Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABAJO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN TECNOLOXÍAS DA INFORMACIÓN

Sistema de compra y consumo de tíckets para viajes ferroviarios

Estudiante: Cristina Busto Noya
Dirección: Fernando Bellas Permuy

A Coruña, setembro de 2019.

A mi madre Mariné y a mi pareja Óscar, por vuestra continua fé en mi.

Agradecimientos

En primer lugar, quiero dar las gracias a mi tutor Fernando Bellas, por ayudarme a hacer posible este proyecto y asesorarme en todo momento.

A mi madre, que me ha dado fuerzas durante todo el camino, cuando yo las perdía. Gracias por estar ahí siempre que me bloqueo y necesito impulso para continuar. Ella siempre confió en mi talento para hacer todo lo que me propusiese.

A Óscar, el mejor compañero de vida. Gracias por tu paciencia, tu apoyo incondicional y por enseñarme a creer en mí misma. Él me ha motivado a comenzar esta aventura confiando en mis capacidades, cuando yo no lo hacía.

A mis amigos de universidad, Sandra, Laura, Jose y David. Esta etapa no hubiese sido lo mismo sin vosotros.

Resumen

Uno de los medios de transporte terrestres más utilizados hoy en día, es el ferrocarril. Es un medio muy cómodo, tanto para viajes ocasionales de larga distancia, como para viajes diarios a nuestra ciudad de trabajo. Por eso, **FlashTicket**, la aplicación desarrollada en este TFG, es un servicio que surge de la necesidad de proprocionar más comodidades a los viajeros para adquirir los tickets.

El objetivo de este TFG, es desarrollar un sistema de gestión de tickets, para una compañía ferroviaria. De esta manera, los viajeros tendrán la posibilidad de comprar billetes electrónicamente en cualquier momento, a través de una plataforma digital. Este sistema también permite al viajero adquirir abonos. Un abono, es un ticket de temporada que permite viajar entre dos estaciones concretas, durante un periodo de tiempo limitado. Lo único que el usuario tiene que hacer es asociar (formalizar) nuevos viajes a ese abono sin ningún coste adicional.

Además, el sistema proporciona a los revisores la posibilidad de validar los tickets de los viajeros.

El sistema desarrollado en este TFG, se compone de una aplicación multiplataforma para los viajeros, una aplicación móvil para los revisores y un servicio REST con la lógica de negocio para ambas aplicaciones.

Cabe destacar, que queda fuera del alcance de este proyecto, todo lo relacionado con la gestión de la base de datos de estaciones, trenes, etc.

Palabras clave:

- Servicio REST
- Spring Boot
- Aplicación Web
- Aplicación movil
- Java
- Maven
- Mysql
- Ticket
- Abonos
- Tren
- Viaje
- Viajero
- Angular
- Ionic
- Scanner QR
- Code QR

Keywords:

- REST Server
- Spring Boot
- Web application
- Mobile application
- Java
- Maven
- Mysql
- Ticket
- Abonos
- Train
- Trip
- Voyager
- Angular
- Ionic
- Ticket
- Scanner QR
- Code QR

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introdución

1.1 Motivación

La idea de este trabajo de fin de grado, surge de mi experiencia como usuaria asidua del ferrocarril durante cuatro años. Mi rutina consistía en la previa adquisición de un abono, en mi caso de durabilidad mensual, con el que podía desplazarme diariamente desde mi residencia habitual (Santiago) hasta la universidad, ubicada en A Coruña.

El abono consiste en una tarjeta de papel en la que, por la cara frontal, aparecen los datos del viajero, y en la cara posterior, se registran los viajes asociados al abono. Para poder viajar en un tren, primero debes registrar el viaje asociado a ese tren. Este registro se lleva a cabo a través de unas máquinas situadas en las estaciones de ferrocarril.

Cada vez que realizaba un viaje en tren, unos minutos antes de su partida me acercaba a una de las máquinas anteriormente mencionadas, e insertaba el abono. A continuación, seleccionaba en la pantalla el viaje que deseaba registrar. En ese momento, la máquina se encarga de registrar dicho viaje, imprimiendo la información del mismo en la parte posterior del abono.

Cuando el revisor necesita verificar la validez de un abono, tiene que comprobar las características del mismo y además confirmar manualmente que el viaje hs sido formalizado correctamente y por ende, sus características están impresas en la parte posterior del mismo.

La idea del TFG, es desarrollar un sistema que automatice, tanto el proceso de compra de tickets, como su formalización y verificación por parte del revisor.

Se debe puntualizar, que la acción de formalizar un viaje, es el hecho de registrar o asociar dicho viaje en el abono correspondiente. Será un termino que se usará durante toda la memoria a partir de este momento.

1.2 Objetivos

Los objetivos descritos a continuación son una descomposición del objetivo principal descrito anteriormente.

- **Consultar los horarios de viajes:** Los usuarios de la aplicación, podrán consultar los horarios de los viajes, según diferentes criterios (origen, destino, fecha). De esta manera obtendrán un listado de los viajes disponibles.

- **Adquirir tíckets:**

  - **Tíckets de un solo uso o "billete" :** La aplicación permite adquirir tickets de un solo uso (válidos únicamente para realizar un solo viaje), de forma rápida y sencilla. Simplemente, tendrán que seleccionar aquel viaje que mejor se adapte a sus necesidades. Además, tendrá la posibilidad de elegir el asiento y vagón, del tren en el que viajarán.

  - **Tíckets de más de un uso o "Abono" :** Los abonos, son tickets que te permiten realizar más de un viaje entre dos estaciones concretas, durante un periodo de tiempo concreto. Para obtener un ticket de más de un uso, el usuario tiene que cubrir un formulario, en el que principalmente tendrá que indicar el tipo de ticket que desea obtener, y las estaciones entre las cuales viajará durante ese periodo de tiempo. Además deberá indicar, el tipo de bono que desea obtener.

    En este proyecto existen dos tipos de Abono: **abono mensual**, que te permite realizar viajes durante un mes entero y **abonos 10**, que solo te permiten formalizar 10 viajes en total, siempre entre dos estaciones seleccionadas en la compra de dicho abono.

    Además existe un subtipo de abonos: **abonos libres**, que te permiten viajar en cualquier tipo de tren media distancia (AVANT, MD, REGIONAL) y **abonos regionales**, que sólo te permiten desplazarte en trenes de tipo REGIONAL.

  Posteriormente, la aplicación se encargará de componer un código QR asociado a dicho ticket. Además, si el usuario lo desea, recibirá un correo electrónico con un PDF adjunto donde se encontrarán los datos del ticket. De esta manera, el usuario podrá decidir si desea llevar su ticket físicamente impreso o en su móvil.

- **Facilitar la formalización y cancelación de viajes:** Una vez que se haya adquirido un abono, se podrá formalizar nuevos viajes con él. Para ello, el usuario debe elegir el sentido del viaje y el asiento del tren, si lo desea. En ese momento, se registrará el viaje en el abono, y el usuario lo visualizará en los detalles del mismo. A mayores, para

facilitar la acción de formalización a los usuarios asiduos del ferrocarril, cada vez que el usuario registre un nuevo viaje, éste también será añadido a una lista de viajes más frecuentados. De esta manera el usuario tendrá la posibilidad de registrar de nuevo ese viaje simplemente seleccionándolo. En ese momento, el sistema formalizará un viaje con las mismas características para el día siguiente.

- **Facilitarle la validación de tickets al revisor:** Como ya explicamos anteriormente, el revisor suele verificar los tickets manualmente. Con nuestro sistema, el revisor simplemente tendrá que escanear el código QR del ticket y sabrá al momento si es válido o no. Además se le mostrará una serie de datos adicionales e interesantes para el revisor, como puede ser el DNI del usuario, el asiento, etc.

  Cabe destacar que a información del código QR, será firmada digitalmente, para evitar que la validación del mismo requiera conexión con el servicio.

En la figura Figura 1.1, podemos ver cual sería el flujo habitual del sistema. El viajero mediante la aplicación de viajeros vía web o móvil compra un ticket, lo que hace que el sistema genere un código QR asociado a dicho ticket. A continuación, el viajero descarga el código QR. Cuando se dirige al tren, el revisor con su aplicación móvil, se encarga de escanear el QR y verificar la validación del mismo.

Figura 1.1: Flujo habitual del sistema

1.3 Visión global del sistema:

La arquitectura del sistema está compuesta por tres capas:

- **Aplicaciones cliente :** Esta capa está directamente conectada con el usuario. Le presenta el sistema al usuario y éste interacciona con ella directamente. Por debajo, la capa de presentación, se comunica con el servicio REST, mediante peticiones HTTP.

- **Servicio REST :** En esta capa reside la lógica de negocio del sistema. Se comunica con las aplicaciones cliente, respondiendo a las peticiones HTTP que recibe. También se comunica con la capa modelo, para solicitar al Sitema de Gestión de Base de Datos operaciones de almacenamiento y/o recuperación de datos.

- **Capa de datos** En esta capa residen los datos que se usan en toda la aplicación y es la encargada de acceder a los mismos. Se comunica con la capa de negocio, respondiendo a sus peticiones.

Este tipo de arquitectura, se basa en el modelo **Cliente-Servidor** y tiene como objetivo el desacoplamiento de las partes. De esta manera, cada capa solo conoce a aquellas con las que se comunica, sin importarle la lógica interna de cada una, permitiendo la abstracción, escalabilidad, alto rendimiento y tolerancia a fallos

En la Figura 1.2, podemos ver un esquema que representa las capas comentadas anteriormente. Si nos fijamos en esa figura, podemos hacernos una idea de que frameworks se han usado en el desarrollo de cada capa. Para el desarrollo de las aplicaciones cliente se ha usado Angular e Ionic, lo que hace posible que puedan ser desplegadas como aplicaciones web o móvil gracias a Apache Cordova. Para el servicio REST se ha usado Spring Boot, Hibernate y JPA y para el SGBD se usa MySQL.

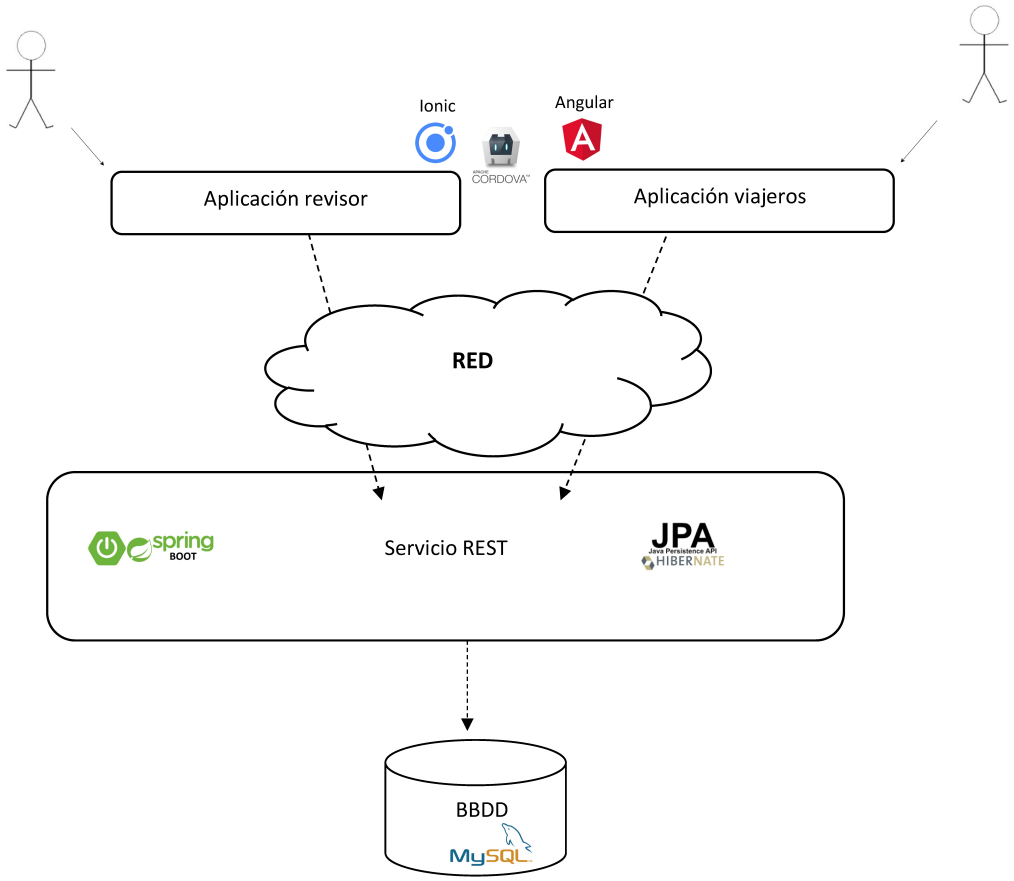


Figura 1.2: Arquitectura del sistema a alto nivel

Capítulo 2

Estado del arte

Una vez han sido descritos los objetivos del proyecto, se ha procedido a un estudio del mercado, en busca de aplicaciones o servicios similares.

2.1 Renfe

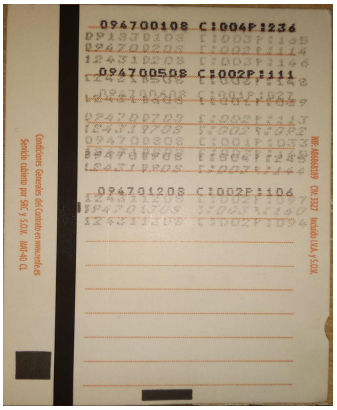
La principal plataforma considerada para el estudio de este proyecto ha sido la de **Renfe Operadora** [1]. Renfe, es la principal operadora ferroviaria de España, su objetivo principal es prestar servicios de transporte de viajeros y mercancías.

Renfe tiene un sistema de billetes automatizado a través de su web `http://www.renfe.com/` o de su aplicación móvil llamada "RenfeTicket", Figura 2.3. Esta aplicación te permite, consultar los horarios de los trenes y los viajes comprados entre otras cosas. Pero en cambio, aunque tiene una opción llamada "Abonos", no contempla los abonos de media distancia.

El sistema de gestión de Abonos de media distancia de renfe, consta de una tarjeta de papel impresa, con un código de barras y los datos asociados a ese mismo abono. Podemos ver un ejemplo, en la Figura 2.1a.

Para formalizar un viaje en el Abono, es necesario acercarse a una estación de ferrocarril de Renfe, e introducir la tarjeta de papel en una de las "máquinas formalizadoras". A continuación, la máquina te muestra los posibles viajes durante ese día y cuando seleccionas uno, lo registra en el abono imprimiendo las características del viaje en la cara inversa del abono. En la Figura 2.1b podemos ver un ejemplo de la información que la máquina imprime en el Abono, cada vez que formalizas un viaje.

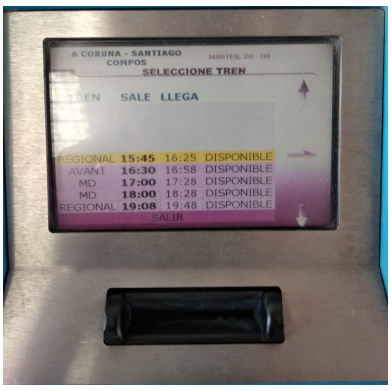
(a) Abono mensual parte delantera



(b) Abono mensual parte trasera

Figura 2.1: Parte delantera y trasera de un abono mensual

Una vez finalizado este proceso, cuando el usuario se dirige al tren con su bono y el viaje formalizado, el revisor debe comprobar tanto los datos del propio bono: tipo de bono, mes del bono, dni asociado al mismo, como los viajes formalizados. Para ello en la parte trasera del bono, visualizará la lista de esos viajes y observará si el viajero registro el viaje actual. Podemos ver en la figura Figura 2.1b, como es el resultado de formalización de viajes. Y en la Figura 2.2, podemos ver la máquina formalizadora.



(a) Pantalla de la máquina formalizadora de viajes

(b) Formalizadora de viajes

Figura 2.2: Máquina formalizadora de viajes

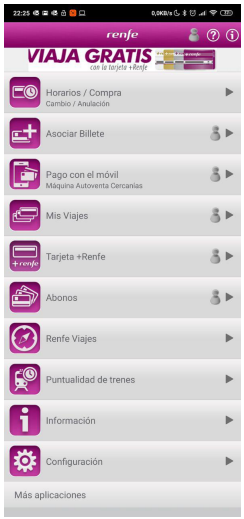


Figura 2.3: Página principal de la aplicación RenfeTicket

2.2 Metro de Madrid

Otro sistema de tickets, que inspiró éste proyecto fue el del **Metro de Madrid** [2]. El Metro, es una red de ferrocarril metropolitano que proporciona uno de los servicios de transporte más usados, en la ciudad de Madrid y a su área metropolitana.

El sistema de gestión de billetes del Metro de Madrid, consta de una tarjeta física, como la de la Figura 2.4, que debes comprar obligatoriamente para poder viajar en Metro. Una vez adquirida esa tarjeta, deberás introducirla en las máquinas de las estaciones de Metro y seleccionar el tipo de billete o bono que quieres comprar.

(a) Parte delantera tarjeta metro

(b) Parte trasera tarjeta metro

Figura 2.4: Parte delantera y trasera de la tarjeta Metro de Madrid

Las estaciones de metro están limitadas por unas barreras metálicas, que no te permiten acceder a las vías, a no ser que tengas la tarjeta. En ese caso, debes acercarla a los lectores que se encuentran en la parte superior de dicha barrera, de esta manera, te descontará un viaje y se abrirán las barreras permitiéndote el acceso a las vías de Metro.

(a) Máquinas del Metro de Madrid

(b) Barreras de acceso metro de Madrid

Figura 2.5: Estación metro de Madrid

Una vez estudiadas los sistemas de ambas compañías, se examinan aquellas características que se echan en falta y se estudia la viabilidad de implementación en la aplicación **FlashTicket**. Inicialmente, se habla únicamente de automatizar el sistema de abonos de trenes, con el fin de proporcionar más comodidades a los usuarios diarios del ferrocarril. Además, se decide evitar el uso de tarjetas físicas con riesgo de pérdida, y únicamente centrarnos en tickets digitales mediante códigos QR. Aún así, no se elimina definitivamente la opción física, permitiendo a los usuarios realizar impresiones en papel de los códigos de los tickets, si lo consideran necesario.

Capítulo 3
Metodología

En este capítulo hablaremos de la metodología usada durante todo el desarrollo del proyecto. Una **metodología de desarrollo** en el ámbito de la ingeniería de software, se refiere a un marco de trabajo que te guía en la estructuración y planificación de un proyecto.

3.0.1 SCRUM

En este proyecto se optó por una metodología ágil, concretamente SCRUM, una de las más utilizadas hoy en día. SCRUM es un marco de trabajo de procesos, que se caracteriza por su insistencia en propagar el trabajo en equipo.

SCRUM [3] es una metodología ágil e iterativa. Por tanto, un proyecto que se desarrolle en base a este marco de trabajo, se ejecutará en ciclos temporales cortos y de duración fija (normalmente dos semanas). Estos ciclos o iteraciones, se denominan **sprints**. Cada sprint, tiene que proporcionar un resultado completo y un incremento del producto final entregable al cliente.

Además de sprints, SCRUM también define un conjunto de roles, artefactos y reuniones.

Roles:

Se refiere a las personas que forman parte del equipo. SCRUM define tres roles esenciales

- **Scrum master:** Es el jefe de proyecto. Se encuentra entre el equipo y el cliente. Sus tareas son las de agilizar el trabajo del equipo y negociar con el cliente, entre otras. Se puede considerar un guía para el equipo.
- **Product Owner:** Es el representante del cliente y por lo tanto se encarga de definir el conjunto de requisitos **"Product Backlog"** y de validarlos posteriormente.
- **Equipo:** Equipo de trabajo
- **Usuarios:** Los usuarios finales que usarán el proyecto que se está desarrollando.

Sprints:

Un sprint es un periodo de tiempo acotado, normalmente de entre 2 semanas y un mes, en el que se desarrollan una serie de ítems de los requisitos del sistema. Cada sprint comienza con una reunión de planificación con todo el equipo, donde se asignan y explican las tareas que se llevaran a cabo en ese sprint.

Artefactos:

Son los elementos que garantizan la transparencia y el registro de la información del proceso Scrum.

- **Product Backlog**: Lista ordenada que contiene todos los requerimientos que se necesitan implementar en el producto final.
- **Sprint backlog**: Lista de requisitos que van a llevarse a cabo en cada sprint.
- **Incremento**: Es la suma de todos los elementos de la lista de producto completados durante el Sprint.

Reuniones:

- **Sprint planning:** Es la reunión de inicio de cada sprint, que comentamos anteriormente. Se definen las tareas que se llevaran a cabo en el sprint.
- **Daily Scrum:** Reunión que tiene lugar al inicio de cada jornada, informando sobre lo que se hizo el día anterior y lo que se hará durante ese día.
- **Sprint Review:** Valoración que se realiza al final de cada sprint.
- **Sprint Retrospective:** Una vez realizdala revisión del sprint, equipo junto con el Scrum Master, hacen un balance general de los puntos positivos y negativos durante el desarrollo del mismo.

En la figura Figura 3.1, podemos ver un esquema del flujo de trabajo de la metodología SCRUM.

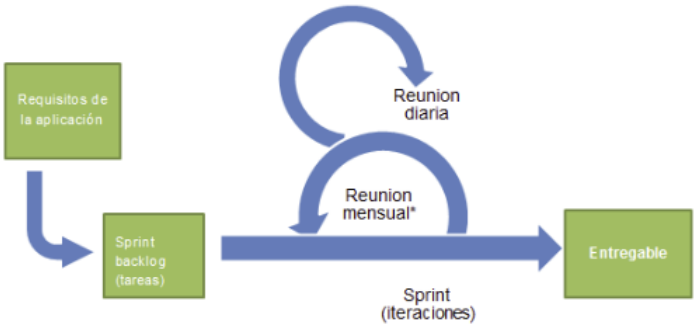


Figura 3.1: Diagrama de SCRUM

3.0.2 Aplicación de Scrum en el proyecto

La elección de esta metodología, surgió de la intención de ir desarrollando casos de uso funcionales que pudiesen verse reflejados al final de cada sprint. Es decir, en un caso real, es satisfactorio tanto para el cliente como para el propio desarrollador, observar el avance de la aplicación a través de incrementos funcionales.

Debemos tener en cuenta que en este proyecto, únicamente existen dos participantes, el director del proyecto y la autora. De esta manera, el director del proyecto realiza el rol de Scrum Master y la autora, hace el papel de equipo y de Product Owner verificando las funcionalidades finales y el cumplimiento de los requisitos.

Con respecto a los artefactos, cada vez que se iniciaba un sprint se listaba una serie de tareas que tenían que ser completadas al finalizar el mismo. Esta decisión se tomaba en la reunión inicial de cada sprint. Además cada día se decidía que tareas se abordarían durante esa jornada.

Por lo tanto, aunque no existe la diferenciación de roles tan completa, sí sé ha seguido la metodología SCRUM en el ámbito de artefactos, reuniones y sprints.

Capítulo 4

Análisis

En este capítulo nos centraremos en el análisis de la aplicación.

4.1 Actores:

En esta sección describiremos los actores que se han tenido en cuenta a la hora de realizar este proyecto, representados en la figura Figura 4.1.

- **Usuario anónimo:** Se refiere al usuario no autenticado. Únicamente podrá iniciar sesión, registrarse o consultar horarios de trenes.

- **Usuario viajero:** Se refiere al usuario autenticado en la aplicación del viajero. Puede realizar cualquier gestión de la aplicación cliente: compra, consulta, cancelación de bonos y/o billetes.

- **Usuario revisor:** Se refiere al usuario que usará la aplicación móvil de comprobación de billetes y bonos. Es necesario que inicie sesión previamente para poder acceder a las funcionalidades de la aplicación. Es el único tipo de usuario que puede acceder a la aplicación de validación de tickets.

