

MEMORIA



ParqLink

Integrantes:

- Samir Shyamdasani Sadarangani
- Elvis Yahir Rodríguez Santos.
- Perla Cristal Baylon Granda.

ÍNDICE

1. Introducción.....	2
1.1. Objetivo del documento:.....	2
1.2. Alcance del sistema:.....	2
1.3. Público objetivo:.....	2
2. Descripción General del Sistema.....	2
2.1. Propósito del sistema:.....	2
2.1 Funcionalidades clave:.....	2
2.3. Requisitos no funcionales:.....	3
2.4. Restricciones y dependencias:.....	3
3. Arquitectura del sistema.....	3
3.1. Diagrama de arquitectura:.....	3
3.2. Tecnología a utilizar:.....	4
4.Posibles mejoras.....	5
5. Funcionalidades de la app.....	5
Presentación de la app:.....	5
5.1 Registro de usuario:.....	6
5.2 Inicio de sesión:.....	6
5.3 Pantalla principal:.....	7
5.4 Navigation Drawer:.....	7
5.5 Abrir/Cerrar sesión NFC con TAG:.....	8
5.6 Detalles de Sesión:.....	9
5.7 Perfil:.....	10
5.8 Historial:.....	10
6. Conclusión.....	11
7. Link a repositorio de GitHub.....	11
Backend:.....	11
Fronted:.....	11
8. Despliegue en Azure.....	12
8.1 ¿Por qué Azure?.....	12
8.2 Diagrama de Backend y la base de datos en Azure :.....	12
8.3 Proceso de despliegue en Azure:.....	13
9. Uso de la aplicación por parte del usuario:.....	14
10. Pruebas de Backend:.....	15
10.1 Autenticación (Authentication):.....	15
10.2 Gestión de Estacionamientos (Parking):.....	15
10.3 Sesiones de Estacionamiento (Parking Sessions).....	16
11.Referencias Bibliográficas.....	17
Fronted:.....	17
Backend:.....	17

1. Introducción

1.1. Objetivo del documento:

ParqLink es una aplicación móvil diseñada para mejorar la experiencia de aparcamiento en España. Utilizando tecnologías avanzadas como Google Maps API y NFC, la aplicación permite a los usuarios encontrar estacionamientos cercanos, conocer horarios y tarifas sin tener que ir a una máquina y sacar su ticket.

1.2. Alcance del sistema:

El sistema cubrirá todo el ciclo de búsqueda, selección y recordatorio de aparcamiento para usuarios en distintas ciudades de España, comenzando por Madrid. Permitirá a los usuarios localizar parkings disponibles, recibir alertas en tiempo real y guardar sus preferencias. También integrará funcionalidades como lectura automática de entradas/salidas por NFC.

1.3. Público objetivo:

- Conductores particulares que buscan estacionamiento en zonas urbanas.
- Usuarios recurrentes de parkings.
- Usuarios tecnológicos que valoran comodidad, integración y eficiencia.

2. Descripción General del Sistema

2.1. Propósito del sistema:

- Facilitar la búsqueda de estacionamiento en España.
- Utilizar Google Maps API para sugerir la mejor ruta en tiempo real desde la ubicación del usuario hasta el estacionamiento seleccionado.
- Mejorar la experiencia del usuario mediante preferencias para que se sienta más cómodo con la app
- Avisar al usuario a través de una notificación sobre el tiempo restante para evitar multas o costos adicionales.

2.1 Funcionalidades clave:

a) Inicio de Sesión y Registro de Usuarios

- Autenticación segura mediante correo electrónico y contraseña.
- Gestión de perfiles con historial de estacionamientos.

b) Búsqueda de Parkings

- Localización de estacionamientos según la ubicación del usuario mediante Google Maps API.

c) Filtros Avanzados

- El parking más cercano.
- Colocar mínimo y máximo precio.

d) Alertas y Notificaciones

- Recordatorios de tiempo restante en un estacionamiento para evitar multas o costos no esperados.

- Informar a los usuarios sobre eventos que puedan afectar la disponibilidad de aparcamiento en ciertas zonas.

e) Perfil de usuario con preferencias personalizadas

- Permitir a los usuarios guardar sus preferencias, como parkings favoritos.

2.3. Requisitos no funcionales:

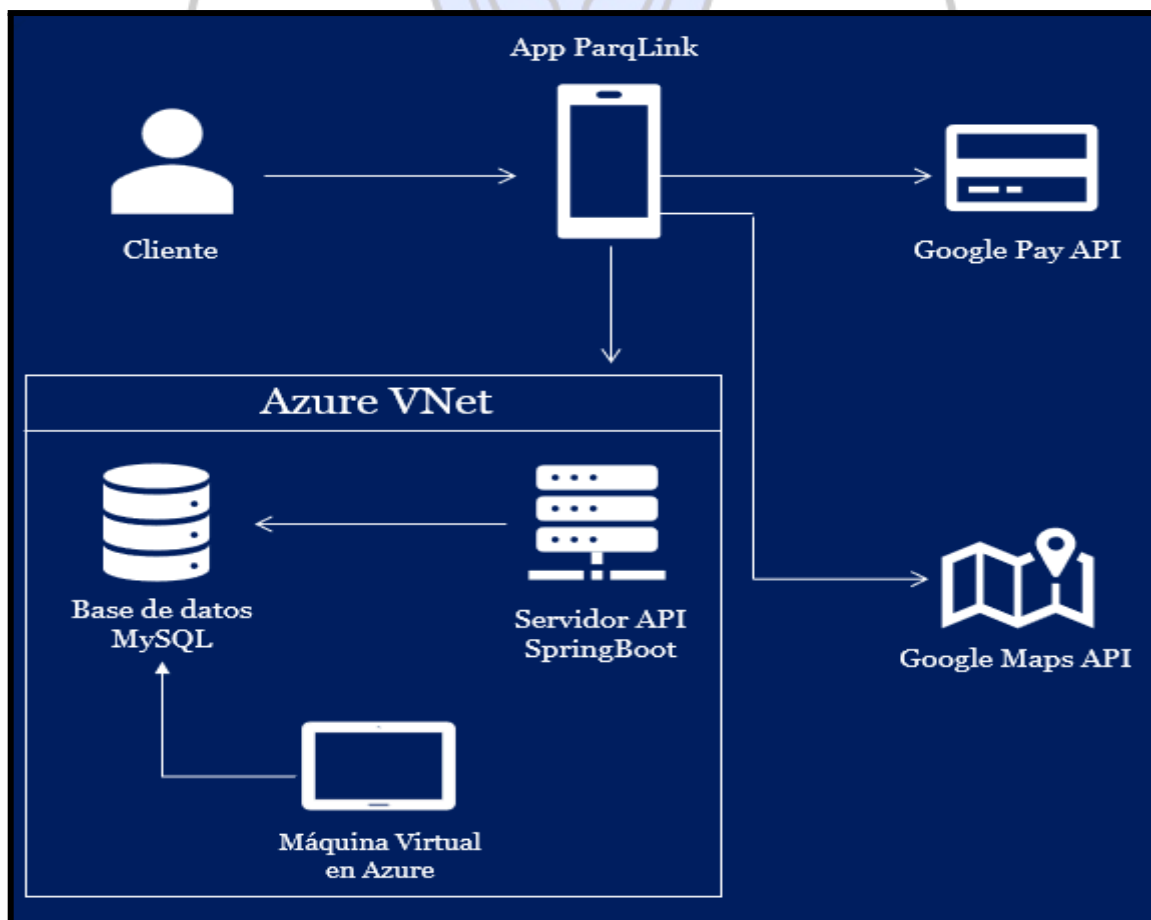
- **Compatibilidad:** Compatible con Android.
- **Usabilidad:** Interfaz intuitiva accesible incluso para usuarios con poca experiencia tecnológica.

