



**UNIVERSIDAD PONTIFICIA DE
COMILLAS - ICAI**

**Grado en Ingeniería en Tecnologías de
Telecomunicación**

TRABAJO FINAL DE GRADO
DESARROLLO DE SOFTWARE PARA LA
GESTIÓN DE PARQUES TEMÁTICOS
CON INTERNET OF THINGS

Autor: Yago Tobío Souto

Director: Atilano Fernández-Pacheco Sánchez-Migallón

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título
**Desarrollo de Software para la Gestión de Parques Temáticos con
Internet of Things**

.....
en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el
curso académico2022/2023..... es de mi autoría, original e inédito y
no ha sido presentado con anterioridad a otros efectos. El Proyecto no es
plagio de otro, ni total ni parcialmente y la información que ha sido tomada
de otros documentos está debidamente referenciada.



Fdo.: **Yago Tobio Souto**

Fecha: ...05.../ ...06.../ ...2023...

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

71216314B

ATILANO RAMIRO

FERNÁNDEZ-

PACHECO

Firmado digitalmente
por 71216314B

ATILANO RAMIRO

FERNÁNDEZ-PACHECO

Fecha: 2023.06.05

11:01:01 +02'00'

Fdo.: **Atilano Ramiro Fernández-Pacheco** Fecha: ...05.../ ...06.../ 2023

Fecha: 05/06/2023

Agradecimientos:

Me gustaría agradecerle a **mi padre y a mi madre**, por mostrarme el valor del trabajo duro y la perseverancia.

Me gustaría agradecerles a **mis hermanos**, por ser una luz constante en mi día a día y sacar lo mejor de mí.

Me gustaría agradecerle a **mi abuela**, por haberme dado la oportunidad de haber cursado una carrera tan especial.

Me gustaría agradecerle a **mis amigos de la universidad**, por ser mi fuente de positividad, entusiasmo y apoyo en todo momento alto y bajo.

y finalmente, me gustaría agradecerle a **mi director del Trabajo de Final de Grado, Atilano**, por su constante entusiasmo y guía experta durante el desarrollo del proyecto.

Índice general

1. Project Abstract	7
1.1. Introduction	7
1.2. Project Structure	7
1.3. Project Description	8
1.4. Results	9
1.5. Conclusions	9
2. Resumen del proyecto	11
2.1. Introducción	11
2.2. Definición del proyecto	11
2.3. Descripción del sistema	12
2.4. Resultados	13
2.5. Conclusiones	14
3. Introducción	15
4. Descripción de las tecnologías	17
4.1. Front-end	17
4.1.1. React.js	17
4.1.2. Lenguajes Web: HTML, CSS, JavaScript	21
4.2. Back-end	22
4.2.1. Python	22
4.2.2. Django	23
4.2.3. SQLite	24
4.3. Otras tecnologías usadas	25
4.3.1. Git y GitHub	25
4.3.2. Visual Studio Code	26
5. Estado del arte	29
5.1. Microservicios	29
5.1.1. Análisis de ventajas y desventajas de ambas arquitecturas	30
5.2. <i>Internet of Things</i> (IoT)	31
5.3. API, Metodología REST y CRUD	32
5.4. Desarrollo Cross-Platform	33
5.5. Cultura DevOps	33
5.6. Teoría de Colas	34