Debemos tener en cuenta que debido al alcance del proyecto, no se han tenido en cuenta ciertos actores que existirían en la vida real. Por ejemplo, el actor administrador, que se encargaría de la gestión del catálogo de estaciones, viajes, trenes, que como ya hemos comentado no forma parte del alcance de este TFG.

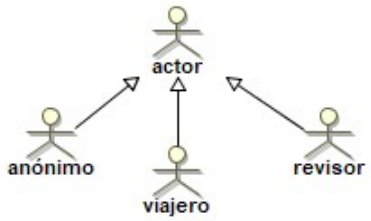


Figura 4.1: Conjunto de actores del proyecto

4.2 Requisitos:

Un paso importante antes de comenzar a desarrollar un proyecto, es realizar la Especificación de Requisitos (ERS). En esta sección, detallaremos los requisitos del proyecto.

4.2.1 Requisitos del Cliente:

Los requisitos del cliente, deben describir los requisitos del producto de tal forma que sean comprensibles para los usuarios del sistema, sin conocimiento previo. Por lo tanto, únicamente deben especificar el comportamiento externo del sistema.

Aplicación de viajeros:

1. **Autenticación:**
   - Los usuarios deben autenticarse para poder realizar cualquier funcionalidad que la aplicación ofrece, excepto consultar horarios de viajes, que es pública.
   - Los usuarios autenticados podrán realizar cualquier función de la aplicación.
   - Los usuarios autenticados podrán cerrar sesión.
   - La autenticación se realizará mediante el uso de JWT (JSON Web Token)
2. **Consulta de horarios:**
   - Tanto el usuario revisor como el usuario anónimo, podrán consultar los horarios de los trenes. Para ello deben cubrir un formulario eligiendo: origen, destino y fecha salida. A continuación le aparecerá una lista de los trenes en función de los parámetros seleccionados.
3. **Billetes:**
   El usuario, previamente autenticado, podrá:
   - Comprar un billete de tren, eligiendo el viaje que desea realizar. La aplicación le proporciona la opción de seleccionar el asiento y vagón, y a mayores podrá recibir un correo electrónico con los datos específicos de dicho viaje.

- Consultar los billetes comprados ordenados por su estado actual: billetes disponibles (aún no se ha realizado el viaje), billetes cancelados y billetes caducados (el viaje ya ha sido realizado).
- Cancelar un billete
- Descargar el código QR de un billete.

4. **Abonos:**
   El usuario, previamente autenticado podrá:
   - Comprar un abono cubriendo los datos del formulario: tipo y subtipo del abono que desea comprar y las dos estaciones entre las que desea viajar. También tendrá que elegir el mes en el que desea usar dicho abono. Una vez rellenados los campos, podrá seleccionar "Comprobar precio" para saber cuanto le costaría la compra con las condiciones seleccionadas. A mayores podrá seleccionar la opción de recibir un correo electrónico con los datos específicos de dicho abono, en caso de que decida realizar la compra.
   - Consultar los abonos comprados ordenados por su estado actual: abonos disponibles (cuya fecha de expiración es posterior a la actual), abonos cancelados y abonos caducados (fecha de expiración anterior a la actual).
   - Cancelar un abono, lo que implica la cancelación de los viajes asociados al mismo.
   - Formalizar un viaje asociado al abono actual.
   - Cancelar un viaje previamente formalizado.
   - Formalizar un viaje para el día siguiente, de entre la lista de viajes más frecuentados con ese abono.
   - Descargar el código QR de un abono.

Aplicación del revisor:

1. **Autenticación:**
   - El revisor debe autenticarse para poder acceder a cualquier funcionalidad que la aplicación le ofrece.
2. **Selección del tren:**
   - El revisor una vez autenticado, deberá seleccionar el tren en el que va a realizar la revisión de billetes y abonos.
3. **Validación de Billetes y Abonos:**
   El usuario revisor, previamente autenticado y después de haber seleccionado un tren, podrá:
   - Comprobar si un billete o abono es válido, a través del Scanner QR.

4.2.2 Requisitos Funcionales del Sistema:

A continuación mostraremos los requisitos del Sistema. La diferencia entre los requisitos del usuario y los del sistema, es que estos últimos agregan más detalle y explican como el producto debe proporcionar los requisitos del usuario.

Concretamente en esta sección, describiremos los **requisitos funcionales**[1]; cada requisito irá acompañado de sus diagramas del mismo.

Casos de uso comunes a ambas aplicaciones:

1. **CU-01 Autenticación**
   Este caso de uso, representa una generalización de los casos de uso **CU-01.1** (Registro) y **CU-01.2** (Inicio de Sesión). En la figura Figura 4.2, podemos ver un diagrama de este caso de uso.

2. **CU-01.1 Registro**
   Este caso de uso se inicia cuando el usuario, selecciona **"Registrarse"**. Al usuario le aparece un formulario que debe cubrir con sus datos. Si algún dato es incorrecto o es requerido y no ha sido cubierto, el usuario será avisado. En caso contrario, podrá pulsar el botón aceptar, y si todo ha salido satisfactoriamente, será redirigido a la página principal donde podrá iniciar sesión.

3. **CU-01.2 Inicio de Sesión**
   Este caso de uso se inicia cuando el usuario, selecciona **"Iniciar sesión"**. En ese momento, al usuario le aparece un formulario que debe cubrir con su login y contraseña para poder iniciar sesión. A continuación, en caso de que se haya logueado exitosamente, será redirigido a la página principal donde podrá ver nuevas funcionalidades. Si ha ocurrido algún error el usuario será informado.

4. **CU-01.3 Cerrar Sesión**
   Si actualmente el usuario está atuenticado, podrá activar el caso de uso cerrar sesión, pasando de ser un usuario de tipo cliente viajero a ser un usuario anónimo no autenticado.

[1] Los requisitos funcionales, describen lo que el sistema debe hacer y las interacciones entre el sistema y su entorno.

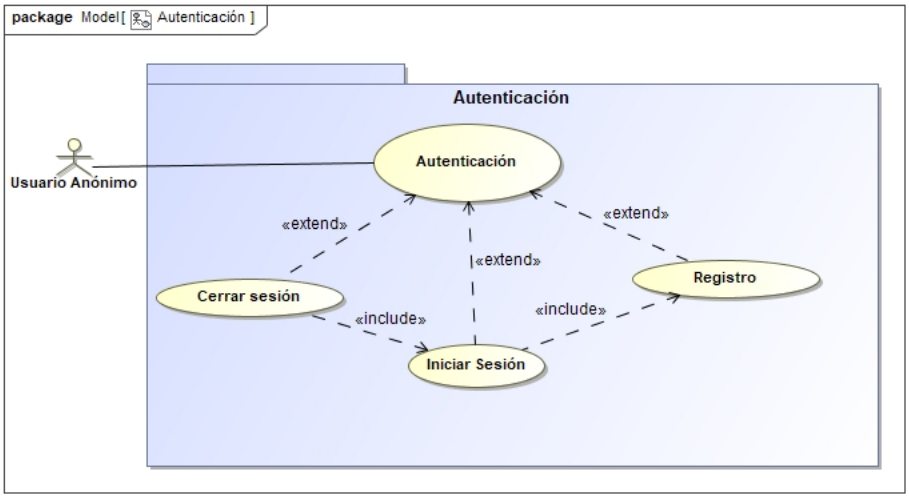


Figura 4.2: Caso de uso CU-01: Autenticación

Aplicación de viajeros:

1. **CU-02: Consultar horarios**
   Este caso de uso se activa cuando un usuario anónimo o usuario cliente, seleccionan **"Consultar horarios"**. En ese momento se presentará un formulario donde deberán indicar el origen, destino y fecha de salida del usuario. Una vez rellenados todos los campos, y seleccionado el botón aceptar, al usuario se le mostrará una lista con los viajes de tren del día seleccionado.

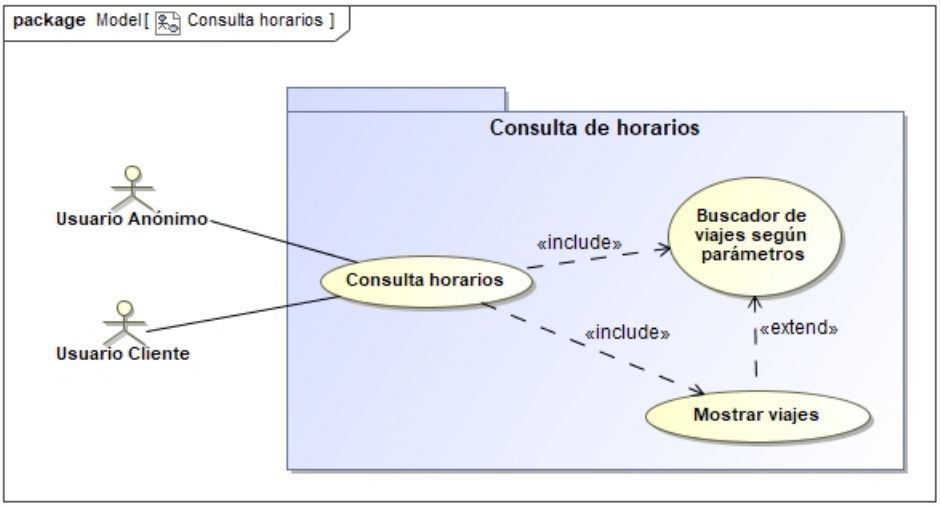


Figura 4.3: Caso de uso CU-02: Consulta de horarios

1. **CU-03: Gestión de billetes**

La gestión de billetes está compuesta por los siguientes casos de uso. En la figura Figura 4.4, podemos ver un diagrama de flujo que refleja este caso de uso.

2. **CU-03.1: Comprar billete**
   Este caso de uso se activa, cuando el usuario selecciona un billete dentro de la lista de billetes mostrada en el caso de uso CU-02. En ese momento al usuario le aparecerá una ventana emergente, donde si lo desea, podrá seleccionar el asiento, el modo de pago y el número de billetes que desea. Además si el usuario lo desea, recibirá un correo electrónico con el billete que ha comprado. La compra de un billete implica la generación de un código QR asociado al mismo.

3. **CU-03.2: Cancelar billete**
   Si el billete se encuentra previamente en estado "Disponible", y si el usuario actual es el mismo que compró el billete anteriormente, podrá cancelarlo. De esa manera, el asiento que se le asignó quedará liberado, y el estado del billete será modificado.

4. **CU-03.3: Visualizar billetes:**
   Si el usuario selecciona "**Mis billetes**", visualizará un menú, en donde cada pestaña constará de una lista con todos los billetes categorizados por su estado: *Disponible*, *Cancelado* y *Caducado*.

5. **CU-03.4: Descarga de billete:**
   Este caso de uso se activa, cuando el usuario visualiza un billete y decide seleccionar el botón de descarga. De esta manera, el usuario obtendrá una copia del billete con el código QR.

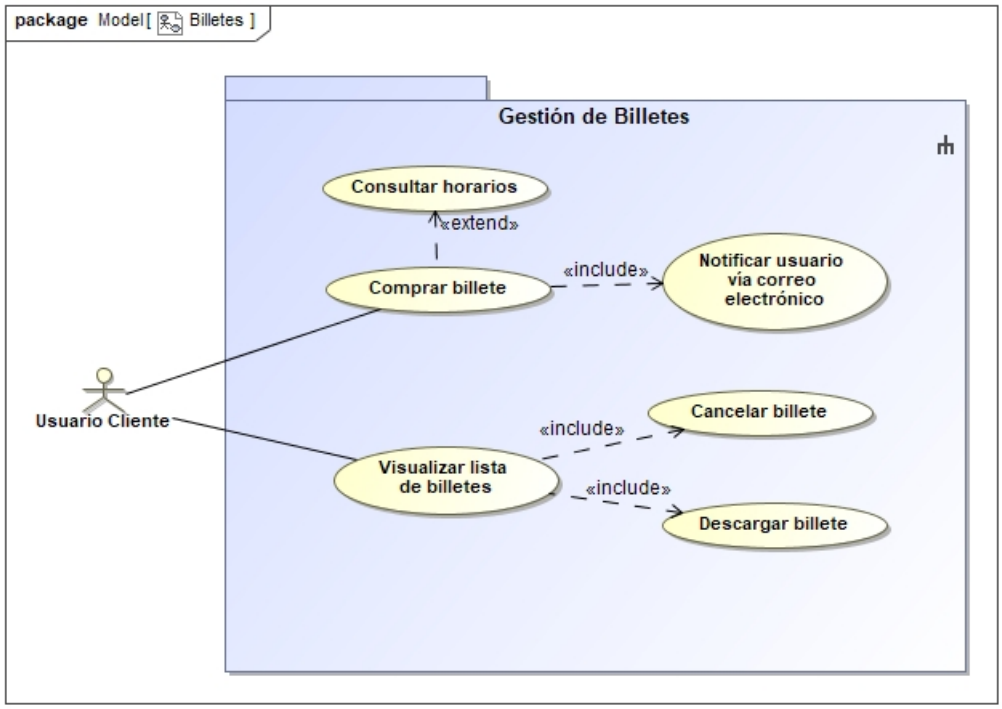


Figura 4.4: Caso de uso CU-03: Gestión de billetes

6. **CU-04: Gestión de abonos**
   La gestión de abonos está compuesta por los siguientes casos de uso. En la figura Figura 4.5, podemos ver un diagrama de flujo que los refleja.

7. **CU-04.1: Comprar abono:**
   Este caso de uso se activa, en el momento que el usuario selecciona "Comprar abono". En ese momento al usuario le aparece un formulario donde debe seleccionar dos estaciones entre las cuales desea viajar, el mes en el que desea utilizar el abono y el método de pago. Además deberá escoger entre los tipos (bono mensual, bono 10) y subtipos (libre, regional) de abonos existentes. Si lo desea podrá recibir una notificación mediante correo electrónico con los detalles del abono.

8. **CU-04.2: Consultar precio abono:**
   El usuario puede seleccionar "Consultar precio", después de cubrir el formulario de compra. De esa manera podrá saber el precio del abono asociado a las características seleccionadas, antes de comprarlo.

9. **CU-04.3: Cancelar abono:**
   Este caso de uso se activa si el usuario selecciona "Cancelar", dentro de la información del abono. Para que la cancelación se realice, es necesario que el estado anterior del abono sea "Disponible" y que el usuario que desea cancelar dicho

abono, sea el mismo que el que lo compró.

10. **CU-04.4: Visualizar lista de abonos**
    Si el usuario selecciona "Mis abonos", podrá visualizar una menú en donde cada pestaña le mostrará una lista de abonos categorizada por su estado: *Disponible*, *Cancelado* y *Caducado*. Si selecciona un abono, podrá acceder a su detalle.

11. **CU-04.5: Visualizar detalle abono**
    Si el usuario selecciona un abono dentro de la lista mencionada en el caso de uso CU-04.4, podrá acceder al detalle de dicho abono. En esa pantalla, podrá visualizar tanto el detalle del propio abono, como los viajes que han sido asociados al abono. Además, aparecerá una lista con aquellos viajes más frecuentados por el usuario con dicho abono.

    El usuario podrá formalizar nuevos viajes, activando en ese momento el caso de uso CU-04.6 o cancelar aquellos ya formalizados, caso de uso CU-04.7 . También podrá, a través de los viajes más frecuentados, formalizar un nuevo viaje para mañana activando el caso de uso CU-04.8.

12. **CU-04.6: Formalizar viaje**
    Si el usuario selecciona "Nuevo viaje", a partir del caso de uso anterior (CU-04.4), podrá formalizar un nuevo abono. Para eso, será redirigido a la misma pantalla que en el caso de uso CU-02. El formulario que verá el usuario, tendrá fijados origen y destino, según las estaciones del abono comprado. Por lo tanto, el usuario no podrá modificar esos datos, pero sí podrá cambiar el sentido del mismo, es decir, elegir la orientación del viaje entre esas dos estaciones.

13. **CU-04.7: Cancelar viaje formalizado**
    Si el usuario selecciona "Cancelar" en un viaje formalizado (tabla de viajes formalizados del caso de uso CU-04.5), inmediatamente le aparecerá un mensaje de confirmación de la acción. Si el usuario decide continuar y cancelar el viaje, la página se refrescará y se actualizará el estado del mismo. En caso contrario, todo seguirá igual y no se realizará ninguna acción.

14. **CU-04.8: Formalizar viaje para el día siguiente**
    Si el usuario selecciona el símbolo de compra, en un viaje frecuentado (tabla de viajes frecuentados del caso de uso CU-04.5), le aparecerá un mensaje de confirmación preguntando si desea formalizar dicho viaje, para la fecha del día siguiente. En caso de que el usuario acepte, el viaje aparecerá en la lista de formalizados. En caso contrario, todo seguirá igual.

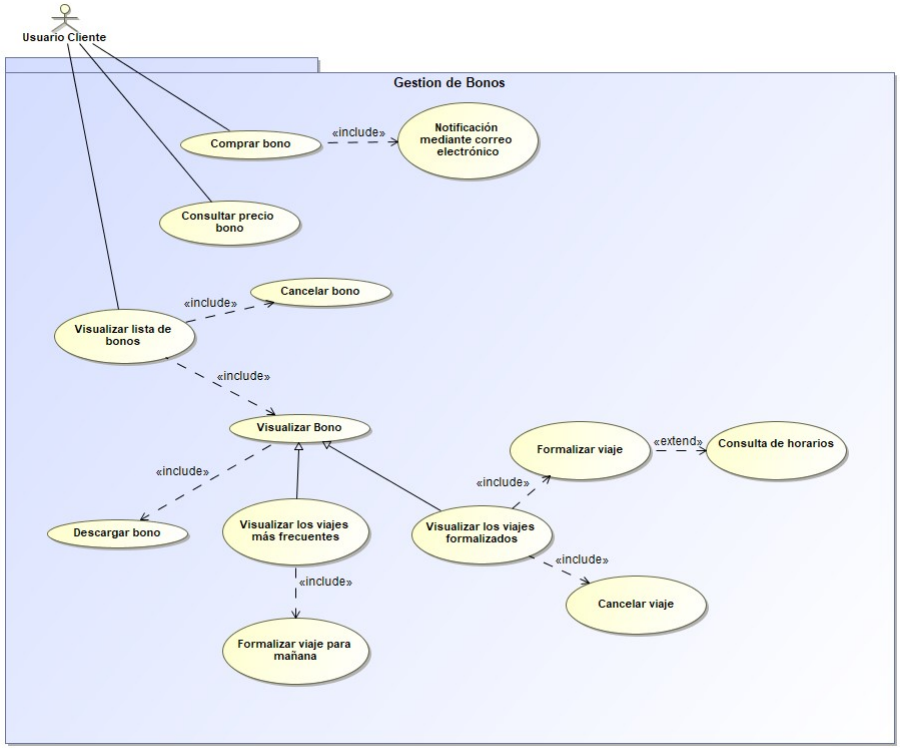


Figura 4.5: Caso de uso CU-04: Gestión de abonos

Aplicación del revisor:

1. **CU-05.1: Selección del tren:**
   Este caso de uso solo debe ser realizado por el actor **revisor**. De esta manera, en la aplicación de validación de billetes, el revisor previamente autenticado, deberá primero seleccionar el viaje en el que va a realizar dichas validaciones. Para ello, si selecciona "Seleccionar viaje", podrá visualizar un formulario similar al buscador de viajes de la aplicación de los viajeros.
2. **CU-05.2: Validación de abonos y billetes**
   Cuando ya haya seleccionado el viaje, podrá realizar validaciones tanto de abonos como de billetes. Cuando seleccione "Validar", le aparecerá un Scanner de códigos QR, con el que deberá escanear el código de dicho billete o abono. A continuación la aplicación le comunicará la validez o la ausencia de ella, del QR escaneado. Además le proporcionará ciertos datos interesantes para el revisor (DNI del usuario, asiento, vagón ...).

Capítulo 5

Planificación

En esta sección se mostrará la planificación inicial del proyecto. Además, también se comentarán las desviaciones que se han producido desde el inicio del proyecto y el coste.

5.1 Iteraciones

Los primeros sprints se corresponden con las tareas de estudio de viabilidad, alcance de proyecto, análisis de requisitos y diseño. A partir de ese momento cuando ya se sabía que hacer, cómo hacerlo y que tecnologías eran necesarias, se comenzó el desarrollo.

- **Primer sprint:** En este primer sprint se presentó la idea del proyecto y se estableció un alcance del mismo.
- **Secundo sprint:** Una vez establecido el alcance, se definió el proyecto de manera más concreta. Es decir, se escribió un enunciado explicando la idea del proyecto acompañándola con mockups.
- **Tercer sprint:** Cuando ya se sabía exactamente que hacer y cómo sería el resultado de las aplicaciones, se pasó a realizar la elección de las tecnologías que se usarían a lo largo del proyecto, tanto para la parte de servicio REST como para las aplicaciones clientes.
- **Cuarto sprint:** Se hizo un análisis inicial de requisitos, estableciendo los posibles casos de uso y tareas del desarrollo del proyecto.
- **Quinto sprint:** El objetivo de este sprint, era comprobar y estudiar como realizar la comunicación entre una aplicación de Angular simple y un servidor con un api REST.
- **Sexto sprint:** Este sprint, tenía como objetivo, el desarrollo de una aplicación de **Scanner QR**. De esta manera construimos la base de la aplicación del revisor y comprobamos su viabilidad.

- **Séptimo sprint:** En este sprint se desarrolla todo lo relacionado con la autenticación de los usuarios, que se realizaría mediante el uso de JWT.
- **Octavo sprint:** Este sprint tiene como objetivo implementar la lógica asociada al buscador de horarios de viajes por parámetros (Origen, destino, fecha salida).
- **Noveno sprint:** Este sprint se centra en los billetes de tren. Se implementa la parte de contratación de un viaje concreto, consulta de los billetes del usuario y cancelación de los mismos.
- **Décimo sprint:** Este sprint se centra en los bonos. Se desarrolla la parte de contratación de un bono y formalización de viajes asociados a un bono. También se abordan las acciones de cancelación de un abono y de un viaje formalizado.
- **Undécimo sprint:** Este sprint tiene como objetivo la implementación de una funcionalidad adicional sobre los abonos mensuales: mostrar al usuario una lista de viajes que más ha frecuentado con dicho abono, y permitirle que formalice uno de esos viajes, para el día siguiente. De esta manera si el usuario viaja diariamente, no es necesario que pase la fase de buscador de viajes. Además en este sprint, se investigó acerca de la firma digital del contenido del código QR.
- **Duodécimo sprint:** Este sprint implica la finalización de la aplicación del revisor. Primero, se añadió el apartado de registro e inicio de sesión de un revisor. A continuación se desarrolló la validación del código QR, mostrándole al revisor información interesante y verificando la firma del mismo.
- **Decimo tercer sprint:** En este último sprint se lleva a cabo la redacción de la memoria del proyecto.

Cabe destacar que la implementación de estas anteriores iteraciones, implica por una parte, el desarrollo de lógica en el servidor de API REST y a continuación la correspondiente codificación en las aplicaciones cliente.

5.2 Planificación temporal

Por otro lado, en este proyecto podemos destacar dos **recursos**: Director y Analista/Programador.

- **Director del proyecto:** Se encarga de guiar y supervisar el proyecto, así como de dar el visto bueno al finalizar el mismo.

- **Analista/Programador:** Se encarga del análisis, diseño, implementación y pruebas de cada una de las iteraciones que se comentaron anteriormente.

Para mostrar la **planificación** inicial del proyecto, se ha realizado un diagrama de Gantt en el que cada tarea implicaba una iteración. Debemos tener en cuenta, que aunque la fecha de inicio del proyecto fue el 31 de Noviembre de 2018 han ocurrido una serie de inconvenientes por el camino que han retrasado considerablemente el proyecto.