2.4. Restricciones y dependencias:

- Dependencia de conexión estable a internet para funcionalidades en tiempo real.
- Dependencia de APIs externas como Google Maps API y Google Pay.
- Requiere que los parkings estén registrados en la base de datos para mostrar información detallada.

3. Arquitectura del sistema

3.1. Diagrama de arquitectura:



Componentes	Tecnología principal	Descripción
App Móvil (Frontend) - ParqLink	Android Studio Retrofit Google Maps API Google Pay API	Interfaz de usuario para búsqueda, filtro, favoritos y lista de parkings.
Backend (API) - IntelliJ	Spring Boot JWT MySQL	Manejo de lógica de negocio, sesiones NFC y seguridad.
Base de datos	MySQL (XAMPP)	Almacena usuarios, parkings y sesiones de usuarios.
Despliegue - Microsoft Azure	Azure CLI: Azure App Service Azure Database for MySQL Máquina Virtual en Azure	Una red virtual en Azure con tres subredes para controlar la parte código backend, base de datos y una máquina para administrar la BBDD.

3.2. Tecnología a utilizar:

1.- Lenguajes de Desarrollo

- **Frontend Móvil:**

Java (Android): Creación de la interfaz de usuario (UI), donde los usuarios pueden buscar parkings, recibir notificaciones.

- **Backend:**

Spring Boot (Java): Creación del servidor backend que manejará las solicitudes de la app móvil, como la búsqueda de parkings y la autenticación de usuarios.

SQL(MySQL): La gestión de datos de los estacionamientos y usuarios.

2.- Frameworks y librerías

- **Google Maps API:** Ofrece funcionalidades avanzadas como búsqueda de direcciones, cálculo de rutas y visualización de tráfico en tiempo real.
- **Google Pay API:** Se utiliza como método de pago en nuestra aplicación, está en modo prueba.

3.- Herramientas y Entorno de Desarrollo

- **IDE:**

1. Android Studio (Android), IntelliJ : Desarrollar y probar la interfaz de usuario de la app móvil.
2. Integrar Google Maps API dentro de la aplicación, implementar funcionalidades como notificaciones y autenticación de usuarios.

- **Control de versiones:** GitHub: Guardar y gestionar el código fuente de la app móvil y el backend. Colaborar con otros desarrolladores en el equipo.

4. Posibles mejoras

- **Implementar el funcionamiento de Pago de Google Pay:** esta implementado pero como una simulación y un pago fake utilizando el siguiente atributo **ENVIRONMENT_TEST** si se cobrará de verdad se cambiaría este atributo por el de producción **ENVIRONMENT_PRODUCTION**, luego registrarnos como comerciantes en Google Pay donde obtendremos un ID de comerciante, que lo colocaremos en nuestro código en la clase **GooglePayUtils**.
- **Ampliación de la información sobre los parkings:** Incluir detalles adicionales como disponibilidad de cargadores eléctricos, servicio de lavado de autos, vigilancia, o si el parking es cubierto o descubierto, para ayudar al usuario a tomar una mejor decisión al escoger un parking.
- **Verificación de cuenta por correo electrónico:** Enviar un correo de confirmación al registrarse para activar la cuenta, asegurando que el email sea válido y mejorando la seguridad del sistema.
- **Integración de cámara con reconocimiento de matrícula:** Implementar un sistema automatizado de acceso al parking mediante reconocimiento de matrículas a través de una cámara en la entrada/salida del aparcamiento que reemplazaría el uso de NFC. Al detectar la matrícula registrada de un usuario, el sistema iniciará automáticamente una sesión en la aplicación del móvil y terminará la sesión cuando vuelva a salir del parking.
- **Parte administrativa de la aplicación:** tenemos para gestionar la base de datos a través de una máquina virtual en Azure, pero quisiéramos a futuro una interfaz de administración para las siguientes funciones:
 1. Gestionar parkings: altas, bajas, edición de precios, horarios y disponibilidad.
 2. Visualizar estadísticas de uso: cantidad de sesiones activas, parkings más utilizados.
 3. Gestión de incidencias: reportes de fallos o quejas de usuarios.

