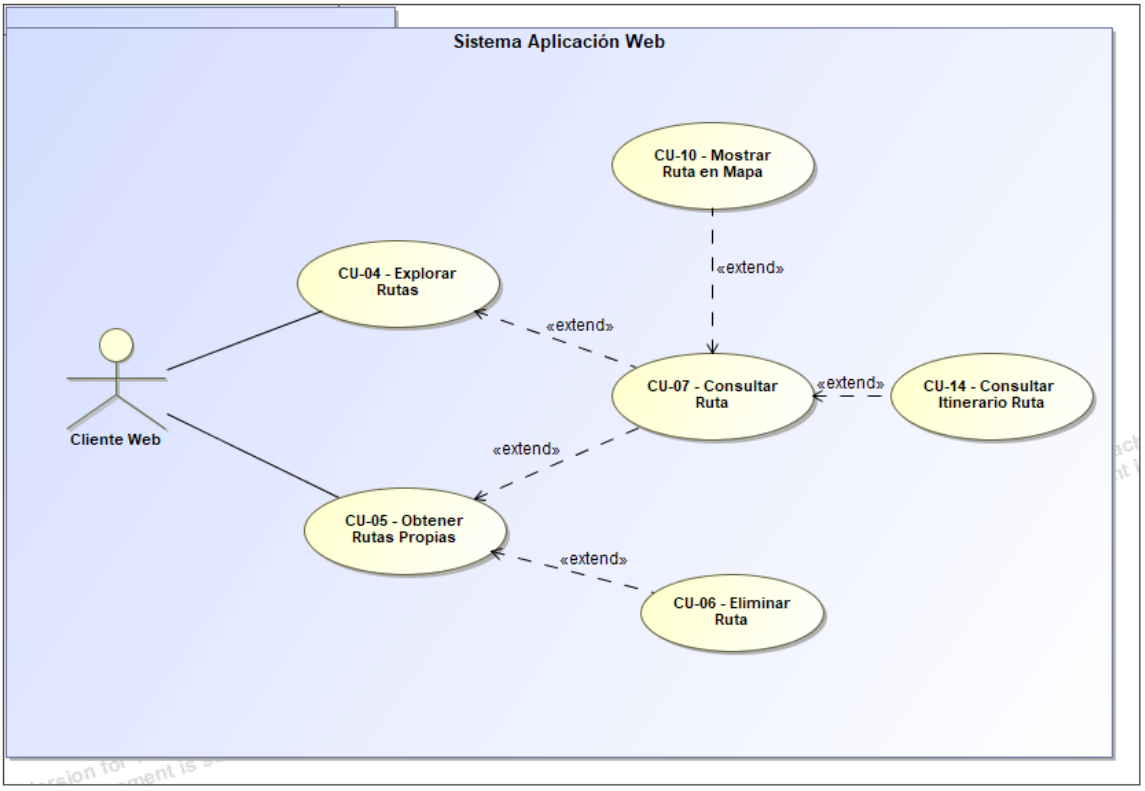

Figura 5.5: Diagrama casos de uso - Sistema aplicación web.

Diagrama sistema administración

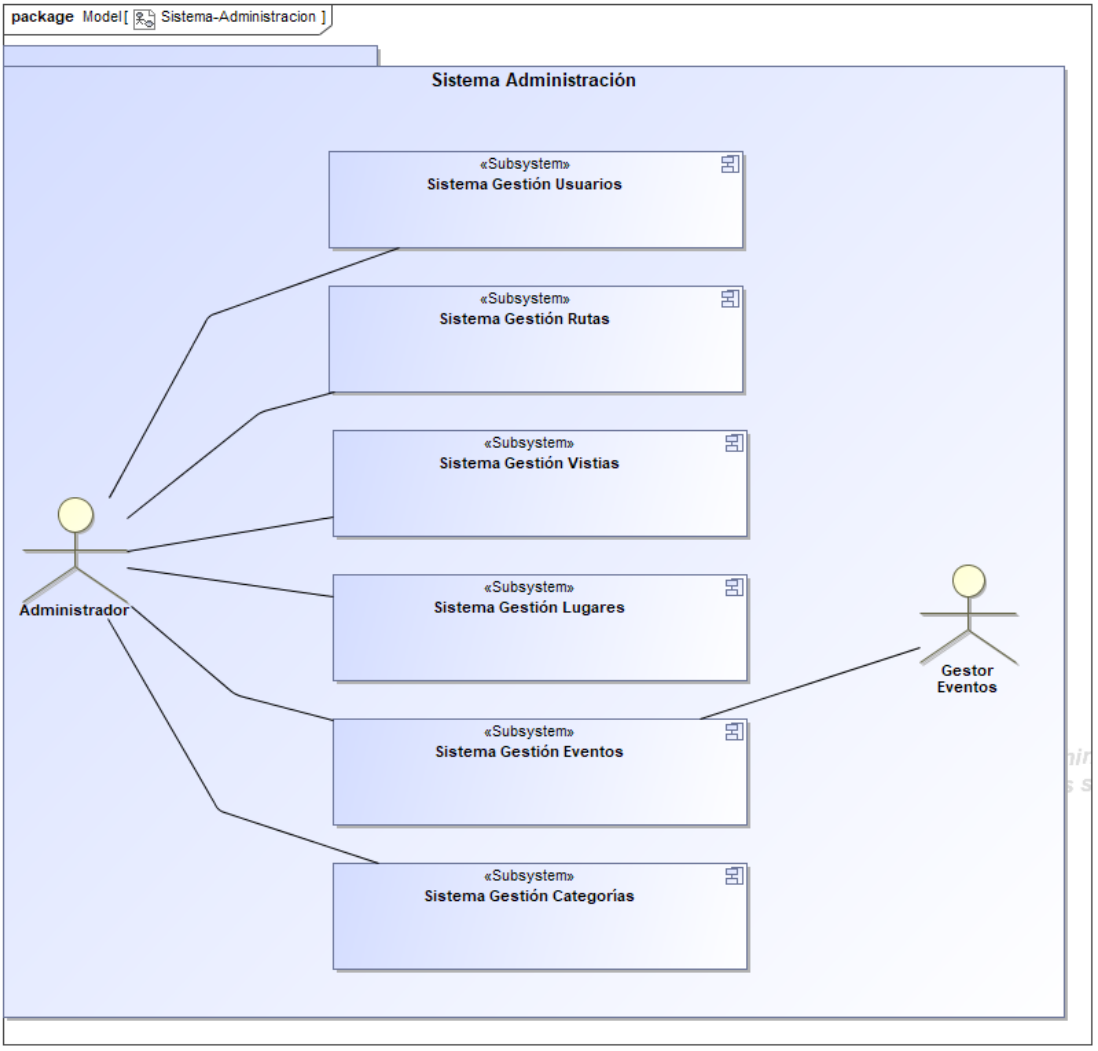

Figura 5.6: Diagrama Casos de Uso - Sistema Administración.

Diagrama sistema gestión usuarios

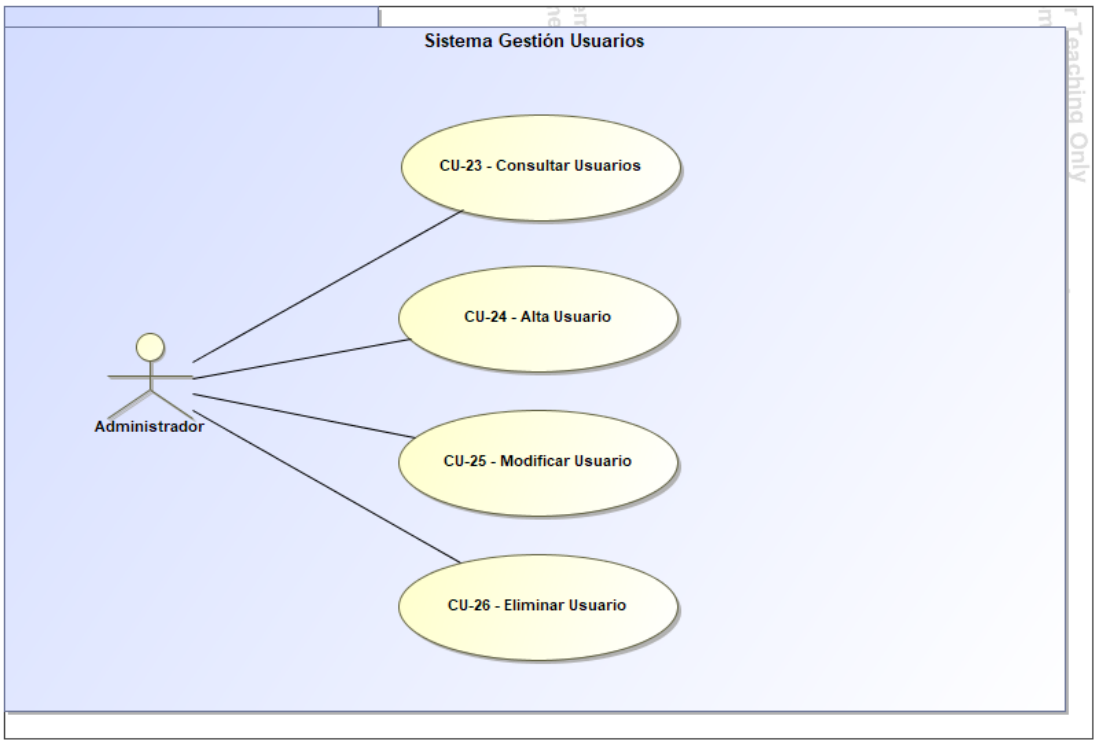

Figura 5.7: Diagrama casos de uso - Sistema gestión usuarios.

Diagrama sistema gestión rutas

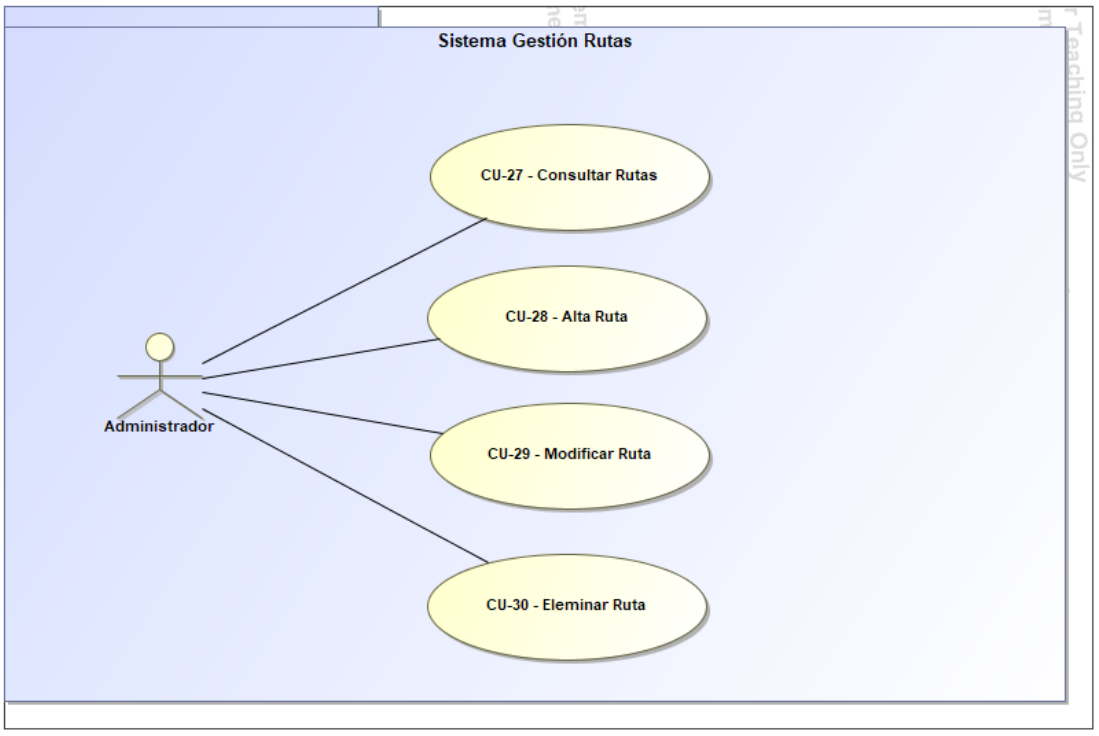

Figura 5.8: Diagrama casos de uso - Sistema gestión rutas.

Diagrama sistema gestión visitas

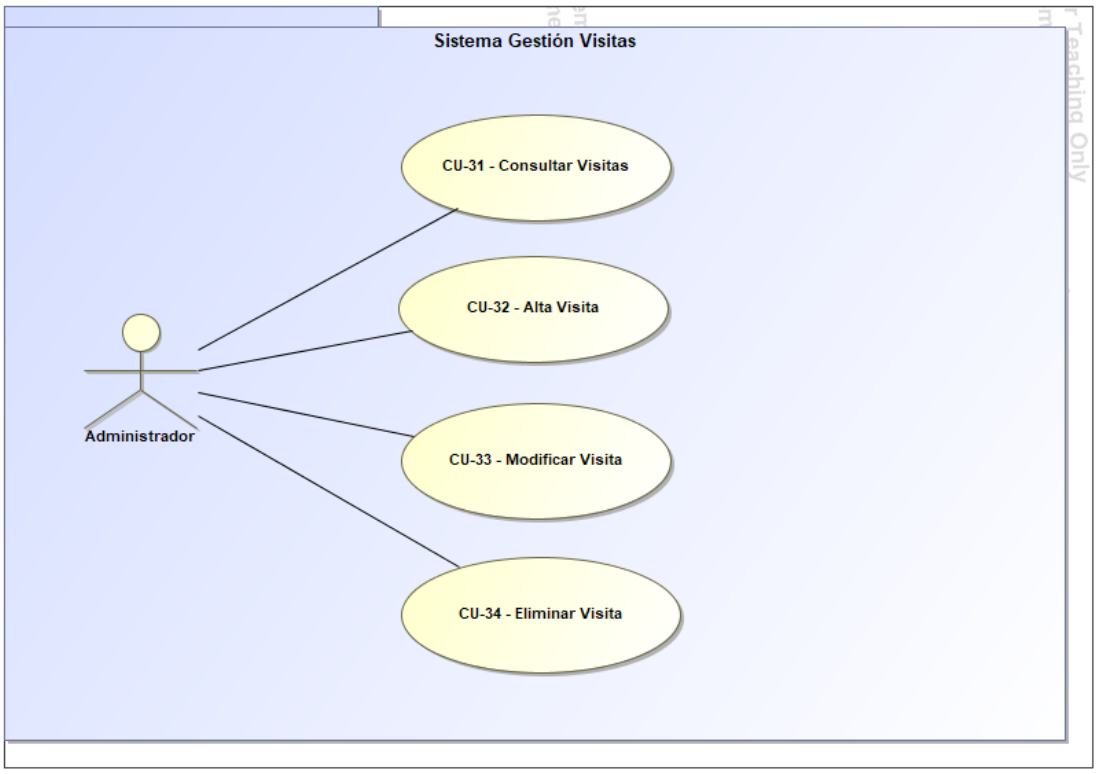

Figura 5.9: Diagrama casos de uso - Sistema gestión visitas.

Diagrama sistema gestión lugares

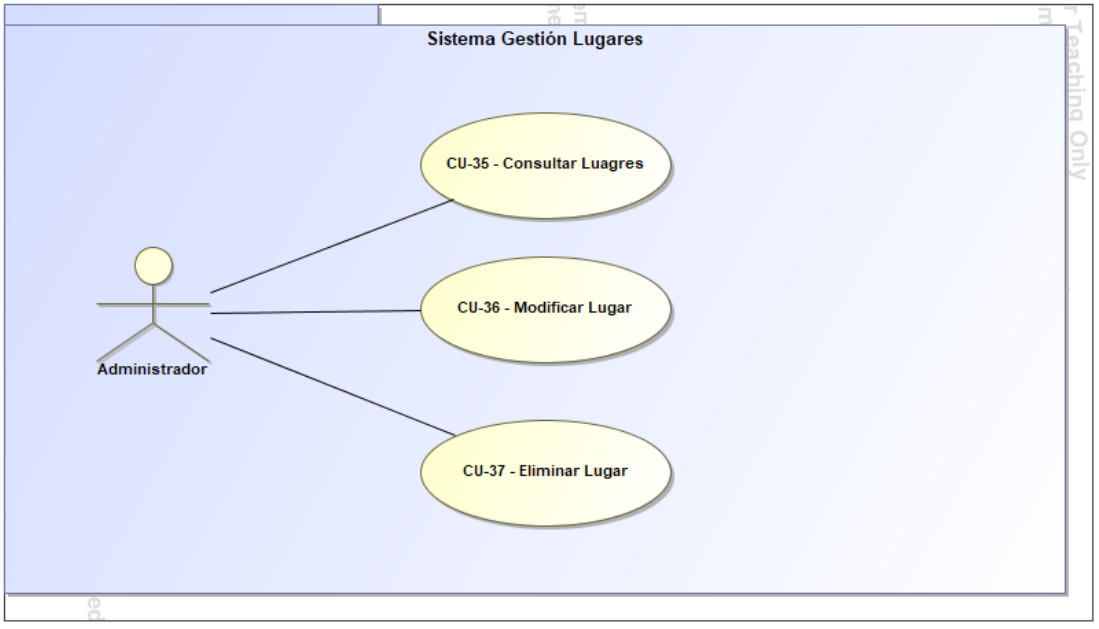

Figura 5.10: Diagrama casos de uso - Sistema gestión lugares.

Diagrama sistema gestión eventos

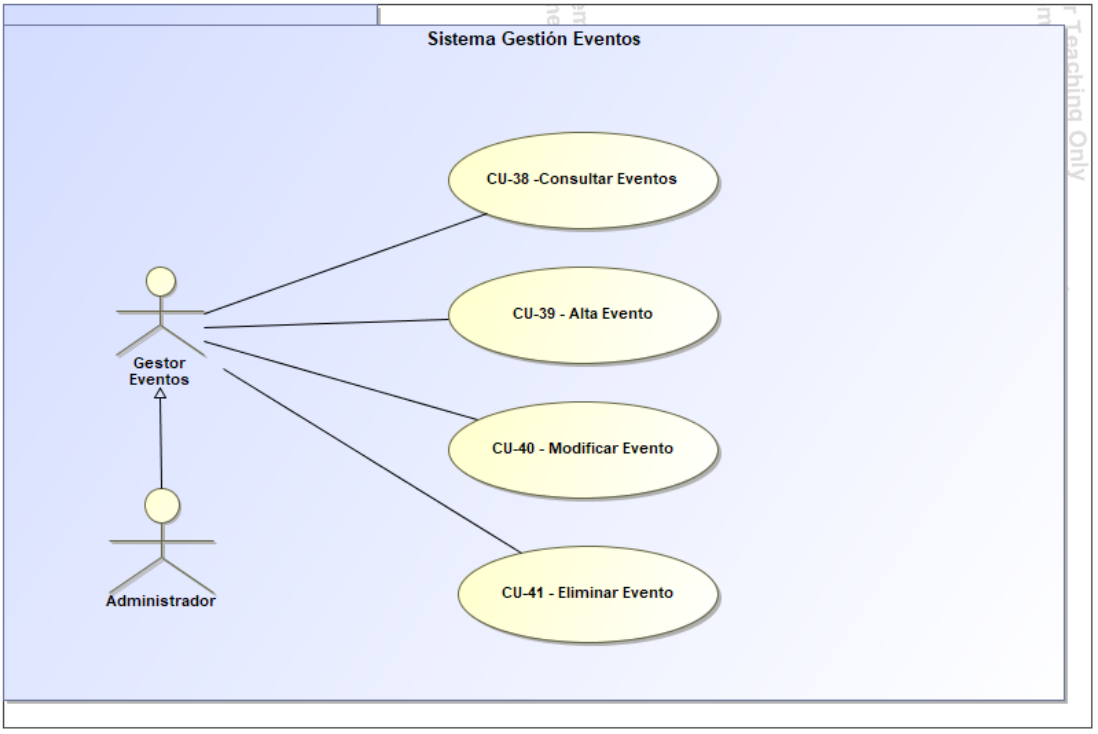

Figura 5.11: Diagrama casos de uso - Sistema gestión eventos.

Diagrama sistema gestión categorías

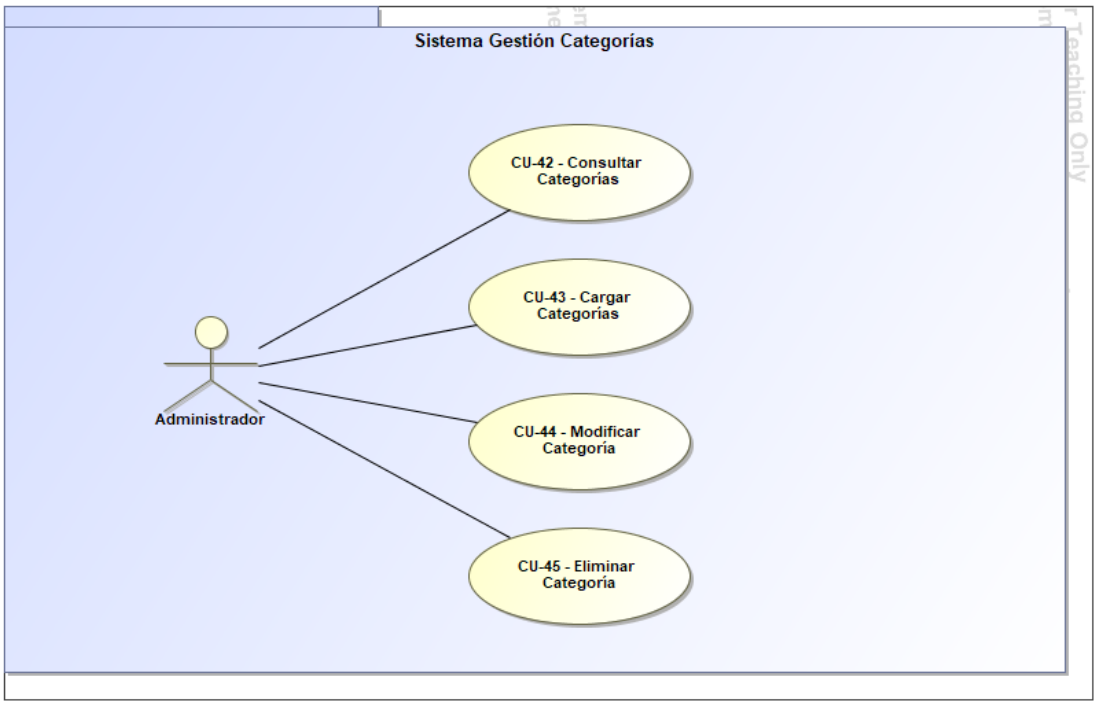

Figura 5.12: Diagrama casos de uso - Sistema gestión categorías.