A continuación se muestra el diagrama de Gantt asociado a este proyecto. En la Figura 5.1, podemos ver el diagrama de los sprints. Además podemos ver en la tabla Tabla 5.1 las tareas en formato tabla, con la fecha de inicio de ese sprint, la de finalización y las horas invertidas en el mismo.

Sprint	Nombre	Data inicio	Data fin	Horas
1	Alcance del proyecto	30/11/18	30/11/18	3h
2	Descripción del proyecto	03/12/18	18/02/19	50h
3	Elección de las tecnologías	19/02/19	19/02/19	3h
4	Análisis inicial de requisitos	20/02/19	25/02/19	25h
5	Comunicación entre servidor y cliente	26/02/19	28/02/19	10h
6	Aplicación Scanner QR	01/03/19	10/03/19	45h
7	Autenticación de usuarios	11/03/19	31/03/19	30h
8	Buscador de horarios	01/04/19	30/04/19	70h
9	Gestión de billetes	01/05/19	31/05/19	75h
10	Gestión de bonos I	01/06/19	30/06/19	90h
11	Gestión de bonos II y firma digital	01/07/19	28/07/19	75h
12	Validación de tickets	29/07/19	11/08/19	60h
13	Memoria	12/08/19	06/09/19	134h
Total				670h

Tabla 5.1: Coste total del proyecto

5.3 Cálculo de costes

Podemos realizar una estimación del coste total del proyecto, teniendo en cuenta que en la planificación inicial hemos reflejado una duración de 640 h. Multiplicaremos por ese número el salario del trabajador, 25€/hora obteniendo un total de 16000 €.

$$25 euros/hora \times 640 horas = 16000 \tag{5.1}$$

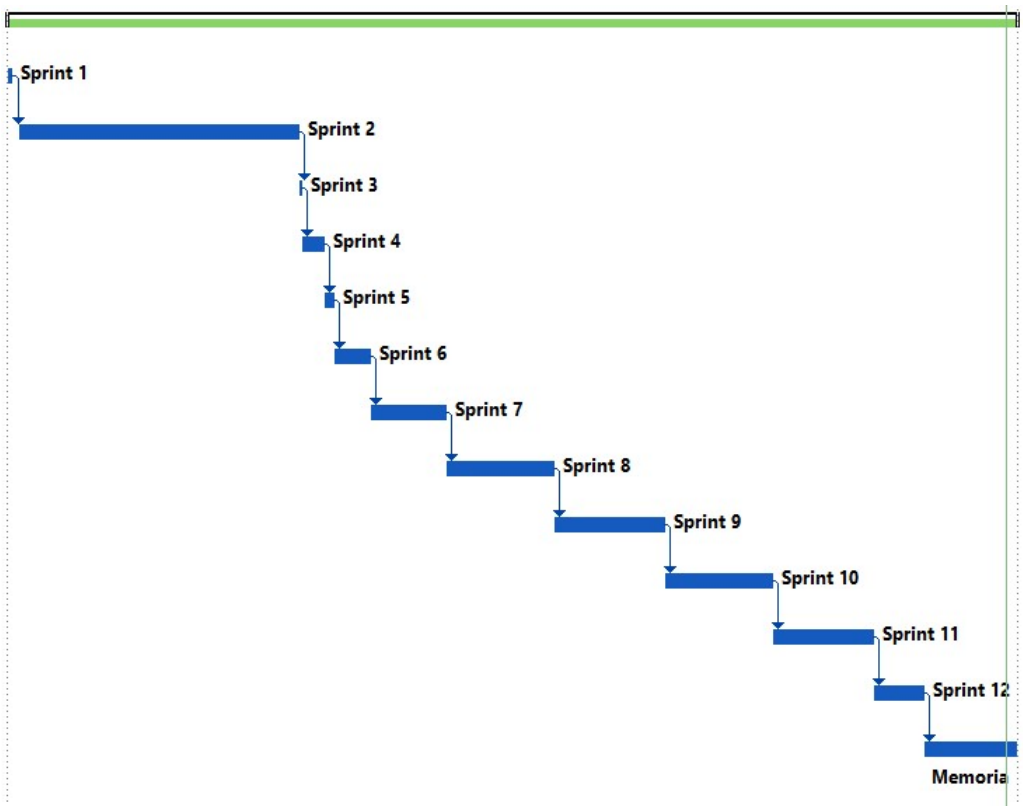


Figura 5.1: Diagrama de Gantt I

Capítulo 6

Fundamentos tecnológicos

En este capítulo abordaremos las tecnologías e herramientas que hemos usado a lo largo de todo el proyecto. Para ello, lo dividiremos en tres bloques:

- Tecnologías usadas en back-end
- Tecnologías usadas en front-end
- Herramientas de edición y creación de diagramas

6.1 Tecnologías usadas en back-end

- **Java**: Lenguaje de programación interpretado orientado a objetos. Una de sus características principales es la de ser un lenguaje multiplataforma, gracias a la máquina virtual de java (JVM).
- **Apache Maven:** [4] [5]
  Apache Maven es una herramienta software para la gestión y construcción de proyectos Java.
- **Spring Boot:** [6] [7]
  Spring Boot es un framework que facilita la creación de aplicaciones independientes basadas en Spring. Concretamente facilita la selección de los artefactos con Maven y el despliegue de la aplicación.
- **Hibernate:** [8] [9] [10]
  Hibernate es una herramienta de mapeo objeto/relacional (ORM) para Java. Por lo tanto, nos ayuda en el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación mediante archivos declarativos o anotaciones en los beans.

- **JPA Framework (Java Persistence API):** [11] [12]
  JPA es un framework del lenguaje Java, que maneja datos relacionales en las aplicaciones. Hibernate para Java EE incorpora el estándar JPA, mediante el componente Hibernate Annotation.

- **MapStructs:** [13][14]
  MapStructs es una herramienta que facilita el mapeo de objetos entre sí, por lo que es considerado un "**Mapeador de Beans de Java**". MapStructs se encarga de generar el código de mapeo a través de una serie de anotaciones en una interfaz.

- **MySQL:** [15] [16]
  MySQL es un Sistema de Gestión de Base de Datos relacional (SGBD).

6.2 Tecnologías usadas en front-end

- **Node:**[17] Es un entorno multiplataforma basado en JavaScript, que está diseñado para generar aplicaciones de forma altamente optimizada. Node contiene la herramienta **npm** (Node Package Manager), que es un gestor de paquetes con el que podremos gestionar los módulos proporcionados por node.

- **Angular 7: [18] [19]**
  Angular es un framework para crear aplicaciones web móviles y de escritorio, desarrollado en TypeScript, de código abierto. Se basa en el patrón de desarrollo Modelo Vista Controlador (MVC).

- **TypeScript:** [20]
  TypeScript es un lenguaje de programación libre y de código abierto. Es un superconjunto de JavaScript, que esencialmente añade tipado estático y objetos basados en clases. TypeScript extiende la sintaxis de JavaScript, por tanto cualquier código desarrollado con este lenguaje, en tiempo de compilación se traduce en JavaScript

- **Bootstrap:** [21]
  Bootstrap es una biblioteca multiplataforma o un conjunto de herramientas de código abierto, para desarrollar con HTML, CSS y JS. Su objetivo es facilitar mediante plantillas, el diseño de aplicaciones web.

- **Ionic:** [22]
  Ionic es un framework de código abierto, para el desarrollo de aplicaciones móviles híbridas. Actualmente, permite al usuario elegir el marco de interfaz con el que desea trabajar (Angular, Vue). Además, tiene integrado Cordova permitiendo la generación multiplataforma.

- **Apache Cordova:** [23] [24]
  Apache Cordova es un marco de desarrollo móvil de código abierto. Permite utilizar las tecnologías estándar web (HTML, CSS y JavaScript), para desarrollo multiplataforma. En este proyecto, se usará junto con Ionic y Angular. Por lo tanto, no se trabajará directamente sobre este framework sino que será usado para generación de ".apk".

- **Visual Studio Code:** [25] Es un editor de código desarrollado por Microsoft que proporciona funcionalidades de depuración, resaltado de sintaxis... Además también incorpora acceso directo al terminal de windows. En este proyecto se ha usuado para desarrollar las aplicaciones cliente con TypeScript.

6.3 Herramientas de edición y creación de diagramas

Las herramientas de edición y creación de diagramas, que se han usado para las fases de análisis, diseño y documentación han sido las siguientes:

- **Git:** Software de control de versiones, que te permite almacenar tus proyectos y todas las versiones de los mismos. En este caso, se usó el repositorio Git proporcionado por la facultad de informática.

- **Balsamiq:** [26] Balsamiq es una herramienta que permite realizar maquetas de diseño de una aplicación, con un editor fácil y simple de usar.

- **MagicDraw:** Herramienta de diseño de diagramas. En este proyecto se ha usado para los diagramas de clases y los casos de uso.

- **Latex:** [27] Editor de texto plano que permite desarrollar documentos con una metodología programática.

- **Postman:** Aplicación de escritorio que permite realizar peticiones http, configurando cabeceras, cuerpo, url, token... En este caso se ha usado para las pruebas contra el servidor.

- **Eclipse:** [28] Plataforma software que proporcionan un conjunto de herramientas de código abierto multiplataforma que permite desarrollar en diferentes lenguajes de programación.

- **Draw.io** [29] Programa de creación de diagramas de todo tipo (UML, entidad-relacion...) Además la herramienta es totalmente gratuita y tiene una versión online que te permite desarrollar los diagramas sin necesidad de descargar ningún programa.

Capítulo 7

Diseño

En este capítulo se abordarán todos los temas relacionados con el diseño.

7.1 Modelo de datos:

En este apartado comentaremos los detalles relacionados con el modelo de datos de la aplicación. Ademas en la figura Figura 7.1, podemos ver como se relacionan las entidades.

User

Entidad que representa a los usuario.

- id: Identificador del usuario.
- dni: DNI del usuario
- email: Correo electrónico del usuario
- login: Login del usuario
- password: Contraseña del usuario
- name: Nombre completo del usuario
- authorities: Conjunto de roles del usuario

Authority

Enumerado que representa los posibles roles que puede tener un usuario. En este caso, solo consideramos dos roles el de revisor y el de viajero.

Train

Entidad que representa a los trenes

- id: Identificador del usuario
- uuid: Localizador del tren

- typetrain: Tipo de tren

Carriage

Representa un vagón de tren

- id: Identificador del vagón
- description: Descripción del vagón
- train: Tren al que pertenece este vagón

Seat

Representa un asiento de un tren

- id:Identificador del asiento
- description: Descripción del asiento
- carriage: Vagón al que pertenece el asiento

GenericTrip

Representa la información básica no variable de un viaje.

- id: Identificador del tren genérico
- uuid: Localizador del viaje
- stationsTime: Mapa que guarda la ruta de estaciones por las que pasaría el tren y para cada estación guarda la hora de salida, de dicha estación.
- train: Tren asociado al viaje
- timeArrival: Hora de salida
- timeDestination: Hora de finalización del viaje completo
- price: Precio del viaje
- days: Días de la semana en los que se lleva a cabo el viaje.

ActualTrip

Representa la información de un viaje real, con fecha y día concreto.

- id: Identificador del viaje concreto.
- uuid: Localizador del viaje concreto
- genericTrip: Viaje genérico al que está asociado este viaje concreto
- state: Estado del viaje
- dateDeparture: Fecha de salida.
- dateDestination: Fecha de llegada.

- busySeats: Conjunto de sitios ocupados.

Ticket

Entidad que representa los bonos.

- id: Identificador del bono
- uuid: Clave única que representa el localizador del bono
- trips: Conjunto de viajes formalizados con el bono
- firsStation y secondStation: Estaciones entre las que este bono permite viajar.
- creationDate: Fecha de creación del bono
- expirationDate: Fecha de expiración del bono
- user: Usuario que representó la compra del bono
- QR: Código QR asociado al bono
- type: Tipo de bono (TicketType).
- subType: Tipo de bono que representa los tipos de trenes en los que el usuario deseará viajar.
- state: Estado actual del bono (TicketState)
- paymentMethod: Método de pago del bono
- price: Precio del bono
- frequentTrips: Conjunto de viajes frecuentes asociados al bono.

TicketTrip

Representa la relación entre un viaje concreto y un bono, es decir, representa a un viaje formalizado.

- id: Identificador del viaje
- seat: Asiento asignado al viaje
- actualTrip: Viaje concreto al que está asociado este viaje formalizado
- ticket: Bono al que está asociado este viaje formalizado
- state: estado del viaje formalizado.

Station

Representa una estación de ferrocarril

- id: Identificador de la estacion
- name: Nombre de la estación.

- location: Localización de la estación (ciudad, pueblo..)

SimpleTicket

Representa un billete simple

- id: Identificador del billete.
- uuid: Localizador del billete.
- seat: Asiento asignado para este billete.
- actualTrip: Viaje concreto para el que este billete es válido.
- source: Estación origen.
- destination: Estación destino.
- creationDate: Fecha de compra del billete.
- expirationDate: Fecha de expiración del billete.
- user: Usuario propietario del billete.
- QR: Código QR asociado al billete.
- state: Estado actual del billete.
- paymentMethod: Método de pago, con el que se adquirió el billete
- price: Precio del billete.

CodeQR

Representa un código QR

- id: Identificador del código QR.
- name: Nombre del fichero que contiene el código QR.
- path: Ruta con la ubicación del fichero
- creationDate: fecha de creación del Código Qr.

PriceTicket

Representa el conjunto de posibles tipos de bonos y sus precios asociados.

- id: Identificador del código QR.
- firstStation: Una de las dos estaciones asociadas al bono.
- secondStation: Una de las dos estaciones asociadas al bono.
- ticketType: Tipo de bono que el usuario desea comprar. Este tipo representa el número de viajes que el usuario podrá formalizar con este bono. Puede ser **bono mensual**, que permite formalizar viajes durante todo un mes o **bono 10**, que permite formalizar únicamente 10 viajes en total.

- price: fecha de creación del Código Qr.
- ticketSubType: Subtipo del bono que el usuario desea comprar, que representa el tipo de trenes en los que el usuario podrá viajar. Puede ser **regional**, el usuario únicamente podrá viajar en trenes regionales o **libre**, que permite viajar en cualquier tipo de tren.

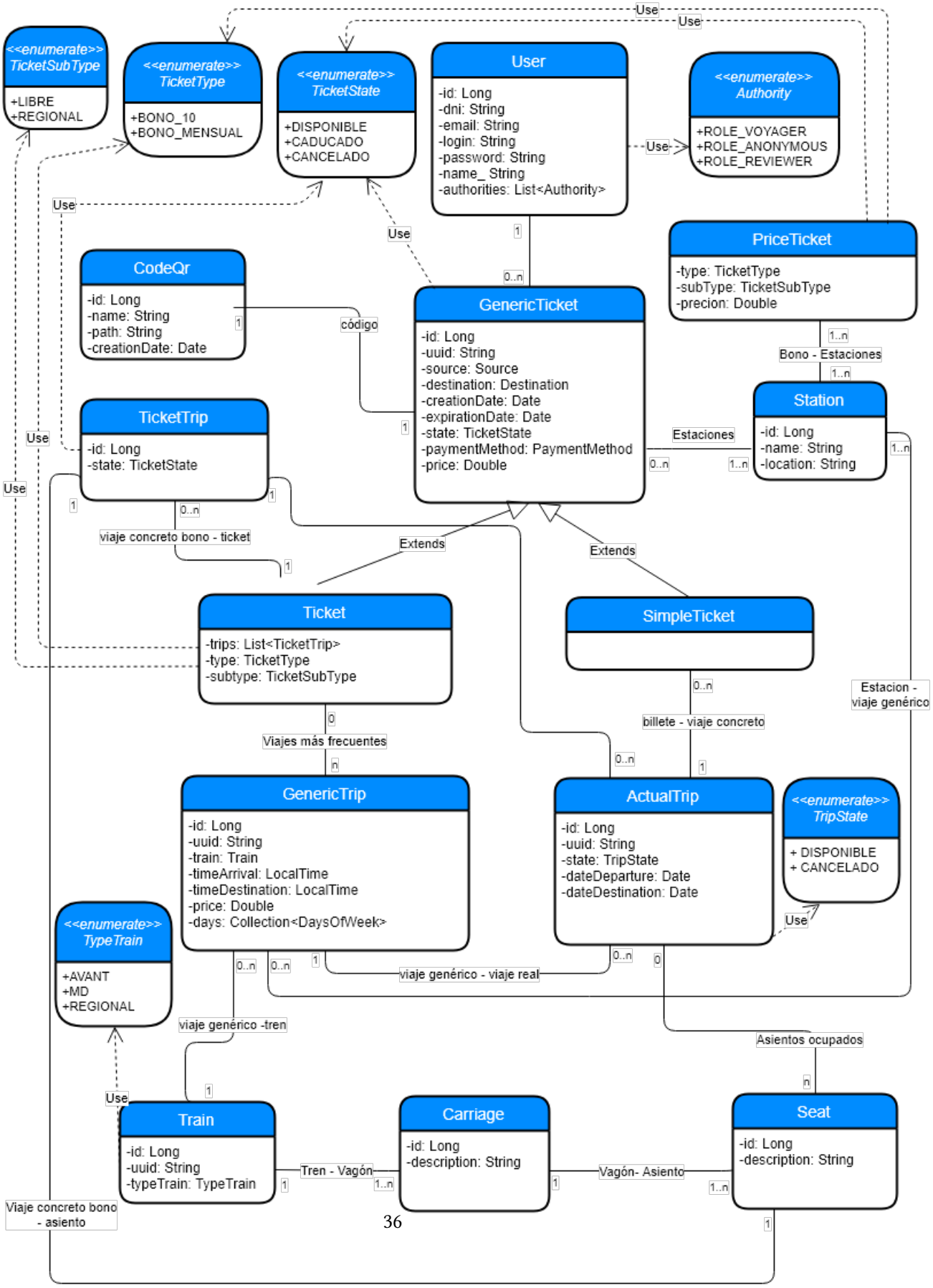


Figura 7.1: Diagrama entidad-relación

7.2 Arquitectura tecnológica del sistema:

En esta sección comentaremos las tecnologías utilizadas en cada capa a lo largo del proyecto.

Por un lado, en la **capa de datos** de la arquitectura, se ha utilizado MySQL. Hemos delegado la conexión con la base de datos, en JPA e Hibernate, que usan JDBC (Java Database Connectivity), que es un estándar para el acceso a base de datos. Debemos destacar, que en el servidor hemos necesitado la capa repositorio, en donde delegamos en JPA.

En cuanto a la **capa intermedia** (capa de negocio), hemos usado Spring e Hibernate. Esta capa a su vez también está subdividida en capas:

- **Capa servicio REST(Controladores):** Capa pública que expone un API REST. Es con la que se comunican las aplicaciones clientes.
- **Capa modelo:** Esta capa a su vez está subdividida:
  - **Capa de lógica de negocio:** Está compuesta por las fachadas y los servicios. Aquí es en donde se implementa la lógica de negocio de la aplicación. En este proyecto se ha hecho diferenciación entre dos capas, de manera que en la fachada se realiza la mayor parte de la codificación y en el servicio solo se codifica aquella lógica que necesita comunicación con la capa repositorio. Se puede decir que los servicios son una capa de comunicación entre la fachada y la capa repositorios, que a su vez implementa cierta lógica del sistema.
  - **Capa de repositorio:** Es la capa de acceso a datos. En ella se delega toda la comunicación con la base de datos.

Para la comunicación entre capas, hemos necesitado utilizar tres tipos de datos:

- **Entities:** Conjunto de datos que se usan para la comunicación con la BD. Concretamente son usados por la capa repositorio. Cada entidad se corresponde con una tabla en BD.
- **DTOs:** Conjunto de datos utilizados para la lógica de negocio.
- **TOs:** Conjunto de datos, expuestos al cliente, a través de la Capa REST.

Por otro lado, con respecto a esta capa, debemos tener en cuenta que para realizar el mapeo entre tipos de datos, se ha usado el framework **MapStructs**.

Finalmente, para las **aplicaciones cliente**, hemos usado Angular 7 e Ionic, ambas herramientas están basadas en la programación con TypeScript, HTML y CSS. Para los estilos CSS, nos hemos apoyado en Bootstrap. También debemos tener en cuenta que para la el despliegue

en móvil de las aplicaciones, hemos usado el framework Cordova. La comunicación entre esta capa y el servicio, se realiza a través de la capa API REST que mencionábamos anteriormente, mediante peticiones HTTP.

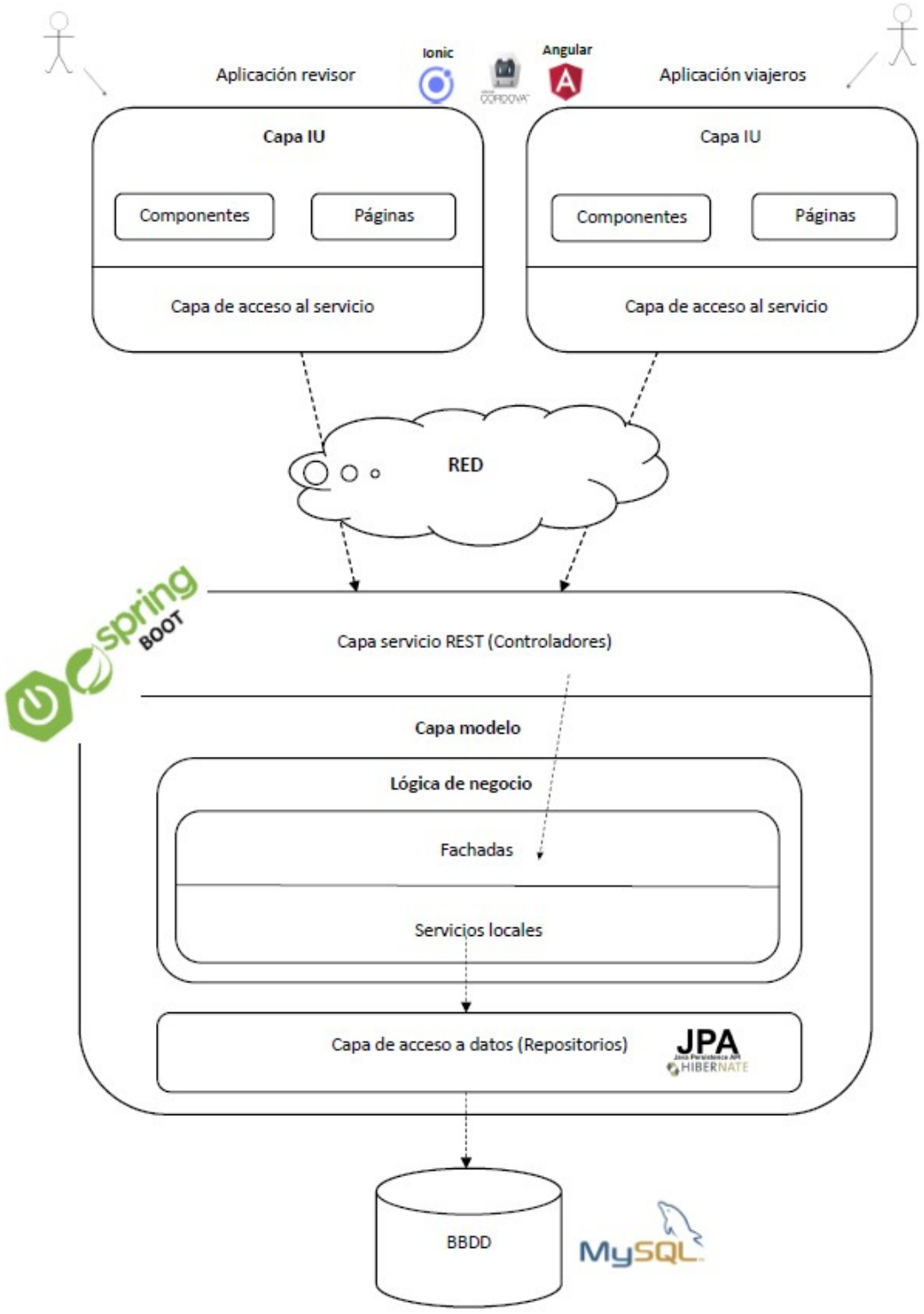


Figura 7.2: Arquitectura más detallada de la aplicación

A continuación expondremos los pasos de diseño de la aplicación. Por un lado, explicaremos el diseño Back-end, servicio y por otro lado, Front-End, cliente.