5. Funcionalidades de la app

Presentación de la app:

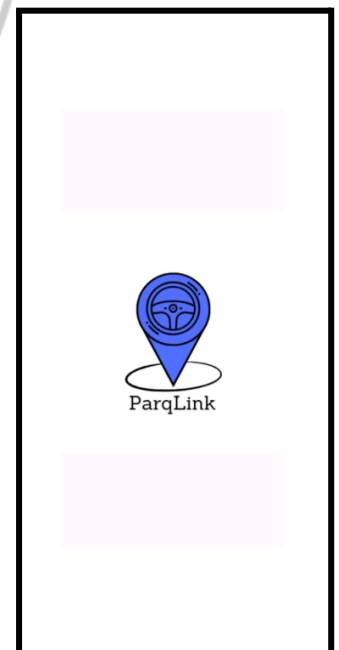
La clase PresentacionApp actúa como una pantalla de bienvenida animada que mejora la experiencia del usuario desde el primer momento en que interactúa con la aplicación ParqLink. Su función principal es reproducir un video introductorio que presenta la propuesta de valor de la app antes de dirigir al usuario a la pantalla correspondiente según su estado de autenticación.

Una vez iniciado, éste Activity reproduce un video alojado en los recursos locales de la aplicación (R.raw.intro). El video se adapta dinámicamente al tamaño de pantalla para garantizar una correcta visualización, manteniendo la proporción original del contenido. Al finalizar la reproducción, el sistema verifica si el usuario ya ha iniciado sesión previamente, y según eso, redirige automáticamente al MainActivity (pantalla principal de la app) o al LoginActivity (pantalla de inicio de sesión).

Tecnologías utilizadas:

- VideoView y MediaPlayer para la carga, reproducción y gestión de eventos del video de presentación.
- SharedPreferences para comprobar si el usuario ya está autenticado mediante la variable booleana "isLoggedIn".
- AppCompatActivity como base del ciclo de vida de la actividad.

Importancia dentro del sistema: Ofrece una introducción profesional y amigable que contribuye a la identidad de marca de ParqLink. Además, integra de forma transparente una verificación de sesión activa para asegurar una transición fluida hacia la experiencia principal del usuario.



5.1 Registro de usuario:

La clase RegisterActivity es una parte fundamental del flujo de incorporación de nuevos usuarios en la aplicación ParqLink. Su principal objetivo es permitir que personas que aún no tienen una cuenta puedan registrarse proporcionando sus datos personales básicos: nombre, correo electrónico y contraseña. Esta actividad representa el primer contacto formal del usuario con el sistema, por lo tanto, su diseño y funcionamiento impactan directamente en la experiencia de usuario y en el éxito de la adopción de la aplicación comprueba si escribiste bien el formato correo sino no te dejará crear un usuario, también verifica si la contraseña está lo suficientemente compleja para crearla.

Tras un registro exitoso, la aplicación guarda el token de acceso y con el botón volver el usuario regresa a la pantalla de inicio de sesión. También ofrece la opción de mostrar u ocultar la contraseña para mejorar la usabilidad.

Tecnologías utilizadas:

- Retrofit2 para enviar la solicitud de registro al servidor.
- SharedPreferences para almacenar el token de acceso de forma local.
- Actividad basada en AppCompatActivity.

Importancia dentro del sistema:

Es el punto de entrada para nuevos usuarios que desean unirse a ParqLink, permitiendo su autenticación futura y el acceso completo a las funcionalidades de la aplicación.



5.2 Inicio de sesión:

La clase LoginActivity permite a los usuarios iniciar sesión en la aplicación ingresando su correo electrónico y contraseña. Verifica las credenciales con el backend y, si son correctas, mantiene activa la sesión mediante almacenamiento local. Además, gestiona la visibilidad de la contraseña para mejorar la experiencia del usuario.

En caso de autenticación exitosa, se guarda un token de acceso (JWT) y se redirige al usuario a la pantalla principal. Si falla, muestra mensajes de error apropiados al usuario.

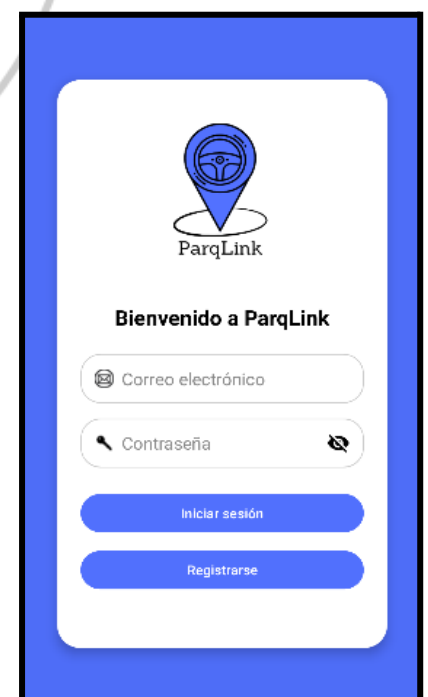
Esta clase también detecta si el usuario ya ha iniciado sesión previamente y, en ese caso, lo dirige directamente a la pantalla principal sin necesidad de volver a ingresar sus datos.

Tecnologías utilizadas:

- Retrofit2 para la conexión con el backend.
- SharedPreferences para la persistencia de la sesión del usuario.
- Actividad basada en AppCompatActivity.

Importancia dentro del sistema:

Actúa como la puerta de entrada a la aplicación y es responsable de la autenticación segura del usuario.



5.3 Pantalla principal:

La clase MainActivity representa la interfaz principal de ParqLink, donde el usuario puede visualizar, buscar y filtrar los parkings disponibles en un mapa de Google Maps junto a una lista. Esta actividad también gestiona funcionalidades clave como la ubicación del usuario, el control de sesiones NFC y la navegación hacia otras secciones de la app.