5.6.1. Introducción	34
5.6.2. Notación y Terminología	34
5.6.3. Modelos y clasificación	35
5.6.4. Sistema de colas o procesos	36
5.6.5. Proceso de Llegada	36
5.6.6. Mecanismo o Facultad de Servicio	38
5.6.7. Definición de estados transitorios y estacionarios	39
5.6.8. Notación de Kendall y la clasificación de los modelos.	39
5.6.9. Distribuciones en sistemas de colas	40
5.6.10. Modelo Markoviano M/M/1 – Cola actual en parques de atracciones	42
5.6.11. Modelo de cola multicanal: M/M/c: (α /FCFS)	43
6. Definición del Trabajo	45
6.1. Motivación	45
6.2. Objetivos del proyecto	46
6.3. Metodología	46
6.4. Planificación y estimación económica	47
6.4.1. Estudio económico a 5 años vista	49
7. Sistema Desarrollado	51
7.1. Simulación del sistema de colas	52
7.2. Diagrama de casos de usos	53
7.3. Diseño de microservicios	54
7.3.1. Domain Driven Design y el diagrama de clases	57
7.3.2. Microservicio del Administrador	59
7.3.3. Microservicio de Clientes	59
7.3.4. Autorización mediante Token en el Microservicio de Clientes	60
7.3.5. Microservicio de Manejo de Entradas	62
7.3.6. Cifrado para la generación de códigos QR como entradas	64
7.3.7. Microservicio de las Colas	65
7.3.8. Microservicio de tiendas	67
7.3.9. Microservicio de restaurantes	68
7.4. Desarrollo de Back-End usando Django	69
7.4.1. models.py	69
7.4.2. serializers.py	72
7.4.3. admin.py	75
7.4.4. urls.py	76
7.4.5. views.py	77
7.4.6. SQLite y Django Admin	79
7.5. Vista del Administrador	80
7.5.1. Añadir empleados al parque	80
7.5.2. Añadir atracciones al parque	82
7.6. API	87
7.6.1. Diagramas de Secuencia	91
7.7. Desarrollo de front-end móvil usando React Native	92
7.7.1. Desarrollo en React Native	92
7.7.2. Inicio de sesión y gestión de la cuenta	95

7.7.3. Compra de entradas y gestión de huéspedes	101
7.7.4. Exploración de los parques de atracciones	106
7.7.5. Reservas de atracciones	109
8. Resultados del Trabajo	113
8.1. Análisis de los resultados de la simulación	114
8.2. Análisis de la aplicación móvil front-end	115
8.3. Análisis del back-end y el dashboard del administrador	116
9. Trabajo Futuros	119
9.1. Conclusiones y resultados principales	119
9.2. Trabajos Futuros	120
9.2.1. Desarrollo de mapa virtual - GPS	120
9.2.2. Visualización de tiendas y restaurantes e implementación Pay2Go	120
9.2.3. Aplicación Empleado - Scanner QR y gestión particular	121
9.2.4. Desarrollo de generador de itinerarios usando Machine Learning basado en las características del grupo	121
9.2.5. Dashboard de Analíticas para los administradores del parque de atracciones	121
9.2.6. Deployment de la aplicación en la nube y en los mercados de aplicaciones móviles	122
9.2.7. Desarrollo del sistema de colas virtual y refinamiento basado en modelos reales	123
10. Bibliografía	125
11. Anexo - Manual de Instalación	129
11.1. Instalación del Back-end y del entorno de desarrollo	129
11.1.1. Instalación de Visual Studio Code	129
11.1.2. Instalación de Python y pip	130
11.1.3. Descarga del proyecto	131
11.1.4. Abriendo el proyecto en Visual Studio Code	131
11.1.5. Abriendo una terminal con permisos de administrador	133
11.1.6. Navegación al directorio del proyecto	133
11.1.7. Instalación de los requisitos del proyecto	134
11.1.8. Arranque del servidor	134
11.2. React Native (front-end)	135
11.2.1. Instalación de Node.js y npm	136
11.2.2. Verificación de la instalación	136
11.2.3. Reinicio del sistema	136
11.2.4. Instalación de Yarn	137
11.2.5. Instalación de Expo CLI	137
11.2.6. Instalación de las dependencias del proyecto	137
11.2.7. Inicio de la aplicación	138
11.3. Instalación de la aplicación Expo en su dispositivo móvil	138
11.3.1. Escaneo del código QR	139
12. Anexo - Manual de Instalación	141