7.3 Diseño Back-End de la aplicación:

7.3.1 Capa acceso a datos:

En este proyecto la capa de acceso a datos, como mencionamos en capítulos anteriores, está compuesto por un conjunto de repositorios y de entidades. Las entidades nos permiten representar los datos almacenados en la base de datos, que en nuestro caso, se ven representados en la Figura 7.1.

Cada entidad, tiene asociado un repositorio (Repository). Un repositorio en JPA, se representa mediante una interfaz y se encarga de gestionar todas las operaciones de persistencia contra la tabla de base de datos asociada a la entidad. En este proyecto, se ha creado un respositorio por entidad, de manera que todos los repositorios extienden de **JpaRepository**, clase que nos proporciona métodos CRUD básicos. Todos los repositorios, deben estar anotados con **@Repository**, de esta manera le hacemos saber a JPA que esa interfaz se debe instanciar en el contexto de Spring, como un Bean de tipo "Repository".

Además, si deseamos implementar métodos nuevos dentro de un repositorio, JPA nos lo permite mediante la anotación a nivel de método **@Query**. De esta manera, le hacemos saber a JPA que el método que le sigue, es una query y debe ser transformado a lenguaje SQL. Acompañando a dicha anotación, debe ir la sentencia JPQL que deseemos.

A continuación podemos observar un ejemplo, de como funciona la anotación @Query.

```
@Query("Select s From Seat s
    JOIN s.carriage c
    JOIN c.train t
    WHERE t.id = :idTrain and
    s NOT IN (Select bs From ActualTrip at
        JOIN at.busySeats bs
        WHERE at.id = :idActualTrip)")

List<Seat> findValidSeatsFromActualTrip(Long idTrain, Long
   idActualTrip);
```

En la Figura 7.3 veremos el conjunto de repositorios más importantes del proyecto. Aunque como hemos dicho, cada entidad tiene asociado un repositorio, los que aparecen en el figura referenciada, son aquellos repositorios que tienen lógica a mayores de la proporcionada por JpaRepository. Los restantes, usan los métodos ya proporcionados por JPA, de tipo CRUD.

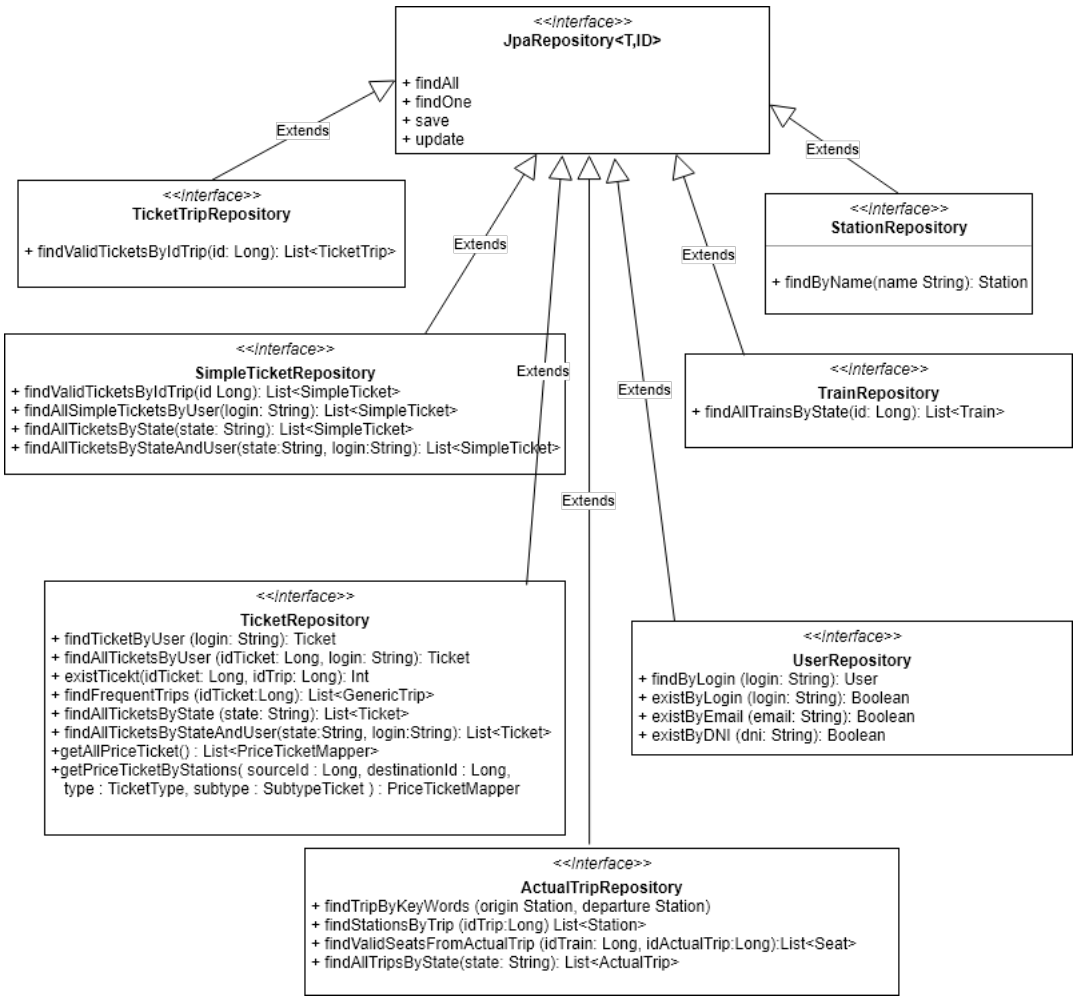


Figura 7.3: Repositorios más importantes del proyecto

7.3.2 Servicios:

A continuación hablaremos de la capa de servicios, esta capa es que se comunica con los repositorios anteriormente mencionados. Por lo tanto, se encarga de encapsular la comunicación entre la capa de fachada y la capa de acceso a datos. Debemos tener en cuenta que estos servicios, deben incorporar la anotación **@Transactional**, permitiendo que contengan lógica transaccional y también la anotación **@Service**, indicando que son beans de tipo servicio.

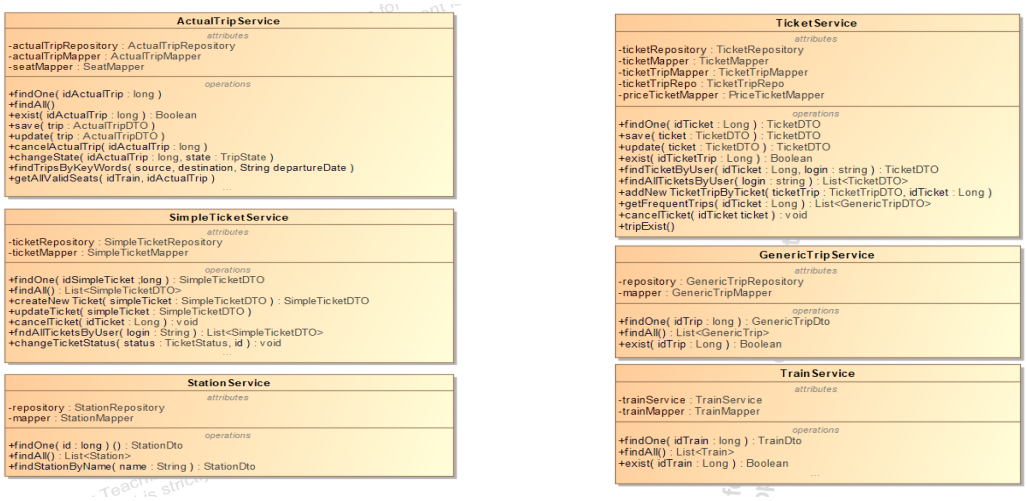


Figura 7.4: Diagrama de clase de los servicios I

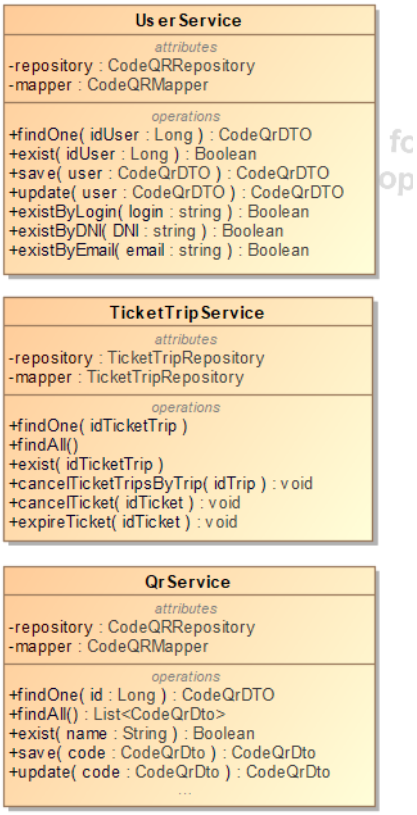


Figura 7.5: Diagrama de clase de los servicios III

7.3.3 Fachada:

La capa de fachada, es la encargada de implementar toda la lógica de negocio de la aplicación. Esta capa se comunica continuamente con los servicios, que se encargan del acceso a datos, y las fachadas usarán esos datos para su lógica. De esta manera, diferenciamos la lógica correspondiente a los datos de la lógica del flujo de la aplicación.

Figura 7.6: Diagrama de clase de las fachadas I

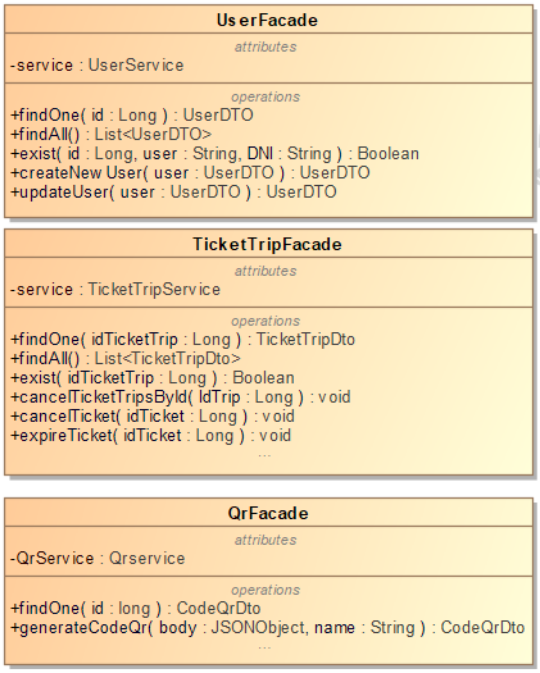


Figura 7.7: Diagrama de clase de las fachadas II

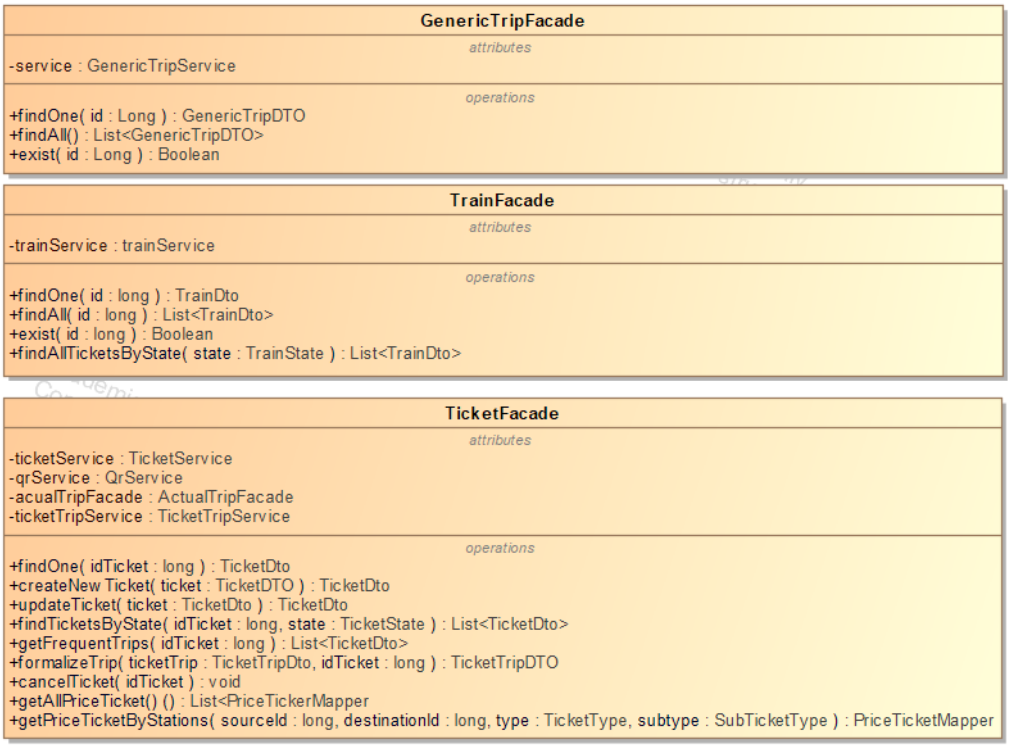


Figura 7.8: Diagrama de clase de las fachadas III

7.3.4 Controladores:

La capa REST, contiene un conjunto de controladores que son por los que entra el flujo de la aplicación. Cabe destacar, que tanto los objetos recibidos como los devueltos por esta aplicación, deben requieren representación JSON.

A continuación mostraremos el API REST:

URL	Método HTTP	Descripción
/api/auth/new	POST	Creación de un nuevo usuario. Debe recibir en el cuerpo de la petición el objeto que represente al usuario.
/api/auth/login	POST	Autenticación de un usuario. Necesita recibir en el cuerpo de la petición el login y password del usuario. Devuelve un Token de tipo Bearer JWT.
/api/auth/user/login	GET	Obtener un usuario a través de su login. Devuelve un usuario
/api/auth/user/id	GET	Obtener un usuario a través de su identificador. Devuelve un usuario

Tabla 7.1: API REST del controlador de autenticación

URL	Método HTTP	Descripción
/api/trip/search	GET	Obtención de una lista de viajes, mediante parámetros (origen, destino, fecha)
/api/trip/id	GET	Obtencion de un viaje a través de su identificador.

Tabla 7.2: API REST del controlador de viajes

URL	Método HTTP	Descripción
/api/station/	GET	Obtener la lista de todas las estaciones.
/api/station/id	GET	Obtener una estación a través de su identificador
/api/station/name	GET	Obtener una estación a través de su nombre

Tabla 7.3: API REST del controlador de estaciones

URL	Método HTTP	Descripción
/api/train/	GET	Obtener la lista de trenes.
/api/train/tipo	GET	Obtener la lista de trenes de un tipo (AVANT,MD,REGIONAL).
/api/train/id	GET	Obtener un tren por su id

Tabla 7.4: API REST del controlador de trenes

URL	Método HTTP	Descripción
/api/ticket	POST	Crear un billete. Por parámetro recibe un flag que permite saber si el usuario desea recibir un email con la información del billete
/api/ticket/id/cancel	DELETE	Cancelar un bono en función de su identificador
/api/ticket/id	PUT	Actualizar un bono
/api/ticket/id	GET	Obtener un bono a partir de su identificador
/api/ticket/login	GET	Obtener el conjunto de bonos asociados a un usuario
/api/ticket/state/	GET	Obtener el conjunto de bonos con un estado actual concreto.
/api/ticket/state/login	GET	Obtener el conjunto de bonos asociados a un usuario, y con un estado actual concreto.
/api/ticket/id/newTrip/login	POST	Formaliza un nuevo viaje en el bono cuyo id es id
/api/ticket/id/cancelTrip/idTrip	DELETE	Cancelar un viaje formalizado según su identificador
/api/ticket/id/idTrip	GET	Obtener un viaje formalizado según su identificador
/api/ticket/id/trips	GET	Obtener el conjunto de viajes formalizados de un bono
/api/ticket/id/frequentTrips	GET	Obtener el conjunto de viajes más frecuentados de un bono
/api/ticket/id/qr	GET	Obtener el código QR asociado a un bono
/api/ticket/prices	GET	Obtener todos los objetos PriceTicket
/api/ticket/price	GET	Obtener un PriceTicket según los parámetros: id primera estación, id segunda estación, tipo de bono, subtipo de bono

Tabla 7.5: API REST del controlador de bonos

URL	Método HTTP	Descripción
/api/simple/ticket	POST	Crear uno o varios billetes. Por parámetro recibe el número de billetes que se desea crear y un flag que permite saber si el usuario desea recibir un email con la información del billete
/api/simple/ticket/id/cancel	DELETE	Cancelar un billete a partir de su identificador
/api/simple/ticket/id	GET	Obtener un billete a partir de su identificador
/api/simple/ticket/id	PUT	Actualizar un billete
/api/simple/ticket/id	GET	Obtener un billete a partir de su identificador
/api/simple/ticket/login	GET	Obtener el conjunto de billetes asociados a un usuario
/api/simple/ticket/state	GET	Obtener el conjunto de billetes con un estado concreto
/api/simple/ticket/state/login	GET	Obtener el conjunto de billetes asociados a un usuario, y con un estado actual concreto.
/api/simple/ticket/id/qr	GET	Obtener el código QR asociado a un billete

Tabla 7.6: API REST del controlador de billetes simples

7.4 Diseño Front-end de las aplicaciones

A nivel Front-end, se ha usuado Angular, Ionic y Cordoba. El framework Angular, divide la progración en varias capas: componentes, servicios y router. A su vez en este proyecto se ha añadido una nueva capa a nivel Front-End. Las páginas, de esta manera se diferencia, aquellos componentes que componen una página y los que se centran en una lógica concreta.

7.4.1 Componentes:

Un componente en Angular, es considerado el bloque de construcción de una interfaz de usuario más básico de una aplicación cliente. Los componentes de Angular, son un subconjunto de directivas, asociados a una plantilla.

Los componentes pueden referenciarse desde el template de otro componente. Es decir, un componente puede estar compuesto de uno o varios componentes más, de manera que cuando se compila el código, Angular inyecta el código del componente hijo en el componente padre. La comunicación entre dos componentes puede realizarse de varias formas:

Mediante "accessors": Un componente hijo, a menudo suele necesitar recibir parámetros para poder realizar su lógica. Una manera de transmitir esos parámetros de un componente a otro, es mediante "accessors". Un **"accessor"** se representa, como vemos en la Figura 7.9, mediante la anotación **@Input()**, de esa manera estamos declarando que el

parámetro con esa anotación es de tipo entrada. Para enviar parámetros a un componente, debemos hacerlo referenciando a su "selector", como se ve en la Figura 7.10. El selector es el identificador del componente, y podemos encontrarlo en el archivo **".ts"** del mismo, Figura 7.11

```
@Input()
ticket: Ticket;

@Input()
login: string;
```

Figura 7.9: Ejemplo de accessor en angular

```
<div class="row justify-content-center" *ngFor="let ticket of caducadosTickets">
    <app-ticket-little-detail [ticket]="ticket" [login]="currentUser.login"></app-ticket-little-detail>
</div>
```

Figura 7.10: Ejemplo de llamada entre componentes

```
@Component({
  selector: 'app-ticket-little-detail',
  templateUrl: './ticket-little-detail.component.html',
  styleUrls: ['./ticket-little-detail.component.scss']
})
```

Figura 7.11: Selector de un componente angular

A través de su URL: Si un componente tiene asociado una ruta en el archivo "app-routing.module.ts de angular", entonces éste puede ser llamado desde otro componente. Esta llamada se puede realizar desde el template, con la directiva **"routerLink"** , como se visualiza en la figura Figura 7.12 o a través del archivo ".ts", usando el método navigate del objeto **Router** de Angular, Figura 7.13.

```
<a [routerLink]="['/user/simple/tickets/',info.nombreUsuario]" class="btn btn-success mt-4">Mis billetes</a>
```

Figura 7.12: Ejemplo de uso de routerLink en Angular

```
this.router.navigate(['home']);
```

Figura 7.13: Ejemplo del uso del método navigate del objeto Router de Angular

Mediante inyección: Un componente puede ser inyectado en otro componente, y de esa manera tener acceso a los métodos y atributos públicos del mismo. Normalmente esta inyección se hace a nivel de constructor, Figura 7.14

```
constructor(
  private route: ActivatedRoute,
  private router:Router,
  private ticketService: TicketService,
  private simpleTicketService: SimpleTicketService,
  private tokenService: TokenService,
  private tripService: TripService,
  private modalService: NgbModal,
  private authService: AuthUserService

) {

  this.idTicket = +route.snapshot.params.idTicket;
}
```

Figura 7.14: Ejemplo inyección de componentes Angular a través de constructor

Debemos tener en cuenta, que un componente angular está dividido en tres archivos:

- **Archivo ".ts"**: Fichero donde se implementa la lógica de tipo TypeScript.
- **Template ".html"**: Fichero donde se implementa el código HTML. Angular trabaja con código HTML puro, pero a mayores proporciona herramientas para realizar lógica dentro de este fichero. Nos permite utilizar variables declaradas en el archivo .ts, utilizar sentencias condicionales y bucles.
- **Fichero de estilos ".scss"**: Aquí añadiremos los estilos asociados a ese componente. Angular en la creación del proyecto te permite seleccionar el tipo de lenguaje de estilos que deseas utilizar (scss, css,..).

En el Apartado 7.5, tenemos un listado con los componentes de la aplicación.

7.4.2 Servicios:

Angular a parte de los componentes, también proporciona "Servicios". Estos servicios son archivos TypeScript, que nos permiten agrupar la lógica de comunicación con el servidor. Recordemos que la comunicación entre la aplicación cliente y el servidor, se realiza mediante peticiones HTTP a un API REST expuesto por el servidor.

En este proyecto, se ha creado un conjunto de servicios que permiten separar la parte de comunicación con el servidor, de la lógica interna de la aplicación cliente. De esta manera, los componentes cuando necesiten acceder al API REST, lo harán a través de los servicios mencionados.

Podemos observar en la Figura 7.15, las correspondencias entre servicios Angular y los controladores del servidor. Podemos observar, que el servicio TokenService no se comunica con ningún controlador, eso es porque su objetivo es controlar la funcionalidad del JWT a nivel de sesión.

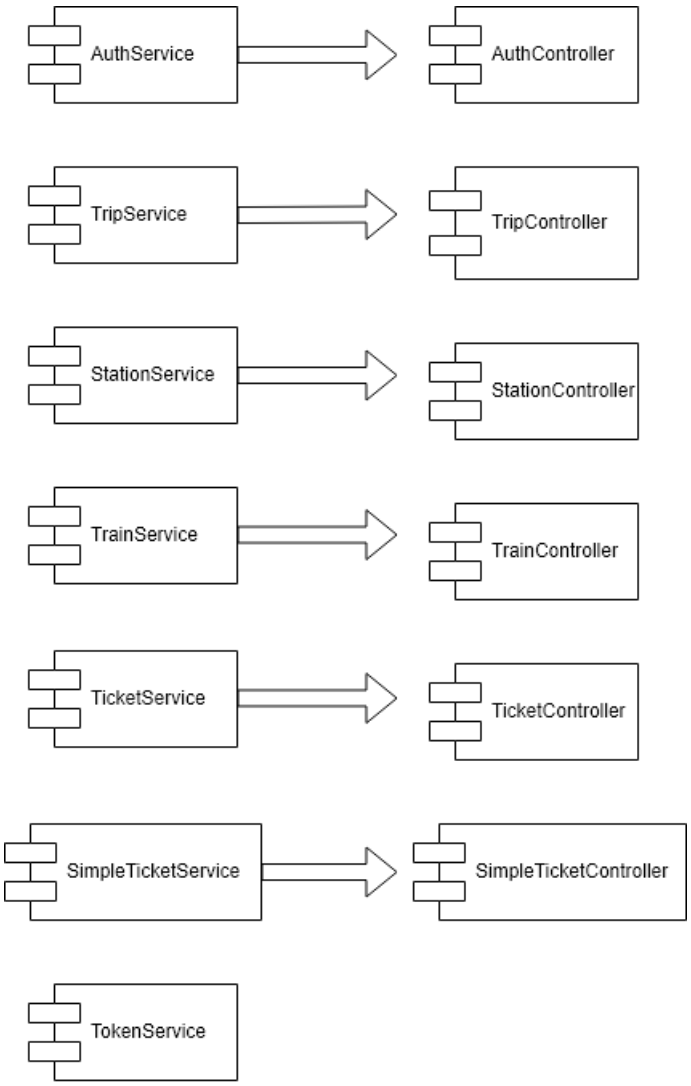


Figura 7.15: Comunicación entre aplicación cliente y servidor

En la Figura 7.16, muestra como se realiza una comunicación entre componente y servicio. Por otro lado en la Figura 7.17, se refleja como se lleva a cabo la correspondiente comunicación HTTP con el servidor.