Funcionalidades principales:

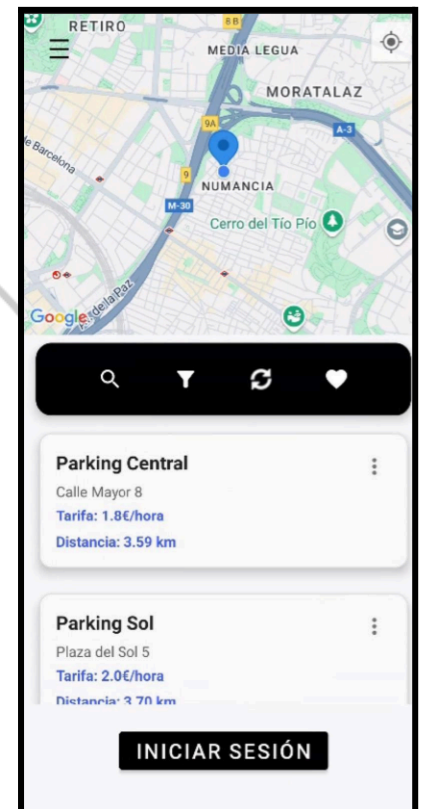
- **Visualización de parkings:** Muestra un mapa con marcadores de los parkings disponibles y una lista detallada con su nombre, precio, distancia y dirección.
- **Orden y filtros:** Permite ordenar los parkings por cercanía o aplicar filtros por rango de precios.
- **Búsqueda:** Incluye una función para buscar parkings por nombre.
- **Refrescar:** Utiliza la ubicación del usuario para ordenar los parkings por proximidad.
- **Sesión NFC:** Si hay una sesión activa mediante NFC, muestra un botón con acceso a los detalles. En caso contrario, permite iniciar una nueva sesión.
- **Gestión de favoritos:** Ofrece una opción para mostrar únicamente los parkings marcados como favoritos.
- **Navegación:** A través de un menú lateral se puede acceder al perfil del usuario, historial de actividad o cerrar sesión.

Tecnologías utilizadas:

- Google Maps API para el mapa interactivo.
- FusedLocationProviderClient para obtener la ubicación del usuario.
- SharedPreferences para guardar el estado de la sesión y otros datos locales.
- Retrofit (en otras clases) para conectar con el backend.
- NFC (opcionalmente habilitado) para gestión de sesiones físicas de aparcamiento.

Importancia dentro del sistema:

Es el núcleo operativo de la aplicación. Desde esta pantalla el usuario puede interactuar con las principales funciones de ParqLink, acceder a la información de los parkings, iniciar o cerrar sesiones, y navegar al resto de funcionalidades.

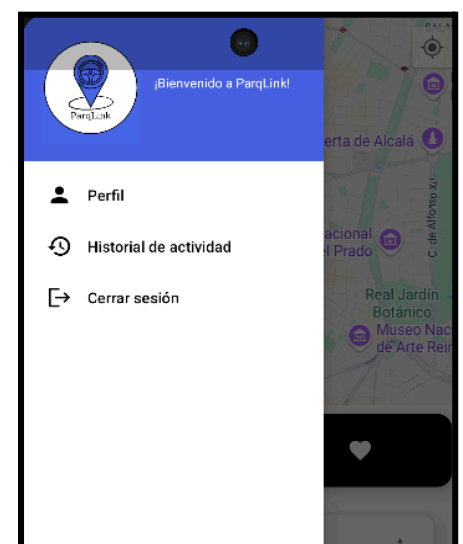


5.4 Navigation Drawer:

El Navigation Drawer de la aplicación proporciona un panel lateral de navegación que permite al usuario acceder de forma estructurada y eficiente a las principales funcionalidades de la aplicación. Este componente está basado en DrawerLayout e implementa una Navigation View anclada a la izquierda.

La Navigation View carga:

- Un encabezado personalizado que muestra el logotipo de ParqLink y un mensaje de bienvenida.
- Se abre de dos formas deslizando de izquierda a derecha o apretando el botón de tres líneas superior en main activity.



- Un menú definido en XML que incluye accesos directos a:
 - Perfil del usuario
 - Historial de sesiones de parking
 - Cierre de sesión (logout)

Importancia dentro del sistema:

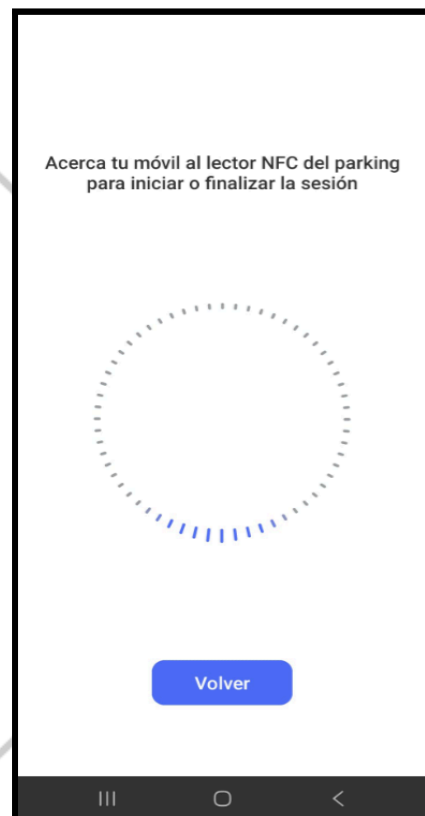
El Navigation Drawer es un componente clave que facilita el acceso rápido a funciones críticas como el perfil del usuario, el historial de sesiones y el cierre de sesión. Su integración mejora la experiencia de usuario, reduce la fricción en la navegación y contribuye a mantener una interfaz limpia.

5.5 Abrir/Cerrar sesión NFC con TAG:

La clase `NfcReaderActivity` permite al usuario iniciar o finalizar una sesión de estacionamiento escaneando una etiqueta NFC. Está diseñada para detectar etiquetas con un identificador de parking codificado en su contenido.