5.2.3. Especificación casos de uso

Caso de uso: Autenticarse

CU<01> - Autenticarse	
Descripción	El usuario se identifica introduciendo las credenciales de acceso en el sistema
Actores	Cliente Móvil Cliente Web Administrador Gestor de Eventos
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	
Secuencia Normal	1. El usuario introduce sus credenciales en la ventana de login. 2. El usuario pulsa el botón de *Acceder*. 3. El sistema valida las credenciales. 4. El usuario accede a la aplicación.
Secuencia Alternativa	3. Los datos introducidos no son correctos. 1. El sistema muestra un mensaje de error y regresa al *Paso 1*.
Postcondiciones	El usuario queda autenticado en el sistema

Tabla 5.1: Caso de Uso: Autenticarse

Caso de uso: Registrarse

CU<02> - Registrarse	
Descripción	El usuario introduce los datos para darse de alta en la aplicación.
Actores	Cliente Móvil
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	
Secuencia Normal	1. El usuario selecciona la opción de registro. 2. El sistema muestra un formulario indicando los campos necesarios para realizar el registro. 3. El usuario rellena los campos y pulsa el botón de *Registrarse*. 4. El sistema valida los datos introducidos por el usuario. 5. El usuario accede a la aplicación.
Secuencia Alternativa	3. El usuario pulsa el botón de *Cancelar*. 1. El sistema cancela el registro y redirige al usuario a la pantalla de login.

Continúa en la siguiente página

Continúa de la página anterior

	4. Los datos introducidos por el usuario no son válidos. 1. El sistema muestra un mensaje de error y regresa al *Paso 2*.
Postcondiciones	El usuario queda registrado y autenticado en el sistema

Tabla 5.2: Caso de Uso: Registrarse

Caso de uso: Crear ruta

CU<03> - Crear Ruta	
Descripción	El usuario crea una ruta para una ciudad o lugar especificado.
Actores	Cliente Móvil
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	El usuario está autenticado en el sistema
Secuencia Normal	1. El usuario selecciona la opción de crear una nueva ruta. 2. El sistema muestra buscador para que el usuario indique en qué ciudad o lugar desea crear dicha ruta. 3. El usuario rellena el buscador. 4. El sistema ayuda al usuario autocompletando con los datos de diferentes ciudades y lugares. 5. El usuario selecciona el lugar en la lista de autocompletado ofrecida por el sistema. 6. El sistema muestra un mapa indicando la ubicación del lugar seleccionado y permite al usuario completar el proceso de creación. 7. El usuario pulsa el botón para crear la ruta. 8. El sistema ejecuta la acción.

Continúa en la siguiente página

Continúa de la página anterior

Secuencia Alternativa	3-5-7. El usuario pulsa el botón de *Atrás*. 1. El sistema cancela la acción y regresa a la pantalla anterior. 8. El sistema no puede dar de alta la ruta. 1. El sistema muestra un error indicando que no se pudo realizar la acción.
Postcondiciones	La ruta queda registrada en el sistema

Tabla 5.3: Caso de Uso: Crear ruta

Caso de uso: Explorar rutas

CU<04> - Explorar Rutas	
Descripción	El usuario explora las diferentes rutas creadas por los demás usuarios.
Actores	Cliente Móvil Cliente Web
Punto de Extensión	CU<07> - Consultar Ruta
Extiende	
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción de explorar rutas. 2. El sistema obtiene y muestra las rutas públicas de los demás usuarios.
Secuencia Alternativa	
Postcondiciones	

Tabla 5.4: Caso de Uso: Explorar rutas

Caso de uso: Obtener rutas propias

CU<05> - Obtener Rutas Propias	
Descripción	El usuario obtiene las rutas creadas por él.
Actores	Cliente Móvil Cliente Web
Punto de Extensión	CU<07> - Consultar Ruta
Extiende	
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción de obtener rutas propias. 2. El sistema muestra las rutas que el usuario tiene almacenadas en el sistema. 3. El sistema clasifica las rutas del usuario en función de su progreso; las que aún no empezaron, las que están en curso y las que ya se realizaron. 4. El usuario selecciona una de las posibilidades ofrecidas por el sistema. 5. El sistema filtra las rutas del usuario según lo solicitado.
Secuencia Alternativa	
Postcondiciones	

Tabla 5.5: Caso de Uso: Obtener rutas propias

Caso de uso: Eliminar ruta

CU<06> - Eliminar Ruta	
Descripción	El usuario elimina una ruta propia del sistema
Actores	Cliente Móvil Cliente Web
Punto de Extensión	CU<05> - Obtener Rutas Propias
Extiende	
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción de *Eliminar* sobre una ruta. 2. El sistema muestra un mensaje solicitando al usuario confirmar la acción. 3. El usuario pulsa en el botón de *Confirmar*. 4. El sistema realiza la acción
Secuencia Alternativa	3. El usuario pulsa el botón de *Cancelar*. 1. El sistema cancela la eliminación de la ruta. 4. El usuario no tiene permisos necesarios para realizar determinada acción. 1. El sistema muestra un mensaje de error indicando que no está autorizado a realizar dicha acción.

Continúa en la siguiente página

Continúa de la página anterior

	4. La ruta que quiere eliminar no se encuentra en el sistema 1. El sistema muestra un mensaje de error indicando que no es posible realizar la acción.
Postcondiciones	La ruta queda eliminada del sistema

Tabla 5.6: Caso de Uso: Obtener rutas propias

Caso de uso: Consultar ruta

CU<07> - Consultar Ruta	
Descripción	El usuario obtiene la información detallada de una ruta concreta.
Actores	Cliente Móvil Cliente Web
Punto de Extensión	CU<08> - Modificar Fechas de Viaje CU<09> - Modificar Privacidad CU<10> - Mostrar Ruta en Mapa CU<11> - Consultar Eventos CU<14> - Consultar Itinerario Ruta
Extiende	CU<04> - Explorar Rutas CU<05> - Obtener Rutas Propias
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona una ruta concreta. 2. El sistema obtiene y muestra un panel con los datos detallados de la ruta.
Secuencia Alternativa	2. La ruta solicitada no se encuentra en el sistema. 1. El sistema muestra un mensaje de error indicando que no es posible acceder a la ruta solicitada.

Continúa en la siguiente página

Continúa de la página anterior

	2. El usuario no tiene permisos necesarios para realizar determinada acción. 1. El sistema muestra un mensaje de error indicando que el usuario no está autorizado a realizar dicha acción.
Postcondiciones	

Tabla 5.7: Caso de Uso: Consultar ruta

Caso de uso: Modificar fechas de viaje

CU<08> - Modificar Fechas de Viaje	
Descripción	El usuario selecciona las fechas en las que se realizará la ruta que está planificando.
Actores	Cliente Móvil
Punto de Extensión	
Extiende	CU<07> - Consultar Ruta
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción de *Seleccionar Fechas*. 2. El sistema muestra una pantalla con las fechas del calendario. 3. El usuario selecciona la fecha de inicio y fin y pulsa el botón de *Aceptar*. 4. El sistema modifica las fechas de la ruta.
Secuencia Alternativa	3. El usuario pulsa el botón de *Cancelar*. 1. El sistema cancela la modificación de las fechas y cierra la pantalla. 4. El usuario no tiene permisos necesarios para realizar determinada acción. 1. El sistema muestra un mensaje de error indicando que no está autorizado a realizar dicha acción.

Continúa en la siguiente página

Continúa de la página anterior

	4. La ruta que quiere actualizar no se encuentra en el sistema 1. El sistema muestra un mensaje de error indicando que no es posible realizar la acción.
Postcondiciones	Las fechas de la ruta quedan actualizadas en el sistema.

Tabla 5.8: Caso de Uso: Modificar fechas de viaje

Caso de uso: Modificar privacidad

CU<09> - Modificar Privacidad	
Descripción	El usuario puede alternar la privacidad de cada una de sus rutas permitiendo que sean visible para todos o solo para él.
Actores	Cliente Móvil
Punto de Extensión	
Extiende	CU<07> - Consultar Ruta
Incluye	
Precondiciones	El usuario está autenticado en el sistema.

Secuencia Normal	1. El usuario selecciona la opción de *Detalles de la Ruta*.
	2. El sistema muestra una pantalla con los detalles de la ruta.
	3. El sistema incluye un botón que permite alternar entre los estados de privacidad (*Privado* y *Público*).
	4. El usuario pulsa el botón.
	5. Si la privacidad tenía el valor *Privado*:
	    1. El sistema actualiza la privacidad a *Pública*.
	6. Si la privacidad tenía el valor *Público*:
	    1. El sistema actualiza la privacidad a *Privado*.

Continúa en la siguiente página

Continúa de la página anterior

Secuencia Alternativa	4. El usuario pulsa el botón de *Atrás*. 1. El sistema retorna a la pantalla anterior. 5-6. El usuario no tiene permisos necesarios para realizar determinada acción. 1. El sistema muestra un mensaje de error indicando que no está autorizado a realizar dicha acción. 5-6. La ruta que quiere actualizar no se encuentra en el sistema 1. El sistema muestra un mensaje de error indicando que no es posible realizar la acción.
Postcondiciones	La privacidad de la ruta queda actualizada.

Tabla 5.9: Caso de Uso: Modificar privacidad

Caso de uso: Mostrar ruta en mapa

CU<10> - Mostrar Ruta en Mapa	
Descripción	El usuario puede mostrar la información de la ruta en el mapa.
Actores	Cliente Móvil Cliente Web
Punto de Extensión	
Extiende	CU<07> - Consultar Ruta
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción de *Mostrar Ruta en Mapa*. 2. Para cada día de la ruta, el sistema muestra un mapa con las marcas de las visitas asignadas al día concreto. 3. Si la ruta ya ha transcurrido en el tiempo y tiene información de su ejecución en tiempo real. 1. El sistema incorpora al mapa la ruta real realizada por el usuario permitiendo compararla con la ruta planificada. 4. El usuario puede cambiar entre los días haciendo click sobre ellos.
Secuencia Alternativa	3. El usuario pulsa el botón de *Atrás*. 1. El sistema retorna a la pantalla anterior.

Continúa en la siguiente página

Continúa de la página anterior

Postcondiciones	

Tabla 5.10: Caso de Uso: Mostrar ruta en mapa

Caso de uso: Consultar eventos

CU<11> - Consultar Eventos	
Descripción	El usuario obtiene los eventos dados de alta en el sistema
Actores	Cliente Móvil
Punto de Extensión	CU<12> - Mostrar Evento en Mapa CU<13> - Asignar Evento a Ruta
Extiende	CU<07> - Consultar Ruta
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción de *Consultar Eventos*. 2. El sistema muestra una pantalla con las opciones: *En el viaje* y *Próximos*. 3. Si el usuario selecciona *En el viaje*. 1. El sistema muestra los eventos que coinciden durante las fechas de viaje del usuario. 4. Si el usuario selecciona *Próximos*. 1. El sistema muestra los eventos futuros a las fechas de viaje del usuario.
Secuencia Alternativa	3-4. El usuario pulsa el botón de *Atrás*. 1. El sistema retorna a la pantalla anterior.
Postcondiciones	

Tabla 5.11: Caso de Uso: Cosnultar eventos

Caso de uso: Mostrar evento en mapa

CU<12> - Mostrar Evento en Mapa	
Descripción	Muestra un evento concreto en el mapa.
Actores	Cliente Móvil
Punto de Extensión	
Extiende	CU<11> - Consultar Eventos
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción de *Mostrar Evento en Mapa*. 2. El sistema muestra un mapa indicando la ubicación del evento incluyendo la ubicación de los lugares asignados al día de la ruta que transcurre el mismo día del evento, permitiendo ubicar el evento en función de la ruta ya creada.
Secuencia Alternativa	2. El usuario pulsa el botón de *Atrás*. 1. El sistema retorna a la pantalla anterior.
Postcondiciones	

Tabla 5.12: Caso de Uso: Mostrar evento en mapa

Caso de uso: Asignar evento a ruta

CU<13> - Asignar Evento a ruta	
Descripción	Permite incorporar o eliminar eventos a la ruta.
Actores	Cliente Móvil
Punto de Extensión	
Extiende	CU<11> - Consultar Eventos
Incluye	
Precondiciones	El usuario está autenticado en el sistema.

Secuencia Normal	1. Si el evento no está asignado a la ruta.
	  1. El sistema muestra un botón para añadirlo en la ruta del usuario.
	2. Si el evento ya está asignado a la ruta.
	  1. El sistema muestra un botón para eliminarlo de la ruta.
	3. El usuario hace click en el botón correspondiente.
	4. El sistema muestra un una ventana de confirmación.
	5. El usuario pulsa en *Aceptar*
	6. El sistema incorpora o elimina el evento al día correspondiente.

Secuencia Alternativa	2. El usuario pulsa el botón de *Atrás*. 1. El sistema retorna a la pantalla anterior.

Continúa en la siguiente página

Continúa de la página anterior

	5. El usuario pulso en *Cancelar* en la ventana de confirmación. 1. El sistema cancela la confirmación y retorna al *Paso 1* 6. El usuario no tiene permisos necesarios para realizar determinada acción. 1. El sistema muestra un mensaje de error indicando que no está autorizado a realizar dicha acción. 6. La ruta a la que quiere asignar el evento no se encuentra en el sistema 1. El sistema muestra un mensaje de error indicando que no es posible realizar la acción.
Postcondiciones	El evento queda asignado como visita en la ruta.

Tabla 5.13: Caso de Uso: Asignar evento a ruta

Caso de uso: Consultar itinerario ruta

CU<14> - Consultar Itinerario Ruta	
Descripción	Obtiene la información de la ruta desglosada por los días que la componen.
Actores	Cliente Móvil Cliente Web
Punto de Extensión	CU<15> - Editar Orden Visitas CU<16> - Modificar Hora Salida CU<17> - Eliminar Visita CU<18> - Modificar Tiempo Visita CU<19> - Modificar Modo de Viaje CU<21> - Asignar Lugares a Ruta
Extiende	CU<07> - Consultar Ruta
Incluye	
Precondiciones	El usuario está autenticado en el sistema. La ruta tiene unas fechas de viaje asignadas.

Continúa en la siguiente página

Continúa de la página anterior

Secuencia Normal	1. El usuario selecciona la opción *Itinerario*.
	2. El sistema muestra en la parte superior un selector con los días y en la inferior, la información correspondiente a cada día. Para cada día, el sistema muestra:
	  1. El conjunto de visitas a lugares y/o eventos asignadas por el usuario.
	  2. La hora de comienzo de la ruta para el día determinado.
	  3. La hora de llegada, estimada, a cada visita.
	  4. El tiempo asignado, como parada, en cada visita.
	  5. La hora de salida, estimada, para cada visita.
	  6. El modo de viaje, junto a su duración y distancia, entre cada visita.
	  7. Un botón para eliminar cada visita.
	3. El usuario cambia de día haciendo uso del selector.
	4. El sistema actualiza la pantalla con los datos del día seleccionado.

Secuencia Alternativa	3. El usuario pulsa el botón de *Atrás*. 1. El sistema retorna a la pantalla anterior.
Postcondiciones	

Tabla 5.14: Caso de Uso: Consultar itinerario ruta

Caso de uso: Editar orden visitas

CU<15> - Editar Orden Visitas	
Descripción	Modifica el orden de las visitas establecidas para un día.
Actores	Cliente Móvil
Punto de Extensión	
Extiende	CU<14> - Consultar Itinerario Ruta
Incluye	
Precondiciones	La ruta tiene visitas asignadas a sus días El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción *Habilitar Edición*. 2. El sistema muestra la vista de edición para las visitas. 3. El usuario modifica el orden de las visitas. 4. El sistema actualiza el orden establecido y recalcula las distancias y tiempos de viaje para el nuevo orden de las visitas. 5. El usuario selecciona *Terminar Edición*. 6. El sistema vuelve a la vista normal.
Secuencia Alternativa	3-5. El usuario pulsa el botón de *Atrás*. 1. El sistema retorna a la pantalla anterior.

Continúa en la siguiente página

Continúa de la página anterior

	4. El usuario no tiene permisos necesarios para realizar determinada acción. 1. El sistema muestra un mensaje de error indicando que no está autorizado a realizar dicha acción. 4. La visitas que quiere actualizar no se encuentra en el sistema 1. El sistema muestra un mensaje de error indicando que no es posible realizar la acción.
Postcondiciones	El nuevo orden de las visitas queda actualizado en el sistema.

Tabla 5.15: Caso de Uso: Editar orden visitas

Caso de uso: Modificar hora de salida

CU<16> - Modificar Hora de Salida	
Descripción	Modifica la hora de salida para un día concreto de la ruta.
Actores	Cliente Móvil
Punto de Extensión	
Extiende	CU<14> - Consultar Itinerario Ruta
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción *Modificar hora de salida*. 2. El sistema muestra un formulario para indicar la hora deseada. 3. El usuario selecciona la hora en el formulario y pulsa en *Confirmar*. 4. El sistema valida la hora y la actualiza.
Secuencia Alternativa	3. El usuario pulsa el botón de *Cancelar*. 1. El sistema deshecha el formulario y cancela la acción. 4. El usuario no tiene permisos necesarios para realizar determinada acción. 1. El sistema muestra un mensaje de error indicando que no está autorizado a realizar dicha acción.

Continúa en la siguiente página

Continúa de la página anterior

	4. La ruta sobre la que quiere actualizar la hora de salida ya no se encuentra en el sistema. 1. El sistema muestra un mensaje de error indicando que no es posible realizar la acción.
Postcondiciones	La hora de salida queda actualizada en el sistema.

Tabla 5.16: Caso de Uso: Modificar hora de salida

Caso de uso: Eliminar visita

CU<17> - Eliminar Visita	
Descripción	El usuario elimina una visita de un día de la ruta.
Actores	Cliente Móvil
Punto de Extensión	
Extiende	CU<14> - Consultar Itinerario Ruta
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario hace uso del botón para eliminar la visita. 2. El sistema muestra una pantalla de confirmación. 3. El usuario confirma la acción. 4. El sistema elimina la visita.
Secuencia Alternativa	3. El usuario pulsa el botón de *Cancelar*. 1. El sistema cancela la eliminación de la visita. 4. El usuario no tiene permisos necesarios para realizar determinada acción. 1. El sistema muestra un mensaje de error indicando que no está autorizado a realizar dicha acción.

Continúa en la siguiente página

Continúa de la página anterior

	4. La visita que quiere eliminar no se encuentra en el sistema 1. El sistema muestra un mensaje de error indicando que no es posible realizar la acción.
Postcondiciones	La visita queda eliminada del sistema.

Tabla 5.17: Caso de Uso: Eliminar visita

Caso de uso: Modificar tiempo en la visita

CU<18> - Modificar Tiempo en la Visita	
Descripción	El usuario modifica el tiempo de parada en una determinada visita.
Actores	Cliente Móvil
Punto de Extensión	
Extiende	CU<14> - Consultar Itinerario Ruta
Incluye	
Precondiciones	La ruta tiene visitas asignadas a sus días El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción *Editar Tiempo Visita*. 2. El sistema muestra un formulario para indicar el tiempo deseado. 3. El usuario selecciona indica el tiempo en el formulario y pulsa en *Confirmar*. 4. El sistema actualiza el tiempo para la visita.
Secuencia Alternativa	3. El usuario pulsa el botón de *Cancelar*. 1. El sistema cancela la operación. 4. El usuario no tiene permisos necesarios para realizar determinada acción. 1. El sistema muestra un mensaje de error indicando que no está autorizado a realizar dicha acción.

Continúa en la siguiente página

Continúa de la página anterior

	4. La visita que quiere actualizar no se encuentra en el sistema. 1. El sistema muestra un mensaje de error indicando que no es posible realizar la acción.
Postcondiciones	El tiempo de la visita queda actualizado en el sistema.

Tabla 5.18: Caso de Uso: Modificar tiempo en la visita

Caso de uso: Modificar modo de viaje

CU<19> - Modificar Modo de Viaje	
Descripción	El usuario modifica el modo de viaje entre dos visitas. Los modos de viaje habilitados son: *En coche*, *En bicicleta* y *Andando*
Actores	Cliente Móvil
Punto de Extensión	
Extiende	CU<14> - Consultar Itinerario Ruta
Incluye	
Precondiciones	La ruta tiene al menos dos visitas para un día concreto El usuario está autenticado en el sistema.

Secuencia Normal	1. El usuario selecciona la opción *Modificar Modo de Viaje*.
	2. Si el modo de viaje era *Andando*.
	    1. El sistema actualiza el modo de viaje a *En coche*.
	3. Si el modo de viaje era *En coche*.
	    1. El sistema actualiza el modo de viaje a *En bicicleta*.
	4. Si el modo de viaje era *En bicicleta*.
	    1. El sistema actualiza el modo de viaje a *Andando*.

Continúa en la siguiente página

Continúa de la página anterior

Secuencia Alternativa	2-3-4. El usuario no tiene permisos necesarios para realizar determinada acción. 1. El sistema muestra un mensaje de error indicando que no está autorizado a realizar dicha acción. 2-3-4. La visitas en las que quiere actualizar el modo de viaje no se encuentra en el sistema. 1. El sistema muestra un mensaje de error indicando que no es posible realizar la acción.
Postcondiciones	El modo de viaje queda actualizado en el sistema.

Tabla 5.19: Caso de Uso: Modificar modo de viaje

Caso de uso: Consultar lugares

CU<20> - Consultar lugares	
Descripción	El usuario consulta los diferentes lugares en función de diferentes criterios.
Actores	Cliente Móvil
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	El usuario está autenticado en el sistema.

Secuencia Normal	1. El sistema muestra en la parte superior un selector y en la inferior los lugares más relevantes para la ciudad o lugar donde está creada la ruta.
	2. Si el usuario selecciona la opción *Lista* en el selector.
	  1. El sistema muestra los lugares en una lista.
	  2. Para cada lugar el sistema incluye.
	  1. Foto, nombre, categoría a la que pertenece y dirección en la que se encuentra.
	  2. Un indicador con el número de días en el que está asignado dicho lugar en la ruta del usuario.
	  3. Un botón para asignar o desasignar el lugar a los días de la ruta.

Continúa en la siguiente página

Continúa de la página anterior

	3. Si el usuario selecciona la opción *Mapa* en el selector. 1. El sistema muestra los lugares en el mapa. 2. Para cada lugar el sistema incluye. 1. Una marca, que incluye nombre y categoría, en la ubicación exacta en el mapa. 2. Un color diferente en función de si el lugar está asignado o no a la ruta. 3. Un botón para asignar o desasignar el lugar a los días de la ruta.
Secuencia Alternativa	2-3. El usuario pulsa el botón de *Atrás*. 1. El sistema retorna a la pantalla anterior.
Postcondiciones	

Tabla 5.20: Caso de Uso: Consultar lugares

Caso de uso: Asignar lugares a ruta

CU<21> - Asignar Lugares a Ruta	
Descripción	El usuario añade o elimina lugares a la ruta.
Actores	Cliente Móvil
Punto de Extensión	
Extiende	
Incluye	CU<20> - Consultar Lugares
Precondiciones	El usuario está autenticado en el sistema.

Secuencia Normal	1. El usuario selecciona la opción *Añadir Lugar*.
	2. El sistema implementa *CU<20> - Consultar Lugares*
	3. El usuario hace click en el botón para asignar o desasignar el lugar.
	4. El sistema muestra una ventana con el conjunto de días que forman la ruta.
	5. Si el día aparece *Activado*.
	  1. Dicho día ya tiene incorporado el lugar a la ruta.
	6. Si el día aparece *Desactivado*.
	  1. Dicho día no tiene incorporado el lugar a la ruta.
	7. El usuario, activando y desactivando, indica los días en los que desea incluir dicho lugar en la ruta y pulsa el botón de *Aceptar*.