Cabe destacar, que todo método que realice una petición al servidor, deberá devolver un objeto de tipo Observable. De esta manera, la aplicación no se verá bloqueada a la espera de la respuesta del servidor. Los métodos que devuelven Observables, cuando se invocan debe hacerse a través del método "subscribe", creando una petición asíncrona. Esto proporciona una mejor experiencia de usuario, sin esperas.

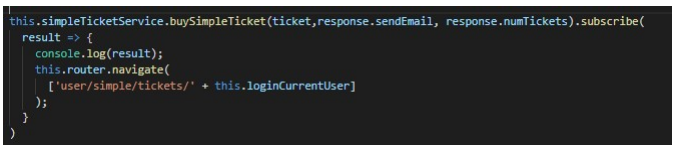


Figura 7.16: Comunicación entre un componente y el servicio

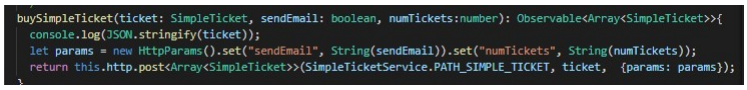


Figura 7.17: Comunicación entre un servicio y el servidor

7.4.3 Enrutador:

Angular en cada aplicación crea un nuevo archivo específicamente para definir las rutas de la aplicación. Ese fichero se llama “app.routing-module.ts”, y define una constante de tipo Routes[](objeto de Angular).

Enrutador de la aplicación de viajeros:

PATH	Componente	Descripción
/flashticket/home	Home	Página principal de la aplicación.
/flashticket/login	Login	Página de login.
/flashticket/register	Home	Página de registro
/flashticket/train-schedule	BuyTripPage	Página de búsqueda de trenes.
/flashticket/tickets/:login	UserTicketsPage	Página de bonos de un usuario
/flashticket/tickets/:login	UserTicketsPage	Página de bonos de un usuario
/flashticket/tickets/:login/:idTicket	TicketDetailsPage	Página del detalle de un bono
/flashticket/tickets/:login/:idTicket/newTrip	TicketDetailsPage	Página de compra y formalización de nuevos viajes, tanto a nivel de billete como de bono.
/flashticket/user/simple/tickets/:login/	TicketDetailsPage	Página con la lista de billetes de un usuario.
/flashticket/buy-ticket/	BuyTicketPage	Página de compra de bonos.

Tabla 7.7: Conjunto de rutas de la aplicación Angular de viajeros

Enrutador de la aplicación de revisor:

PATH	Componente	Descripción
/flashticket/home	Home	Página principal de la aplicación.
/flashticket/login	Login	Página de login.
/flashticket/register	Home	Página de registro
/flashticket/search-trip	SelectTripPage	Página de selección de viajes
/flashticket/ticketScanner	ScannerTicketsPage	Página Scanner de abonos
/flashticket/simpleTicketScanner	ScannerSimpleTicketsPage	Página Scanner de billetes

Tabla 7.8: Conjunto de rutas de la aplicación Ionic de revisores

7.4.4 Ionic y Cordova

Debemos destacar que Ionic funciona de la misma manera que Angular, porque en este caso en la creación de la aplicación Ionic se eligió Angular como base de trabajo. Por lo tanto las explicaciones anteriores se pueden aplicar al framework Ionic, con una ligera diferencia en cuanto a los componentes. Ionic sí proporciona componentes de tipo "Page" que aportan su propio fichero NgModules.

Además Ionic incorpora la herramienta cordova que como ya comentamos en su momento, nos permite generar los ficheros .apk para las versiones móvil.

Teniendo en cuenta las tecnologías usadas para desarrollar las aplicaciones cliente, es posible desplegar cada aplicación como web (para acceder desde un navegador) o como móvil (para instalarla como applicación). En este caso, la aplicación de los viajeros se desplego tanto en web como móvil para permitir el acceso multiplataforma. En cambio, la del revisor únicamente se desplego en móvil ya que no tiene sentido acceso desde navegador.

Para realizar el despliegue en móvil de la aplicación de viajeros, se ha usado ccordova, como bien hemos comentado. Para ello ha sido necesario realizar los siguientes pasos:

1. Añadir un proyecto cordova a nuestro proyecto Angular y añadir las plataformas tanto de navegador como móvil, de esta manera podemos desplegar la aplicación en ambas plataformas. En este caso, el despliegue de la aplicación móvil se hizo en Android, pero para iOS el funcionamiento es idéntico. Simplemente es necesario añadir la plataforma ios. Además tenemos que añadir el plugin, para que el enrutamiento se comporte correctamente.

```
cordova create cordova
cordova platform add android
cordova platform add browser
cordova plugin add cordova-plugin-ionic-webview
```

```
5
```

2. Cambiar el campo <base href="/"> del archivo index de nuestro proyecto Angular, por <base href="./">.

3. Añadir al apartado de scripts del archivo "package.json", los scripts que usaremos para generar el apk.

```
1   "cordova": "ng build --prod --base-href . --output-path
     cordova/www",
2   "cordova-android": "npm run cordova && cd cordova &&
     cordova run android --device",
3   "cordova-browser": "npm run cordova && cd cordova &&
     cordova run browser"
4
```

   Si nos fijamos en el comando de android, el despliegue se realiza con el parámetro "–device", esto es debido a que en este caso se ha usado adb para generar el despliegue directamente en el propio dispositivo móvil.

En el caso de Ionic es más sencillo el despliegue en móvil gracias a su aplicación DevApp, que te permite ver directamente en el dispositivo móvil el resultado de la aplicación. Simplemente tienes que tener el móvil y el ordenador desde el que realices el despliegue, en la misma red y ejecutar a continuación el siguiente comando

```
1   ionic serve --devapp
```

7.5 Interfaz gráfica de las aplicaciones:

En esta sección describiremos a alto nivel, la interfaz de usuario de las aplicaciones.

7.5.1 Componentes y páginas de la aplicación de viajeros

La interfaz de usuario está estructurada en componentes, páginas y navegación como ya hemos comentado. En este apartado concretaremos con más detalle, como están estructuradas las dos aplicaciones en cuanto a los componentes, páginas y la navegación entre ellas.

La tabla 7.9 muestra el conjunto de componentes de tipo modal que se usan para informar al usuario

Componente Modal	**Descripción**
CancelAlert	Componente de tipo modal. Muestra una ventana emergente, que permita al usuario confirmar que desea cancelar un billete o bono.
ConfirmModal	Componente de tipo modal. Muestra una ventana emergente, que permita al usuario confirmar un evento concreto.
ErrorModal	Componente de tipo modal. Muestra una ventana emergente, con la información asociada al error que se ha producido.
InfoModal	Componente de tipo modal. Muestra una ventana emergente, mostrando información al usuario.

Tabla 7.9: Lista de componentes alerta de la interfaz de usuario

Las tablas 7.10 y 7.11 muestran el conjunto de componentes base que usan a lo largo de la aplicación.

Componente	**Descripción**
Login	Componente de tipo modal. Contiene el formulario que el usuario debe rellenar para autenticarse en la aplicación.
Register	Componente de tipo modal. Contiene el formulario que el usuario debe rellenar para registrarse en la aplicación.
BuyTripForm	Este componente, contiene el formulario que el usuario debe rellenar cuando desea comprar un viaje concreto. Desencadena un evento cuando el usuario selecciona “Aceptar”, para comunicarle al componente padre, los parámetros seleccionados por el usuario en el formulario.

Tabla 7.10: Lista de componentes de la interfaz de la aplicación de viajeros I

Componente	Descripción
BuyTripModal	Componente de tipo modal. Contiene el formulario que el usuario debe rellenar para completar su compra final. En este formulario podrá seleccionar datos como el vagón y asiento, método de pago y si desea recibir un correo electrónico de notificación. Es un componente compartido para los bonos y billetes, por lo tanto adicionalmente, en el caso de billetes, podrá indicar el número de billetes que desea comprar
FrequentTripsTable	Componente que pinta en vista la tabla de "Viajes más frecuentados", dentro del detalle de un bono. Desencadena eventos para comunicación con el componente padre, cuando el usuario desee formalizar un nuevo viaje concreto, a partir de un viaje genérico de la lista.
SimpleTicketDetails	Componente que representa la información de un billete de tren. Contiene eventos para comunicación con el componente padre, en caso de que el usuario seleccione "Cancelar", para cancelar un billete.
TicketLitleDetails	Componente que representa la información más importante de un bono.
TicketTripsTable	Componente que contiene la tabla de "Viajes Formalizados" del detalle de un bono mensual. Contiene eventos para la comunicación con el componente padre, en caso de que el usuario seleccione "Cancelar", para cancelar un viaje ya formalizado.
TripDetail	Componente que muestra la información asociada a un viaje concreto. Desencadena un evento cuando el usuario selecciona un viaje genérico, para comunicarle al componente padre cual ha sido el viaje elegido por el usuario.

Tabla 7.11: Lista de componentes de la interfaz de la aplicación de viajeros II

A continuación, se muestra un listado con las páginas de la aplicación de los viajeros. Debemos tener en cuenta, que todas las páginas de la aplicación, están compuestas por un barra de navegación y un menú. Desde el menú, si el usuario anónimo selecciona "Iniciar Sesión", se invocará al modal Login. Dentro de este modal, si el usuario selecciona "Registrarse", se invocará al modal Register.

Páginas	Descripción
Home	Página principal de la aplicación. Cuando el usuario está logueado, ésta página contiene enlaces a: BuyTicketPage, BuyTripPage, UserSimpleTicketsPage y UserTicketsPage. En cambio, cuando el usuario actual es Anónimo, y por lo tanto no se ha autenticado, solo contiene enlaces a BuyTripPage, donde los botones de compra de viajes estarán desactivados.
BuyTicketPage	Página que contiene el formulario que se debe cubrir para comprar un bono. Cuando el usuario finaliza la compra, es redirigido a la página TicketDetailsPage.
BuyTripPage	Página que contiene la lógica relacionada con la compra y formalización de viajes. Está compuesta por los componentes BuyTripForm y TripDetail. En el momento que la página recibe el evento del componente BuyTripForm, realiza la comunicación con el servidor para obtener los viajes segun los parámetros seleccionados por el usuario. Cuando el usuario selecciona un viaje de la lista proporcionada por el componente BuyTripForm, la página lanza el modal BuyTripModal. A continuación, cuando el usuario pulsa "Aceptar" sobre el componente modal anteriormente mencionado, la página realiza la comunicación correspondiente con el servidor y posteriormente redirecciona al usuario a la página UserSimpleTicketsPage o a la página TicketDetailsPage mostrando la información del bono, en caso de que la invocación a esta página venga desde esa misma página.
TicketDetailsPage	Página que contiene la información correspondiente a un bono. Está compuesta por el componente TicketLitleDetails, TicketTripsTable y FrequentTripsTable. Además cuando el usuario selecciona "Cancelar" sobre algún viaje formalizado de la tabla del componente TickeTripsTable, se invoca al modal CancelAlert. Cuando el usuario selecciona "Formalizar Nuevo Viaje", es redireccionado a la página BuyTripPage. Cuando el usuario selecciona un viaje de la tabla del componente FrequentTripsTable, se invoca al modal ConfirmModal.
UserSimpleTicketsPage	Página que muestra la lista de billetes disponibles adquiridos por el usuario. Está compuesta por el componente SimpleTicketDetails. Además la página contiene un menú, donde el usuario podrá visualizar las listas de billetes cancelados y caducados. Cuando el usuario, en la lista de billetes disponibles, seleccione "Cancelar", se invocará al modal CancelModal. Si el usuario selecciona el icono de descarga de un billete, se le descargará el código QR, asociado a dicho billete.
UserTicketsPage	Página que muestra la lista de bonos disponibles adquiridos por el usuario. Está compuesta por el componente . Además la página contiene un menú, donde el usuario podrá visualizar las listas de bonos cancelados y caducados. Cuando el usuario, en la lista de bonos disponibles, seleccione "Cancelar", se invocará al modal CancelModal. Si el usuario selecciona el icono de descarga de un bono, se le descargará el código QR, asociado a dicho billete. Además si selecciona "Más detalles", será redireccionado a la página

Tabla 7.12: Lista de páginas de la interfaz de la aplicación de viajeros I

7.5.2 Componentes y páginas de la aplicación del revisor

A continuación se muestra un listado con las páginas relativas a la aplicación del revisor. Debemos tener en cuenta que además de estas páginas, también contiene las de registro e inicio de sesión, cuyo funcionamiento es idéntico al de la aplicación de viajeros con la única diferencia de que únicamente permite autenticarse a usuarios cuyo rol sea "ROLE_REVIWER".

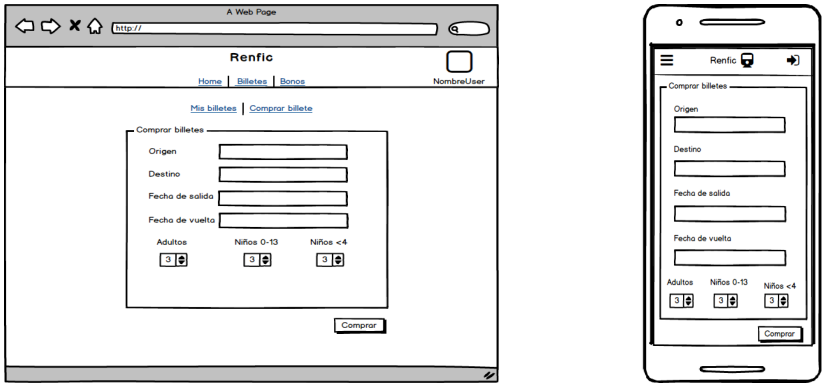
Páginas	Descripción
SelectTripPage	Página que permite al revisor seleccionar el viaje en el que va a realizar la verificación de billetes. Para ello el revisor, deberá seleccionar "Seleccionar viaje". En ese momento le aparecerá un formulario, en donde deberá rellenar el origen, destino y la fecha del viaje. Cuando presione "Aceptar", le aparecerá una lista de viajes asociados a las características que anteriormente indicó. Entre todos ellos deberá seleccionar uno.
SimpleTicketsScannerPage	Página que permite la verificación de un billete. Dispone de un botón "Validar billete", que si selecciona activará el Scanner QR con el que debe validar el billete. Una vez escaneado, le aparecerá en la pantalla la información relativa a dicho billete y un mensaje que le indicará si el billete es válido o no.
TicketsScannerPage	Página que permite la verificación de bono. Dispone de un botón "Validar bono", que si selecciona activará el Scanner QR con el que debe validar el bono actual. Una vez escaneado, le aparecerá en la pantalla la información relativa a dicho bono y un mensaje que le indicará si es válido o no.

Tabla 7.13: Lista de páginas de la interfaz de la aplicación del revisor

7.5.3 Mockups principales de la aplicación para viajeros

A continuación se mostrarán los mockups más importantes de la aplicación de viajeros. Observaremos que cada figura está compuesta por dos subfiguras: en el lado izquierdo, encontramos el mockup correspondiente al despliegue como aplicación web y en el lado derecho, el mockup correspondiente con la aplicación desplegada en móvil.

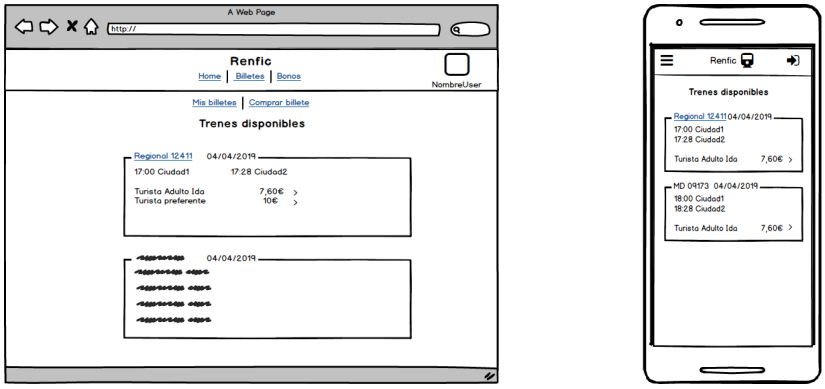
La Figura 7.18 muestra los mockups correspodiente a la compra de un billete. Como podemos observar, muestra un formulario que debe cubrir el usuario, para poder visualizar los viajes del día seleccionado.



(a) Versión web (b) Versión móvil

Figura 7.18: Comprar billete

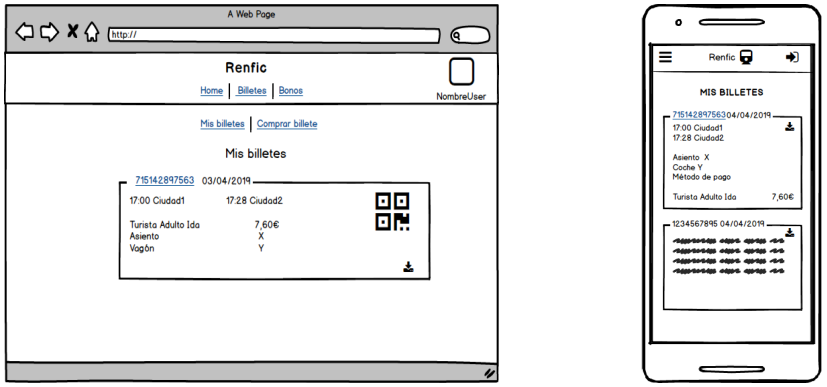
La Figura 7.19, muestra un ejemplo de la lista de trenes resultado de la búsqueda anterior. Como podemos ver aparecería la información básica del tren, para que el usuario pueda decidir que viaje desea escoger.



(a) Versión web (b) Versión móvil

Figura 7.19: Trenes disponibles

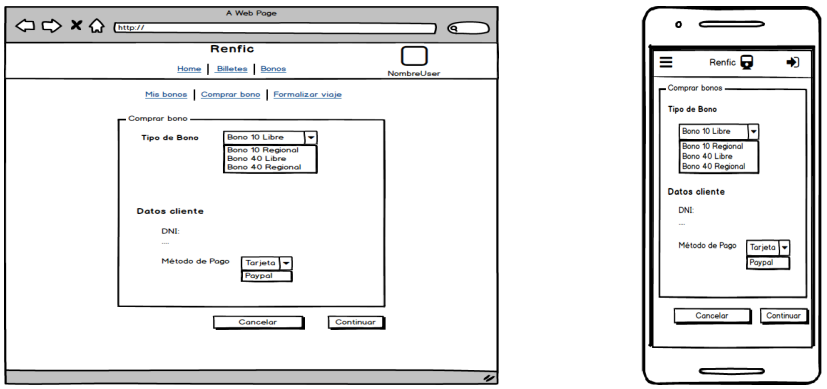
La Figura 7.20, muestra la lista de billetes comprados por un usuario. Podemos observar que tanto en la versión móvil como web, que el usuario tiene la posibilidad de descargar el código QR para llevar su billete sin necesidad de acceder a la aplicación.



(a) Versión web (b) Versión móvil

Figura 7.20: Billetes comprados por el usuario

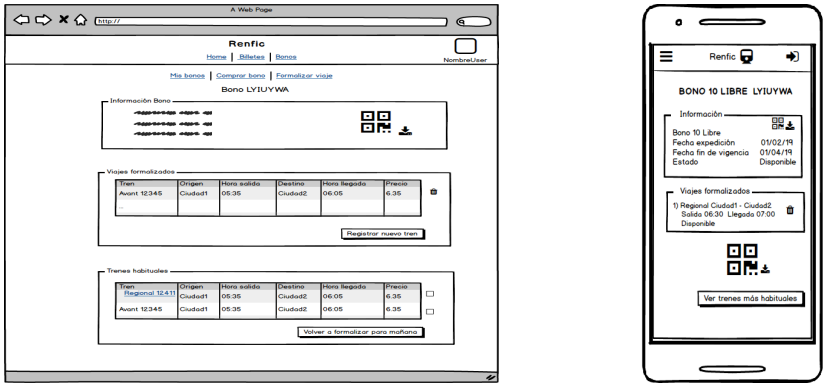
La siguiente figura, Figura 7.21, nos muestra el formulario que el usuario debería cubrir para poder comprar un bono, tanto en versión web como móvil.



(a) Versión web (b) Versión móvil

Figura 7.21: Comprar bono

Los mockups de la Figura 7.22 muestran la diferente información de esa página, y podemos hacernos una idea sobre las listas de viajes formalizados y viajes más frecuentes. Además también podremos descargar el código QR del bono.

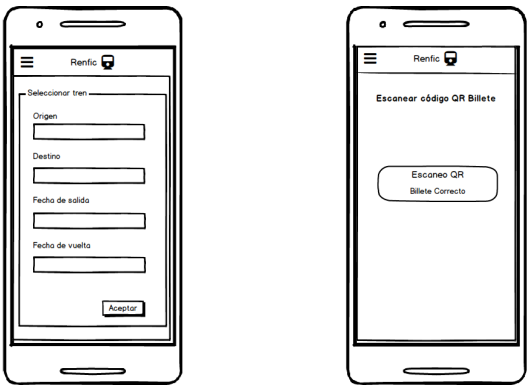


(a) Versión web (b) Versión móvil

Figura 7.22: Información del bono comprado

7.5.4 Mockups principales de la aplicación para revisores

En la Figura 7.23, podemos ver los mockups más importantes de esta aplicación. En la izquierda podemos observar como sería la página de selección de tren, Figura 7.23a ,que se compone de un formulario básico. En el lado derecho, podemos hacernos una idea de como sería el scanner QR Figura 7.23b.



(a) Seleccionar el tren (b) Escanear codigo QR de un billete

Figura 7.23: Mockups aplicación del revisor

Capítulo 8

Implementación y pruebas

8.1 Implementación

8.1.1 Generación código QR, PDF y envío de correo electrónico

Cuando el usuario compra un billete o un abono, el sistema crea un nuevo código QR asociado al mismo, un documento con la información necesaria y envía un correo electrónico al usuario con esta última información. En esta sección comentaremos como se han codificado estos tres pasos.

Para crear el código QR nos hemos basado en la información del ticket. El generador de códigos QR, recibe un JSONObject con los campos que se plasmarán en el QR y un String con el nombre que llevará el archivo QR. El método establece una serie de variables que se usarán en la generación del código (extensión del fichero, path completo, tamaño). Lo primero que se hace es establecer el formato del código QR, como por ejemplo el tipo de caracteres que se codificarán. En nuestro caso, el siguiente paso es firmar el cuerpo del código y añadir la firma resultante en el conjunto de elementos del cuerpo.

En este proyecto para generar los códigos QR, hemos seguido los siguientes pasos:

1. Decidimos todas las variables necesarias para generar el código: contenido de la imagen, path en el que guardaremos la imagen, tipo de la imagen (ṗng), el tamaño de la misma y el nombre que le daremos a la imagen definitiva.