Funcionamiento paso a paso:

- Inicialización:**
 - Se obtiene la instancia de `NfcAdapter`.
 - Se comprueba si el dispositivo soporta NFC.
 - Se solicita permiso para mostrar notificaciones.
 - Se inicializa un reproductor de sonido para retroalimentación auditiva al usuario.
- Detección NFC con TAG:**
 - Se escanea con la etiqueta (tag) que contiene un id de su respectivo parking. Cuando se detecta el ID se abre la sesión.
- Lectura del TAG:**
 - Se extrae el mensaje NDEF y se interpreta el texto del primer `NdefRecord` para obtener el ID del estacionamiento.
- Comunicación con el backend:**
 - Se envía una solicitud POST al endpoint `/scan` con el token JWT del usuario y el ID del tag escaneado.
 - El backend responde con un objeto `ParkingSessionResponse`, que indica si se debe **iniciar** o **finalizar** una sesión.
- Lógica condicional según la respuesta:**
 - Si la sesión tiene una hora de **fin**, se considera una **finalización de sesión**. Se eliminan datos de la sesión del almacenamiento y se detiene el servicio en primer plano.
 - Si no tiene hora de fin, se trata de un **inicio de sesión**. Se guarda la hora de inicio, se inicia un servicio en primer plano (`NfcSessionService`) y se muestra una notificación persistente.
- Notificaciones y sonidos:**
 - Se muestra un `AlertDialog` para informar al usuario del resultado.
 - Se reproducen sonidos específicos para retroalimentación inmediata.
 - Se crean o finalizan notificaciones persistentes para reflejar el estado de la sesión.



7. GooglePay:

- Cuando se cierra la sesión crea un AlertDialog que te dice que debes pagar con los datos del coste total, al presionar en el botón pagar ahora, se vincula con tu google pay y finalmente al pagar después te confirma el pago exitoso.

Tecnologías utilizadas:

- NFC (NfcAdapter, NdefMessage) para lectura de etiquetas.
- Notificaciones persistentes para informar al usuario del estado.
- SharedPreferences para guardar datos locales de sesión.
- Retrofit para comunicación con el backend.
- Reproductor de sonidos personalizado para retroalimentación auditiva.
- Java Time API (LocalDateTime) para manejo de fechas y horas.

5.6 Detalles de Sesión:

Pantalla que muestra la sesión de estacionamiento en curso. Al iniciarse, recibe el nombre del parking y la hora de inicio desde el intent. Calcula en tiempo real la duración transcurrida y actualiza la interfaz cada segundo. Además, busca en la lista local de parkings para mostrar dirección y tarifa si hay coincidencia con el nombre recibido. Detiene las actualizaciones temporales cuando la actividad se pausa.

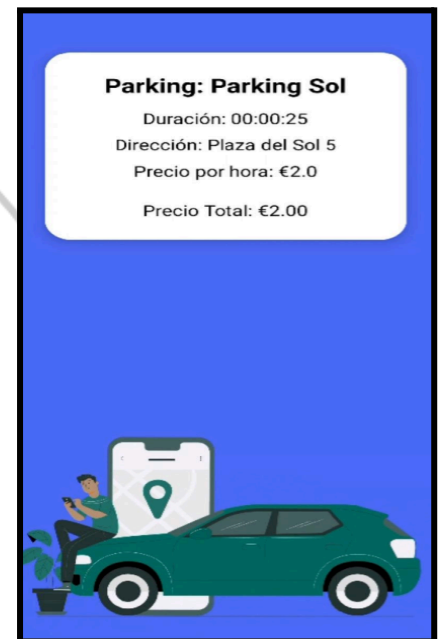
Importancia funcional: Esta pantalla proporciona una vista clara y en tiempo real del estado actual de la sesión NFC.

Permite al usuario:

- Verificar que la sesión está activa.
- Controlar cuánto tiempo ha transcurrido.
- Consultar los datos del estacionamiento en curso (dirección y tarifa).

Lottie: Es un lottie que se repite una y otra vez para darle un poco de movimiento a la interfaz.

Gracias a esta funcionalidad, el usuario puede **gestionar mejor el uso del estacionamiento** y evitar sobrepasar el tiempo deseado.



5.7 Perfil:

La clase Profile gestiona la visualización y edición de los datos personales del usuario dentro de la aplicación ParqLink. Esta pantalla permite consultar el nombre y correo electrónico almacenados localmente, y modificar el nombre de usuario de forma sencilla a través de la interfaz. La información se conserva mediante almacenamiento persistente utilizando SharedPreferences.

Funcionalidades principales:

- **Visualización de datos del usuario:** Muestra el nombre y correo electrónico previamente guardados en las preferencias de la aplicación.
- **Guardado local:** El nuevo nombre introducido se guarda en SharedPreferences y se refleja inmediatamente en la interfaz al pulsar "Guardar".
- **Cambio dinámico de interfaz:** La interfaz alterna entre modo de edición y visualización para mejorar la experiencia de usuario.
- **Volver a la pantalla anterior:** Un botón dedicado permite salir de la pantalla y regresar al flujo principal de la aplicación.
- **Lottie:** Es un lottie que se repite una y otra vez para darle un poco de movimiento a la interfaz.

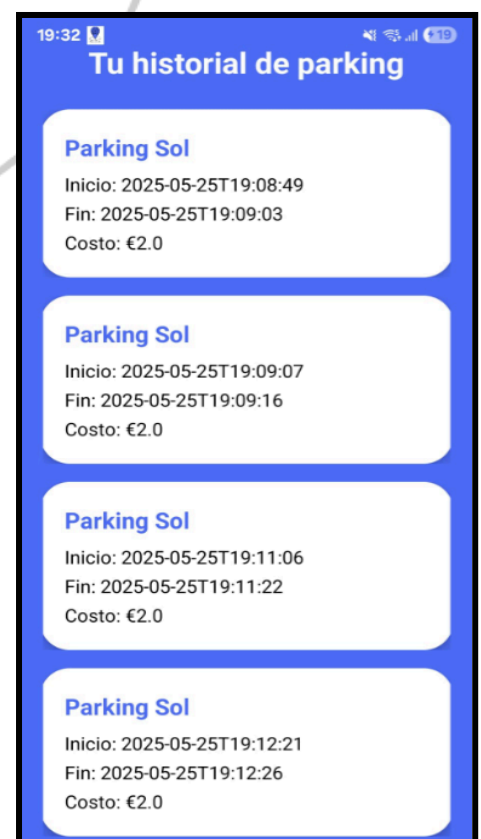


5.8 Historial:

Esta clase representa la pantalla de historial dentro de ParqLink, donde el usuario puede consultar un registro de todas las sesiones de aparcamiento anteriores. Utiliza una conexión al backend para recuperar los datos y los muestra en una lista con formato amigable mediante un RecyclerView.