12.1. Introducción	141
12.2. Manual por parte del administrador	141
12.2.1. Acceso al dashboard del administrador	141
12.2.2. Manejo de modelos	145
12.3. Manual por parte del cliente	147
12.4. Vistas de un usuario no autenticado	147
12.4.1. Vista de las atracciones	148
12.5. Vistas de un usuario autenticado	150
12.5.1. Inicio de sesión	151
12.5.2. Gestión de datos personales	152
12.5.3. Sacar entradas para el parque de atracciones	153
12.5.4. Visualizar entradas en la aplicación	155
12.5.5. Editar la información de sus acompañantes	156
12.5.6. Hacer reservas para parques de atracciones a horas específicas.	157
12.5.7. Visualizar reservas en la aplicación	158
13. Anexo - Alineación con los Objetivos de Desarrollo Sostenible	159

Capítulo 1

Project Abstract

1.1. Introduction

This project proposes the development of a software solution for the management of theme parks, incorporating the principles of microservices and Internet of Things (IoT) features. The primary focus will be the establishment of a virtual queuing system, minimizing idle time for park visitors. The project encompasses two key components: a mobile application for visitors and a web application for the park's administrators and employees.

1.2. Project Structure

Motivated by the evolution of technologies such as IoT, cloud, cross-platform development, and virtual queuing systems, this project employs the concept of virtualization [\[Services \(2018b\)\]](#). This technology enables the creation and operation of virtual resources and instances of physical resources, such as computers, servers, and storage, within a single device, thereby promoting resource efficiency.

The application follows a microservice-like architecture. Microservices involve building an application as a suite of small services that run in their own processes and communicate using lightweight mechanisms, such as an HTTP-based API. Each microservice is dedicated to performing an individual, complete, and independent function [\[Services \(2018a\)\]](#), contrasting with monolithic architectures where functionalities are interconnected, hence less fault-tolerant.

The services are instantiated and contained within virtual containers, managed by Docker and Kubernetes. Docker is responsible for creating these containers, while Kubernetes ensures their efficient deployment, scaling, and load-balancing [\[Atlassian \(2017\)\]](#).

The project aims to:

- Implement a microservice architecture using Docker and Kubernetes.
- Develop a user-friendly mobile application for visitors to enhance their theme park experience, particularly via virtual queueing for theme park rides.

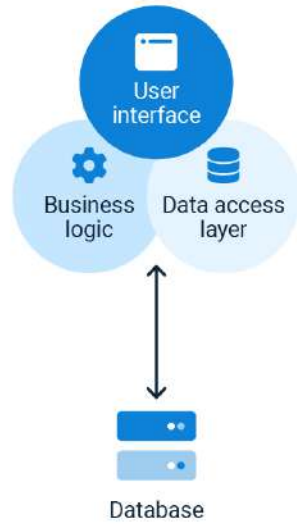
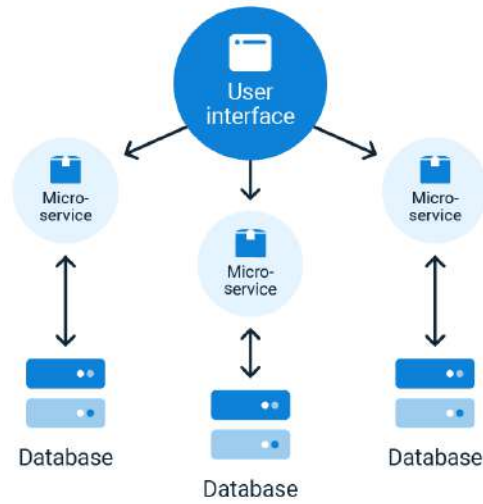
Monolithic Architecture**Microservice Architecture**

Figura 1.1: Monolithic vs. Microservice Architectures

- Establish a real-time, analytic dashboard for park managers to modify park attributes and extract valuable usage data.

1.3. Project Description

The microservices architecture, as illustrated below, forms the backbone of the system. Each microservice is tasked with a specific, independent function, and their interactions with different applications are highlighted.