2. Primero creamos una variable de tipo BitMatriz, que representará la imagen en bits antes de almacenarla. A este objeto debemos pasarle, el contenido con el que generaremos el código, especificarle el tipo de código (BarcodeFormat.QR_CODE) y el tamaño.

3. A continuación guardaremos la imagen en memoria a través de un objeto de tipo BufferedImage.

4. Crearemos un objeto de tipo Graphics2D, para generar la imagen en el disco duro. Este objeto nos deja darle formato a la imagen (fondo, tamaño..)

5. Finalmente, a través de ImageIO escribimos la imagen pasándole los parámetros: el objeto imagen, extensión y el path

Un código Qr, puede parecer una secuencia de segmentos aleatorios, pero en realidad tiene una estructura bien definida. Si nos fijamos en la figura Figura 8.1, podremos diferenciar varias partes del código:

1. Patrón de detección de posición (Finder): Estos patrones se encuentran en tres esquinas del código, de manera que indican la posición.

2. Marcas de alineación (Alignment): Se usa para detectar la posición cuando se produce un desplazamiento del código. Si el código QR es grande, este elemento ayuda con la orientación.

3. Margen: Área en blanco al rededor el código, que lo delimita.

4. Patrón de tiempo (timing): Estas líneas al escanear, determina el tamaño de la matriz de datos.

5. Información de versión (Version info): Especifican la versión del código QR. Normalmente la versión del código, va ligada a la densidad del mismo.

6. Información del formato (Format info): Contiene información sobre la toleracia a errores, facilitando el escaneo de datos

7. Datos y claves de corrección de errores: Contiene los datos reales codificados del código QR.

Figura 8.1: Estructura de un código QR

En la imagen **??**, podemos ver como al escanear el código, captura los patrones de detección de posición y la marca de alineación. De esa manera, una vez los tiene ubicados, ya sabe donde está el conjunto de datos.

Figura 8.2: Código QR en el momento de escanearlo

En cuanto a la firma digital, debemos tener en cuenta, que para realizar dicha firma, fue necesario generar un par de claves con GPG (GNU Privacy Guard). Después, se exportó tanto la clave privada como la pública, añadiéndolas al proyecto. Los comandos utilizados para la generación de las claves y exportación de las mismas, fueron los siguientes:

```
###GENERAR CLASES
  #gpg --gen-key
###EXPORTAR CLAVES
  #gpg --armor --output flashticket-pub.asc --export
 1E7EAE6A60467F5A96818E44A104A8B03A36A0DE
  #gpg --armor --output flashticket-priv.asc --export
 --secret-key cristinoya@gmail.com
###LISTAR CLAVES
  #gpg --list-keys
  #gpg --list-secret-keys
```

Una vez generado el código QR, se compone el fichero ".pdf" que será enviado al usuario a través de un correo electrónico, en caso de que así lo haya indicado. Para poder generar el pdf, inicialmente debemos componer el path del mismo. A continuación crearemos un Out-

putStream que será el que guarde el fichero en el path anteriormente indicado. En este caso, se ha decidido que el archivo constará de un título con el nombre de la aplicación **FlashTicket**, un subtítulo con el texto **"Información de la compra"**, una tabla con la información más relevante del ticket y el código QR del mismo.

Para finalizar este proceso, el sistema enviará un correo electrónico al usuario con el PDF anteriormente generado. Para ello debemos pasarle al método el correo destinatario, el asunto, el texto que se añadirá al cuerpo y una lista de ficheros adjuntos. De esa manera compondrá el correo electrónico que recibirá el usuario.

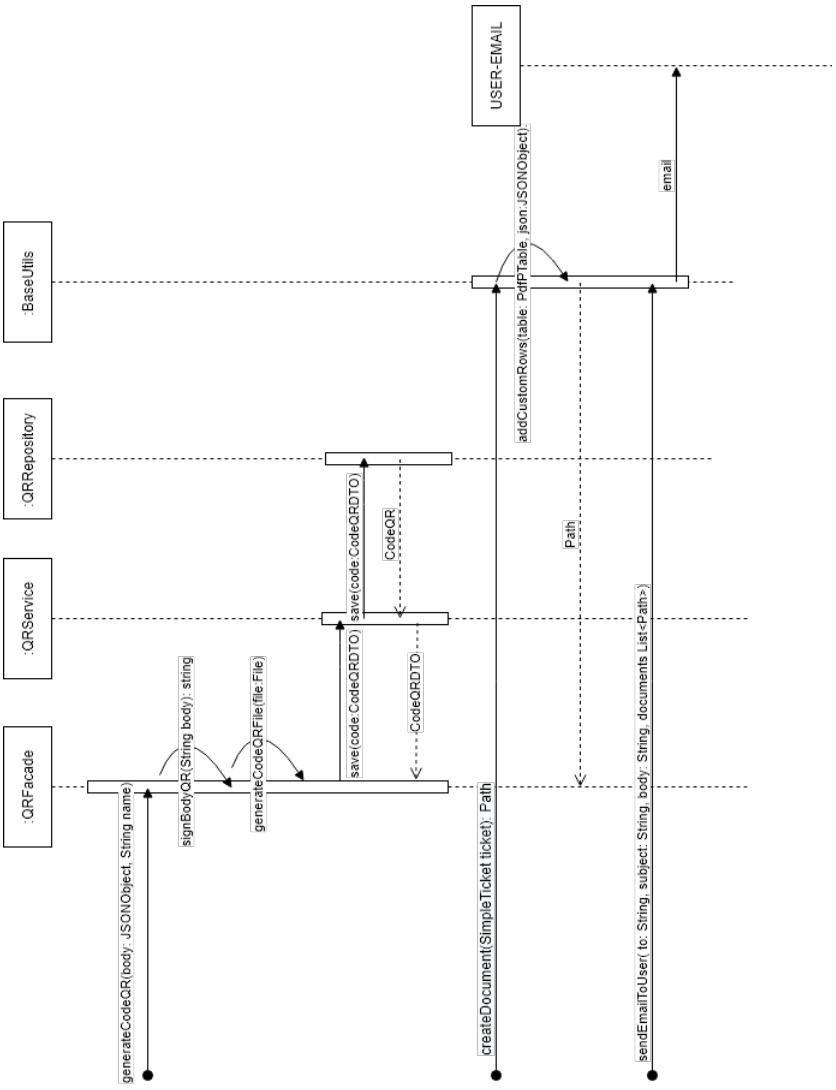


Figura 8.3: Diagrama del proceso de generación de QR, PDF y envío de email

8.1.2 Autenticación mediante JWT

En este proyecto se hace uso de JWT para autenticar al usuario. Recordemos que un token es una cadena de caracteres alfanumérica. Concretamente un JWT (JSON Web Token), es un tipo de token con un estructura concreta que puede ser desencriptada por el servidor, y de esta forma obtener las credenciales de usuario y autenticarnos en la aplicación.

Un token de tipo JWT tiene una estructura dividida en tres secciones separadas por un punto ".":

- **Cabecera:** almacena el tipo de token y el algoritmo de encriptado que se usó.
- **Payload:** Es el cuerpo del token y contiene los datos que identifican al usuario. En esta parte del token podemos incluir lo que nosotros deseemos
- **Firma:** Firma digital de las partes anteriores, que sirve para validar que el contenido no haya sido alterado.

En la figura Figura 8.4 se puede ver un ejemplo de las partes de un token de tipo JWT.

Figura 8.4: Estructura JWT

La página web `https://jwt.io/`, nos permite decodificar un token JWT, y ver sus diferentes partes. En la figura Figura 8.5 se puede ver un ejemplo de un JWT usado en esta aplicación.

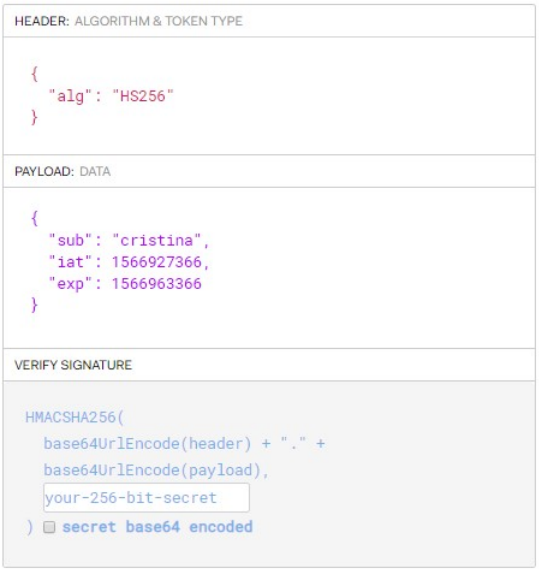


Figura 8.5: Decodificación de JWT

El proceso de autenticación mediante JWT usado en el proyecto es el siguiente. Cuando el usuario se loguea en nuestra aplicación con su login y contraseña, el servidor valida su correcta autenticación, forma el token y lo devuelve a la aplicación cliente. De esa manera todas las peticiones que se realicen al servidor a partir de ese momento, deben incluir el JWT en la cabecera **Authentication**, para que el servidor pueda validarlo y desencriptarlo En caso de que el token no sea válido o no exista, el servidor devolverá un error 401 NOT AUTHORIZED, pero si es válido, permite realizar la petición deseada y devolver la response de la misma.

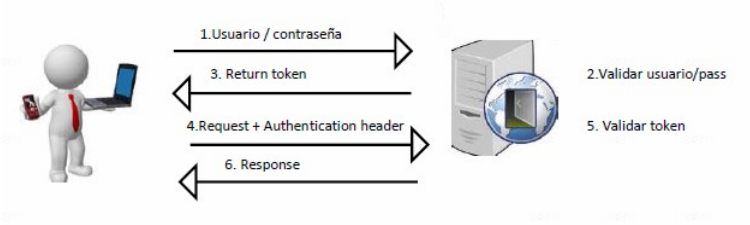


Figura 8.6: Diagrama de flujo de autenticación JWT

8.1.3 MapStructs

MapStructs es una herramienta que permite realizar traducciones entre objetos de manera rápida y eficaz. En nuestro caso, hemos usado MapStructs para las traducciones entre Entidades, DTOs y TOs.

Para usar MapStructs es necesario implementar al menos una interfaz anotada con **@Map-**

per, anotación que incorpora la biblioteca de MapStructs. En esa anotación debes indicar el "componentModel", en este caso spring y las clases de tipo Mapper que debería usar a mayores. Esto último es necesario porque si un objeto tiene un atributo cuyo tipo es otro objeto, mapstructs necesita su mapper para poder realizar la traducción completa. Dentro de esa interfaz, debes añadir las firmas de los métodos que deseas que MapStructs implemente.

```
1 @Mapper(componentModel = "spring", uses = {
      StationMapper.class })
```

Por ejemplo, en esta aplicación tenemos una interfaz mapper por cada entidad en ella añadimos las firmas de los métodos de traducción entre la Entidad, DTO y TO, **??**

```
@Mapper(componentModel = "spring")
public interface CodeQRMapper {

    CodeQR toEntity(CodeQRDTO qr);

    CodeQRDTO toDTO(CodeQR qr);

    CodeQRTO toTO(CodeQRDTO qr);

    CodeQRDTO toDTO(CodeQRTO qr);
}
```

Figura 8.7: Ejemplo de interfaz Mapper

8.1.4 Manejador de excepciones

A lo largo del proyecto se han creado excepciones nuevas, para casos concretos. Para poder gestionarlas, se crea un manejador de excepciones, que extenderá la clase ***"ResponseEntityExceptionHandler"*** e incorporará la anotación ***@RestControllerAdvice***.

La anotación *@RestControllerAdvice*, nos permite manejar las excepciones a través de un único manejador. Además, nos proporciona control sobre el cuerpo y el código HTTP de la respuesta y permite mapear más de una excepción con un mismo código HTTP.

En este proyecto se ha creado una clase llamada *RestResponseEntityExceptionHandlerController*, que extiende la clase *ResponseEntityExceptionHandler* y está anotada con *RestControllerAdvice*.

```
@ExceptionHandler(value = { InputValidationException.class, InvalidPreviousStateException.class,
        InvalidUserDNIException.class, UserAlreadyExistException.class, TripAlreadyExistException.class})
protected ResponseEntity<Object> handleBadRequest(Exception ex, WebRequest request) {
    String bodyOfResponse = ex.getMessage();
    return handleExceptionInternal(ex, bodyOfResponse, new HttpHeaders(), HttpStatus.BAD_REQUEST, request);
}
```

Figura 8.8: Como manejar las excepciones en Spring

A continuación mostraremos una tabla, con las excepciones usadas en esta aplicación, su

código y descripción correspondiente.

Excepción	Código HTTP	Descripción
InvalidChangeStateException	400	Excepción que se lanza cuando se intenta cambiar a un estado de un bono o viaje, desde un estado anterior inadecuado.
InputValidationException	400	Argumentos de la petición inválidos.
TripAlreadyExistException	400	El viaje que se intenta formalizar en el bono, ya ha sido formalizado previamente.
InvalidUserDNIException	400	El DNI del usuario no es válido
UserAlreadyExistException	400	Excepción que se lanza cuando ya existe un usuario con el mismo login, DNI o email.
InvalidLoginException	401	El login es incorrecto
Unauthorized	401	Petición no autorizada
InstanceNotFoundException	404	La instancia del objeto no existe.
CompleteTrainException	404	Excepción de tren completo.
CompleteTicketException	404	Excepción que se produce cuando no se pueden formalizar más viajes con dicho bono.

Tabla 8.1: Tabla con la excepciones usadas en el proyecto

En las aplicaciones clientes, lo que se hace es capturar la excepción a través del código HTTP recibido. Como hemos mencionado en el Apartado 7.4.2, las respuestas del servidor se reciben en la aplicación cliente mediante objetos observables, usando el método **".subscribe"**. Este método maneja tres tipos de notificaciones: *next*, devuelve el objeto de retorno en caso de que todo vaya bien; *error*, devuelve una notificación de error; *complete*, controlador para notificación de ejecución completa.

En este caso las excepciones las tratamos a través del método *error*, obteniendo el código y el mensaje.

```
this.ticketService.cancelarviaje(this.ticket.id, this.loginCurrentUser, trip.id).subscribe(
  result => {
    window.location.reload();
  },
  error => {
    const alert = {
      id: 1,
      type: 'danger',
      strong: '¡Error!',
      message: error.error.message,
      icon: 'ni ni-support-16'
    } as IAlert

    const modal = this.modalService.open(ErrorModalComponent);
    modal.componentInstance.alert = alert;
  }
);
```

Figura 8.9: Como manejar las excepciones en Angular

8.1.5 Caso de compra de un billete

En esta sección comentaremos el flujo del caso de uso de compra de un billete. Este caso de uso, recordemos que se activa en la aplicación final de los viajeros, cuando el usuario cubre el formulario de buscador de viajes, selecciona uno y cubre los datos necesarios para la compra del mismo (asiento, método de pago y número de billetes deseados). Cuando el usuario selecciona aceptar, la página se comunica con el servicio **SimpleTicketService** y este a su vez le manda la petición **POST HTTP** al servicio.

Esta petición entrará por el controlador **SimpleTicketController**, que inicialmente se encarga de realizar una serie de validaciones sobre los datos recibidos (usuario válido, estaciones seleccionadas válidas, viaje seleccionado válido). Una vez hechas todas la validaciones, se comunica con la fachada **SimpleTicketFacade**, que se encarga de realizar una serie de operaciones tantas veces como número de billetes haya solicitado el usuario.

Lo primero que hace la fachada es comprobar si el usuario seleccionó un asiento, en caso contrario, se comunica con la fachada de viajes concretos **ActualTripFacade**, pidiéndole que le indique un asiento válido y libre para asignar a ese billete. Ésta selección se realiza al azar entre todos los asientos disponibles en ese viaje. A continuación, se vuelve a comunicar con la fachada ActualTripFacade, para comunicarle que añada a su lista de asientos ocupados, el asiento asignado a este billete. Una vez finalizado este proceso se procede a generar el código QR correspondiente al ticket y por ende firmarlo digitalmente. Después se crea el documento asociado al mismo y se guarda el ticket, comunicándose con el servicio **SimpleTicketService** y este a su vez con el repositorio **SimpleTicketRepository**.

Finalmente, se envía un correo electrónico cuyo destinatario es el email del usuario que compra el billete y al que se le añade como ficheros adjuntos los pdf de los billetes comprados.

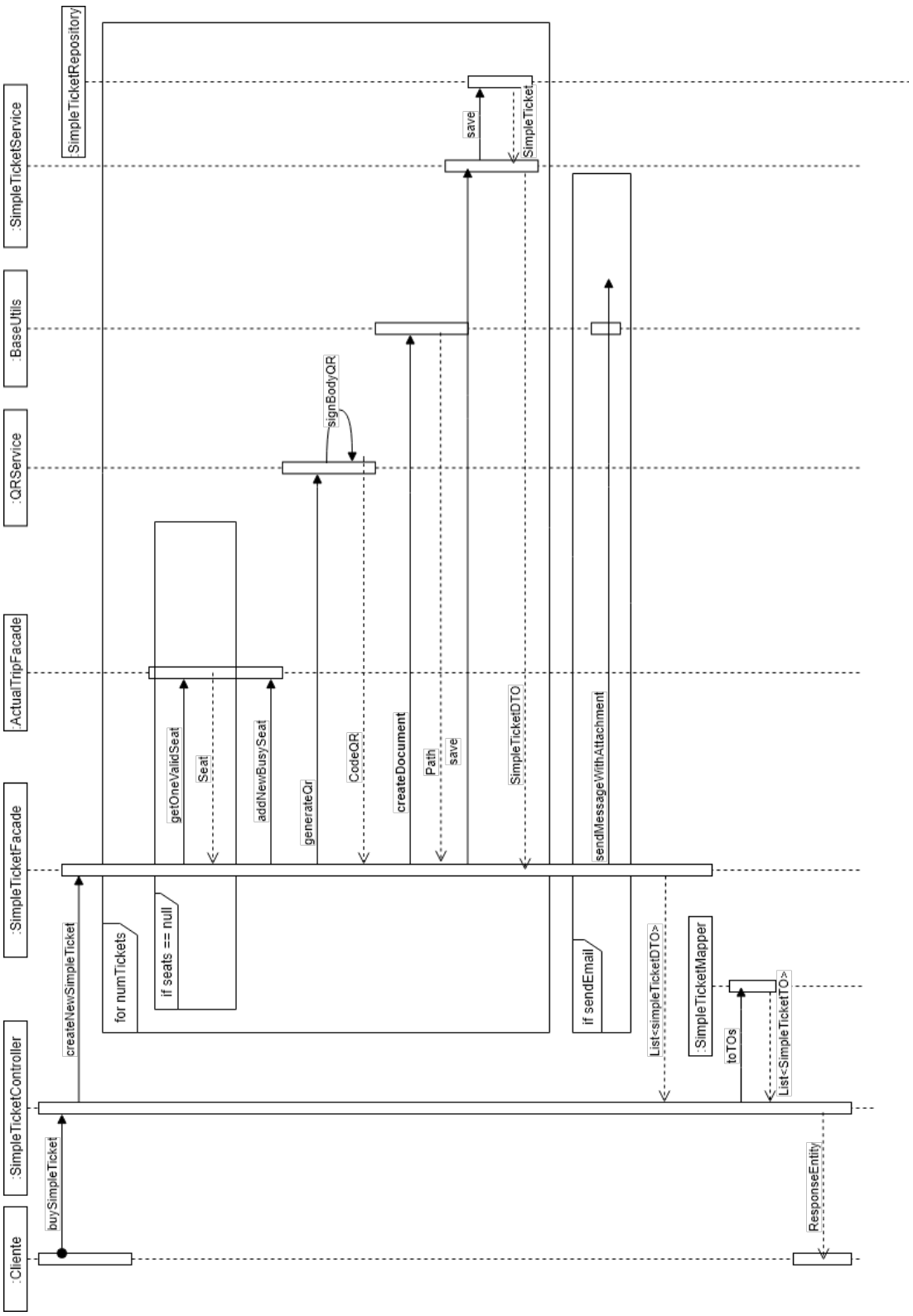


Figura 8.10: Diagrama de flujo de la compra de un billete

8.2 Pruebas

Las pruebas de la implementación del servicio, se realizaron mediante JUnit y Postman.

8.2.1 JUnit

Las pruebas unitarias, se realizan para comprobar el correcto funcionamiento de la capa de negocio. Para realizar estas pruebas nos apoyaremos en las ventajas que nos proporciona **Spring Boot Test** [30].

Para empezar debemos etiquetar nuestras clases test con las siguientes anotaciones:

- **@RunWith(SprinRunner.class)**: Se usa para indicar que la clase de prueba, usará las características tanto de Spring Boot como de JUnit. Siempre que usemos JUnit con Spring Boot se requerirá esta anotación

- **@SpringBootTest**: Esta anotación se usa cuando necesitamos hacer uso de nuestro archivo de propiedades y del contexto de spring.

Además de las anotaciones anteriormente mencionadas, Spring Boot nos proporciona otra adicional a nivel de método que nos permite simular un usuario. La anotación es @WithMockUser, y nos permite simular un usuario y personalizar sus características: authorities, contraseña, roles, nombre de usuario.

8.2.2 Postman

Con Postman hemos llevado a cabo las pruebas del API REST. Es una herramienta que permite realizar peticiones HTTP a cualquier API, configurando los parámetros que necesitemos (método de petición, cuerpo,cabeceras,token,parámetros)

Para hacer las pruebas con PostMan, primero es necesario crear un usuario, como se refleja en la Figura 8.11

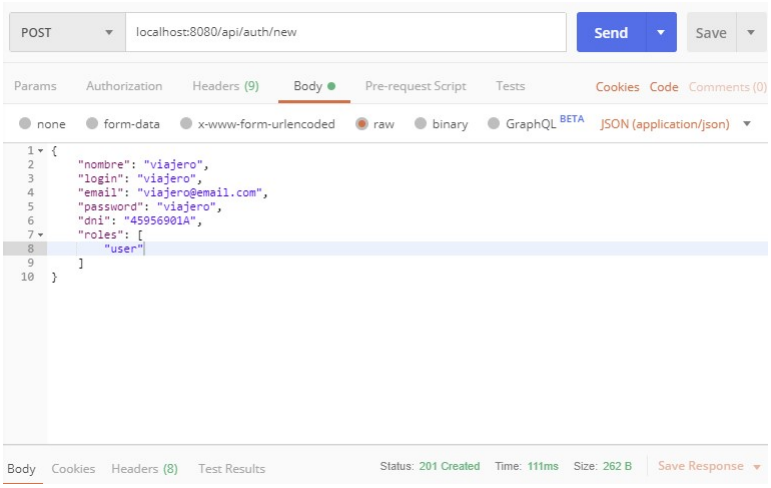


Figura 8.11: Registro con Postman

A continuación nos tenemos que autenticar, para que nos devuelva el token JWT, que useramos para realizar el resto de peticiones. En la Figura 8.12, vemos un ejemplo de login.

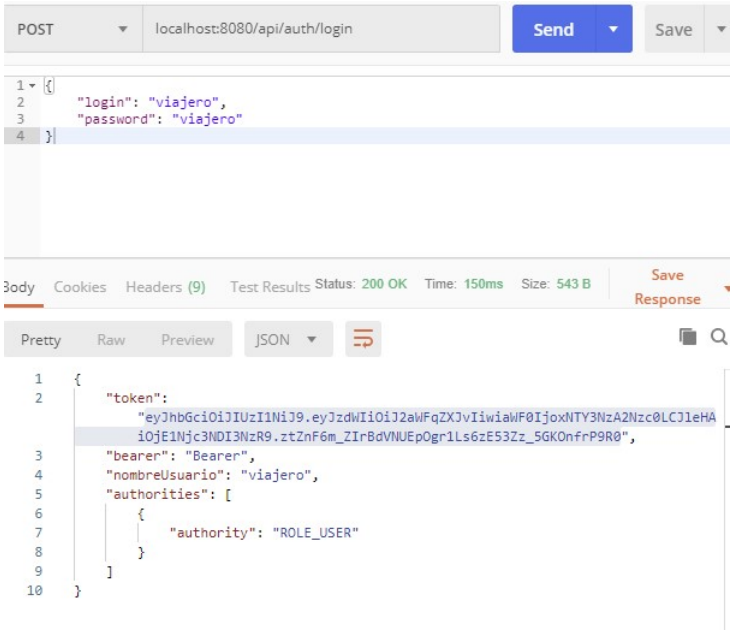


Figura 8.12: Login con Postman

Ahora ya podemos realizar cualquier petición al servidor siempre y cuando añadamos en la cabecer Authorization el JWT devuelto anteriormente.