Funcionalidades principales:

- **Recuperación del historial desde el backend:** Se realiza una llamada a la API para obtener todas las sesiones de aparcamiento del usuario autenticado. El token JWT se recupera desde SharedPreferences y se utiliza en la cabecera de autenticación.
- **Visualización estructurada con RecyclerView:** Las sesiones se presentan en una lista vertical usando un RecyclerView y un Adapter personalizado, que renderiza cada elemento con los datos relevantes como nombre del parking, fecha y duración.
- **Manejo de errores:** Se proporciona retroalimentación al usuario en caso de errores de red, respuesta inválida del servidor o problemas con el token de autenticación.



Tecnologías utilizadas:

- **Retrofit:**
Para realizar llamadas HTTP y obtener los datos desde el backend de ParqLink.
- **RecyclerView:**
Para mostrar las sesiones de forma eficiente y dinámica.
- **SharedPreferences:**
Para recuperar el token de autenticación JWT almacenado localmente.

6. Conclusión

ParqLink busca optimizar la gestión del aparcamiento en España ofreciendo una solución tecnológica moderna, eficiente y segura. Con funcionalidades avanzadas y una integración con NFC y transporte público, la aplicación mejorará la experiencia de estacionamiento y facilitará el acceso a información relevante para los usuarios.

7. Link a repositorio de GitHub

Contamos con dos repositorios privados para mejor eficiencia y orden, tenemos la parte del Frontend y la del Backend.

Backend:

https://github.com/samirshyam/ParqLink_Backend

Frontend:

https://github.com/samirshyam/ParqLink_Frontend

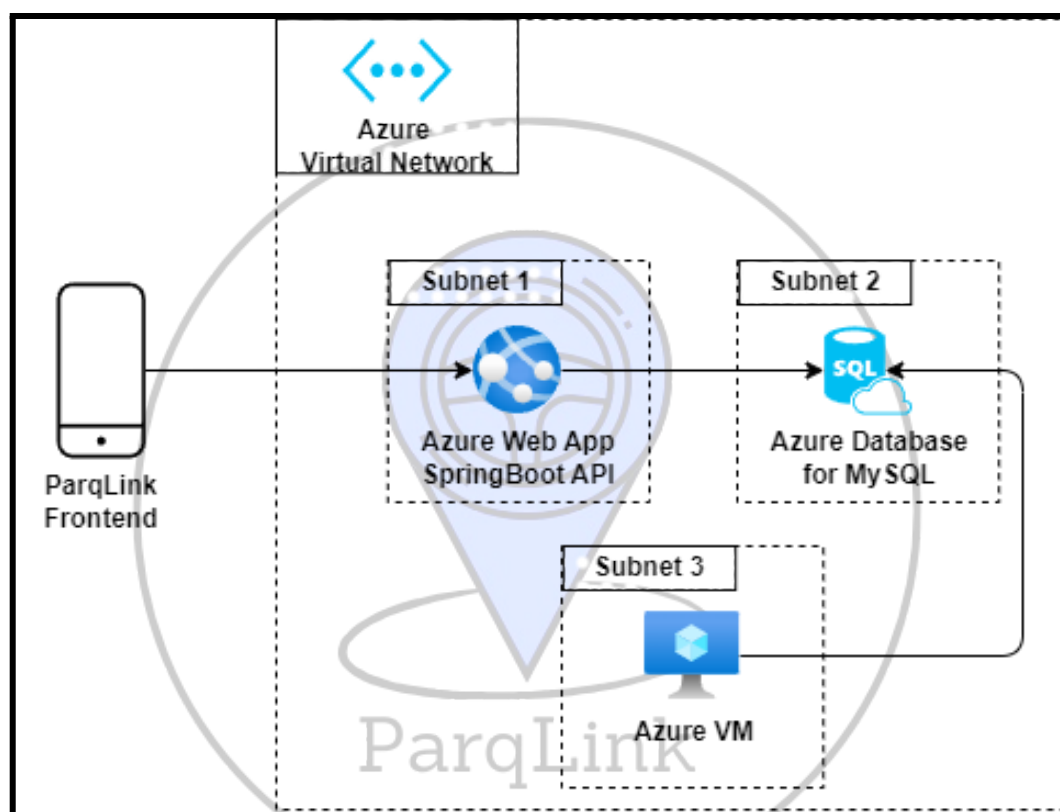
ParqLink

8. Despliegue en Azure

8.1 ¿Por qué Azure?

Para garantizar la escalabilidad, disponibilidad y seguridad del sistema, se ha optado por desplegar tanto el backend como la base de datos en la plataforma en la nube Microsoft Azure.

8.2 Diagrama de Backend y la base de datos en Azure :



1. Azure Virtual Network (VNet)

Toda la infraestructura está contenida dentro de una Red Virtual de Azure (VNet), lo que permite la comunicación segura entre los distintos recursos de la nube. Dentro de esta red virtual, se han definido tres subredes (subnets) para aislar y organizar los componentes de la solución.

2. Subnet 1 – Azure Web App (SpringBoot API)

En esta subred se despliega el backend de la aplicación, desarrollado con Spring Boot.

Este servicio está publicado como una Azure Web App, lo que permite una gestión sencilla del entorno, escalado automático, e integración continua., es un servicio serverless.

La Web App actúa como intermediario entre el frontend móvil y la base de datos, gestionando las peticiones y la lógica del negocio.

3. Subnet 2 – Azure Database for MySQL

Aquí se encuentra el servicio de base de datos relacional, implementado como Azure Database for MySQL.

La base de datos almacena toda la información relevante de la aplicación: usuarios, sesiones, parkings, etc.

Solo se permite el acceso desde la Web App y desde la máquina virtual (para mantenimiento o scripts especiales), restringiendo así los accesos externos no autorizados.

4. Subnet 3 – Azure Virtual Machine (VM)

Esta subred alberga una máquina virtual de Azure que se utiliza para tareas administrativas, monitoreo o pruebas específicas.

Puede contener herramientas de diagnóstico o scripts de respaldo que interactúan directamente con la base de datos.

La comunicación con la VM también está restringida a través de reglas específicas de red dentro de la VNet.

5. ParqLink Frontend

El frontend de la aplicación es una aplicación móvil que se comunica directamente con el backend (SpringBoot API).