Continúa en la siguiente página

Continúa de la página anterior

	8. El sistema confirma la acción y actualiza la ruta del usuario.
Secuencia Alternativa	3. El usuario pulsa el botón de *Atrás*. 1. El sistema retorna a la pantalla anterior. 4-7. El usuario pulsa el botón de *Cancelar*. 1. El sistema cancela la acción y no actualiza la información de la ruta. 8. El usuario no tiene permisos necesarios para realizar determinada acción. 1. El sistema muestra un mensaje de error indicando que no está autorizado a realizar dicha acción. 8. La ruta en la que quiere incorporar los lugares no se encuentra en el sistema. 1. El sistema muestra un mensaje de error indicando que no es posible realizar la acción.
Postcondiciones	El lugar queda incluido en los días que el usuario indica.

Tabla 5.21: Caso de Uso: Asginar lugares a ruta

Caso de uso: Modificar datos personales

CU<22> - Modificar Datos Personales	
Descripción	El usuario modifica sus datos en el sistema.
Actores	Cliente Móvil
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción *Editar Datos*. 2. El sistema muestra un formulario con los datos actuales del usuario. 3. El usuario modifica sus datos y pulsa el botón *Aceptar*. 4. El sistema actualiza los datos del usuario.
Secuencia Alternativa	3. El usuario pulsa el botón de *Cancelar*. 1. El sistema cancela la acción y no actualiza la información del usuario. 4. El usuario no tiene permisos necesarios para realizar determinada acción. 1. El sistema muestra un mensaje de error indicando que no está autorizado a realizar dicha acción.
Postcondiciones	Los datos del usuario quedan actualizados en el sistema.

Tabla 5.22: Caso de Uso: Modificar datos personales

Caso de uso: Consultar usuarios

CU<23> - Consultar Usuarios	
Descripción	Muestra los diferentes usuarios registrados en el sistema.
Actores	Administrador
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	El administrador está autenticado en el sistema.
Secuencia Normal	1. El administrador selecciona la opción *Consultar Usuarios*. 2. El sistema muestra una tabla con todos los usuarios del sistema.
Secuencia Alternativa	
Postcondiciones	

Tabla 5.23: Caso de Uso: Consultar usuarios

Caso de uso: Alta usuario

CU<24> - Alta Usuario	
Descripción	Añade un nuevo usuario al sistema.
Actores	Administrador
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	El administrador está autenticado en el sistema.
Secuencia Normal	1. El administrador selecciona la opción *Añadir Usuario*. 2. El sistema muestra un formulario con los datos a rellenar por el usuario. 3. El administrador introduce los datos y pulsa sobre el botón *Confirmar*. 4. El sistema comprueba los datos y los inserta en el sistema.
Secuencia Alternativa	3. El administrador pulsa el botón de *Cancelar*. 1. El sistema cancela la acción y detiene el proceso de alta. 4. El sistema no acepta los datos introducidos. 1. El sistema cancela la acción y muestra un error al usuario.
Postcondiciones	El usuario queda registrado en el sistema.

Tabla 5.24: Caso de Uso: Alta usuario

Caso de uso: Modificar usuario

CU<25> - Modificar Usuario	
Descripción	Modifica los datos de un usuario.
Actores	Administrador
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	El administrador está autenticado en el sistema
Secuencia Normal	1. El administrador selecciona la opción de *Modificar* sobre un usuario. 2. El sistema muestra un formulario con los datos actuales del usuario seleccionado. 3. El administrador modifica los datos que desea y pulsa sobre el botón *Confirmar*. 4. El sistema comprueba los datos y los actualiza.
Secuencia Alternativa	3. El administrador pulsa el botón de *Cancelar*. 1. El sistema cancela la acción y no actualiza la información del usuario. 4. El sistema no acepta los datos introducidos. 1. El sistema cancela la acción y muestra un error al usuario.
Postcondiciones	Los datos del usuario quedan actualizados en el sistema

Tabla 5.25: Caso de Uso: Modificar usuario

Caso de uso: Eliminar usuario

CU<26> - Eliminar Usuario	
Descripción	Elimina un usuario del sistema.
Actores	Administrador
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	El administrador está autenticado en el sistema.
Secuencia Normal	1. El administrador selecciona la opción de *Eliminar* sobre un usuario. 2. El sistema muestra una pantalla de confirmación. 3. El administrador pulsa sobre el botón de *Confirmar*. 4. El sistema realiza la acción.
Secuencia Alternativa	3. El administrador pulsa el botón de *Cancelar*. 1. El sistema cancela la acción y detiene el proceso de eliminación.
Postcondiciones	El datos del usuario quedan eliminados del sistema.

Tabla 5.26: Caso de Uso: Eliminar usuario

Caso de uso: Consultar rutas

CU<27> - Consultar Rutas	
Descripción	Muestra los diferentes rutas registradas en el sistema.
Actores	Administrador
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción *Consultar Rutas*. 2. El sistema muestra una tabla con todas las rutas del sistema.
Secuencia Alternativa	
Postcondiciones	

Tabla 5.27: Caso de Uso: Consultar rutas

Caso de uso: Alta ruta

CU<28> - Alta Ruta	
Descripción	Añade una nueva ruta al sistema.
Actores	Administrador
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción *Añadir Ruta*. 2. El sistema muestra un formulario con los datos a rellenar por el usuario. 3. El usuario introduce los datos y pulsa sobre el botón *Confirmar*. 4. El sistema comprueba los datos y los inserta en el sistema.
Secuencia Alternativa	3. El usuario pulsa el botón de *Cancelar*. 1. El sistema cancela la acción y detiene el proceso de alta. 4. El sistema no acepta los datos introducidos. 1. El sistema cancela la acción y muestra un error al usuario.
Postcondiciones	La ruta queda registrada en el sistema.

Tabla 5.28: Caso de Uso: Alta ruta

Caso de uso: Modificar ruta

CU<29> - Modificar Ruta	
Descripción	Modifica los datos de una ruta.
Actores	Administrador
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción de *Modificar* sobre una ruta. 2. El sistema muestra un formulario con los datos actuales de la ruta seleccionada. 3. El usuario modifica los datos que desea y pulsa sobre el botón *Confirmar*. 4. El sistema comprueba los datos y los actualiza.
Secuencia Alternativa	3. El usuario pulsa el botón de *Cancelar*. 1. El sistema cancela la acción y no actualiza la información de la ruta. 4. El sistema no acepta los datos introducidos. 1. El sistema cancela la acción y muestra un error al usuario.
Postcondiciones	Los datos de la ruta quedan actualizados en el sistema.

Tabla 5.29: Caso de Uso: Modificar ruta

Caso de uso: Eliminar ruta

CU<30> - Eliminar Ruta	
Descripción	Elimina una ruta del sistema.
Actores	Administrador
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción de *Eliminar* sobre una ruta. 2. El sistema muestra una pantalla de confirmación. 3. El usuario pulsa sobre el botón de *Confirmar*. 4. El sistema realiza la acción.
Secuencia Alternativa	3. El usuario pulsa el botón de *Cancelar*. 1. El sistema cancela la acción y detiene el proceso de eliminación.
Postcondiciones	Los datos de la ruta quedan eliminados del sistema.

Tabla 5.30: Caso de Uso: Eliminar ruta

Caso de uso: Consultar visitas

CU<31> - Consultar Visitas	
Descripción	Muestra los diferentes visitas registradas en el sistema.
Actores	Administrador
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción *Consultar Visitas*. 2. El sistema muestra una tabla con todas las visitas del sistema.
Secuencia Alternativa	
Postcondiciones	

Tabla 5.31: Caso de Uso: Consultar visitas

Caso de uso: Alta visita

CU<32> - Alta Visita	
Descripción	Añade una nueva visita al sistema.
Actores	Administrador
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción *Añadir Visita*. 2. El sistema muestra un formulario con los datos a rellenar por el usuario. 3. El usuario introduce los datos y pulsa sobre el botón *Confirmar*. 4. El sistema comprueba los datos y los inserta en el sistema.
Secuencia Alternativa	3. El usuario pulsa el botón de *Cancelar*. 1. El sistema cancela la acción y detiene el proceso de alta. 4. El sistema no acepta los datos introducidos. 1. El sistema cancela la acción y muestra un error al usuario.
Postcondiciones	La visita queda registrada en el sistema.

Tabla 5.32: Caso de Uso: Alta visita

Caso de uso: Modificar visita

CU<33> - Modificar Visita	
Descripción	Modifica los datos de una visita.
Actores	Administrador
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción de *Modificar* sobre una visita. 2. El sistema muestra un formulario con los datos actuales de la visita seleccionada. 3. El usuario modifica los datos que desea y pulsa sobre el botón *Confirmar*. 4. El sistema comprueba los datos y los actualiza.
Secuencia Alternativa	3. El usuario pulsa el botón de *Cancelar*. 1. El sistema cancela la acción y no actualiza la información de la visita. 4. El sistema no acepta los datos introducidos. 1. El sistema cancela la acción y muestra un error al usuario.
Postcondiciones	Los datos de la visita quedan actualizados en el sistema.

Tabla 5.33: Caso de Uso: Modificar visita

Caso de uso: Eliminar visita

CU<34> - Eliminar Visita	
Descripción	Elimina una visita del sistema.
Actores	Administrador
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción de *Eliminar* sobre una visita. 2. El sistema muestra una pantalla de confirmación. 3. El usuario pulsa sobre el botón de *Confirmar*. 4. El sistema realiza la acción.
Secuencia Alternativa	3. El usuario pulsa el botón de *Cancelar*. 1. El sistema cancela la acción y detiene el proceso de eliminación.
Postcondiciones	Los datos de la visita quedan eliminados del sistema.

Tabla 5.34: Caso de Uso: Eliminar visita

Caso de uso: Consultar lugares

CU<35> - Consultar Lugares	
Descripción	Muestra los diferentes lugares registrados en el sistema.
Actores	Administrador
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción *Consultar Lugares*. 2. El sistema muestra una tabla con todos los lugares del sistema.
Secuencia Alternativa	
Postcondiciones	

Tabla 5.35: Caso de Uso: Consultar lugares

Caso de uso: Modificar lugar

CU<36> - Modificar Lugar	
Descripción	Modifica los datos de un lugar.
Actores	Administrador
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción de *Modificar* sobre un lugar. 2. El sistema muestra un formulario con los datos actuales del lugar seleccionado. 3. El usuario modifica los datos que desea y pulsa sobre el botón *Confirmar*. 4. El sistema comprueba los datos y los actualiza.
Secuencia Alternativa	3. El usuario pulsa el botón de *Cancelar*. 1. El sistema cancela la acción y no actualiza la información del lugar. 4. El sistema no acepta los datos introducidos. 1. El sistema cancela la acción y muestra un error al usuario.
Postcondiciones	Los datos del lugar quedan actualizados en el sistema.

Tabla 5.36: Caso de Uso: Modificar lugar

Caso de uso: Eliminar lugar

CU<37> - Eliminar Lugar	
Descripción	Elimina un lugar del sistema.
Actores	Administrador
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción de *Eliminar* sobre un lugar. 2. El sistema muestra una pantalla de confirmación. 3. El usuario pulsa sobre el botón de *Confirmar*. 4. El sistema realiza la acción.
Secuencia Alternativa	3. El usuario pulsa el botón de *Cancelar*. 1. El sistema cancela la acción y detiene el proceso de eliminación.
Postcondiciones	Los datos del lugar quedan eliminados del sistema.

Tabla 5.37: Caso de Uso: Eliminar lugar

Caso de uso: Consultar eventos

CU<38> - Consultar Eventos	
Descripción	Muestra los diferentes usuarios registrados en el sistema.
Actores	Administrador Gestor de Eventos
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción *Consultar Eventos*. 2. El sistema muestra una tabla con todos los eventos del sistema.
Secuencia Alternativa	
Postcondiciones	

Tabla 5.38: Caso de Uso: Consultar eventos

Caso de uso: Alta evento

CU<39> - Alta Evento	
Descripción	Añade un nuevo evento al sistema.
Actores	Administrador Gestor de Eventos
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción *Añadir Evento*. 2. El sistema muestra un formulario con los datos a rellenar por el usuario. 3. El usuario introduce los datos y pulsa sobre el botón *Confirmar*. 4. El sistema comprueba los datos y los inserta en el sistema.
Secuencia Alternativa	3. El usuario pulsa el botón de *Cancelar*. 1. El sistema cancela la acción y detiene el proceso de alta. 4. El sistema no acepta los datos introducidos. 1. El sistema cancela la acción y muestra un error al usuario.
Postcondiciones	El evento queda registrado en el sistema.

Tabla 5.39: Caso de Uso: Alta evento

Caso de uso: Modificar evento

CU<40> - Modificar Evento	
Descripción	Modifica los datos de un evento.
Actores	Administrador Gestor de Eventos
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción de *Modificar* sobre un evento. 2. El sistema muestra un formulario con los datos actuales del evento seleccionado. 3. El usuario modifica los datos que desea y pulsa sobre el botón *Confirmar*. 4. El sistema comprueba los datos y los actualiza.
Secuencia Alternativa	3. El usuario pulsa el botón de *Cancelar*. 1. El sistema cancela la acción y no actualiza la información del evento. 4. El sistema no acepta los datos introducidos. 1. El sistema cancela la acción y muestra un error al usuario.
Postcondiciones	Los datos del evento quedan modificados en el sistema.

Tabla 5.40: Caso de Uso: Modificar evento

Caso de uso: Eliminar evento

CU<41> - Eliminar Evento	
Descripción	Elimina un evento del sistema.
Actores	Administrador Gestor de Eventos
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción de *Eliminar* sobre un evento. 2. El sistema muestra una pantalla de confirmación. 3. El usuario pulsa sobre el botón de *Confirmar*. 4. El sistema realiza la acción.
Secuencia Alternativa	3. El usuario pulsa el botón de *Cancelar*. 1. El sistema cancela la acción y detiene el proceso de eliminación.
Postcondiciones	Los datos del evento quedan eliminados del sistema.

Tabla 5.41: Caso de Uso: Eliminar evento

Caso de uso: Consultar categorías

CU<42> - Consultar Categorías	
Descripción	Muestra los diferentes categorías registradas en el sistema.
Actores	Administrador
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción *Consultar Categorías*. 2. El sistema muestra una tabla con todas las categorías del sistema.
Secuencia Alternativa	
Postcondiciones	

Tabla 5.42: Caso de Uso: Consultar categorías

Caso de uso: Cargar categorías

CU<43> - Cargar Categorías	
Descripción	Obtiene las categorías de una fuente externa y las actualiza en el sistema.
Actores	Administrador
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción *Cargar Categorías*. 2. El sistema muestra una pantalla de confirmación. 3. El usuario pulsa sobre el botón de *Confirmar*. 4. El sistema realiza la acción.
Secuencia Alternativa	3. El usuario pulsa el botón de *Cancelar*. 1. El sistema cancela la acción y detiene el proceso de alta. 4. El sistema no puede realizar la acción. 1. El sistema cancela la acción y muestra un error al usuario.
Postcondiciones	Los datos obtenidos de las categorías quedan registrados en el sistema.

Tabla 5.43: Caso de Uso: Cargar categorías

Caso de uso: Modificar categoría

CU<44> - Modificar Categoría	
Descripción	Modifica los datos de una categoría.
Actores	Administrador
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción de *Modificar* sobre una categoría. 2. El sistema muestra un formulario con los datos actuales de la categoría seleccionada. 3. El usuario modifica los datos que desea y pulsa sobre el botón *Confirmar*. 4. El sistema comprueba los datos y los actualiza.
Secuencia Alternativa	3. El usuario pulsa el botón de *Cancelar*. 1. El sistema cancela la acción y no actualiza la información de la categoría. 4. El sistema no acepta los datos introducidos. 1. El sistema cancela la acción y muestra un error al usuario.
Postcondiciones	Los datos de la categoría quedan actualizados en el sistema.

Tabla 5.44: Caso de Uso: Modificar categoría

Caso de uso: Eliminar categoría

CU<45> - Eliminar Categoría	
Descripción	Elimina una ruta del sistema.
Actores	Administrador
Punto de Extensión	
Extiende	
Incluye	
Precondiciones	El usuario está autenticado en el sistema.
Secuencia Normal	1. El usuario selecciona la opción de *Eliminar* sobre una ruta. 2. El sistema muestra una pantalla de confirmación. 3. El usuario pulsa sobre el botón de *Confirmar*. 4. El sistema realiza la acción.
Secuencia Alternativa	3. El usuario pulsa el botón de *Cancelar*. 1. El sistema cancela la acción y detiene el proceso de eliminación.
Postcondiciones	Los datos de la categoría quedan eliminados del sistema.

Tabla 5.45: Caso de Uso: Eliminar categoría

Capítulo 6

DISEÑO

6.1. Arquitectura del sistema

La arquitectura empleada en nuestro sistema será una arquitectura en capas, una de las técnicas de diseño más usadas en las ciencias de la computación. La arquitectura basada en capas es una especialización de la arquitectura cliente-servidor donde la carga de trabajo se divide en diferentes capas con un reparto claro de las funciones.

En esta arquitectura, una capa inferior proporciona un servicio a otra copa superior. El servicio ofrecido se define mediante un contrato de servicio. De esta forma, se consigue independizar el software de ambas capas y los cambios de implementación en una de ellas no tienen repercusión sobre las demás.

Partiendo que la aplicación será accesible desde dispositivo móvil y navegador web, se presentará una solución al diseño basada en dos alternativas de la arquitectura basada en capas: la arquitectura en 3 capas y la arquitectura en 4 capas.

6.1.1. Arquitectura en 3 capas físicas

En este sistema de arquitectura, se diferencian tres capas físicas que cumplen funciones diferentes.

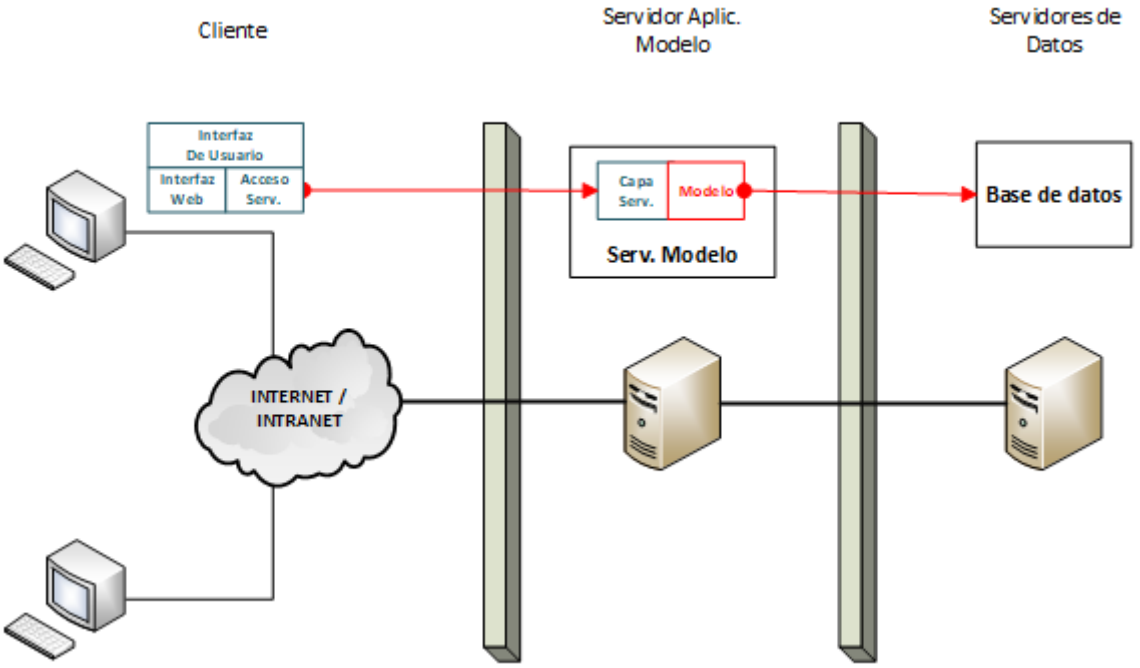

Figura 6.1: Diagrama arquitectura en tres capas

- **Capa Física - Servidor de Datos:** Es la capa encargada de gestionar el almacenamiento de los datos.

- **Capa Física - Servidor Aplicación Modelo:** Formada por las siguientes capas lógicas.

  - **Capa Modelo:** Es la encargada implementar las funcionalidades de la aplicación y mantener la comunicación con el servidor de datos realizando las acciones relacionadas con el acceso a los datos. Comúnmente, se subdivide en dos, la capa de acceso a datos, para la conexión con el servidor de datos, y la capa lógica de negocio, para la implementación de los casos de uso.

  - **Capa Servicios:** Sirve de enlace entre la interfaz de usuario y la capa modelo.

- **Capa Física - Cliente:** Divida en las siguientes capas lógicas.

  - **Capa de Acceso al Servicio:** Capa encargada de acceder e invocar a los métodos de la capa de servicios del modelo.

- **Capa Interfaz de Usuario:** Interfaz gráfica final que se instala en las máquinas cliente y dispositivos finales.

6.1.2. Arquitectura en 4 capas físicas

En esta alternativa, se añade una capa intermedia entre el cliente y el modelo que actúe de servidor de aplicaciones y que proporciones la interfaz web para clientes que accedan desde navegador web.

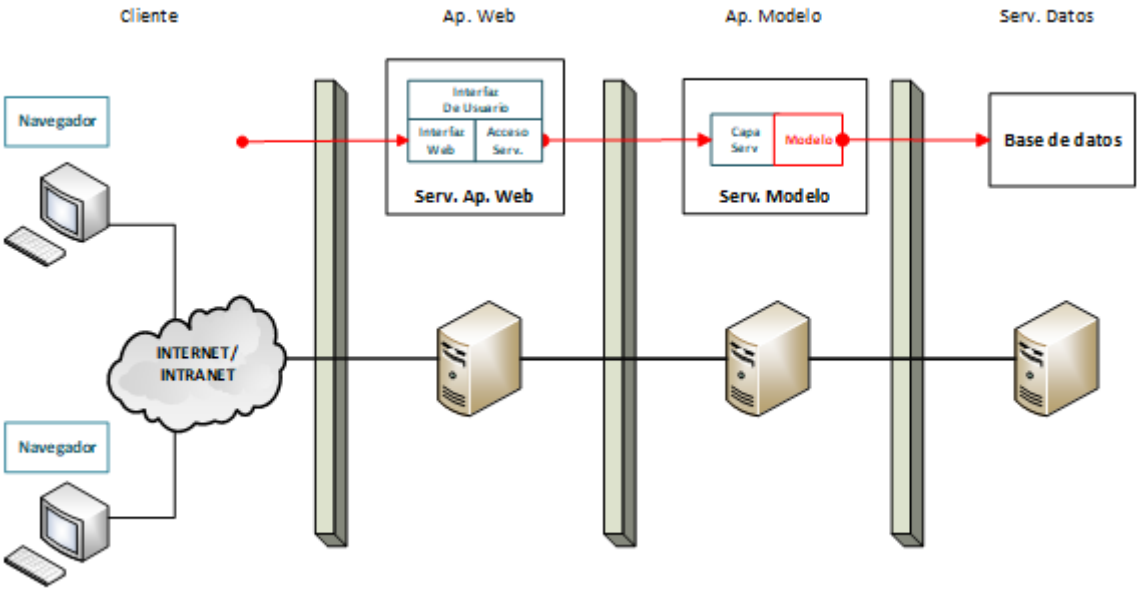

Figura 6.2: Diagrama arquitectura en cuatro capas

Un navegador, para acceder a la aplicación, necesitará de un servidor de aplicaciones que le proporcione la interfaz web. Incorporar esta interfaz dentro del servidor de aplicaciones visto en la arquitectura anterior haría que este y el modelo estén fuertemente acoplados, impidiendo que puedan ser construidos con tecnologías diferentes.