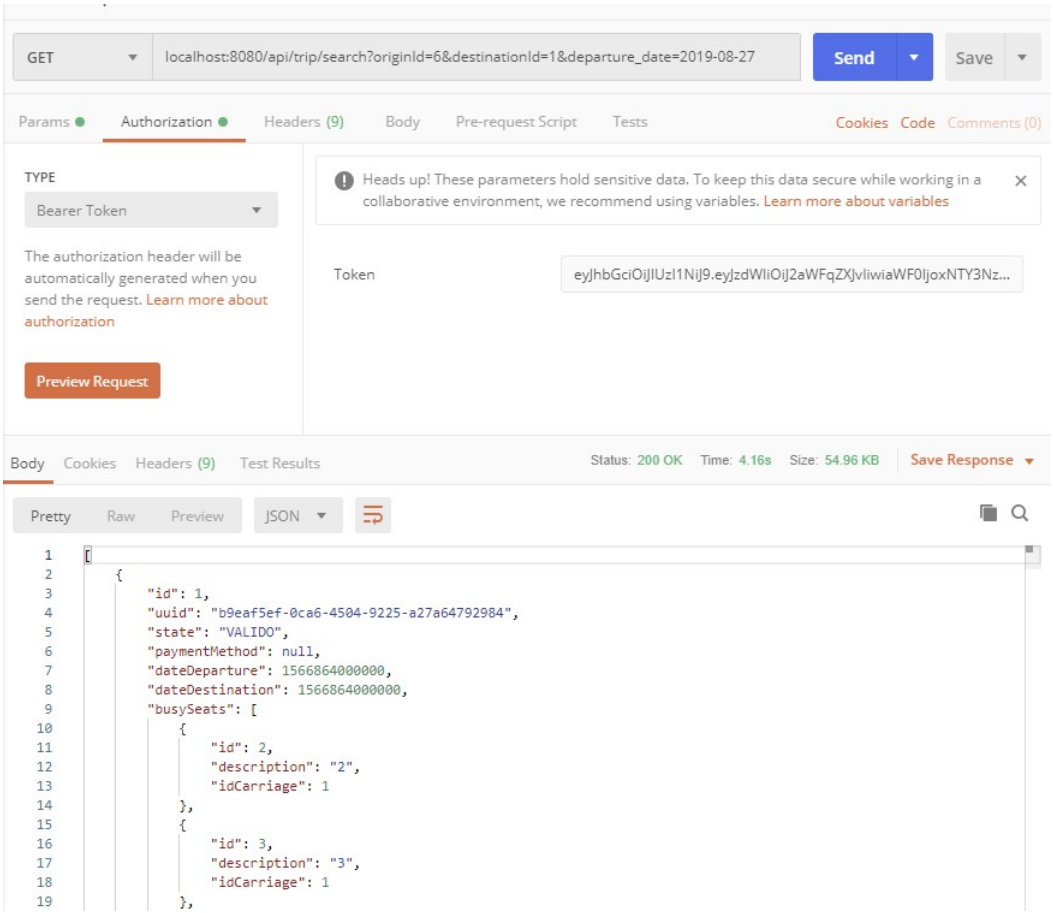


Figura 8.13: Operación get con Postman

8.2.3 Pruebas de integración y aceptación

Estas pruebas se han llevado a cabo, probando los diferentes casos de uso de ambas aplicaciones finales. De esta manera, se comprueba que las operaciones tanto a nivel front-end como back-end funcionan correctamente. Por tanto, se ha probado a comprar billetes, bonos y formalizar viajes, a cancelar billetes, bonos y viajes formalizados, a autenticar usuarios y a validar los códigos QR de los tickets.

Capítulo 9

Conclusiones y trabajo futuro

9.1 Conclusiones

Una vez finalizado el proyecto, podemos decir que los objetivos iniciales han sido gratamente alcanzados. Todas las funcionalidades planteadas fueron desarrolladas satisfactoriamente.

Por un lado, se ha conseguido crear la estructura de una aplicación que permita a los usuarios habituales de trenes y metros, viajar con comodidad y gestionar sus billetes de manera más eficaz. Además se introdujo la gestión de bonos, mediante códigos QR digitales que la aplicación del revisor pudiese validar.

Por otro lado, a nivel académico se han ampliado y afianzado muchos de los conocimientos impartidos a lo largo de estos cuatro años. Además, se han conocido nuevas tecnologías y frameworks, que no se han visto durante la carrera, como puede ser Ángular y Spring boot.

Como autora de este proyecto, estoy orgullosa del trabajo realizado estos últimos meses para llevar a cabo una idea, que desde hace unos años he deseado hacer realidad. Después de haber sido usuaria del ferrocarril casi diariamente, se echan en falta ciertas comodidades que hoy en día todavía no se contemplan. Lo que nunca pensé es que yo pudiese llevar a cabo ese desarrollo. Queda mucho camino por delante, y la versión actual solo es el principio, no es una versión que pueda salir al mercado ya que se basa en datos ficticios. Pero este camino no se ha acabado todavía, por eso en el Apartado 9.2, os podréis hacer una idea de que funcionalidades se incorporarán próximamente en la aplicación.

9.2 Trabajo futuro

Este proyecto se ha comenzado con el objetivo de continuar su desarrollo, aumentando el alcance anteriormente delimitado. Es un proyecto muy interesante que puede abarcar muchos escenarios y actores de la vida real. Por lo tanto, algunos de las mejoras serán las siguientes:

- **Crear una aplicación de administración**: En esta aplicación, se podrá gestionar todo lo referente a los trenes, estaciones, viajes y bonos genéricos. Además desde este panel, permitiremos que se puedan cancelar viajes comprados por otros usuarios. De esta manera, estamos representando el papel de administrador.
- **Lista de viajes favoritos**: En esta aplicación se ha gestionado en los bonos, una lista de viajes más frecuentados para que al cliente habitual le sea más sencillo formalizar diariamente nuevos viajes. A mayores, esta funcionalidad se ampliará para toda la aplicación, permitiendo que un usuario pueda seleccionar un viaje como favorito y posteriormente eliminarlo de su lista de favoritos. Facilitando de esa manera no solo formalizar viajes, sino comprar billetes.
- **Validación de asientos:** Un revisor de un tren que tiene varias paradas, necesita controlar quien sube y baja y quien baja en cada parada, y que asientos debe validar de nuevo entre esas paradas. Por lo tanto, se creará una funcionalidad que permitirá al revisor, saber por que asientos debe volver a pasar para validar el billete.
- **Tarjeta de fidelización:** Actualmente en la aplicación no se está llevando a cabo ningún sistema de puntos. Pues otro de los pasos a llevar a cabo, es añadir una tarjeta de fidelización digital que el usuario podrá solicitar si lo desea, en la que se vayan acumulando automáticamente puntos a medida que el usuario realiza compras. De manera que esos puntos se puedan canjear en nuevos billetes.
- **Descuentos:** Otra ampliación necesaria, es la de descuentos en billetes y bonos. Hoy en día existen múltiples descuentos a la hora de viajar, ya sea por edad o por condiciones especiales, se desarrollará una funcionalidad que permita al usuario seleccionar de entre varios posibles descuentos (estudiante, mayores de 65, menores de 5, discapacidad...).
- **Notificaciones de retrasos y cancelaciones:** Hoy en día si quieres saber si un viajes se canceló o se retrasó debes acudir a la estación de partida del mismo, y observar el panel de horarios. En esta aplicación, se añadirá una nueva funcionalidad en donde, el usuario recibirá notificaciones por retrasos y cancelaciones de viajes, concediéndole más tiempo de decisión ante esa nueva circunstancia eventual.
- **Gestión de compras a través de medios de pago electrónicos reales:** Después de ampliar y completar ambas aplicaciones, se añadirá la posibilidad de gestionar compras reales de billetes. De esta manera, la aplicación estará completamente preparada para salir al mercado y aplicarse en escenarios como los comentados en el Capítulo 2.

Apéndices

Apéndice A

Manual de usuario

En este capítulo se mostrará la solución final desarrollada de la aplicación.

A.1 Acceso a la aplicación usuario anónimo

Al acceder a la aplicación, el usuario sin autenticación previa, podrá visualizar un menú donde únicamente podrá **Iniciar Sesión** o **Consultar horarios**.

Además en el cuerpo de la página aparecerá una bienvenida, y un "Widget" que le permitirá realizar la misma acción de consulta de horarios. Esto lo podemos ver en la Figura A.1.

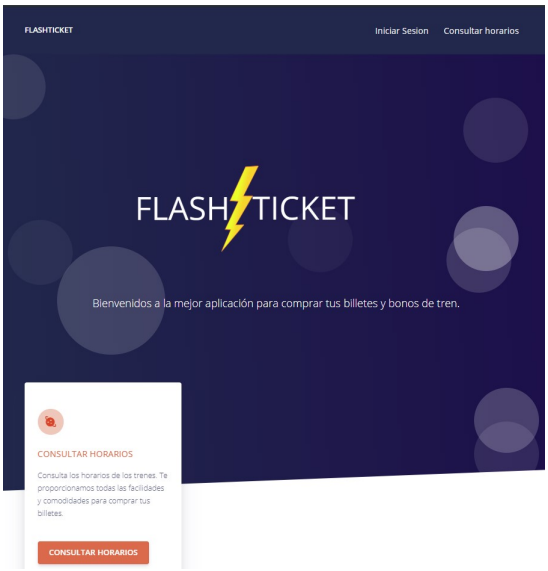


Figura A.1: Acceso a la aplicación

Si el usuario decide acceder al buscador de viajes, le aparecerá un formulario como el de la Figura A.2, en donde podrá seleccionar origen, destino y la fecha concreta de la que desea conocer los viajes disponibles.

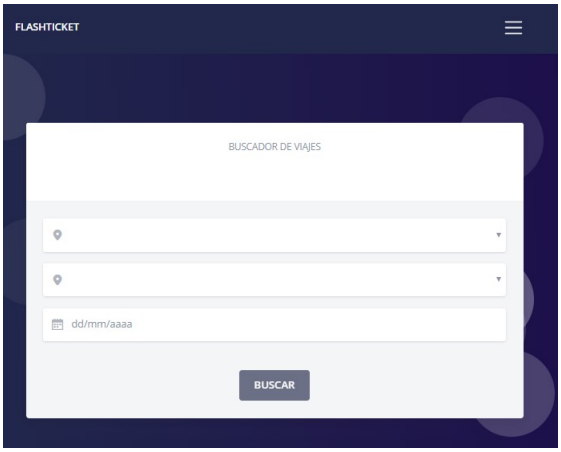


Figura A.2: Página buscador de viajes usuario anónimo I

Al realizar la búsqueda, al usuario le aparece un conjunto de trenes disponibles, con los detalles más importantes para el usuario.

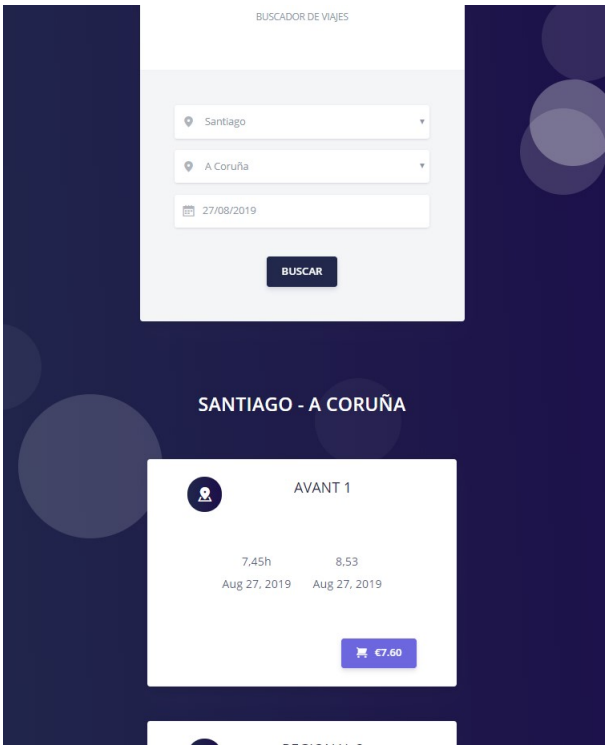


Figura A.3: Página buscador de viajes usuario anónimo II

Recordemos que estamos en el escenario en el que el usuario no está autenticado, por lo tanto, si pulsa el botón de compra de algún tren, le aparecerá la ventana de login, Figura A.4, recordándole que para poder realizar esta acción necesita estar autenticado previamente.

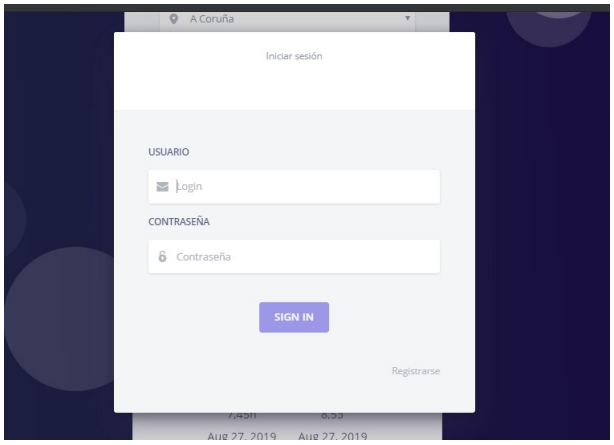


Figura A.4: Página buscador de viajes usuario anónimo III

A.2 Proceso de autenticación

El usuario podrá autenticarse en cualquier momento, seleccionando el botón "Iniciar sesión", de la aplicación, Figura A.5

Figura A.5: Iniciar sesión en la aplicación I: Botón "Iniciar Sesión"

En ese momento, aparecerá una ventana emergente en donde el usuario podrá cubrir el formulario correspondiente al inicio de sesión: login y contraseña. Figura A.6

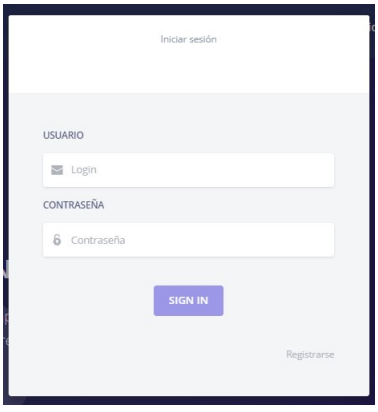


Figura A.6: Iniciar sesión en la aplicación II: Ventana emergente

Si el usuario inserta las credenciales incorrectas, le aparecerá un mensaje en el login avisándole del error. Podemos ver el mensaje concreto, en la Figura A.7

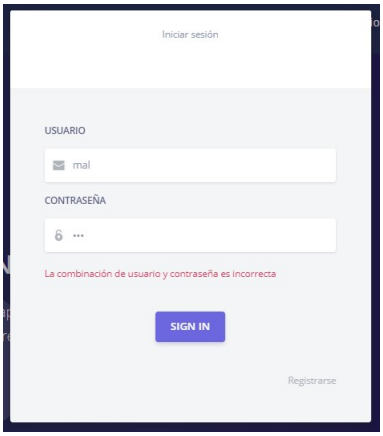


Figura A.7: Iniciar sesión en la aplicación III: Credenciales incorrectas

En caso de que le usuario no se haya registrado todavía, podrá hacerlo seleccionado el botón "Registrarse", que se encuentra en la ventana emergente de login. En ese momento, se cerrará la ventana emergente de login, y se abrirá la de registro.

En esa pantalla, el usuario deberá cubrir todos los campos del formulario para poder registrarse.

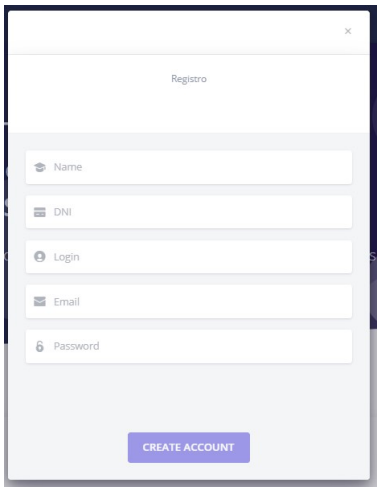


Figura A.8: Registrarse en la aplicación I: Ventana emergente

Si el usuario inserta el campo login, dni o email que le corresponde a un usuario ya registrado en la aplicación, se mostrará un mensaje especificando el campo que debe modificar. Podemos ver un ejemplo de este error en la Figura A.9, en donde se refleja que el usuario con dicho login ya existe.

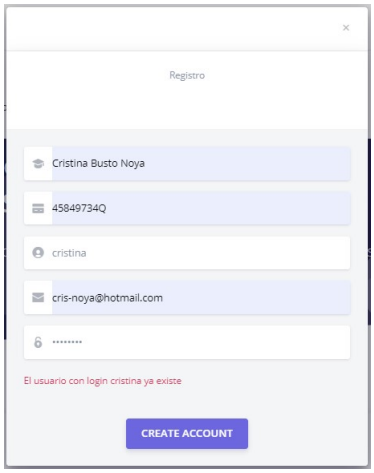


Figura A.9: Registrarse en la aplicación II: Error login ya existente

A.3 Acceso a la aplicación de los viajeros

En el momento que el usuario se registra correctamente e inicia sesión en la aplicación, la vista de la página principal se verá modificada. En el menú aparecerán nuevos enlaces, entre los que se encuentra su nombre de login con un desplegable, Figura A.10. y en el cuerpo de la página, podrá visualizar nuevos widgets. Podemos ver en la figura Figura A.11, como el usuario visualizaría dicha página.

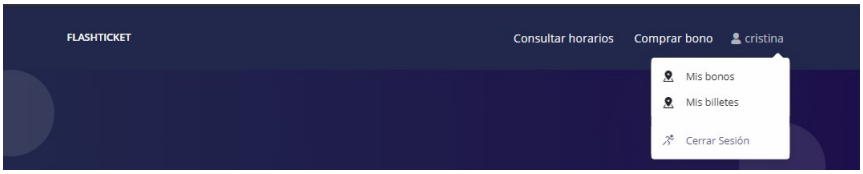
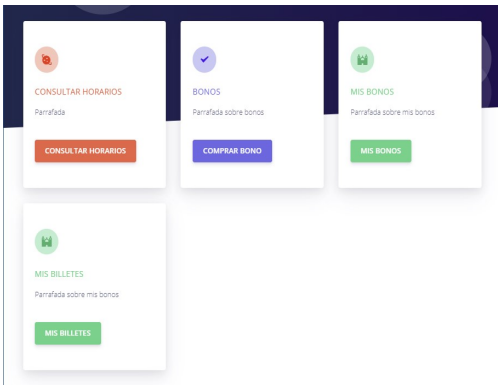


Figura A.10: Menú de la aplicación cuando el usuario ha sido autenticado

(a) Página principal I



(b) Página principal II

Figura A.11: Página principal de la aplicación cuando el usuario se ha registrado.

A.4 Gestión de billetes

En el Apéndice A.1, hemos visto como es la página de buscador de viajes, y que era necesario que el usuario estuviese autenticado para poder comprar un billete. Continuando desde ese punto y con el usuario ya autenticado, si selecciona un viaje le aparecerá una ventana emergente,Figura A.12. En esa ventana, el usuario deberá seleccionar el método de pago y el número de billetes que desea comprar. A mayores podrá decidir si desea recibir un correo electrónico con la información de la compra realizada e indicar vagón y asiento.

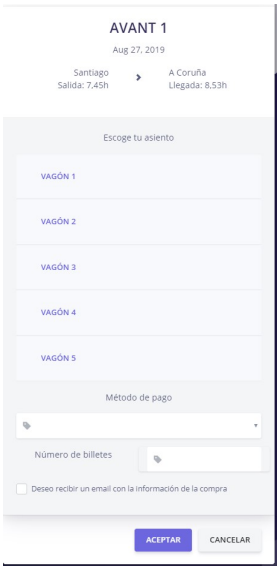


Figura A.12: Ventana de cumplimentación de datos y confirmación de compra

Cuando el usuario confirme la compra, será automáticamente redireccionado a la página de lista de billetes del usuario.

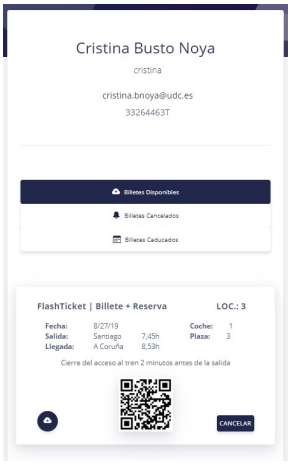
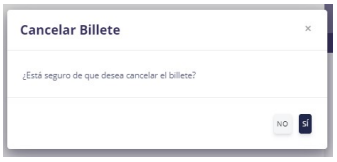
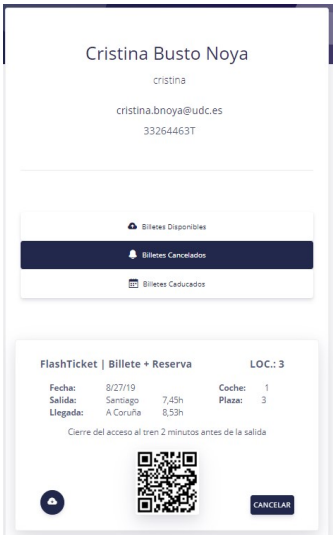


Figura A.13: Página de lista de billetes del usuario

El usuario si lo desea puede cancelar un billete comprado, para ello tendrá que pulsar el botón “Cancelar”. En ese momento, le aparecerá una ventana emergente pidiendo la confirmación del usuario para realizar la cancelación. En el momento que confirme, la página se recargará y el billete aparecerá ahora en la lista de billetes cancelados.



(a) Ventana de confirmación de cancelación



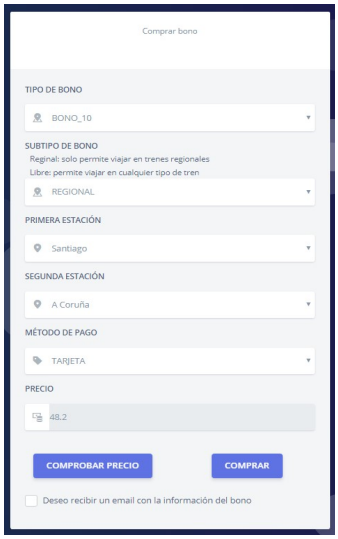
(b) Lista de billetes cancelados

Figura A.14: Anulación de un billete

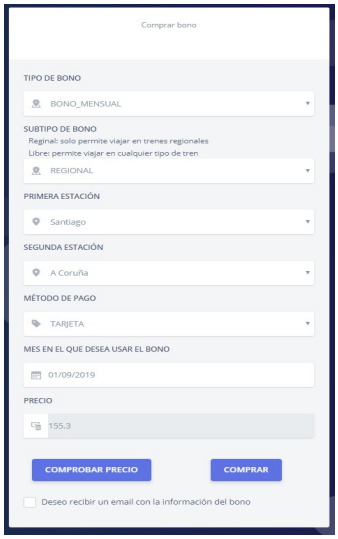
A.5 Gestión de bonos

Si el usuario desea comprar un bono, accederá a una página donde tendrá que cubrir los campos necesarios para dicha compra: tipo de bono, subtipo de bono, estaciones entre las que desea viajar y método de pago. En el caso de seleccionar como tipo de bono "BONO MENSUAL", le aparecerá un nuevo campo obligatorio en el que debe seleccionar un día cualquiera del mes en el que desea usar el bono. Además, al igual que en la compra de billetes, el usuario podrá recibir un correo electrónico con la información de la compra.