A través de esta conexión, los usuarios finales pueden interactuar con los servicios ofrecidos por ParqLink: visualizar parkings, realizar sesiones, Buscar/Filtrar/Agregar a favoritos sus parkings, etc.

Ventajas de esta Arquitectura en Azure

- **Seguridad:** Separación por subredes dentro de una VNet garantiza control de acceso y tráfico.
- **Escalabilidad:** El servicio de Azure Web App se puede escalar automáticamente según la demanda.
- **Mantenimiento:** La VM facilita tareas de mantenimiento sin afectar los servicios principales.
- **Alta disponibilidad:** Azure Database for MySQL y los servicios web garantizan disponibilidad con réplicas y respaldo automático.

8.3 Proceso de despliegue en Azure:

Despliegue de la API Spring Boot en Azure

Para llevar a cabo el despliegue de la API desarrollada con Spring Boot en la plataforma Microsoft Azure, se siguió un procedimiento estructurado que incluyó los siguientes pasos:

1. Instalación de Azure CLI

Se instaló la herramienta de línea de comandos de Azure (Azure CLI) en un entorno Windows, con el fin de facilitar la gestión de los recursos de Azure desde la terminal.

2. Autenticación en Azure

Se realizó el inicio de sesión mediante el comando `az login`, que permitió autenticar la sesión con una cuenta de Microsoft, habilitando así el acceso a los recursos disponibles en la suscripción de Azure.

3. Empaquetado del proyecto con Maven

El proyecto fue empaquetado utilizando Maven. Debido a que la aplicación utiliza variables de entorno para establecer la conexión con la base de datos, las pruebas automatizadas podrían fallar si no están correctamente configuradas. Por esta razón, se omitieron las pruebas durante el empaquetado mediante el siguiente comando:

```
mvn clean package -DskipTests
```

Esto generó un archivo .jar listo para ser desplegado en el entorno en la nube.

4. Despliegue en Azure App Service

Se creó una Web App en Azure App Service, configurada específicamente para ejecutar aplicaciones Java basadas en archivos JAR. El despliegue del archivo se realizó mediante Azure CLI con el siguiente comando:

```
az webapp deploy \  
--resource-group <nombre-del-grupo-de-recursos> \  
--name <nombre-de-la-webapp> \  
--type jar \  
--target-path app.jar
```

5. Configuración de variables de entorno

Desde el Portal de Azure, en la sección de configuración de la Web App, se establecieron las variables de entorno necesarias para el correcto funcionamiento de la aplicación, como los parámetros de conexión a la base de datos y otras propiedades relevantes.

6. Inicio y monitoreo de la aplicación

Una vez iniciada la Web App, se utilizó el siguiente comando para monitorear los logs en tiempo real y verificar que la aplicación estuviera funcionando de forma correcta:

```
az webapp log tail \  
--name <nombre-de-la-webapp> \  
--resource-group <nombre-del-grupo-de-recursos>
```

9. Uso de la aplicación por parte del usuario:

1. El usuario abre la aplicación y visualiza una animación de Parqlink de bienvenida.
2. Si ya está registrado, y es su primera vez entrando accede a su cuenta con su correo y contraseña, después de la primera vez si ya está logueado no será necesario este paso ya que entra directamente a la página principal, si no está registrado tendrá que registrarse y escribir en el formato correcto el correo y contraseña.
3. En la pantalla principal, el Usuario observa un mapa con parkings cercanos gracias a Google Maps API , verá la barra con las herramientas (Buscar, Filtrar, actualizar y favoritos) , como es su primera vez no se sabe los nombre de los parkings así que irá a filtrar parkings según su preferencia por min/max precio o distancia.
4. Imaginemos que el usuario filtra el precio 1 a 3 euros, se filtrará la lista en ese rango y escoge el de su preferencia.
5. Presiona el parking, el mapa se mueve hacia esa ubicación y luego le da a la flecha que está dentro del mapa que le llevará a Google Maps con la dirección colocada desde su ubicación al parking y le mostrará la mejor ruta al aparcamiento.
6. Al llegar a la entrada del parking, le da a Iniciar Sesión en la aplicación y escanea la etiqueta NFC que está en el aparcamiento con su móvil.
7. Se inicia la sesión del parking y el tiempo empieza a correr en su móvil tanto dentro de la aplicación en la ventana de los detalles de sesión que aparecerá como en el panel de notificaciones de su móvil. Luego deja el coche y se va del aparcamiento, estando afuera del aparcamiento recibe una alerta de que ya faltan 15 minutos para que el costo total incremente así que decide volver y llegar a tiempo para salir del aparcamiento.
8. Para salir del aparcamiento, vuelve a escanear la NFC en la entrada del parking, le aparece una notificación y también un diálogo dentro de la aplicación que dice Sesión finalizada y le da a la opción pagar con Google Pay.
9. La sesión queda registrada en su historial y puede consultarse luego para revisar los detalles o el parking donde aparcó.

10. Pruebas de Backend:

Para verificar el correcto funcionamiento del backend de la aplicación, se realizaron diversas pruebas utilizando la herramienta Postman, que permite enviar peticiones HTTP a la API y visualizar las respuestas. A continuación, se describen los endpoints disponibles organizados por funcionalidad, especificando su propósito, el tipo de autenticación necesaria, y cómo se utilizaron durante las pruebas

10.1 Autenticación (Authentication):

Esta sección permite registrar nuevos usuarios y autenticarlos para obtener un token JWT, que será necesario para acceder a endpoints protegidos del sistema.

Método	Endpoint	Descripción	Requiere Autenticación
POST	/api/auth/register	Registra un nuevo usuario	No
POST	/api/auth/login	Inicia sesión y devuelve un token JWT	No

Pruebas realizadas:

- /api/auth/register: Se envió una petición POST con un cuerpo JSON que incluía nombre de usuario, correo electrónico y contraseña. La API respondió con éxito creando el nuevo usuario.
- /api/auth/login: Se utilizó para probar el inicio de sesión. Al enviar las credenciales correctas, el servidor respondió con un token JWT, que se utilizó en las siguientes peticiones que requerían autenticación (mediante el encabezado Authorization: Bearer <token>).