Por ello, con esta arquitectura se pretende hacer ese desacople consiguiendo que múltiples aplicaciones puedan invocar al modelo, independientemente de que sean con interfaz gráfica o mediante navegador, sin necesidad de replicar el código del modelo en cada aplicación.

En consecuencia, analizados los requisitos del sistema y conociendo las necesida-

des del mismo, se diseñará un sistema basado en una arquitectura de cuatro capas desde el punto de vista de un cliente web, y de tres capas, desde el punto del cliente móvil.

6.1.3. Arquitectura completa del sistema

A continuación, se presenta el diseño completo que se elaborará en la aplicación.

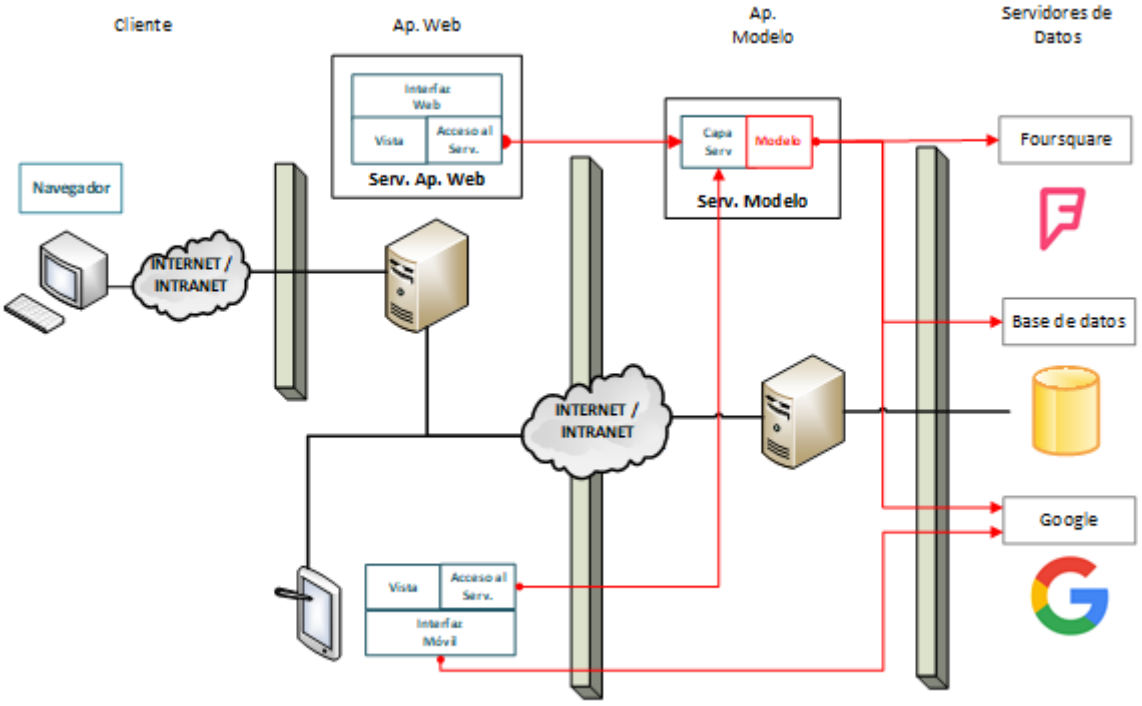

Figura 6.3: Diagrama arquitectura completa del sistema

Como se puede observar en el diagrama, el sistema sigue una arquitectura en cuatro capas.

Al tener la aplicación modelo desacoplada de la aplicación web, la capa de servicios debe servir de enlace entre la capa modelo y la interfaz de usuario. Ese enlace lo ofrece mediante unos servicios REST, que exponen a la capa superior, las funcionalidades implementadas en la capa modelo.

Por su parte, en las aplicaciones cliente, la interfaz web del servidor de aplicaciones y la interfaz del cliente móvil, siguen el patrón de arquitectura Modelo-Vista-Controlador (MVC) y Modelo-Vista-VistaModelo (MVVM), respectivamente. En

ellas, el modelo no se encuentra implementado en la propia aplicación, sino que se accede, a través de la capa de acceso a los servicios, que son ofrecidos por el modelo. De esta forma, ambas aplicaciones finales, invocan al modelo sin necesidad de tenerlo replicado.

6.2. Diseño físico de los datos

6.2.1. Modelo Entidad-Relación

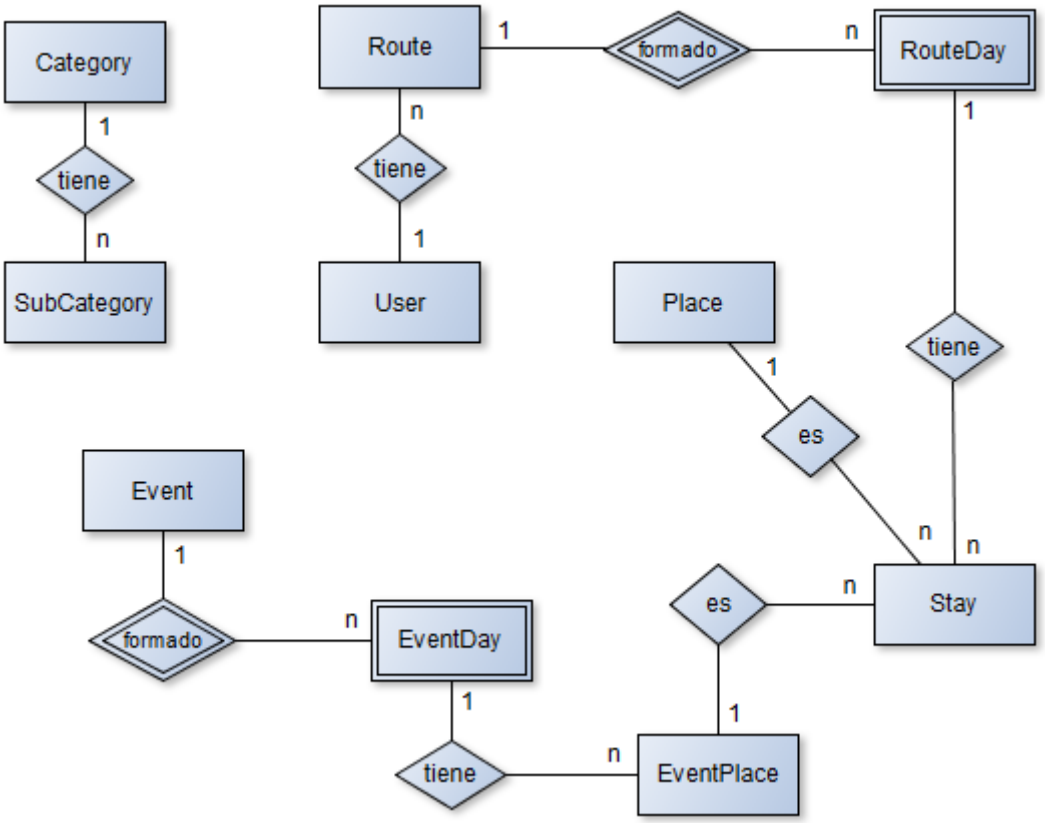

Figura 6.4: Diagrama Entidad-Relación

En el apéndice se incluye un diagrama Entidad-Relación completo, incluyendo entidades y atributos.

6.3. Capa Modelo

Esta capa es la encargada de implementar la lógica de negocio de la aplicación, lo que implica el manejo de las entidades persistentes y el acceso a los datos. Como podemos observar en la Figura 6.4, y debido a la arquitectura establecida, el modelo estará compuesto por la capa de acceso a datos y la capa de lógica de negocio.

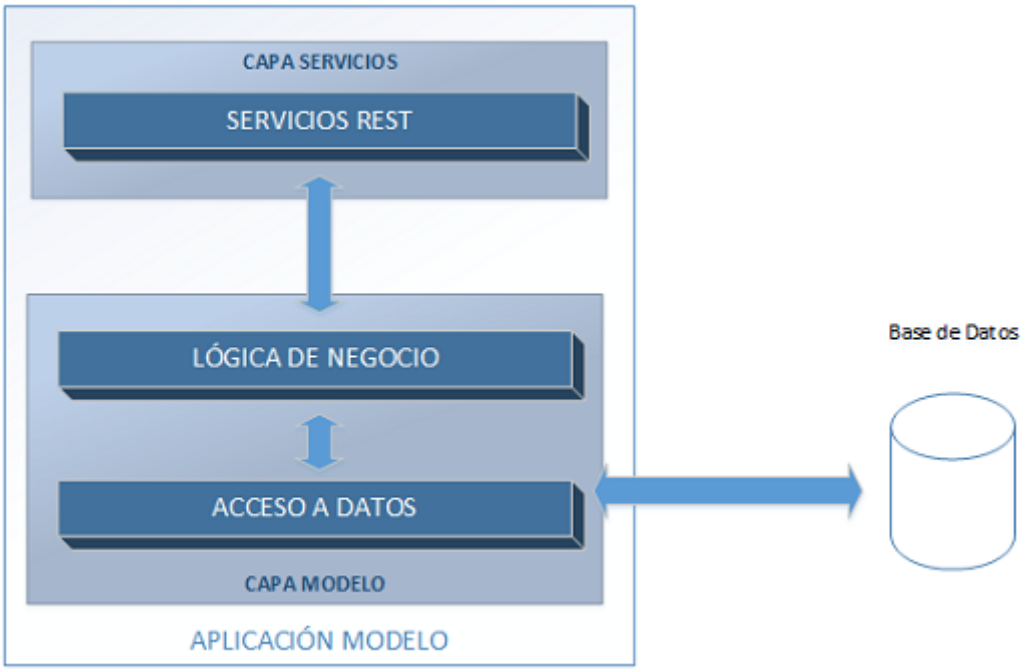

Figura 6.5: Diagrama diseño modelo

6.3.1. Diseño módulo acceso a datos

En la capa de acceso a datos se definirán el conjunto de entidades persistentes que manejará la aplicación y se empleará el patrón de diseño Data-Access-Object (DAO) para abstraer la persistencia de las entidades así como la fuente de datos empleada.

Por cada entidad persistente existirá un DAO encargado de gestionar la comunicación con la fuente de datos empleada.

Diagrama clases persistentes

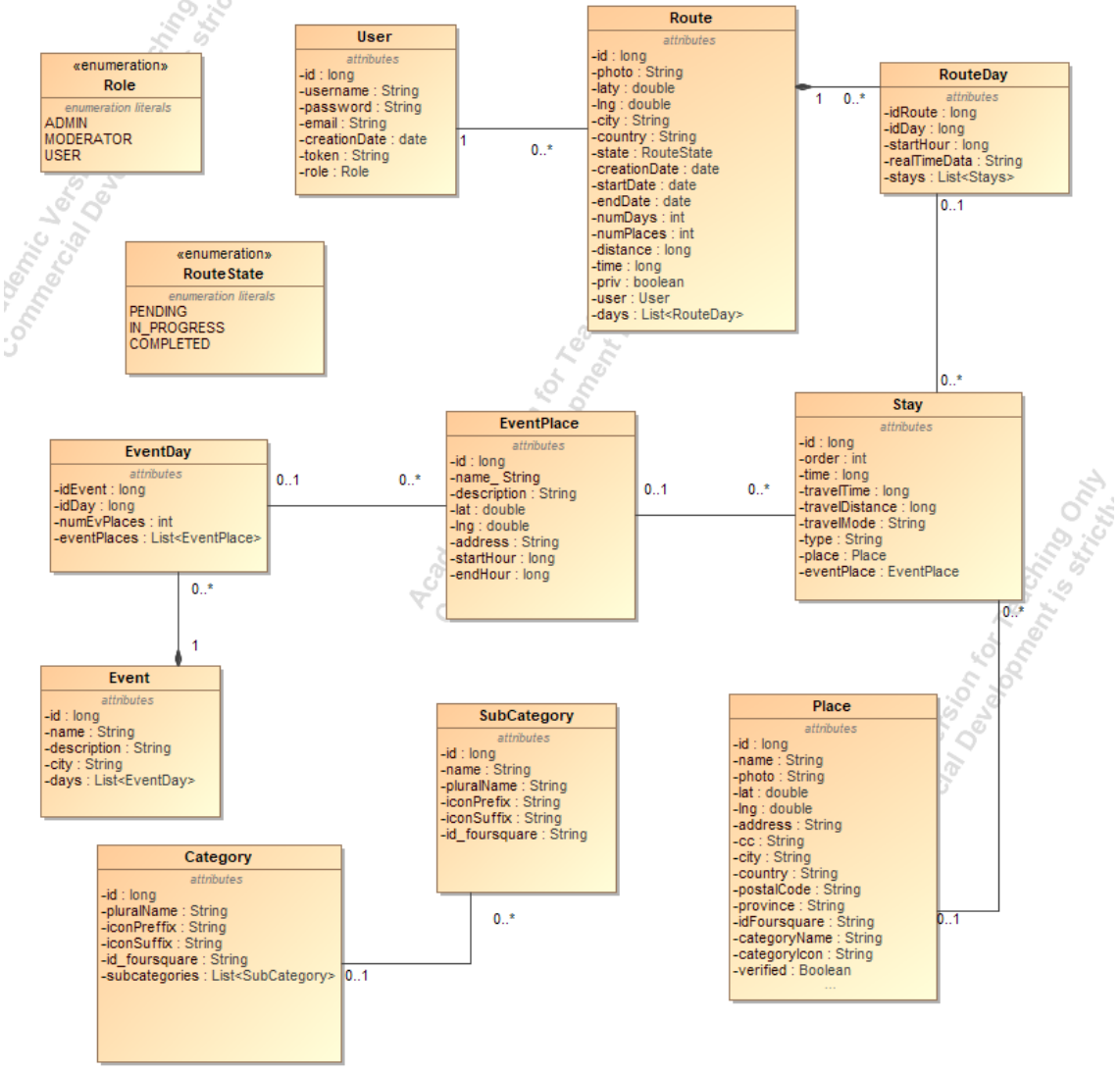

Figura 6.6: Diagrama clases persistentes

En el diagrama se muestran las clases persistentes que manejará la aplicación. A continuación, se detallará brevemente, el significado y funcionalidad de cada una:

- **User:** Es la clase encargada de representar la información de los usuarios registrados en la aplicación.
- **Role:** Enumerado con los roles disponibles para un usuario: *ADMIN, MODERATOR y USER.*

- **Route:** Clase encargada de guardar la información sobre las rutas creadas por los usuarios. Las rutas están compuestas por *RouteDays*.
- **RouteDay:** Clase con una relación fuerte de composición con la clase *Route*. Su tiempo de vida está condicionada por la vida de la clase que la incluye. Mantiene la información para cada uno de los días que componen la duración de una ruta.
- **RouteState:** Enumerado con los diferentes estados por los que pasa una ruta en el tiempo: *PENDING, IN_PROGRESS y COMPLETED*.
- **Stay:** Entidad con la funcionalidad de almacenar las visitas que decida hacer un usuario en un día de una ruta determinada. La visita, puede ser a lugares obtenidos de la fuente externa (Foursquare) o a eventos gestionados por la propia aplicación.
- **Place:** Entidad que registra y almacena los detalles sobre los lugares extraídos de la fuente externa.
- **Event:** Es la clase encargada de representar los eventos dados de alta en el sistema. Los eventos están compuestos por *EventDays*.
- **EventDay:** Clase con una relación fuerte de composición con la clase *Event*. Su tiempo de vida está condicionada por la vida de la clase que la incluye. Mantiene la información para cada uno de los días que componen al evento.
- **EventPlace:** Es la clase donde se maneja toda la información sobre las distintas ubicaciones y actividades que incluye un día determinado del evento.
- **Category:** Clase que almacena la información relevante a las categorías sobre las que se filtran los lugares obtenidos de Foursquare.
- **SubCategory:** Establece una jerarquía con la clase anterior. Almacena las categorías que son un subtipo de una categoría determinada.

Patrón de diseño DAO

Este patrón de diseño intenta desacoplar el acceso a los datos de su almacenamiento subyacente. Los datos persistentes, actualmente, dependen en gran medida

del tipo de base de datos utilizada: base de datos relacional, base de datos orientada a objetos, archivos planos... siendo las bases de datos relacionales las más utilizadas.

Partiendo de que los accesos a diferentes tipos de bases de datos se realizan de manera muy diferente, utilizar este patrón en lugar de acceder directamente a la fuente de datos, nos permite pasar de un tipo de fuente de datos a otro diferente sin tener que realizar modificaciones en la lógica de negocio.

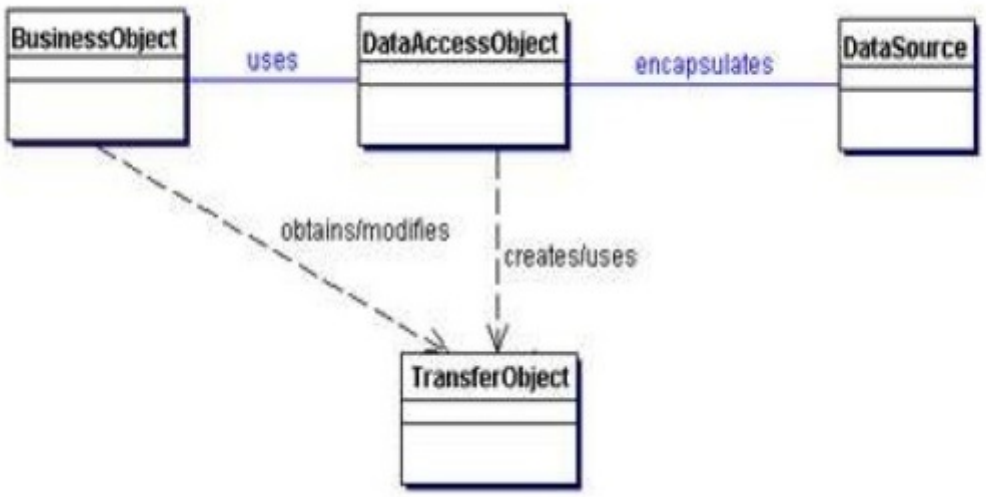

Figura 6.7: Diagrama patrón DAO

En la Figura 6.6 se muestra un pequeño diagrama con los elementos participantes en este patrón de diseño.

- **Business Object:** Representa la clase con la lógica de negocio. Es la responsable de saber qué y cómo modificar el contenido de los datos pero no cómo almacenarlo.
- **Data Access Object:** Se encarga de ocultar la fuente de datos real de manera que el objeto con la lógica de negocio (Business Object) se comunica con este en vez de hacerlo directamente con el objeto de acceso a los datos.
- **DataSource:** Es la clase encargada de obtener las conexiones a la base de datos.
- **Transfer Object:** Es el objeto que se utiliza para transferir el contenido de

los datos reales, del *Data Access Object* al objeto de negocio *Business Object*. Representa los datos almacenados en la base de datos.

Diseño de los DAOs

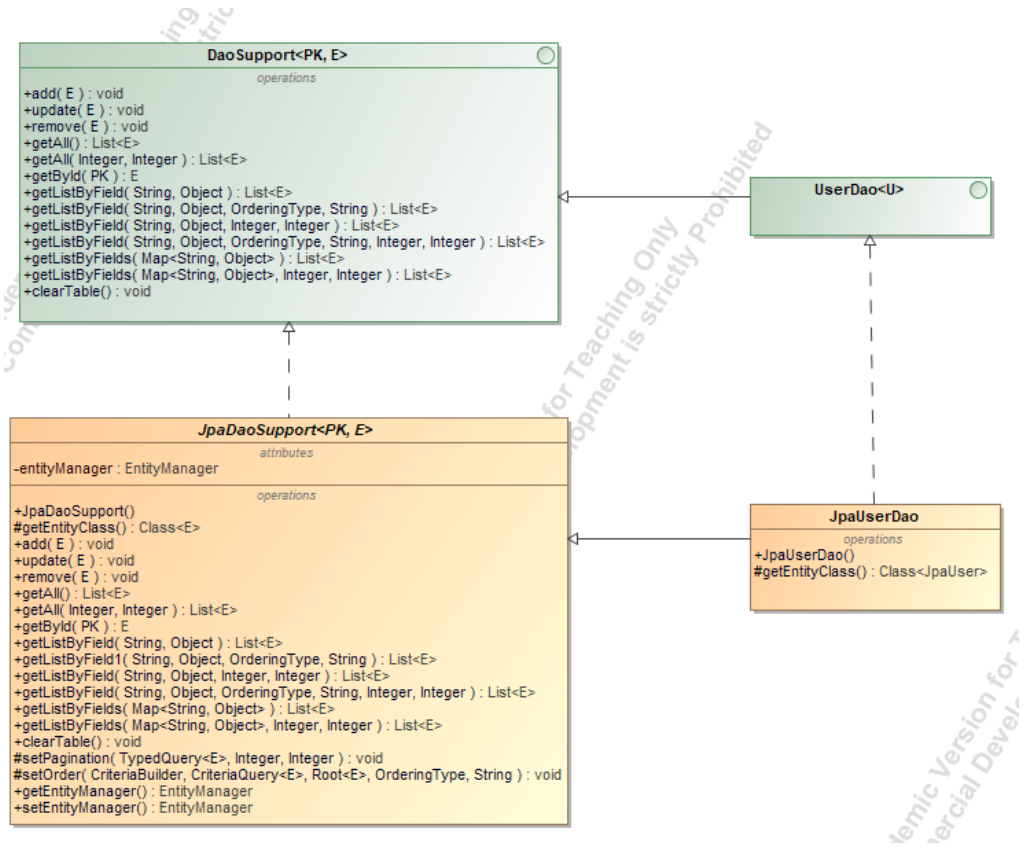

Figura 6.8: Diagrama ejemplo UserDao

En la figura 6.8 se muestra un esquema en el que se define e implementa un DAO genérico que proporciona las principales operaciones de acceso a datos, permitiendo que las definiciones de DAOs, que extiendan a este DAO genérico, tengan las operaciones básicas sin necesidad de definirlas ni implementarlas. Del mismo modo, en los DAOs que extiendan el DAO genérico, también se podrán definir e implementar funcionalidades específicas para cada uno de ellos. Este ejemplo, del DAO de la entidad *User*, se compone de:

- **DaoSupport:** Interfaz que define una serie de métodos y operaciones sobre los datos.

- **JpaDaoSupport:** Clase genérica que implementa las funciones de la interfaz anterior.
- **UserDao:** Definición del DAO específico para la entidad *User*. Extiende las funcionalidades de la interfaz *DaoSupport* y en este caso, no incluye ningún más.
- **JpaUserDao:** Implementación de *UserDao* que extiende la implementación de las funciones de la clase genérica.