Si nos fijamos en la Figura A.15, veremos que además de los campos comentados anteriormente, también tenemos el campo "Precio". En ese campo se mostrará el precio que valdría la compra del bono con los parámetros seleccionados. Para ello, primero el formulario debe estar completamente cubierto y el usuario podrá seleccionar "Consultar precio".



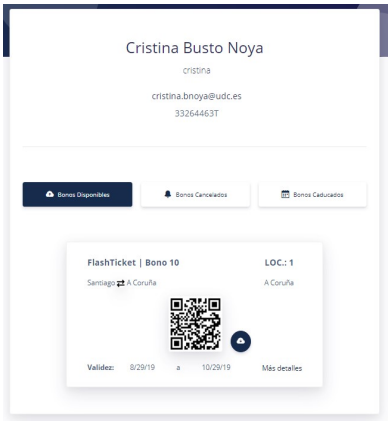
(a) Formulario de compra bono 10



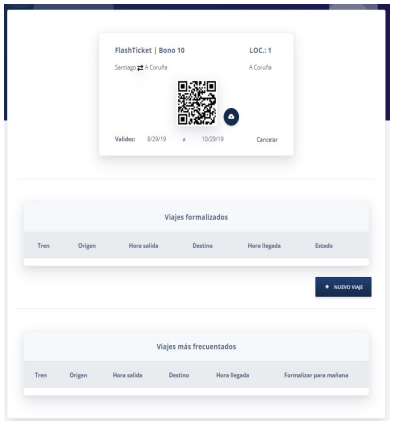
(b) Formulario de compra bono mensual

Figura A.15: Formulario de compra de bonos

Una vez realizada la compra, el usuario será redirigido a la página de lista de bonos del usuario. Al igual que en el caso de los billetes, también se podrá consultar los bonos cancelados y caducados además de los disponibles.



(a) Página de lista de bonos del usuario



(b) Página con los detalles de un bono

Figura A.16: Lista de bonos y detalle de un bono

Si el usuario lo desea, podrá acceder a los detalles del bono seleccionando “Más detalles”. Nos aparecerá una página como la de la Figura A.16b, en donde podemos ver dos listas: la de viajes formalizados y la de viajes más frecuentados. Si nos fijamos en los detalles del viaje, veremos el botón cancelar, de esa manera si el usuario lo desea podrá cancelar el bono y

consecuentemente todos los viajes formalizados en el mismo.

Si el usuario desea formalizar un viaje tendrá que seleccionar “Nuevo viaje” y le aparecerá un formulario como el de la figura Figura A.17, en donde podrá decidir el sentido del viaje y la fecha concreta.

Figura A.17: Formulario para formalizar un nuevo viaje

Cuando el usuario formaliza el viaje, aparecerá en la lista de viajes disponibles del bono y además también aparecerá un nuevo viaje frecuente.

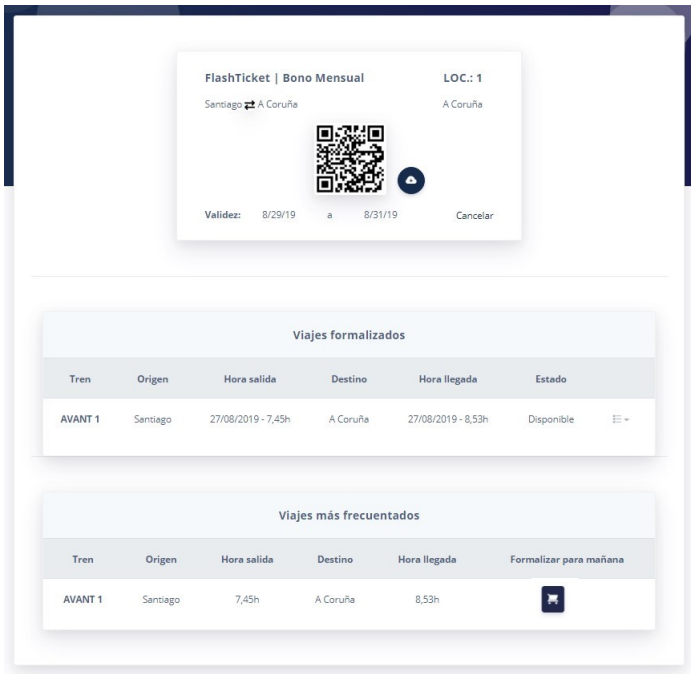


Figura A.18: Detalles de un bono con un viaje comprado

En la lista de viajes formalizados, el usuario podrá cancelar un viaje si lo desea. En ese momento le aparecerá una ventana emergente pidiendo confirmación de la acción para finalmente cambiar el estado del viaje.

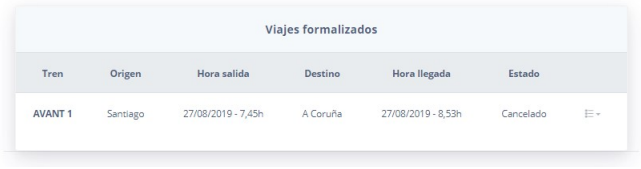


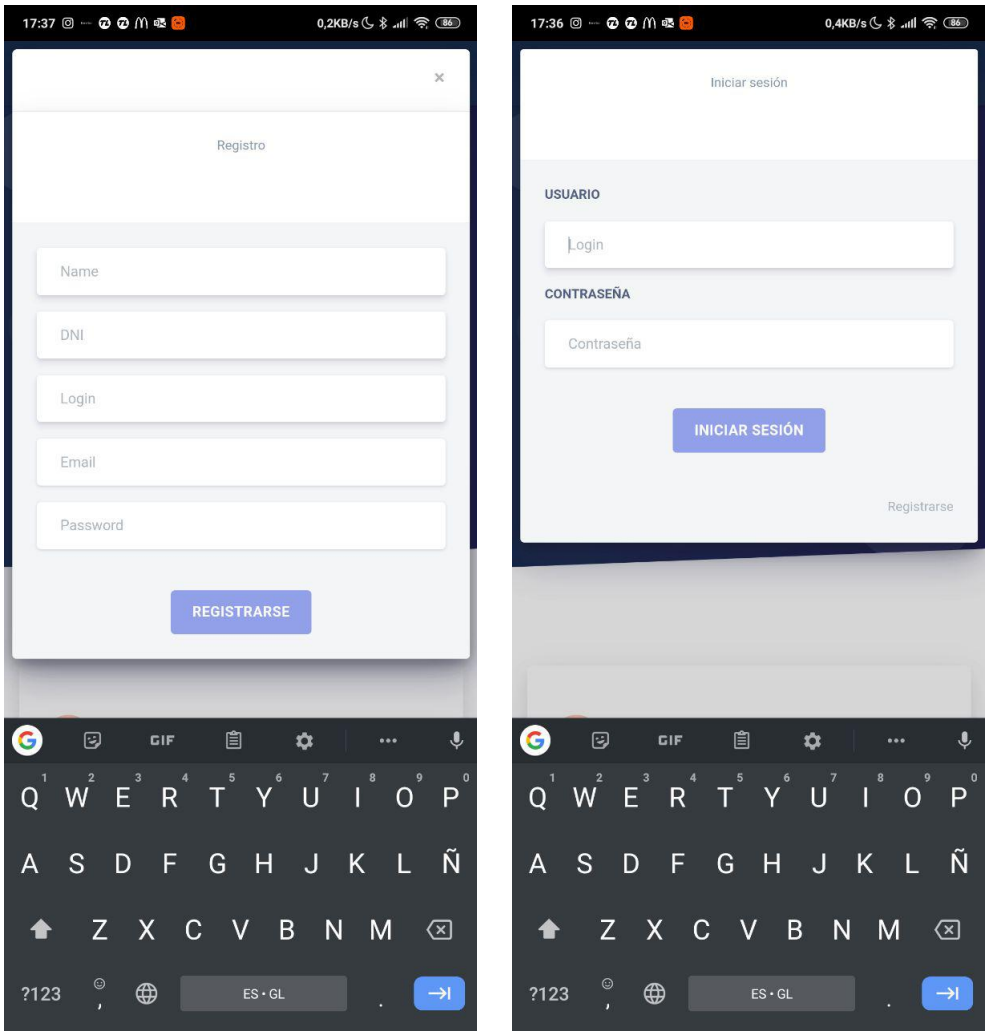
Figura A.19: Página con los detalles de un bono

A.5.1 Validación de tickets

A continuación mostraremos imágenes de la aplicación de validación usada por los revisores. En la primera figura Figura A.20, podemos ver la pantalla principal de la aplicación. Observemos que si el revisor no ha iniciado sesión, le aparecerá un mensaje indicando que debe autenticarse.

Figura A.20: Página principal de la aplicación del revisor

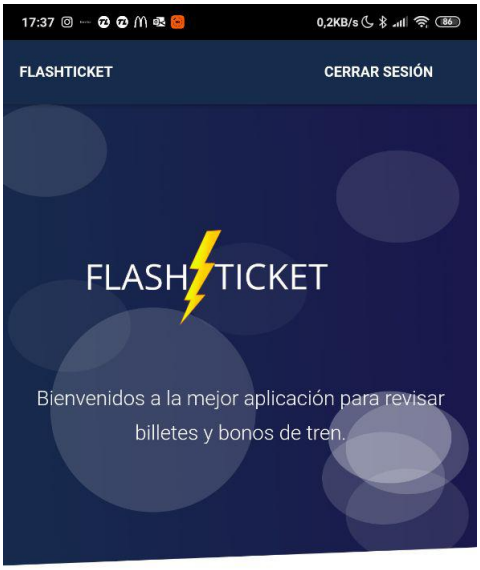
Cuando el usuario selecciona “Iniciar Sesión”, le aparecerá la pantalla de la figura Figura A.21b, en donde podrá loguearse. En caso de no tener una cuenta, deberá registrarse a través del formulario que podemos ver en la página Figura A.21a.



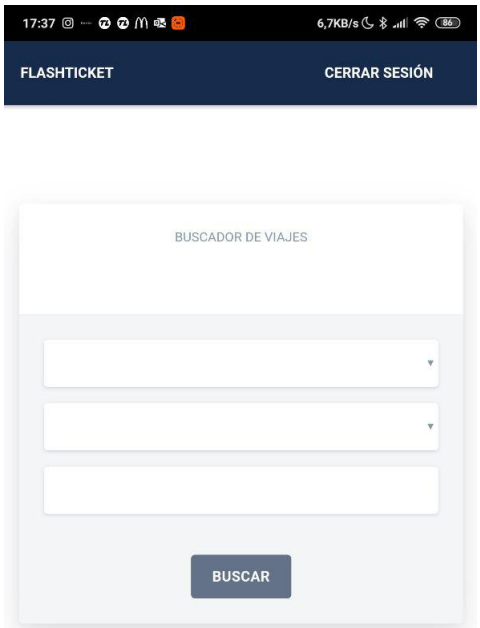
(a) Página de registro del revisor (b) Página de login del revisor

Figura A.21: Páginas de login y registro de la aplicación de validación

Una vez autenticado, el revisor será redirigido a la página principal donde, en el cuerpo de la misma podrá ver un botón para seleccionar el viaje en el que realizará las validaciones, Figura A.22a. Si selecciona el botón, será redirigido a una página con un formulario, Figura A.22b en el que debe seleccionar las estaciones de origen, destino y la fecha del viaje.



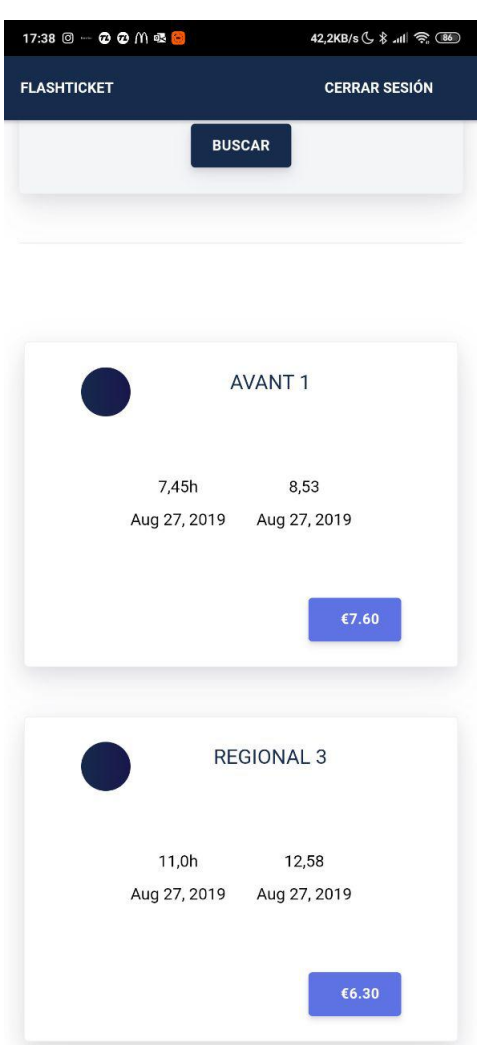
(a) Página principal después del loguearse



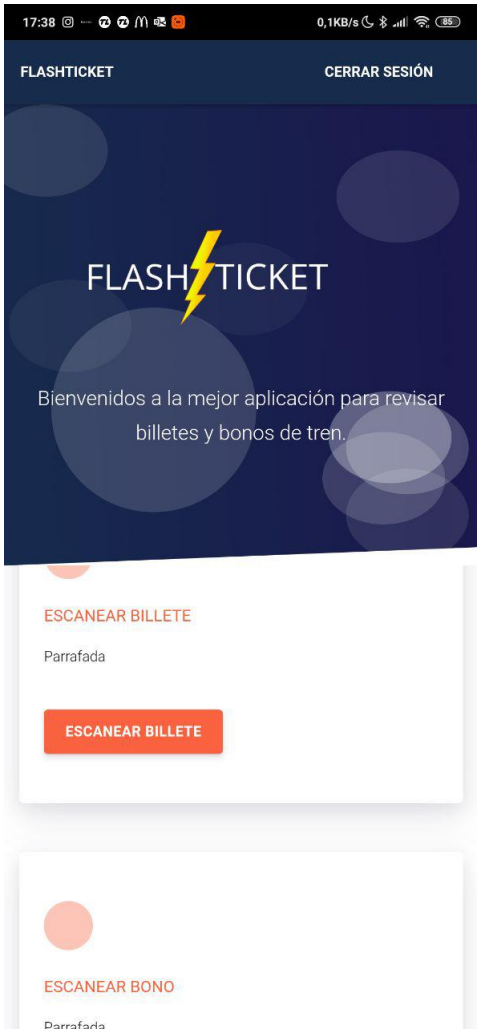
(b) Página para seleccionar el viaje

Figura A.22: Página para seleccionar tren

A continuación después de cubrir el formulario, visualizará una lista de viajes Figura A.23a. Si selecciona uno, será devuelto a la página de inicio en donde ya definitivamente le aparecerán los enlaces a las páginas de escaneo, Figura A.23a.



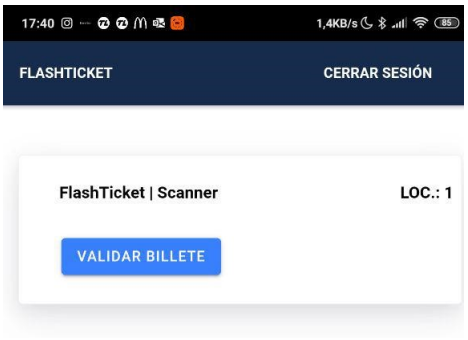
(a) Lista de viajes



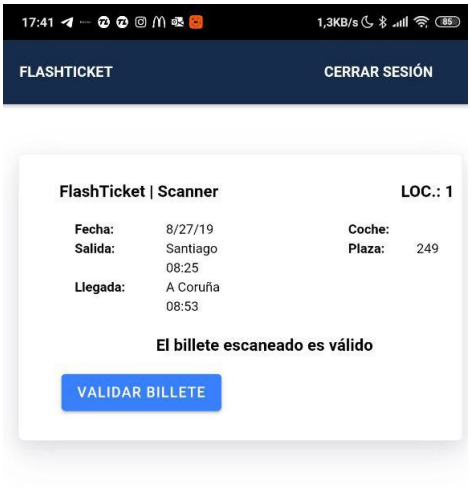
(b) Página principal después de seleccionar un viaje

Figura A.23: Proceso de selección de viaje

El siguiente paso ya es realizar las validaciones. Cuando el revisor vaya a la página de escaneo y seleccione el botón “Validar Ticket”, se le activará su cámara como podemos ver en la Figura A.25. En cuanto el código QR sea escaneado, podrá saber si es válido o no y cierta información del viaje Figura A.24b.



(a) Página de validación



(b) Resultado posterior a la validación

Figura A.24: Proceso de validación de billete

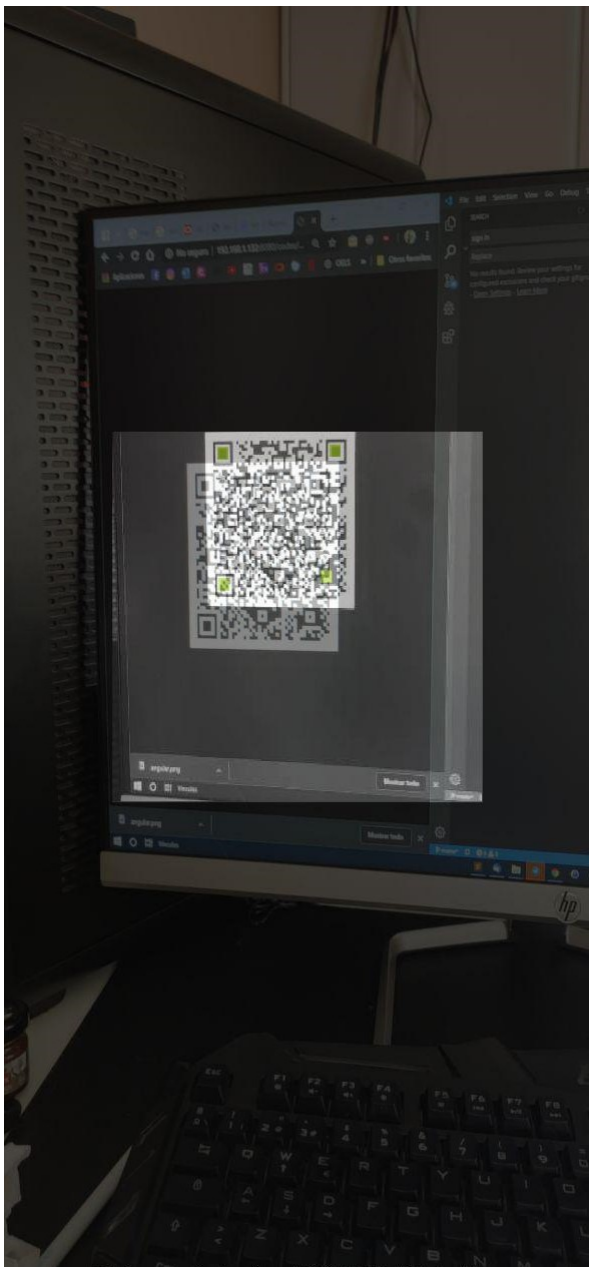


Figura A.25: Cámara de la aplicación de validación

Lista de acrónimos

HTML *HyperText Markup Language.*

CSS *Cascading Style Sheets.*

TS *TypeScript.*

TFG *Trabajo Fin de Grado.*

JPA *Java Persistence API.*

REST *Representational State Transfer.*

API *Application Programming Interface.*

SCRUM

JDBC *Java Database Connectivity.*

DTO *Data Transfer Object*

TO *Transfer Object*

HTTP *Hypertext Transfer Protocol*

BD *Base de Datos*

URL *Uniform Resource Locator.*

CRUD *Create, Read, Update and Delete.*

JWT

JSON *JavaScript Object Notation. 81*

SQL *Structured Query Language*

NPM *Node Package Manager*

QR *Quick Response*

MD *Media Distancia*

RENFE *Red Nacional de los Ferrocarriles Españoles*

GPG *GNU Privacy Guard*

Bibliografía

[1] "Página web de renfe." [En línea]. Disponible en: http://www.renfe.com/

[2] "Metro Madrid." [En línea]. Disponible en: http://www.metromadrid.es/es/pagina-principal-home

[3] "Página oficial de scrum." [En línea]. Disponible en: https://www.scrum.org/

[4] "Maven - Bienvenido a Apache Maven." [En línea]. Disponible en: https://maven.apache.org/

[5] "Maven," Jul. 2019, page Version ID: 117834257. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Maven&oldid=117834257

[6] P. Baeldung, "Spring Boot Tutorial - Bootstrap a Simple App," Jun. 2017, consultado en Agosto de 2019. [En línea]. Disponible en: https://www.baeldung.com/spring-boot-start

[7] "Spring Projects." [En línea]. Disponible en: https://spring.io/projects/spring-boot

[8] "Hibernate. Everything data. - Hibernate." [En línea]. Disponible en: http://hibernate.org/

[9] "HIBERNATE - Persistencia relacional para Java idiomático." [En línea]. Disponible en: https://docs.jboss.org/hibernate/core/3.5/reference/es-ES/html/

[10] baeldung, "Spring Boot with Hibernate," Mar. 2019. [En línea]. Disponible en: https://www.baeldung.com/spring-boot-hibernate

[11] Baeldung, "Spring and jpa - baeldung," 1993. [En línea]. Disponible en: https://www.baeldung.com/the-persistence-layer-with-spring-and-jpa

[12] "Spring data jpa - reference documentation," 1993. [En línea]. Disponible en: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/

[13] "Documentation – MapStruct." [En línea]. Disponible en: http://mapstruct.org/documentation/

[14] P. Baeldung, "Quick Guide to MapStruct," Oct. 2016. [En línea]. Disponible en: https://www.baeldung.com/mapstruct

[15] "MySQL," Jul. 2019, page Version ID: 117748913. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=MySQL&oldid=117748913

[16] "MySQL." [En línea]. Disponible en: https://www.mysql.com/

[17] "Documentación oficial de nodejs," 1993. [En línea]. Disponible en: https://nodejs.org/es/

[18] "Página web oficial de angular," 1993. [En línea]. Disponible en: https://angular.io/

[19] "Documentación oficial de angular," 1993. [En línea]. Disponible en: https://devdocs.io/angular/

[20] "Documentación oficial de typescript," 1993. [En línea]. Disponible en: https://www.typescriptlang.org/docs/

[21] a. B. M. O. contributors, Jacob Thornton, "Bootstrap." [En línea]. Disponible en: https://getbootstrap.com/

[22] "Guía oficial de ionic," 1993. [En línea]. Disponible en: https://ionicframework.com/docs

[23] "Documentación oficial apache cordova," 1993. [En línea]. Disponible en: https://cordova.apache.org/

[24] "Como usar córdova," 1993. [En línea]. Disponible en: https://cordova.apache.org/docs/es/latest/guide/cli/

[25] "Página oficial de visual studio code," 1993. [En línea]. Disponible en: https://code.visualstudio.com/

[26] "Página oficial de balsamiq," 1993. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Balsamiq

[27] "Página oficial de látex," 1993. [En línea]. Disponible en: https://www.latex-project.org/

[28] "Página oficial de eclipse," 1993. [En línea]. Disponible en: https://www.eclipse.org

[29] "Draw.io versión online," 1993. [En línea]. Disponible en: https://www.draw.io/

[30] "Tutorial de spring boot test," 1993. [En línea]. Disponible en: https://www.baeldung.com/spring-boot-testing