10.2 Gestión de Estacionamientos (Parking):

Este grupo de endpoints permite obtener una lista de estacionamientos disponibles, con la posibilidad de aplicar filtros y ordenamientos.

Método	Endpoint	Descripción	Requiere Autenticación
GET	/api/parking/all	Obtiene todos los estacionamientos registrados (con filtros)	No

Filtros opcionales disponibles:

- latitude: latitud actual del usuario.
- longitude: longitud actual del usuario.
- maxDistance: distancia máxima (en metros).
- maxPrice: precio máximo por hora.
- sort: ordenar por distance o price.
- order: orden asc o desc.
- name: nombre del estacionamiento.
- address: dirección parcial o completa.
- page: número de página de la lista de parkings.
- size: número de parkings en cada página.

Ejemplos de pruebas con filtros:

Estacionamientos cercanos

Se buscó obtener los estacionamientos dentro de un radio de 500 metros de una ubicación específica:

GET /api/parking/all?latitude=40.4168&longitude=-3.7038&maxDistance=500

Resultado esperado: Sólo se devuelven los parkings dentro del rango.

10.3 Sesiones de Estacionamiento (Parking Sessions)

Esta sección gestiona el inicio y fin de las sesiones de aparcamiento mediante escaneo de etiquetas NFC, y también permite consultar el historial de sesiones del usuario.

Método	Endpoint	Descripción	Requiere
Autenticación			
POST	/api/parking/scan	Inicia o finaliza una sesión de aparcamiento mediante NFC	Sí
GET	/api/parking/sessions	Devuelve las sesiones pasadas del usuario, con duración y costo	Sí

Pruebas realizadas:

- /api/parking/scan: Se simuló el escaneo de un tag NFC enviando un JSON con el ID del tag. Si no había sesión activa, la API la iniciaba; si ya había una sesión activa para ese usuario y ese parking, la API la finaliza.
- /api/parking/sessions: Se verificó que la API devolviera correctamente la lista de sesiones anteriores del usuario autenticado, incluyendo el tiempo de duración y el costo total de cada sesión.

Consideraciones Finales

- Todas las rutas protegidas se probaron usando el token JWT en los encabezados (Authorization: Bearer <token>).
- Se verificaron los posibles errores, como intentos de acceder a recursos protegidos sin autenticación, o el uso de filtros incorrectos.
- Las respuestas de la API incluyen mensajes informativos y códigos de estado HTTP adecuados (200 OK, 401 Unauthorized, 400 Bad Request, etc.).

11.Referencias Bibliográficas

Fronted:

- **API implementadas:**

El desarrollo se basó en la documentación oficial de Google:

Google. (2024, junio 26). *Google Pay API for Android*.
<https://developers.google.com/pay/api/android/overview>

Google. (2025, mayo 23). *Maps SDK for Android*.
<https://developers.google.com/pay/api/android/overview>

- **Imágenes e Íconos implementados:**

a. Iconos de Android Studio:

Se buscó en la librería de iconos de Android Studio:

Android Open Source Project. (s.f.). *Material Design icons*. <https://fonts.google.com/icons>

b. Imágenes tomadas de internet:

Canva - Perla Baylon. (2025) , creación del logo de ParqLink.

- **Sonido de Notificaciones:**

Se buscó en la página oficial de pixabay:

Modern Stereo - vynadot. (s.f.). <https://pixabay.com/es/sound-effects/search/notificación/>

- **NFC Animation:**

LottieFiles - Hugo Saraiva. (s.f.).

<https://lottiefiles.com/free-animation/scanning-blue-version-W4EesGZYRb>

- **Profile Animation:**

LottieFiles - Fun Spyer (2025, abril). <https://lottiefiles.com/free-animation/profile-CysrKbRX3y>

- **Detalles de Sesión Animation:**

LottieFiles - Oscar luna (2025) . <https://lottiefiles.com/free-animation/carpool-JaMJGSRbtT>

- **Presentación de la app:**

Figma - Perla Baylon. (20205, mayo 25).

<https://www.figma.com/proto/kXuzWqsqu39MGbNluoboKP/LOGO?node-id=55-8&p=f&t=IOrgcuNZSN8WzjDG-1&scaling=scale-down&content-scaling=fixed&page-id=0%3A1&starting-point-node-id=52%3A83>

- **Navegation Drawer:**

Se siguió las indicaciones de el siguiente canal:

Youtube - Stevdza-San. (2020). <https://www.youtube.com/watch?v=pucQsoTUeol&t=73s>

- **NFC:**

Las pruebas de NFC y en la presentación del proyecto se hicieron con un tag.

Backend:

1. Spring Boot (Java) — REST API development

- Se utilizó Spring Web, Spring Security, Spring Data JPA, JWT, etc.
- El desarrollo se basó en la documentación oficial de Spring:
 - Spring Boot. (s.f.). <https://spring.io/projects/spring-boot>
 - Spring Boot. (s.f.). <https://docs.spring.io/spring-security/reference/index.html>

2. JWT (JSON Web Token) para autenticación

Se aplicó el estándar JWT como método de autenticación y autorización en la API, conforme a las recomendaciones de Spring Security.

Recomendación utilizada:

- JWT. (s.f.). <https://jwt.io/introduction>

3. Integración de APIs en Android

En la aplicación móvil se integraron diversas APIs oficiales de Android, como Retrofit para conexiones HTTP, SharedPreferences para almacenamiento local, y la API de NFC para comunicación de campo cercano. Toda la implementación se guió por la documentación oficial de Android:

- Develop for Android. (s.f.). <https://developer.android.com/docs>
- NFCAdapter. (s.f.). <https://developer.android.com/reference/android/nfc/NfcAdapter>

4. Diseño general de API REST y buenas prácticas

Se aplicaron buenas prácticas de diseño REST, incluyendo el uso adecuado de códigos de estado HTTP, patrones de solicitud, uso de DTOs (objetos de transferencia de datos) y arquitectura en capas.

Estas decisiones se tomaron con base en estándares de arquitectura comprobados en el desarrollo profesional de APIs RESTful.