Este esquema se repite con todas las entidades, exceptuando aquellas que necesitan de una relación con otra entidad para existir. Por ejemplo, la entidad *RouteDay* necesita de *Route* para existir, por lo que su identificador es dependiente de la otra entidad. Para este tipo de entidades, que no pueden seguir las definiciones del DAO genérico explicado anteriormente por estar diseñado únicamente para entidades formadas por identificadores de un único atributo, se presenta una solución particular para cada entidad, donde cada DAO define e implementa sus funcionalidades necesarias.

Este diseño y el de todos los DAOs se puede observar en el Apéndice .... (Añadir a apendice)

6.3.2. Diseño módulo lógica de negocio

En este módulo o subcapa es donde se implementan y desarrollan los casos de uso previamente especificados.

En el diseño de los servicios ofrecidos se utilizará el patrón de diseño Fachada. Con este patrón, se crearán una serie de servicios que agruparán y gestionarán un conjunto determinado de entidades y componentes ofrecidos por la capa de acceso a datos. Estos servicios ofrecerán una serie de funcionalidades abstrayendo la complejidad de implementación y de dependencia con demás componentes.

Patrón Fachada

El propósito del patrón de diseño Fachada es proporcionar una interfaz unificada para un conjunto de interfaces en un subsistema. Se define una interfaz de nivel superior lo que permite hacer el subsistema más fácil de utilizar.

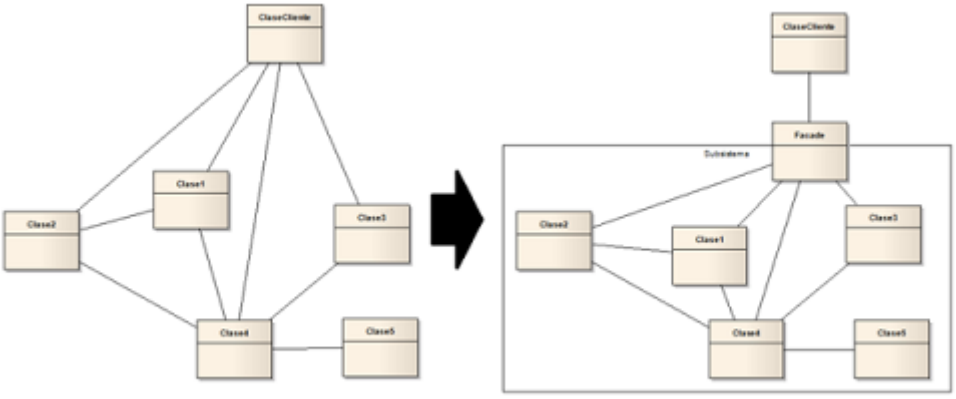

Figura 6.9: Ejemplo patrón de diseño Fachada

Estructurar un sistema en subsistemas ayuda a reducir la complejidad. Uno de los objetivos más comunes de diseño es minimizar la comunicación y las dependencias entre los subsistemas. Esto se puede lograr haciendo uso de este patrón.

Como podemos observar en la Figura 6.7, utilizando el patrón fachada se proporciona una interfaz única de acceso que se encargar de gestionar las comunicaciones y dependencias necesarias con otros módulos o subsistemas para realizar sus funcionalidades. Permite abstraer al cliente de cómo se gestionan las comunicaciones con

los diferentes módulos.

Diseño de los servicios

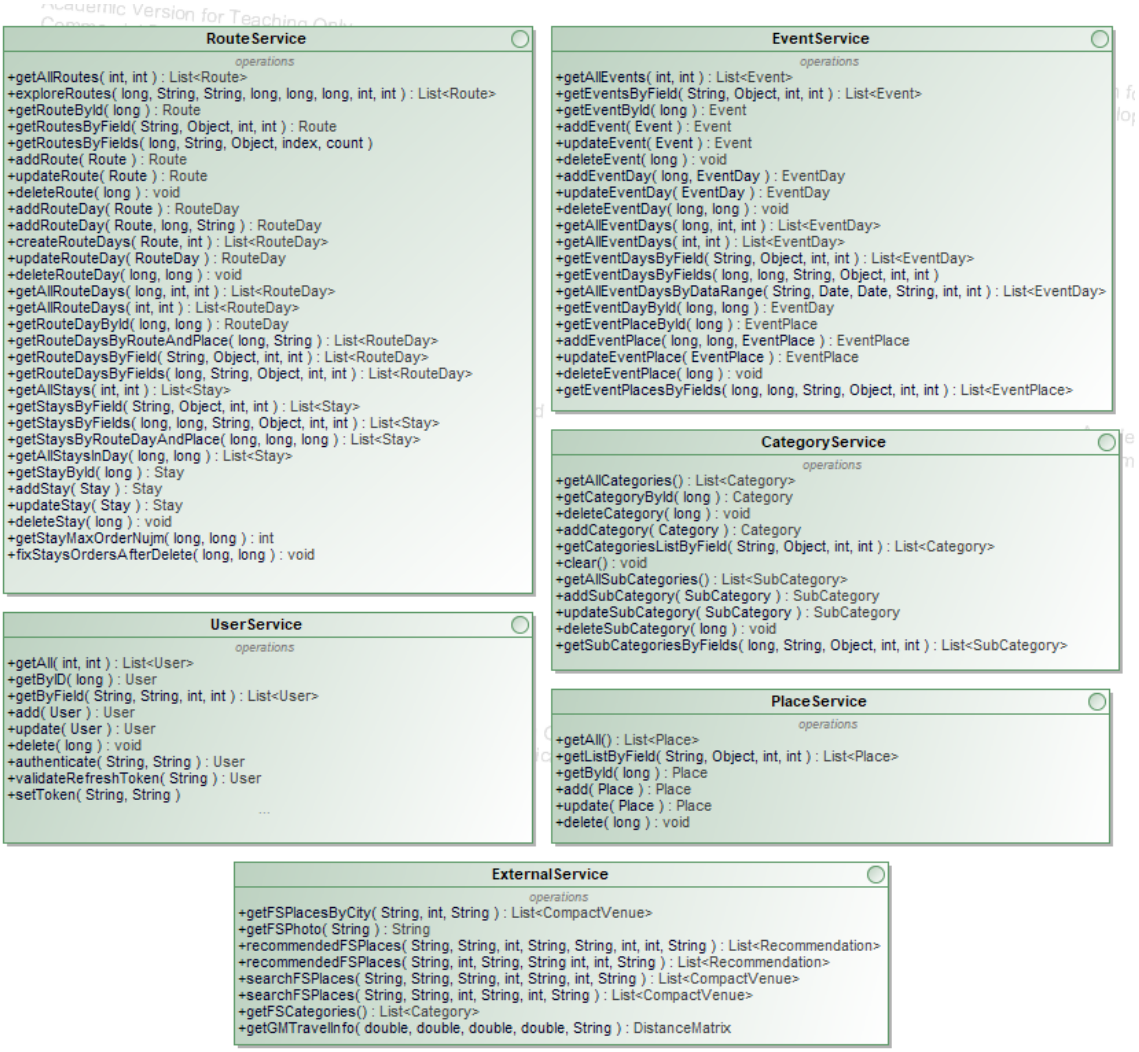

Figura 6.10: Diagrama diseño servicios

En la Figura 6.8 podemos ver el diseño de los servicios que implementarán las funcionalidades de la aplicación. Se detallan brevemente a continuación:

- **RouteService:** Servicio encargado de implementar los casos de uso o funcionalidades referentes a las rutas. Gestionan tanto las propias rutas como su composición por días y las visitas incluidas en cada día.

- **EventService:** Este servicio es el encargado de la gestión de los eventos en la aplicación. Al igual que con el anterior servicio, este incluye las funcionalidades para gestionar la composición por días y los lugares o ubicaciones específicas que existan dentro de un mismo evento.

- **UserService:** Ofrece los casos de uso referente a la entidad usuarios. Incluye desde el manejo de los datos sobre los usuarios hasta las funcionalidades de autenticación.

- **CategoryService:** Es el servicio encargado de administrar las categorías.

- **PlaceService:** Presenta las funcionalidades necesarias para gestionar los lugares que se registran en la aplicación.

- **ExternalService:** Es el servicio encargado de ofrecer las funciones que requieren de una comunicación con fuentes de datos externos. En este caso, se proporcionará una implementación que hará uso de las APIs de Foursquare y de Google Maps.

Diagrama ejemplo patrón fachada

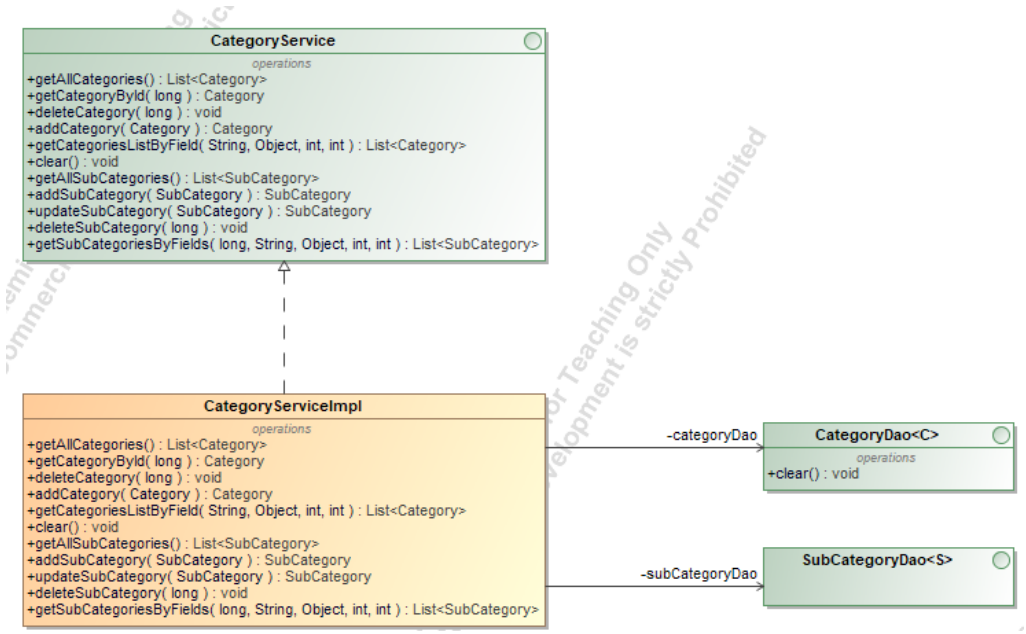

Figura 6.11: Diagrama ejemplo *CategoryService*

En la figura 6.11, podemos ver el ejemplo de un servicio concreto, en este caso *CategoryService*. En este ejemplo, podemos observar la definición del servicio a través de la interfaz *CategoryService* y donde su implementación es la encargada de gestionar las comunicaciones con los diferentes módulo y subsistemas, en este caso, con los DAOs de las entidades *Category* y *SubCategory*.

6.4. Capa servicios

La capa de servicios ofrecerá las funcionalidades de la capa modelo a través de recursos accesibles por la red, mediante peticiones HTTP. Se proporcionará una implementación de la capa de servicios siguiendo el estilo arquitectónico Representational State Transfer (REST) mediante el uso del API de programación JAX-RS.

Se utilizará el patrón de diseño Data Transfer Object (DTO).

Patrón de diseño DTO

Es un patrón de diseño que se utiliza para transferir varios atributos entre un cliente y un servidor, o viceversa. De esta forma se consigue encapsular los objetos de negocio de manera que, cuando un cliente solicita al servidor determinada información, este construye un objeto de transferencia que rellena con los atributos del objeto de negocio y se lo envía finalmente al cliente.

Generalmente, este patrón de diseño está compuesto por las siguientes clases:

- **Objeto de transferencia:** Clase POJO únicamente compuesta por atributos y métodos de lectura y escritura.
- **Clase de negocio:** Clase encargada de implementar las funcionalidades de la aplicación. Envía o recibe la clase *Objeto de transferencia* por parte del cliente, que modifica utilizando los datos de la base de datos.

Diagrama módulo REST

A continuación se mostrará un diagrama de ejemplo, con las clases involucradas en la gestión de los casos de uso de usuarios.

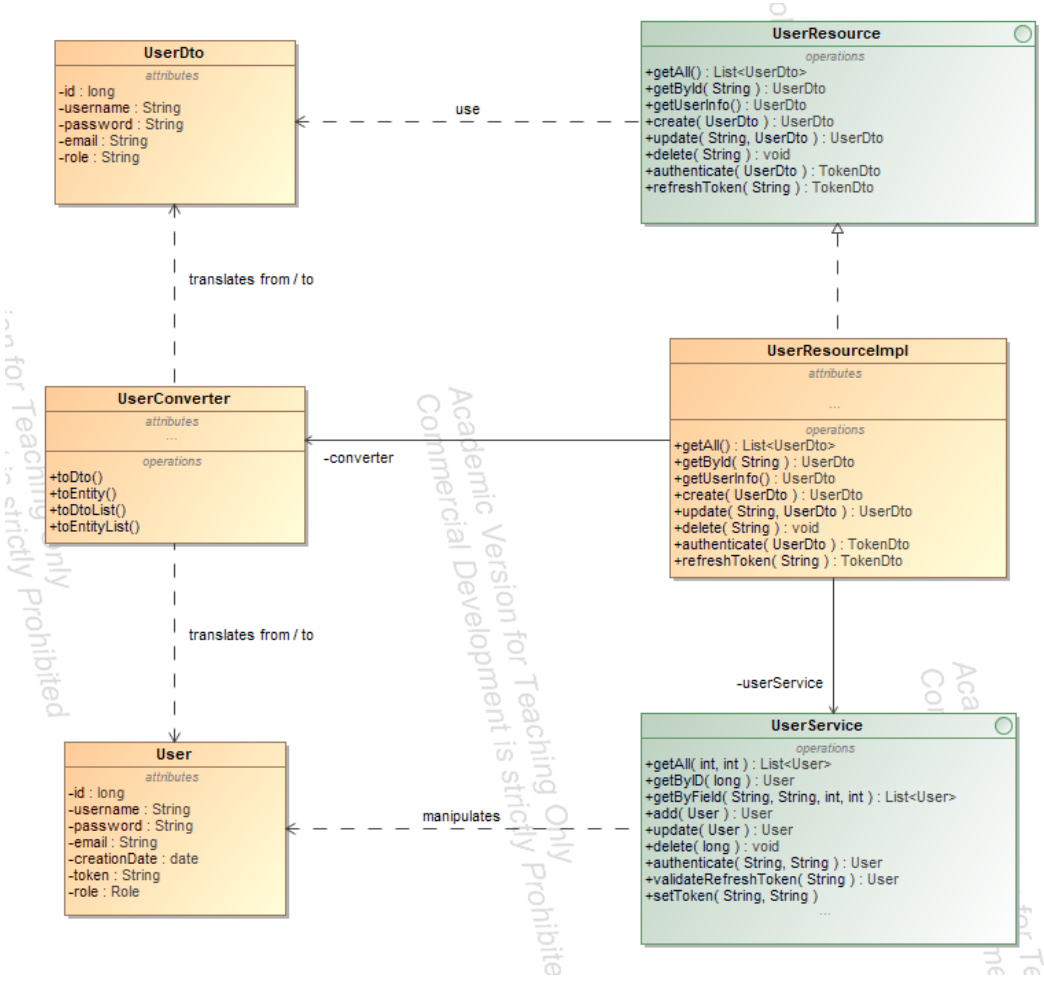

Figura 6.12: Diagrama ejemplo servicios usuario

- **UserDto:** Es el objeto de transferencia.
- **User:** Objeto de negocio para la entidad usuario.
- **UserResource:** Interfaz que expone ciertas funcionalidades a través de un recurso web. En este ejemplo, se define la implementación *UserResourceImpl*, encargada de implementar las funciones establecidas en la interfaz. Los datos e información manejada se hace a través del objeto de transferencia.
- **UserService:** Servicio ofrecido por la capa modelo que gestiona los casos de uso y funcionalidades referentes a usuarios. Manipula el objeto de negocio *User* y sus funcionalidades son invocadas por parte de *UserResouceImpl*.
- **UserConverter:** Clase de utilidad que permite a la implementación del re-

curso *UserResource* transformar el objeto de negocio en un objeto de transferencia, y viceversa.

El resto de servicios definidos en la aplicación seguirán una estructura similar. Se ofrecerán recursos web accesibles mediante HTTP que permitirán la invocación remota de la lógica de negocio de la aplicación.

6.5. Capa Interfaz

La capa interfaz, también conocida como capa cliente o capa de presentación, es la encargada de separar la interacción del usuario respecto a la lógica de negocio. Esta capa simplemente presenta la información al usuario y recibe las peticiones que debe comunicar a la capa modelo.

En nuestro sistema, existen dos presentaciones de la aplicación completamente diferentes e independientes: una aplicación web accesible desde navegador web y una aplicación para dispositivos móviles.

6.5.1. Aplicación web

La aplicación web se ejecutará en un servidor de aplicaciones que recibirá y gestionará todas las peticiones del cliente. Será la encargada de generar la vista web de la aplicación así como comunicarse con la capa modelo, accediendo a la capa de servicios del modelo.

Seguirá el patrón de arquitectura Modelo-Vista-Controlador (MVC).

Patrón Modelo-Vista-Controlador

MVC es un patrón de arquitectura de software que separa los datos y la lógica de negocio, de su representación. Está formado por tres componentes distintos: modelo, vista y controlador.

El modelo es la información de la aplicación y el que proviene de la lógica de negocio. La vista, por su parte, es la representación en pantalla mientras que el

controlador es el encargado de mantener la comunicación con el modelo y definir la forma en que la interfaz reacciona a las peticiones del usuario. Antes del patrón MVC, los diseños de interfaz de usuario tendían a agrupar estos objetos. Con este patrón se consigue desacoplar vistas y modelos estableciendo un protocolo de 'suscripción-notificación' entre ellos.

Diagrama MVC aplicación web

A continuación se mostrará un diagrama donde se verán los componentes MVC anteriormente comentados en el sistema de nuestra aplicación web.

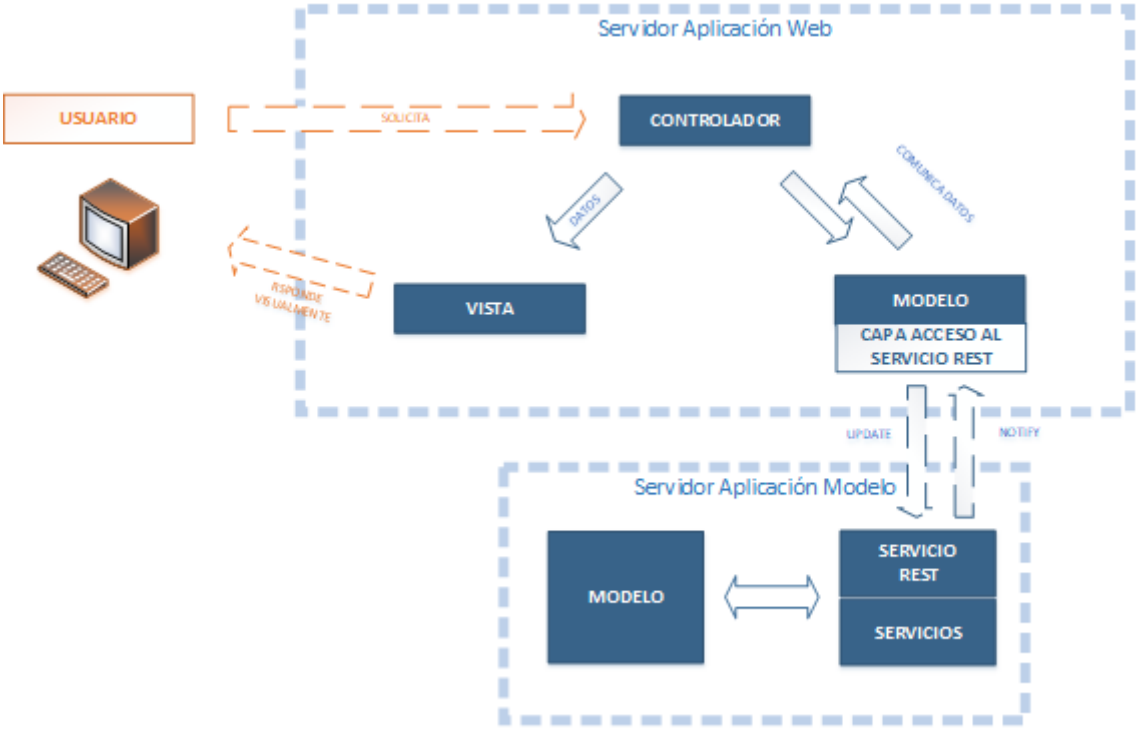

Figura 6.13: Diagrama MVC aplicación web

El controlador recibirá las solicitudes del usuario y obtendrá los datos del modelo a través del servicio REST especificado. Con estos datos, el controlador creará la vista correspondiente, que será entregada al usuario.

Diseño capa acceso al servicio

En la siguiente figura se mostrará el diseño que permitirá invocar a la capa de servicios del modelo.

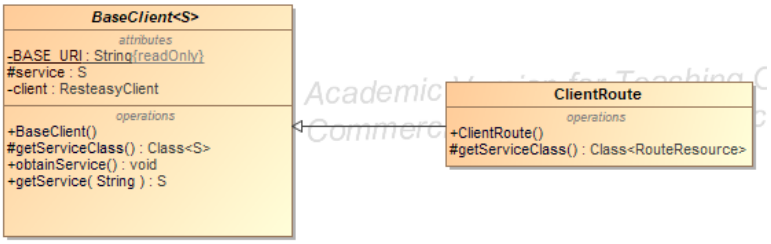

Figura 6.14: Diagrama ejemplo acceso a servicios cliente web

En este diagrama se definen dos clases: *BaseClient* y *ClientRoute*. La primera es la encargada de crear un cliente que permita construir peticiones HTTP que se usarán para invocar los métodos ofrecidos en la capa de servicios.

Por su parte, la clase *ClientRoute*, modifica el comportamiento de la clase anterior indicando sobre qué recurso tiene que elaborar las peticiones HTTP. En este ejemplo, se utiliza el recurso *RouteResource* de la capa de servicios. Análogamente, se crean las diferentes clases clientes, para acceder a los diferentes recursos creados en la capa de servicios del modelo.

Diseño controladores

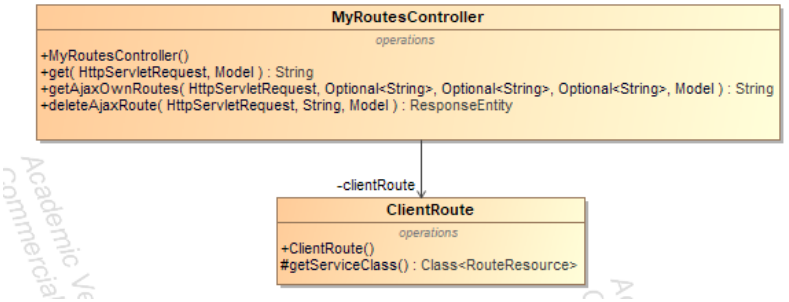

Figura 6.15: Diagrama ejemplo controlador cliente web

En este ejemplo se especifica el controlador *MyRoutesController* que define sobre una URL los métodos que se pueden realizar para la gestión de las rutas propias de un usuario.

- **ClientRoute:** Clase que permite invocar los métodos de la capa de servicios.
- **MyRoutesController:** Controlador web que ofrece las siguientes funcionalidades:
  - **get:** Recibe las peticiones GET sobre la URL especificada.
  - **getAjaxOwnRoutes:** Recibe las peticiones asíncronas para consultar las rutas propias.
  - **deleteAjaxRoute:** Recibe las peticiones asíncronas para eliminar una ruta específica.

6.5.2. Aplicación móvil

La aplicación móvil será desarrollada mediante el SDK Ionic. Una aplicación Ionic es, en esencia, una aplicación Angular que se base en una arquitectura Modelo-Vista-VistaModelo (MVVM).

Patrón Modelo-Vista-VistaModelo

Este modelo se considera una adaptación del patrón anteriormente explicado, el Modelo-Vista-Controlador.

MVC y MVVM siguen un funcionamiento similar pero con unas pequeñas diferencias.

- En el patrón Modelo-Vista-Controlador se separan los datos de una aplicación en los tres componentes anteriormente mencionados (Modelo, Vista y Controlador). Cuando la lógica de negocio realiza un cambio, este es necesario que sea actualizado en la vista.
- El patrón Modelo-Vista-VistaModelo sigue la misma separación que el patrón MVC. En este caso, en vez de controlar los cambios manualmente en la vista o

en los datos como sucede con el patrón MVC, estos se actualizan directamente cuando sucede un cambio o modificación en ellos. De tal manera, que si la vista actualiza un dato que está presentando, este se actualizaría en modelo automáticamente, y viceversa.

El denominado 'Controller' del patrón MVC cambia a 'ViewModel´ en el MVVM. Este último, a diferencia del anterior, incorpora un 'binder' que se encarga de sincronizar la información entre la vista y el modelo.

Figura 6.16: Diagrama patrón MVVM

Diseño capa acceso al servicio

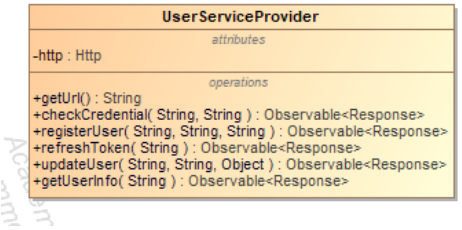

Figura 6.17: Diagrama ejemplo acceso servicios cliente móvil

Para el cliente móvil se definen un conjunto de clases que permitan ejecutar los métodos de la capa de servicios. *UserServiceProvider* es una de ellas, en la que se crean y elaboran las peticiones HTTP necesarias para ejecutar los diferentes métodos ofrecidos por la capa servicios del modelo.

Diseño controladores

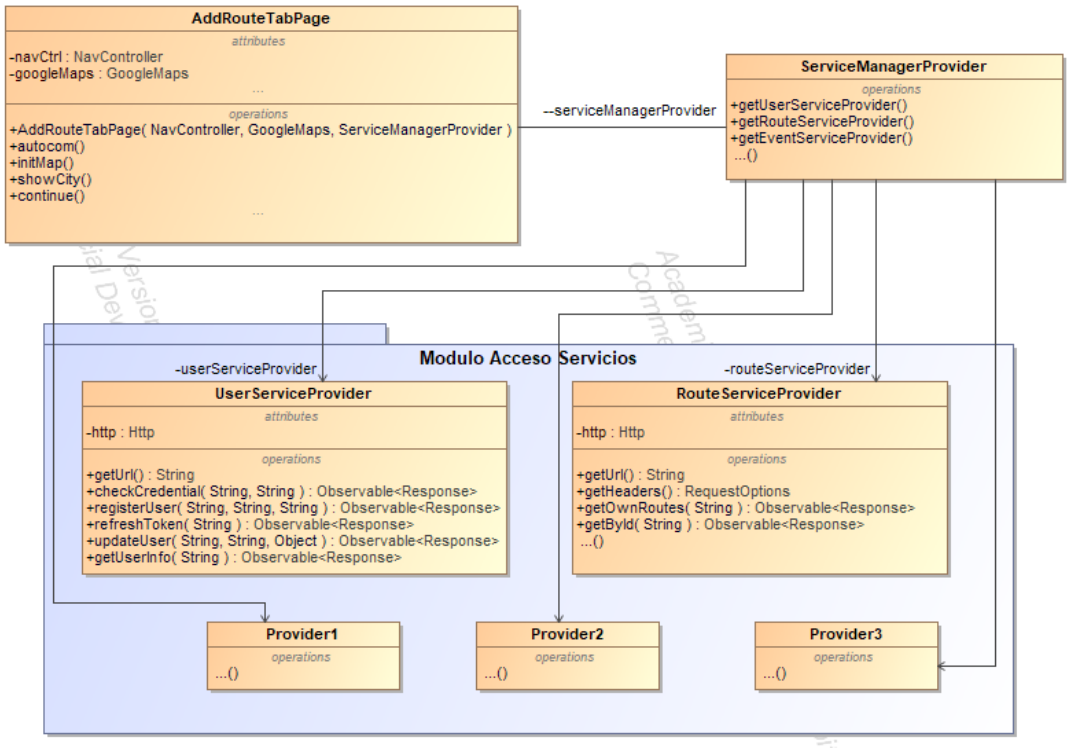

Figura 6.18: Diagrama ejemplo controlador cliente móvil

En la figura 6.18 se muestra el ejemplo de un controlador de la aplicación móvil, *AddRouteTabPage*, encargado de gestionar la pantalla que permite al usuario crear rutas. Dicho controlador hace uso de la clase *ServiceManagerProvider* que permite obtener las diferentes clases que implementan la capa de acceso a los servicios.

6.6. Diseño autenticación

Uno de los principales aspectos de seguridad de las aplicaciones son los procesos de autenticación y de autorización. Para nuestras aplicaciones se propone el siguiente diseño basado en tokens de acceso (*access tokens*) y tokens de refresco (*refresh tokens*) para la autenticación y autorización de los usuarios. A continuación, se mostrará un diagrama de secuencia en el que se podrá observar el proceso de autenticación en el cliente móvil.

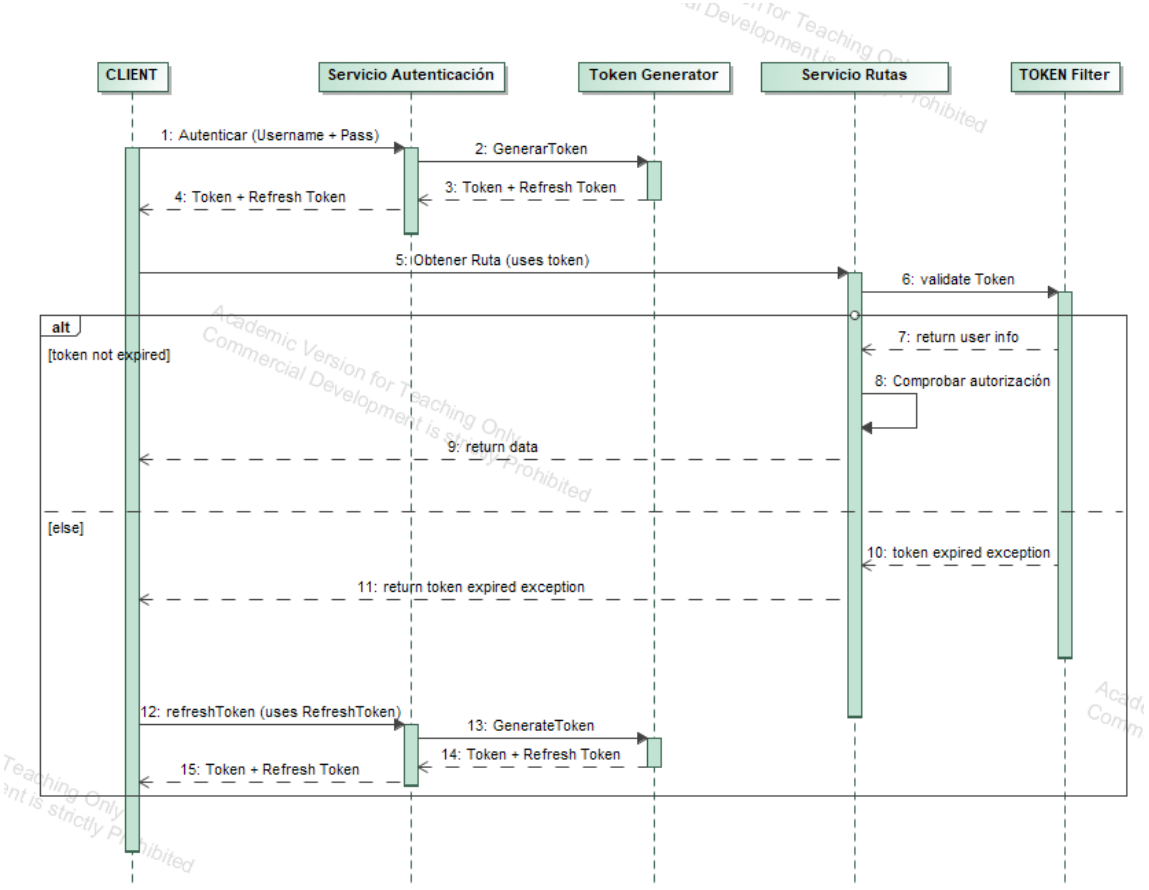

Figura 6.19: Diagrama ejemplo controlador cliente móvil

//TO DO – Incluir como requisito funcional mantener sesión en cliente móvil

Este diseño de autenticación surge debido al requisito funcional para el cliente móvil, que exige mantener la sesión abierta, sin necesidad de autenticarse de nuevo, cada vez que la aplicación es iniciada. Con este diseño, nuestro cliente se autentica en la aplicación y consigue un token de acceso, con el que podrá realizar acciones autenticadas, y un token de refresco. Cuando el primero caduca y deja de ser válido en el tiempo, el sistema devuelve una excepción específica, de tal manera, que el cliente móvil, sabrá que tendrá que actualizar dicho token de acceso realizando una petición específica al servidor indicando su token de refresco.

Con este procedimiento, se consigue que el cliente móvil, almacenando los tokens ofrecidos por el sistema, no tenga que solicitar al usuario las credenciales cada vez que el token de acceso deje de ser válido. Únicamente, solicitará las credenciales al usuario cuando no sea capaz de obtener uno nuevo mediante el servicio del token de

refresco.

El cliente web seguirá el mismo proceso de autenticación mediante token de acceso pero, por las particularidades y características de dicho cliente, no hará uso del token de refresco. Cuando el token de acceso deje de ser válido, el sistema pedirá al usuario que introduzca de nuevo las credenciales.

Capítulo 7

IMPLEMENTACIÓN

7.1. Estructura de la aplicación

7.1.1. Estructura proyecto Java

Para la implementación del modelo y la aplicación web, se ha elaborado un proyecto Maven con los siguientes módulos:

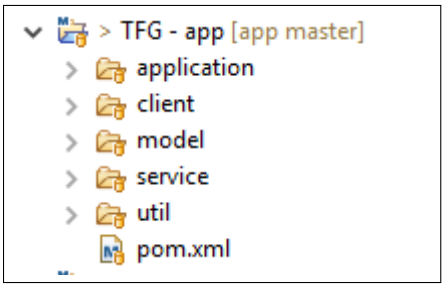

Figura 7.1: Estructura proyecto Java

- **TFG - app:** Es el proyecto Maven de la aplicación Java. Está formado por diferentes módulos que forman la aplicación:
  - **Application:** Es un módulo donde se encuentran las aplicaciones web del sistema.
  - **Client:** Es un módulo donde se implementa el cliente que consume y accede los servicios del modelo.

- **Model:** Módulo donde reside la persistencia y la lógica de negocio de la aplicación.
- **Service :** Es el módulo que define e implementa la capa de servicios.
- **Util :** Módulo de utilidad. Aporta clases y funciones comunes a los demás módulos.
- **pom.xml :** Archivo utilizado por Maven para la construcción del proyecto, manteniendo la gestión de las dependencias y el orden de construcción de los módulos.

Con esta separación en módulos conseguimos hacer más independiente cada uno de los módulos que formarán el desplegable de la aplicación. De tal manera, que si queremos utilizar otra implementación de persistencia, simplemente tenemos que reemplazar el JAR generado por dicho módulo por el que queramos utilizar, en el archivo de aplicación web (WAR).

Módulo Model

El módulo *model* de la aplicación está compuesto por: *core* y *persistence*.

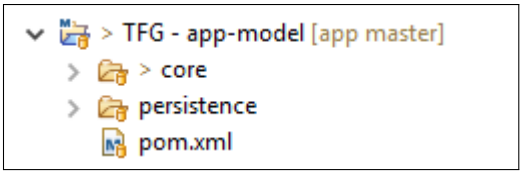

Figura 7.2: Estructura módulo *model*

En el submódulo *persistence* se implementa toda la persistencia de datos, desde las clases persistentes hasta las definiciones e implementaciones de los DAOs. Por su parte, en el submódulo *core* se define e implementa la lógica de negocio.

Estructura directorios Model - Persistence

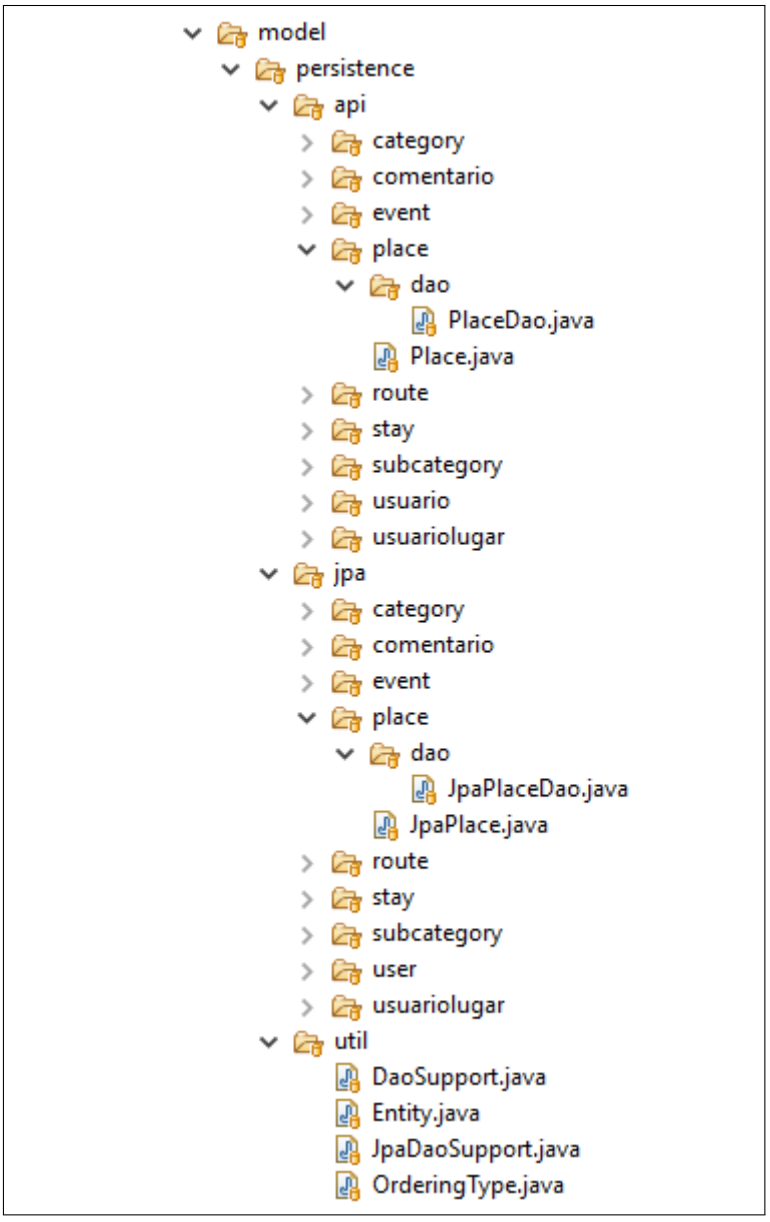

Figura 7.3: Estructura módulo *model-persistence*

- **src/main/java/ ../model/persistence/api.** Es el directorio donde se encuentran todas las definiciones de las clases persistentes (Ej: Place.java) y los DAOs (Ej: PlaceDao.java).
- **src/main/java/ ../model/persistence/jpa.** Directorio donde residen las implementaciones de las definiciones anteriores (Ej: JpaPlace.java, JpaPlace-

Dao.java). Como su nombre indica, se hace uso del API de persistencia JPA.

- **src/main/java/ ../model/persistence/util.** Es el directorio donde se incluyen clases de utilidad (Ej: DaoSupport y JpaDapSupport, para la implementación de los DAOs).

Estructura directorios Model - Core

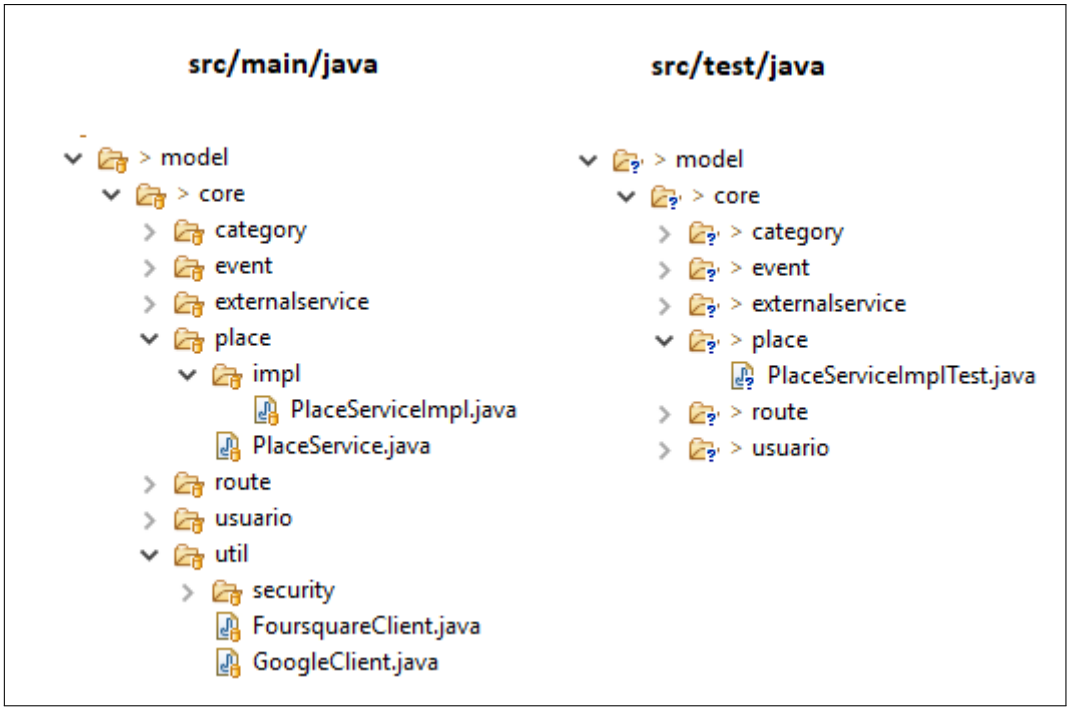

Figura 7.4: Estructura módulo *model-core*

- **src/main/java/ ../model/core.** Directorio donde se especifican e implementan la lógica de negocio de la aplicación.

- **src/main/java/ ../model/core/util.** Directorio de utilidad que incluye los clientes de las APIs externas.

- **src/test/java/ ../model/core.** Incluye las pruebas automatizadas para cada uno de lo servicios de la lógica de negocio.

Módulo Service

El módulo *service* de la aplicación está compuesto por los submódulos *api* y *core*.

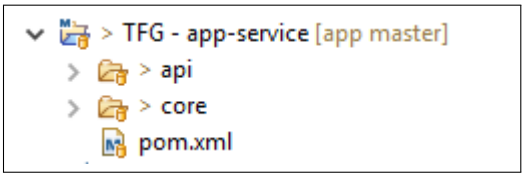

Figura 7.5: Estructura módulo *service*

Estructura directorios Service - Api

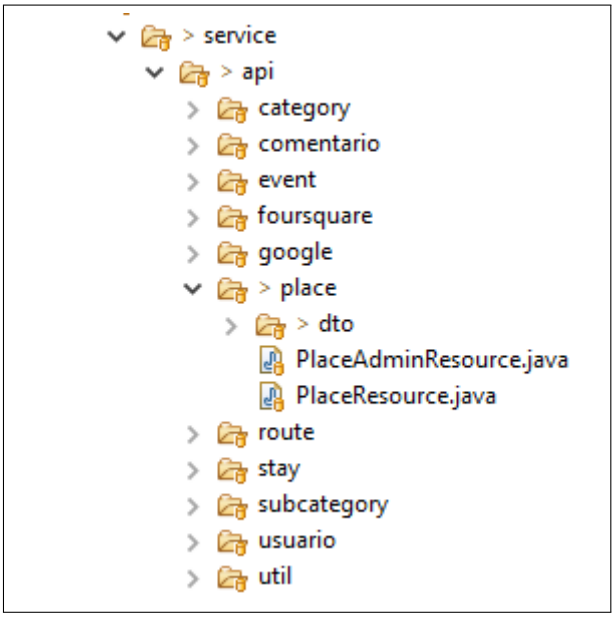

Figura 7.6: Estructura módulo *service-api*

- **src/main/java/ ../service/api.** Directorio donde se definen cada uno de los recursos web mediante la especificación de la API de JAX-RS. Esta API ofrece el soporte para la creación de servicios web, que ofrecerán remotamente, los servicios de la capa modelo. En la figura, se puede observar la definición de dos recursos web, como son: *PlaceResource.java* y *PlaceAdminResource.java*.

Estructura directorios Service - Core

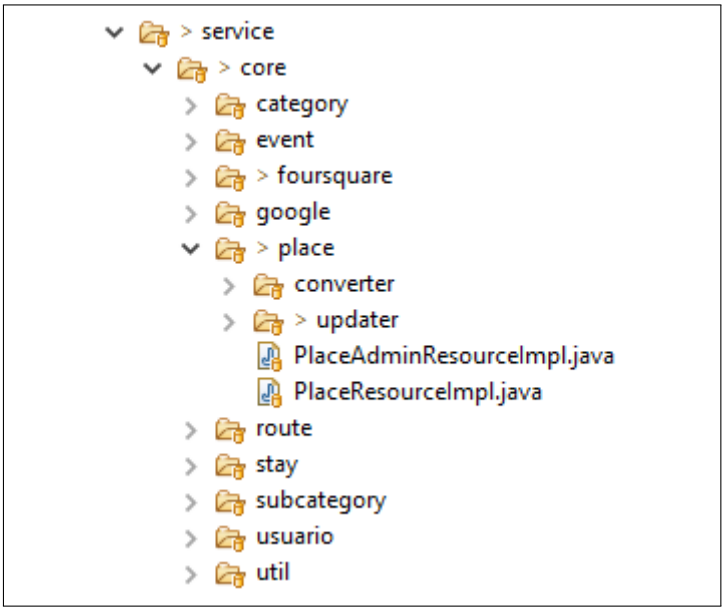

Figura 7.7: Estructura módulo *service-core*

- **src/main/java/ ../service/core.** Directorio con las implementaciones para cada uno de los recursos definidos en el submódulo *service-api*. Se puede observar las clases *PlaceResourceImpl.java* y *PlaceAdminResourceImpl.java* que implementa las clases mostradas en el submódulo anterior.

  - **../service/core/*/converter.** Directorio con las clases necesarias para la conversión de objetos a persistentes a objetos de transferencia de datos.

  - **../service/core/*/updater.** Directorio con las clases necesarias para la creación del objeto persistente a modificar a partir del objeto de transferencia de datos recibido.

  - **../service/core/util.** Directorio en el que se encuentran clases que ofrecen funcionalidades como la conversión de excepciones Java a respuestas HTTP, validadores de datos de entrada, etc...

Módulo Application

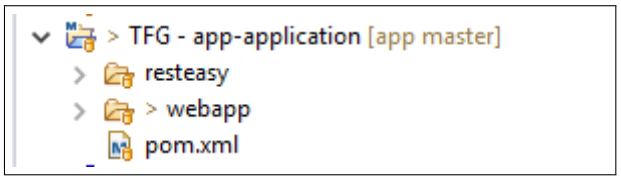

Figura 7.8: Estructura módulo *application*

Formado por los submódulos *resteasy* y *webapp*. El primero de ellos será el encargado de ofrecer una aplicación REST donde se expondrán, remotamente, los servicios creados en la capa de servicios del modelo. Se utilizará el proyecto *JBoss RESTEasy* como implementación del API de JAX-RS.

Por su parte, el módulo *webapp* será el encargado de ofrecer una aplicación web, accesible mediante navegador web. Seguirá una arquitectura MVC.

Estructura directorios Application - Resteasy

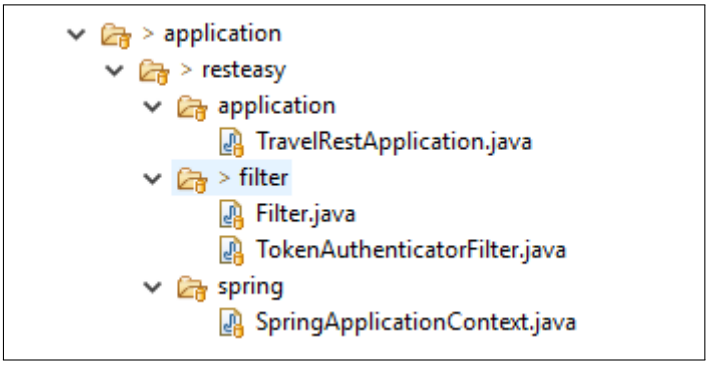

Figura 7.9: Estructura módulo *application-resteasy*

- **src/main/java/ ../application/resteasy.**
  - **../application/resteasy/application.** Directorio con la clase encargada de añadir al contenedor de la aplicación los objetos definidos en la capa de servicios.
  - **../application/resteasy/filter.** Directorio en el que se encuentran los filtros de la aplicación.

- **../application/resteasy/spring.** Directorio con la clase encargada de obtener los objetos del contenedor de Spring y poder incorporarlos al contenedor de la aplicación. para su uso.

Estructura directorios Application - Webapp

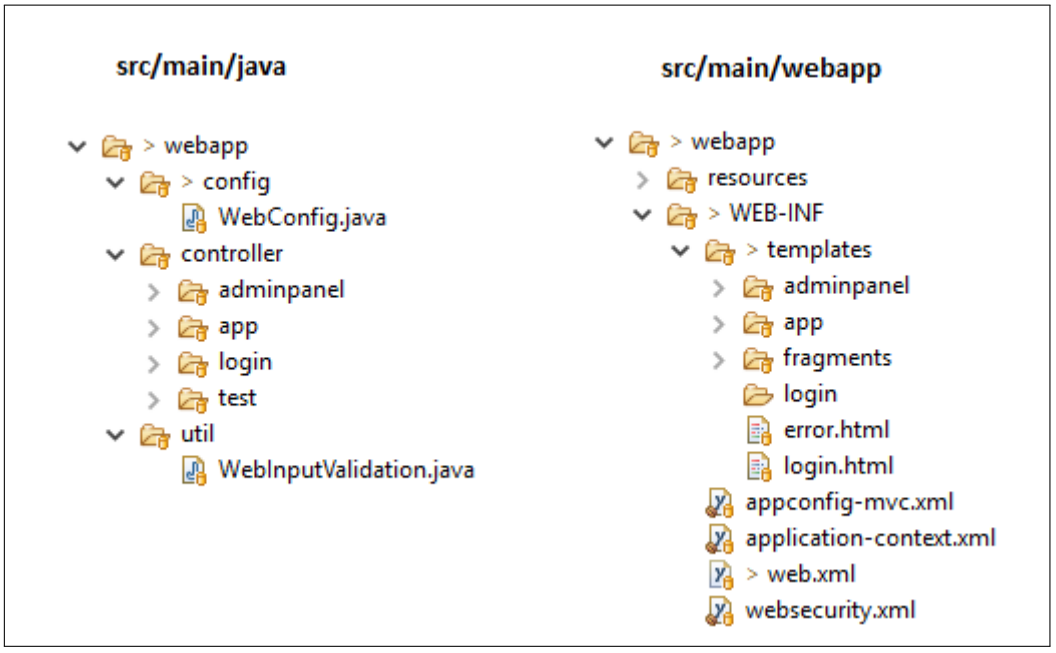

Figura 7.10: Estructura módulo *application-resteasy*

- **src/main/java.**
  - **../application/webapp/config.** Directorio donde residen archivos de configuración de la aplicación.
  - **../application/webapp/controller.** Directorio en el que se encuentran implementados los controladores de la aplicación.
  - **../application/webapp/util.** Directorio con las clases de utilidad.
- **src/main/webapp.**
  - **../application/webapp/resources.** Directorio con los ficheros que aportan un mejor aspecto a la web. Incluye, archivos JavaScript, CSS e imágenes.

- **../application/webapp/WEB-INF.** Directorio en el que se encuentran las plantillas HTML utilizadas para crear las páginas web.

Módulo Client

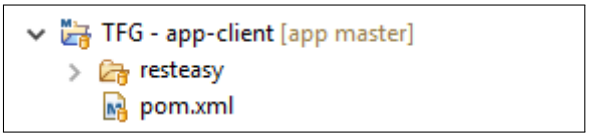

Figura 7.11: Estructura módulo *client*

El módulo *client* está compuesto, únicamente, del módulo *resteasy*. Este módulo, define e implementa un cliente para la aplicación REST anteriormente comentada.

Estructura directorios Client - Resteasy

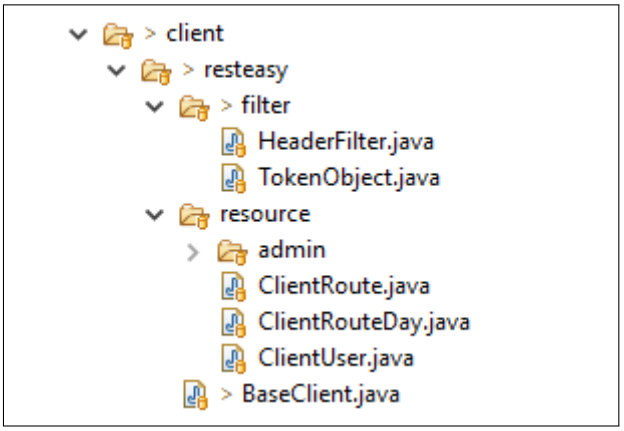

Figura 7.12: Estructura módulo *client-resteasy*

- **src/main/java/ ../client/resteasy.**
  - **../resteasy/filter.** Directorio donde residen los filtros utilizados por el cliente.
  - **../resteasy/resource.** Directorio en el que se encuentran implementados los clientes específicos para cada servicio.

7.1.2. Estructura proyecto Ionic

Ionic presenta una estructura típica de proyecto Cordova donde se pueden instalar complementos nativos y crear archivos de proyecto, específicos para cada plataforma. La estructura de los archivos con el código fuente de la aplicación es la siguiente:

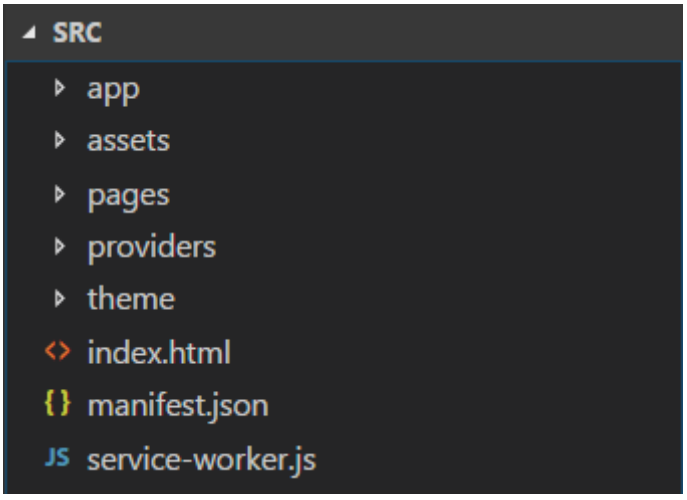

Figura 7.13: Estructura aplicación *Ionic*

- **src/index.html.** Punto de entrada de la aplicación que tiene como propósito la configuración de *scripts* y hojas de estilo para arrancar la aplicación.
- **src/app.** Directorio con las clases que inician la aplicación.
- **src/assets.** Directorio que almacena recursos de estáticos, como imágenes, iconos, etc...
- **src/pages.** Directorio en el que se encuentran las diferentes vistas y controladores que forman la aplicación.
- **src/providers.** Directorio que incluye las clases que implementan el acceso a los servicios y clases que implementan determinadas funcionalidades en la aplicación.
- **src/theme.** Directorio con las clases SASS que especifican ciertos aspectos de estilo de la aplicación.

Estructura directorio pages

En la siguiente figura, se muestra el subdirectorio *pages*, que incluye todas las vistas y controladores de la aplicación.

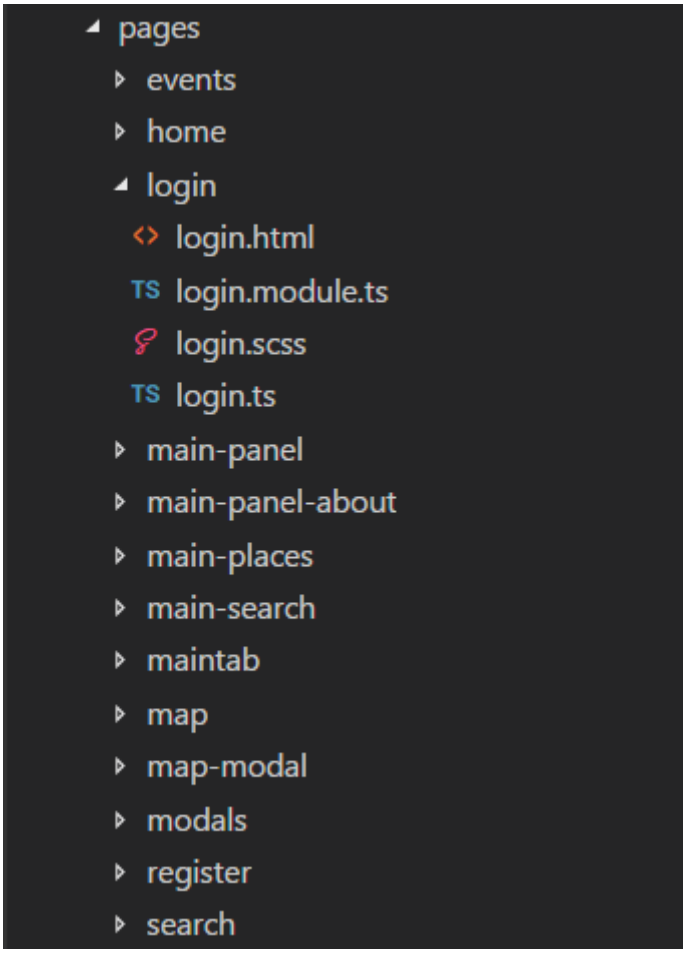

Figura 7.14: Estructura aplicación *Ionic* - *pages*

Para cada *page* tenemos:

- Un archivo HTML que contiene la vista de la página.
- Un archivo SCSS que contiene los estilos de la vista.
- Dos archivos TypeScript. Uno que actúa de controlador y otro que sirve para la configuración de las vistas.

7.2. Implementación capa modelo

7.2.1. Persistencia

Para la implementación de la persistencia de la aplicación se han seguido las definiciones especificadas en la API de persistencia para la plataforma Java, conocida comúnmente como JPA. Esta API nos proporciona una gestión de los datos relacionales en nuestra aplicación Java.

Puesto que JPA solo define un conjunto de interfaces, es necesaria una implementación de las mismas, que en este caso, será gracias al framework de Hibernate. Hibernate no solo implementa las especificaciones de JPA sino que también incorpora funcionalidades propias.

Implementación clases persistentes

A continuación se mostrará un ejemplo de una entidad con las anotaciones definidas por JPA.

```
@Entity
@Table(name = "ROUTE_DAY")
public class JpaRouteDay implements RouteDay<JpaStay> {

    @EmbeddedId
    private RouteDayPK diaPK;

    @Column(name = "START_TIME")
    private Long startTime;

    @Column(name = "REAL_TIME_DATA")
    @Lob
    private String realTimeData;

    @OneToMany(mappedBy = "day", fetch = FetchType.EAGER, orphanRemoval = true,
            cascade = CascadeType.REMOVE)
    @OrderBy("order ASC")
    private List<JpaStay> stays;

    @JoinColumn(name = "ROUTE_X_ROUTE", referencedColumnName = "X_ROUTE",
            insertable = false, updatable = false)
    @ManyToOne
    private JpaRoute route;
```

Figura 7.15: Implementación clases persistentes

A continuación se explican, brevemente, las anotaciones JPA empleadas.

- **@Entity.** Declara una clase POJO como entidad persistente.
- **@Table.** Especifica la tabla empleada para la clase marcada como @Entity.
- **@EmbeddedId.** Denota una clave primaria compuesta que es una clase marcada por @Embeddable. Se aplica sobre un campo o propiedad persistente de la clase.
- **@Column.** Anotación utilizada para *mapear* una columna con la propiedad o campo de la clase. El atributo *name* permite especificar el nombre de la columna.
- **@Lob.** Determina que la propiedad debe persistir como una estructura que permita almacenar gran cantidad de información.
- **@OneToMany.** Define una relación de uno a muchos entre dos entidades.
- **@OrderBy.** Especifica el orden de los elementos de la colección cuando son recuperados.
- **@JoinColumn.** Determina una columna para unir una entidad de asociación.
- **@ManyToOne.** Define una relación muchos a uno.

Implementación DAOs

Como se ha comentado en el apartado de *Diseño*, se ha elaborado un DAO genérico que implementa un conjunto de funcionalidades básicas. Para las consultas a la base de datos, realizadas por el DAO, se ha utilizado *JPA Criteria API*, que nos permite definir estas consultas mediante la creación de una serie de objetos Java. A continuación, se muestra un ejemplo de uso de esta API.

```java
public List<E> getListByField(String fieldName, Object value,
        OrderingType orderingType, String orderingField,
        Integer index, Integer count) {
    CriteriaBuilder criteriaBuilder = this.entityManager.getCriteriaBuilder();
    CriteriaQuery<E> criteriaQuery = criteriaBuilder.createQuery(this.getEntityClass());
    Root<E> root = criteriaQuery.from(this.getEntityClass());

    criteriaQuery.where(criteriaBuilder.equal(root.get(fieldName), value));
    this.setOrder(criteriaBuilder, criteriaQuery, root, orderingType, orderingField);
    TypedQuery<E> typedQuery = this.entityManager.createQuery(criteriaQuery);
    this.setPagination(typedQuery, index, count);
    List<E> result = typedQuery.getResultList();
    return result;
}
```

Figura 7.16: Implementación consultas DAO

En este ejemplo, se define el método *getListByField(...)* que nos devolverá la lista de objetos que cumplan el filtro aplicado. Como se puede observar en la imagen, la consulta a la base de datos se ha realizado mediante la utilización de las clases ofrecidas por la API de Criteria, delegando en ellas la construcción de la *query* necesaria.

Gestión de la transaccionalidad

La lógica de negocio de la aplicación hace uso de los DAOs creados anteriormente y que realizan diferentes operaciones sobre la base de datos. Debido a que en un mismo caso de uso pueden realizar diferentes operaciones de los DAOs, es necesario crear una transacción que nos permita ejecutar todas esas acciones contra la base de datos en bloque y que, en caso de que alguna produjese un error, poder deshacer los cambios ocasionados por las demás.

Para ello, será necesario anotar estos métodos con la anotación *@Transactional*. Con esta anotación, se empezará una transacción antes de la primera línea del método y se terminará justo después de la última, permitiendo ejecutar todo lo que esté dentro del método dentro de una misma transacción.

```java
@Transactional
public S updateStay(S dayPlace) throws UnUpdateableRouteException {
    this.checkRouteStayUpdateable((R) dayPlace.getDay().getRoute());
    this.stayDao.update(dayPlace);
    return dayPlace;
}
```

Figura 7.17: Implementación ejemplo transaccionalidad

En la figura 7.17, se define el método *updateStay* marcado con la anotación comentada anteriormente, de forma que, todas las operaciones ejecutadas dentro del método se encontrarán dentro de una misma transacción.

7.3. Implementación capa servicios

La aplicación está formada por una serie de servicios web REST que ofrecen las funcionalidades de la capa modelo remotamente. Como se había comentado, estos servicios han sido elaborados siguiendo las especificaciones establecidas por la API JAX-RS.

A continuación, se muestra un ejemplo de un recurso web, donde se explica, brevemente, la función de cada anotación.

```java
@Path("route")
@Secured({ Role.USER })
public interface RouteResource extends Serializable {

    @GET
    @Path("/explore")
    @Produces(MediaType.APPLICATION_JSON)
    public List<RouteDto> explore(@DefaultValue("") @QueryParam("city") String city,
            @DefaultValue("") @QueryParam("state") String state,
            @DefaultValue("") @QueryParam("numDays") String numDays,
            @DefaultValue("") @QueryParam("maxDistance") String maxDistance,
            @DefaultValue("") @QueryParam("maxDuration") String maxDuration,
            @DefaultValue("") @QueryParam("index") String index,
            @DefaultValue("") @QueryParam("count") String count)
            throws InputValidationException;

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    @Path("/user/owner")
    public List<RouteDto> getOwnRoutes(@DefaultValue("") @QueryParam("filterBy") String filter,
            @DefaultValue("") @QueryParam("value") String value,
            @DefaultValue("") @QueryParam("index") String index,
            @DefaultValue("") @QueryParam("count") String count) throws InputValidationException;
```

Figura 7.18: Implementación ejemplo transaccionalidad

- **@Path.** Identifica la URI en la que responderá un método o recurso de la clase. Toma un valor relativo, siendo la URI base el *path* de la aplicación.

- **@Secured.** Anotación personalizada. Tiene el objetivo de marcar el recurso como seguro de manera que se aplique un filtro de autenticación en cada petición.

- **@GET.** Indica que el método anotado responde a solicitudes HTTP GET.

- **@Produces.** Define el *mediatype* que pueden generar los métodos. Análogamente, existe la anotación @Consumes, que define el *mediatype* que acepta el método.

- **@QueryParam.** Vincula el valor del parámetro HTTP a uno del método.

- **@DefaultValue.** Especifica un valor predeterminado para @QueryParam.

7.4. Implementación autenticación y autorización

7.4.1. Implementación autenticación

Se ha seguido un proceso de autenticación sin estado con el uso de *Tokens*. Nuestro modelo ofrece una API REST *stateless*, es decir, sin información de estado, donde los tokens son almacenados en lado del cliente, permitiendo que nuestra aplicación sea totalmente escalable.

Para la implementación, se ha seguido el estándar JWT (JSON Web Token) que define una forma compacta y autónoma de transmitir de forma segura información entre dos partes mediante un objeto JSON.

```
public TokenDto authenticate(UsuarioDto usuarioDto)
        throws ForbiddenException {

    U user = this.userService.authenticate(
            usuarioDto.getUsername(),
            usuarioDto.getPassword());
    if (user.getRole().toString() != null) {
        TokenDto token = new TokenDto();
        token.setToken(this.createJwtToken(
                usuarioDto.getUsername(),
                user.getRole().toString()));
        token.setRefreshToken(user.getToken());
        token.setRole(user.getRole().toString());
        token.setName(usuarioDto.getUsername());
        return token;
    } else {
        throw new ForbiddenException();
    }
}
```

Figura 7.19: Implementación autenticación

En la figura anterior, se muestra la funcionalidad *autenticate*. En este método, se delega al servicio *userService* la comprobación de las credenciales ofrecidas por el usuario. Si la comprobación es correcta se procede a la creación del token que es devuelto como respuesta al usuario.

```
private String createJwtToken(String username,
        String role) {
    Calendar cal = Calendar.getInstance();
    cal.add(Calendar.HOUR_OF_DAY, 1);
    return this.tokenService.createToken(username,
            role, cal.getTimeInMillis());
}
```

Figura 7.20: Implementación creación token

La creación del token se delega a la librería Java JWT, a través del servicio *tokenService*, que es la encargada de elaborar las tres partes fundamentales de un JSON Web Token, que son: la cabecera, el *payload* y la firma. El payload puede estar

formado por diferentes atributos, en este caso, lo forman: el nombre de usuario, el rol del usuario y la fecha de expiración del token, fijada en una hora.

```
String authorizationHeader = requestContext.getHeaderString(HttpHeaders
        .AUTHORIZATION);
boolean abort = false;

if ((authorizationHeader == null) || !authorizationHeader
        .startsWith(AUTHENTICATION_SCHEME)) {
    requestContext.abortWith(Response.status(Response.Status.UNAUTHORIZED)
            .build());
} else {
    String token = authorizationHeader.substring(AUTHENTICATION_SCHEME
            .length()).trim();
    Role userRole = null;
    try {
        userRole = Role.valueOf(this.tokenService.validateToken(token));
    } catch (UnsupportedJwtException | MalformedJwtException | SignatureException
            | IllegalArgumentException e) {
        requestContext.abortWith(Response.status(Response.Status.FORBIDDEN)
                .build());
    } catch (ExpiredJwtException ex) {
        abort = true;
        ExpiredTokenExceptionDto exp = new ExpiredTokenExceptionDto("ExpiredJwtToken",
                "Error validating access token.");
        requestContext.abortWith(Response.status(Response.Status.UNAUTHORIZED)
                .entity(exp).build());
    }
```

Figura 7.21: Implementación filtro autenticación

Por su parte, se elabora un filtro para la comprobación de la autenticación, comprobando que las peticiones incluyan la cabecera *AUTHORIZATION* con el valor *'Bearer + token'*. En el ejemplo anterior, de un trozo del filtro, podemos observar como se comprobará la existencia de dicha cabecera y se realizará la evaluación del token.

La evaluación del token se delegará al servicio *tokenService* que obtendrá cada uno de los componentes que forman el token.

7.4.2. Implementación autorización

Implementación autorización por roles

El sistema permitirá la realización de diferentes acciones en función del rol del usuario que solicite la acción. Para ello, en el filtro de autenticación, una vez validado

el token, se obtiene del payload el rol especificado y se comprobará si este está incluido en la anotación *@Secured*, comentada anteriormente, en el recurso web solicitado. Como se especificó, cada recurso web definido en la capa de servicios, incluye una anotación en la que se indica el rol necesario para realizar dichas operaciones.

```
@Path("route")
@Secured({ Role.USER })
public interface RouteResource extends Serializable {
```

Figura 7.22: Implementación autorización recurso web - roles

En el filtro se extraerán los roles especificados en la cabecera *@Secured* de cada recurso y se comprobarán con el rol incluido en el token. De tal manera, que se permitirá ejecutar la acción si ambos roles coinciden. Para el ejemplo anterior, sobre el recurso web *route*, solo se permitirán acciones autenticadas por usuarios cuyo rol sea *USER*.

```
List<Role> classRoles = this.extractRoles(this.resourceInfo
        .getResourceClass());
List<Role> methodRoles = this.extractRoles(this.resourceInfo
        .getResourceMethod());
if (!methodRoles.contains(userRole)) {
    if (!classRoles.contains(userRole)) {
        requestContext.abortWith(Response.status(
            Response.Status.FORBIDDEN).build());
    }
}
```

Figura 7.23: Implementación filtro autorización - roles

Con el filtro anterior conseguimos acciones autorizadas en función del rol especificado.

Implementación autorización basada en recursos

Los roles son solo una parte de la autorización. Para evitar que un usuario acceda de forma no autorizada a un recurso creado por otro usuario, se utilizará el framework

Spring Security. Con Spring Security se pueden definir expresiones en los métodos que permitan comprobar la autorización sobre cada recurso accedido.

```
@PreAuthorize("hasRole('ROLE_ADMIN') or "
        + "@mySecurityService.hasUserPermission("
        + "authentication, #id)")
void delete(Long id) throws InstanceNotFoundException;

@PostAuthorize("hasRole('ROLE_ADMIN') or "
        + "returnObject.username == "
        + "authentication.principal.username")
U getById(Long id) throws InstanceNotFoundException;
```

Figura 7.24: Implementación autorización servicios - basada en recursos

Para que estás anotaciones puedan ser utilizadas es necesario crear un objeto de autorización propio de Spring Security, donde se indica el nombre del usuario y el rol que tiene. Este objeto, se crea en el filtro de autenticación.

A continuación, se detallará cada una de las anotaciones y expresiones indicadas.

- **@PreAuthorize.** Ejecuta la autorización antes de realizar el método. En la anotación se especifica los criterios de autorización.
  - Que el usuario que realice la acción tenga el rol de *ADMIN*.
  - O que el usuario tenga los permisos necesarios. En este caso, se hace uso de una clase externa de utilidad, donde se comprobará que el recurso concreto al que se quiere acceder esté autorizado para el usuario que realiza la acción.
- **@PostAuthorize.** Ejecuta la autorización después de realizar el método. Se especifican los siguientes criterios:
  - Que el usuario que realice la acción tenga el rol de *ADMIN*.
  - O que el objeto de devuelto por el método pueda ser accedido por el usuario que realiza la acción.

7.5. Implementación cliente web

7.5.1. Implementación acceso a servicios

```
@PostConstruct
public void obtainService() {
    this.client = new ResteasyClientBuilder().build();
    ResteasyWebTarget target = this.client.target(BASE_URI);
    HeaderFilter hf = new HeaderFilter();
    this.client.register(hf);
    this.service = target.proxy(this.getServiceClass());
}

public S getService(String token) {
    Set<Object> registered = this.client.getConfiguration().getInstances();
    Iterator<Object> iterator = registered.iterator();
    while (iterator.hasNext()) {
        Object object = iterator.next();
        if (object instanceof HeaderFilter) {
            HeaderFilter headerFilter = (HeaderFilter) object;
            headerFilter.setToken(token);
        }
    }
    return this.service;
}
```

Figura 7.25: Implementación cliente web - Ejemplo acceso a servicios

La capa de acceso a los servicios implementada para el cliente web se elaborará mediante la implementación RESTEasy de la API JAX-RS Client. Con esta implementación se hace uso de *proxy framework*, que actúa de manera opuesta a la especificación JAX-RS indicada en la capa de servicios. En vez de utilizar las anotaciones para determinar las peticiones entrantes al servicio REST, el cliente utiliza dichas anotaciones para construir la petición HTTP necesaria para enviar al servidor.

En el ejemplo, el método *obtainService* se encargar de construir el cliente de RESTEasy. En él, se registra el filtro *HeaderFilter*, utilizado para incluir el token de acceso en las peticiones. Una vez creado el cliente, se especifica a través del *proxy*, la clase del servicio, que está anotada con la especificación de JAX-RS, para elaborar las peticiones HTTP necesarias.

El método *getService* permite obtener el servicio indicando el token que se desea

utilizar. Este método se encargará de indicar en el filtro, el token a incluir en la cabecera HTTP a la hora de elaborar la petición.

7.5.2. Implementación controlador

Para el cliente web se definen una serie de controladores encargados de recibir las peticiones del cliente final y ofrecerles los datos. Estos controladores, han sido implementados mediante el framework Spring MVC. A continuación, se detalla el ejemplo de un controlador.

```
@Controller
@RequestMapping("/login")
@SessionAttributes({ "token" })
public class LoginController {

    @Autowired
    ClientUser clientUser;

    @RequestMapping(method = RequestMethod.POST)
    public String do_login(HttpServletRequest request,
            @ModelAttribute("userDto") UsuarioDto userDto, Model model,
            RedirectAttributes attributes) {

        TokenDto token = null;
        try {
            token = this.clientUser.getService(null).authenticate(userDto);

        } catch (Exception e) {
            model.addAttribute("error", "true");
            return "login";
        }
        ...
```

Figura 7.26: Implementación cliente web - Ejemplo controlador

Este controlador se encarga de realizar las tareas de autenticación en el cliente web. Se puede observar en la imagen, las anotaciones utilizadas para definir este controlador.

- **@Controller.** Registra la clase como controlador.
- **@RequestMapping.** Anotación utilizada para asociar la clase o un método con una petición HTTP.

- **@SessionAttributes.** Define los atributos que deberán ser almacenados, transparentemente, en la sesión del navegador o en otro lugar de almacenamiento.
- **@SessionAttributes.** Define los atributos que deberán ser almacenados, transparentemente, en la sesión del navegador o en otro lugar de almacenamiento.

El método *do_login* recibe las peticiones POST con los datos del usuario a autenticar. El controlador hace uso del cliente creado en la capa de acceso a los servicios para realizar la petición al modelo. En función de la respuesta del modelo, el controlador se encargará de presentar una vista o otra al cliente.

7.5.3. Implementación vistas

Plantillas

El desarrollo de las vistas en el servidor web ha sido elaborado mediante Thymeleaf, un motor de plantillas XML/XHTML/HTML5. Thymeleaf está completamente integrado con Spring MVC ofreciendo una sintaxis sencilla, permitiendo la creación de plantillas de una manera elegante y con un código bien formateado.

```
<div class="row">
    <div class="col-sm-6 col-md-4" th:each="myroute : ${myroutes}">
        <div class="thumbnail">
            <img class="myroute-img" th:if="${myroute.photo} != null"
                th:src="${myroute.photo}" alt=".."
            >
            <div class="caption">
                <h3 th:text="${myroute.city}"></h3>
                <p th:if="${myroute.startDate} != null"
                    th:text="${#dates.format(myroute.startDate, 'dd-MM-yyyy')} +
                    ' - ' + ${#dates.format(myroute.endDate, 'dd-MM-yyyy')}"
                ></p>
                <p th:if="${myroute.startDate} == null">Fechas sin especificar</p>

                <div th:if="${myroute.startDate} != null" class="row">
                    <div class="col-md-6">
                    <p th:text="'Nº de Días: ' + ${myroute.numDays}"></p>
                    </div>
                    <div class="col-md-6">
                    <p th:text="'Nº de Visitas: ' + ${myroute.numPlaces}"></p>
                    </div>
                </div>
                <div th:if="${myroute.numDays} > 0" class="row">
                 ...
```

Figura 7.27: Implementación cliente web - Ejemplo vistas

En la figura anterior se muestra un ejemplo de una plantilla HTML creada con Thymeleaf. El controlador es el encargado obtener los datos y agregar los atributos necesarios a la vista a través de la clase *Model* de Spring MVC. De esta manera, en la plantilla Thymeleaf podremos acceder a estos atributos mediante la sintaxis *${nombre_del_atributo}*. Con expresiones como *th:each*, para iterar una lista u objeto; *th:if*, para comparar atributos o *th:text*, para mostrar un atributo, podremos, fácilmente, manejar los atributos ofrecidos por el controlador en nuestras vistas.

AJAX

El uso de la ténica de desarrollo AJAX nos permite mantener una comunicación asíncrona entre el cliente y el servidor web. Gracias a esta característica, podremos hacer que el cliente realice cambios sobre las páginas sin necesidad de recargarlas, mejorando la interactividad en la aplicación.

Thymeleaf Fragments son unos bloques de plantillas HTML que podemos renderizar directamente mediante AJAX. A continuación se mostrará un ejemplo de un *fragment*.

```
<div th:fragment="header">
    <nav class="navbar navbar-inverse">
       <div class="container">
           <div class="navbar-header">
               <a class="navbar-brand">TFG - Aplicación</a>
           </div>
           <div id="navbar" class="collapse navbar-collapse">
               <ul id="navbarCustom" class="nav navbar-nav">
               </ul>
               <ul class="nav navbar-nav navbar-right">
               </ul>
           </div>
        </div>
    </nav>
</div>
```

Figura 7.28: Implementación cliente web - Ejemplo fragments

En esta figura se muestra la definición de un fragment. En este caso se define un bloque HTML que incluye un diseño de una cabecera. Este fragment puede ser

incrustado en otra plantilla Thymeleaf o ser cargado mediante AJAX. A continuación, se mostrará una figura en la que se podrá ver como se carga un fragment en una plantilla HTML mediante una petición AJAX.

```
var loadMyRoutes = function (status) {
    $("#selector-pending").removeClass("active");
    $("#selector-in_progress").removeClass("active");
    $("#selector-completed").removeClass("active");
    $("#myroutes-content").hide();
    $("#myroutes-loader").show();
    $("#myroutes-content").load("/myroutes/ajax/" +
            "getownroutes?filter=state&value="
            + status, function() {
        $("#myroutes-loader").hide();
        $("#myroutes-content").show();
    });

    var text = "#selector-" + status.toLowerCase();
    $(text).addClass("active");
}
```

Figura 7.29: Implementación cliente web - Ejemplo AJAX

En este ejemplo, se define una función *loadMyRoutes*. Cada vez que esta función sea ejecutada, mediante el método AJAX: *load*, de jQuery, se realizará una petición al servidor consultando las rutas propias de un usuario. El controlador será el encargado de recibir esa petición y devolverá, cuando obtenga los datos, un fragment con los datos de las rutas que serán incorporados en el elemento *#myroutes-content*.

7.6. Implementación cliente móvil

El cliente móvil ha sido desarrollado con el SDK Ionic, construido sobre Angular y utilizando tecnologías web, como HTML o CSS.

7.6.1. Implementación acceso a servicios

```
export class RouteServiceProvider {

  constructor(private http: Http, private authService: AuthServiceProvider) { }
  access: any;


  getUrl() {
    return HTTP_PROTOCOL + global.SERVER_IP + ':' + SERVER_PORT + '/rest/';
  }

  getHeaders() {
    let headers = new Headers();
    headers.append('Authorization', 'Bearer ' + this.authService.getUserInfo().token);
    let options = new RequestOptions({ headers: headers });
    return options;
  }

  getOwnRoutes(state: String) {
    let url = this.getUrl() + 'route/user/owner?filterBy=state&value=' + state;
    return this.http.get(url, this.getHeaders());
  }
```

Figura 7.30: Implementación cliente móvil - Ejemplo acceso a servicios

Las clases encargadas de acceder a los servicios del modelo hacen uso del módulo *http* original de Angular. En la figura anterior, se muestra como se construye la URL de destino y se ejecuta la petición HTTP necesaria. El método *getHeaders* nos permite crear las cabeceras de la petición e incluir el token de acceso necesario para autenticar dicha petición.

7.6.2. Implementación vistas y controladores

Como ya se comentó anteriormente, el cliente móvil sigue un diseño Modelo-Vista-VistaModelo (MVVM). En este caso se utilizará la palabra 'Controlador' para referirnos al elemento Vista-Modelo.

Los controladores inyectan un objeto *$scope* en las vistas. Las propiedades de este objeto estarán disponibles en la vista permitiendo que esta se actualice automáticamente a medida que se modifican los valores del objeto *$scope*. De esta forma se produce un enlace bidireccional de datos.

```
export class MainPlacesPage {

  private lat = "";
  private lng = "";
  private radius = 300;
  private section = "topPicks";
  private query = "";
  private query2 = "";
  private limit = 5;
  private sortByDistance = 0;
  private price = "";
  private photo = false;
  ...
```

Figura 7.31: Implementación cliente móvil - Ejemplo controlador

```
<ion-item class="item-md-pers">
  <ion-label floating>Longitud</ion-label>
  <ion-input color="foursquare2" type="text"
  [ngModel]="lng">
  </ion-input>
</ion-item>

<ion-item class="item-md-pers-range">
  <ion-label *ngIf="radius != 0" class="label-range">
    Radio Búsqueda: {{ radius }} metros</ion-label>
  <ion-label *ngIf="radius == 0" class="label-range">
    Radio Búsqueda: Sin especificar</ion-label>
</ion-item>

<ion-range color="foursquare2" class="filterRange"
min="0" max="1000" step="100" snaps="true"
pin="true" [(ngModel)]="radius">
  <ion-label range-left>N/A</ion-label>
  <ion-label range-right>1000</ion-label>
</ion-range>
```

Figura 7.32: Implementación cliente móvil - Ejemplo vista

De las dos figuras mostradas, la primera hace referencia a la definición de un controlador. En él, se definen el conjunto de variables *lat, lng, radius,...*, entre otras. Estas variables son inyectadas en la vista, de manera como se muestra en la figura 7.32. En la vista, se accede a estas variables del controlador mediante la directiva Angular *ngModel*.

Si se modifica el valor de estas variables en la vista, también se modificará en el controlador, permitiendo un intercambio dinámico de información entre los dos componentes.

Capítulo 8

PRUEBAS

En este apartado se comentarán las pruebas realizas para verificar el correcto funcionamiento de la aplicación.

8.1. Pruebas de integración

Estas pruebas son realizadas para probar la correcta interacción entre dos o más unidades software. Para que estas pruebas sean de validez, es necesario comprobar el correcto funcionamiento de cada una de la unidad software implicada. Para ello, son necesarias las pruebas unitarias, que en nuestro caso, no se automatizaron debido a la simplicidad de los DAOs definidos.

Para cada uno de los servicios definidos, se han realizado pruebas automatizadas con JUnit, exceptuando los servicios externos (Foursquare y Google) que fueron probados de forma manual. A continuación se muestra un el número total de pruebas automatizadas realizadas.

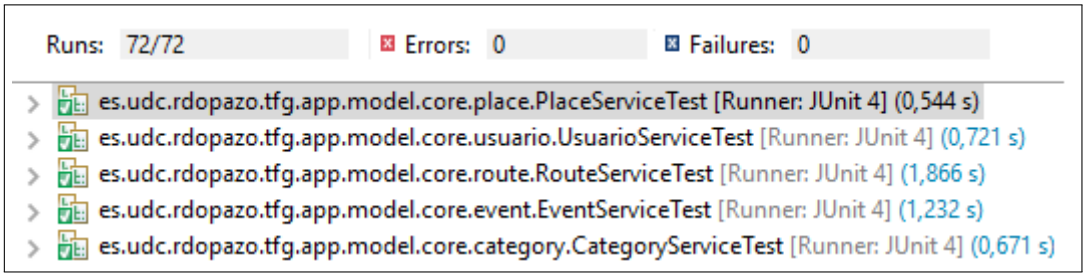

Figura 8.1: Diagrama pruebas ejecutadas

Se han realizado un total de 72 pruebas sobre los diferentes casos de uso de cada servicio. En la figura siguiente, se podrá observar un ejemplo de cómo se han realizado estas pruebas.

```
@Transactional
@Test
public void testGetAllEventPlaces()
        throws InstanceNotFoundException {
    try {
        Long event1 = this.insertValidEvent();
        Long day1 = this.insertValidEventDay(
                this.service.getEventById(event1));
        Long day2 = this.insertValidEventDay(
                this.service.getEventById(event1));
        Long eventplace1 = this.insertValidEventPlace(
                this.service.getEventDayById(event1, day1));
        Long eventplace2 = this.insertValidEventPlace(
                this.service.getEventDayById(event1, day1));
        Long eventplace3 = this.insertValidEventPlace(
                this.service.getEventDayById(event1, day2));

        List<EP> eventplaces = this.service.
                getAllEventPlaces(null, null);

        assertEquals(eventplaces.size(), 3);
        eventplaces = this.service.getAllEventPlaces(0, 1);
        assertEquals(eventplaces.size(), 1);
    } finally {
        this.eventPlaceDao.clearTable();
        this.eventDayDao.clearTable();
        this.eventDao.clearTable();
    }
}
```

Figura 8.2: Diagrama ejemplo - prueba JUnit

A mayores, también se comprobó la cobertura de las pruebas realizas sobre el código de la aplicación, obteniendo los siguiente resultados.

Element	Cov...	Cove...	Miss...	Tota...
TFG - app-model-core	77,2 %	5.327	1.570	6.897
src/test/java	82,9 %	3.934	809	4.743
src/main/java	64,7 %	1.393	761	2.154
es.udc.rdopazo.tfg.app.model.core.event.impl	100,0 ...	342	0	342
es.udc.rdopazo.tfg.app.model.core.place.impl	100,0 ...	71	0	71
es.udc.rdopazo.tfg.app.model.core.category.impl	97,1 %	134	4	138
es.udc.rdopazo.tfg.app.model.core.usuario.impl	85,4 %	246	42	288
es.udc.rdopazo.tfg.app.model.core.route.impl	72,3 %	539	206	745
es.udc.rdopazo.tfg.app.model.core.util	61,1 %	22	14	36
es.udc.rdopazo.tfg.app.model.core.util.security	24,2 %	36	113	149
es.udc.rdopazo.tfg.app.model.core.externalservice.impl	0,8 %	3	382	385

Figura 8.3: Diagrama cobertura ejecución pruebas

Cabe destacar que los servicios externos no fueron incluidos en las pruebas automatizadas así como las clases del sub-paquete *security*, y algunos casos de uso relacionados con usuarios y rutas que requieren una perspectiva global en la que se controle la autenciación de usuarios y el uso de los datos obtenidos de las fuentes externas.

A pesar de estas excepciones, las pruebas cubren más del 77 % de las líneas de código del módulo *model-core*.

8.2. Pruebas de sistema

El objetivo de estas pruebas es comprobar el correcto funcionamiento del sistema software completo e integrado. Para ello, se realizaron el mayor número de pruebas posibles,

Se realizaron pruebas de sistema para cada uno de los grandes bloques de la aplicación global. Por una parte, se realizaron pruebas sobre el servicio REST mediante el uso de herramientas de API testing, como es Postman. Posteriormente, se probaron los dos clientes, web y móvil, independientemente, comprobándose el correcto funcionamiento de cada uno de ellos.

Capítulo 9

CONCLUSIONES

Capítulo 10

TRABAJO FUTURO

Capítulo 11

BIBLIOGRAFÍA

Apéndice A

Diagramas

A.0.1. Diagrama Entidad Relación

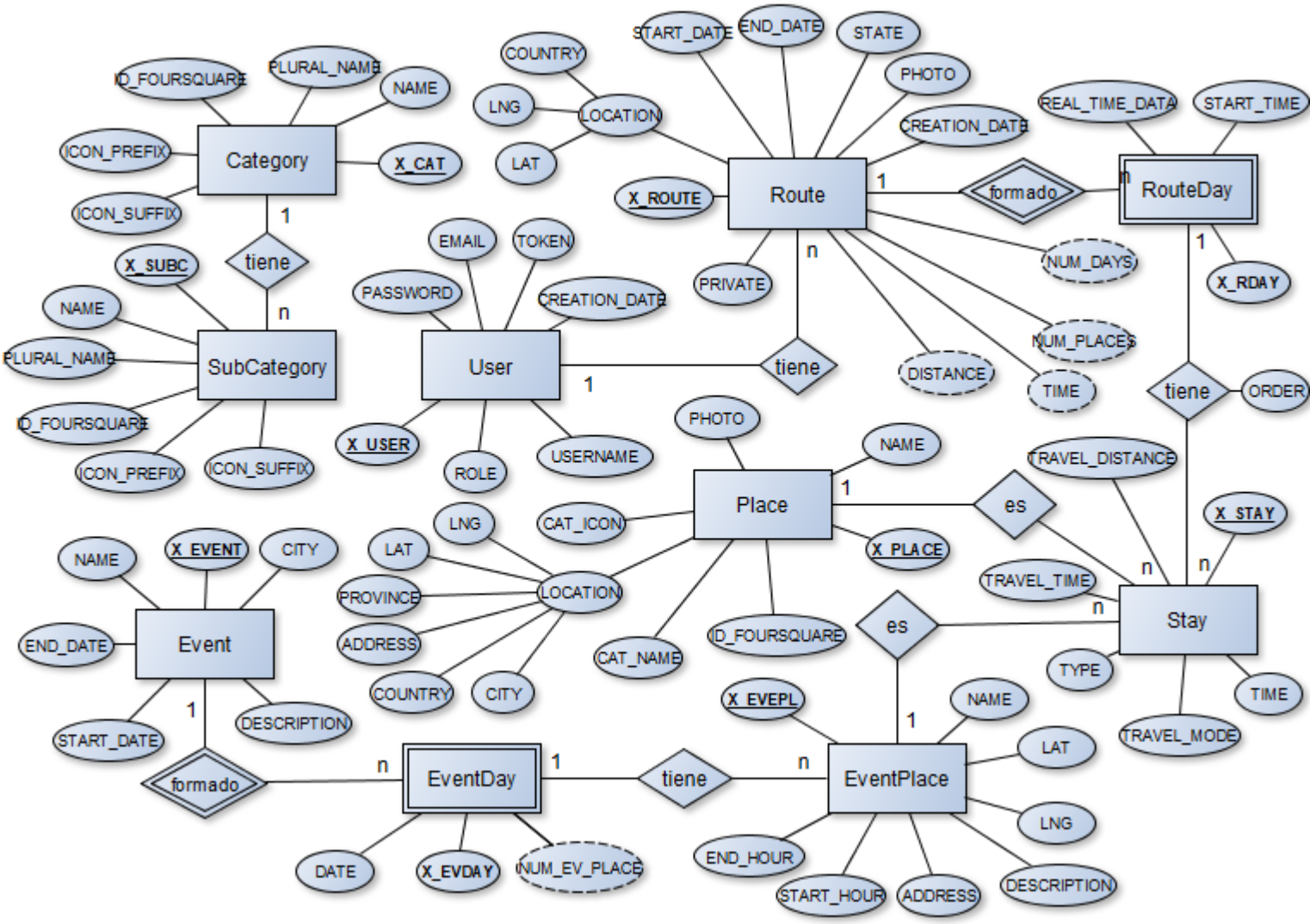

Figura A.1: Diagrama ER con atributos

Apéndice B

MANUAL DE USUARIO

B.1. Manual de usario aplicación móvil