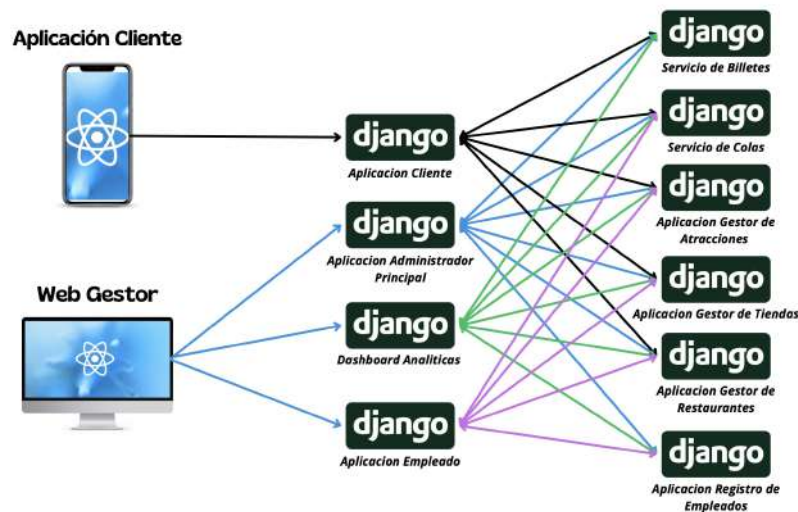


Figura 1.2: Project Microservices Architecture

Key microservices and elements of the architecture include:

- **Client Application:** This is the visitors' mobile application that interfaces with all microservices.
- **Main Administrator Application:** The park management's web portal grants access to all project microservices.
- **Analytics Dashboard:** This microservice interacts with the queue service and gathers data about the park's services.
- **Staff App:** Park administrators manage the theme park's employees via this app, providing different roles and permissions.
- **Ticket Services:** This platform facilitates user registration and ticket purchases for theme park visits.
- **Virtual Queuing Service:** This microservice administers the virtual queues for each theme park ride.
- **Theme Park API Information Service:** This microservice allows modification and communication of all park properties.

1.4. Results

The project has yielded satisfactory results, successfully delivering a web application for theme park administrators and a mobile app for visitors.

The web application enables real-time park management, with changes instantly reflected on the user app. This results in a highly efficient tool for organizers and administrators.

By incorporating a REST API and token-based authentication, I have developed a secure and efficient communication channel between the two apps.

On the client side, a React Native cross-platform application provides an intuitive and interactive experience for park visitors. It offers a variety of features that may prove essential during their visit.

Apart from achieving the goals of this project, I have strived to adhere to best programming practices and methodologies such as DRY (Don't Repeat Yourself), KISS (Keep It Simple, Stupid), and the SOLID framework [Domareski (2020)].

1.5. Conclusions

The project, Virtual Q, stemmed from my initial dissatisfaction due to the inability to fully experience a theme park on peak days. I found it unacceptable that large-scale corporations could subject their customers to such mundane and stagnant periods of waiting, especially in contrast to the excitement of their rides and other attractions.

This endeavour has not only enabled me to devise a potential solution to this issue, but has also provided me with profound insights into queueing theory systems, a multi-

tude of back-end frameworks, database management, and front-end mobile application development.

I must concede that the potential scope of this project could reach astronomical proportions. However, its current state serves as a solid foundation and starting point for the application. Furthermore, it illustrates the array of skills I have acquired during its progression.

Capítulo 2

Resumen del proyecto

2.1. Introducción

El mundo moderno es digital y la tecnología y el software están en constante evolución. Los avances en las aplicaciones y en la web han revolucionado todas las industrias a nivel global. Sin embargo, los parques de atracciones son algunas de las pocas áreas de ocio cuya gestión no ha sido completamente digitalizada [Oracle (2021)]. Por lo tanto, mediante el desarrollo de una aplicación full stack, nuestro objetivo es digitalizar la experiencia en los parques de atracciones tanto para los usuarios como para los administradores.

2.2. Definición del proyecto

Este proyecto se motiva por las nuevas tecnologías de IoT, Cloud, desarrollo multiplataforma y el sistema de colas virtuales. La aplicación utiliza la virtualización [Services (2018b)], una tecnología innovadora para el uso eficiente de recursos, la cual permite crear versiones virtuales o simuladas de recursos físicos como computadoras, servidores, sistemas operativos, redes y almacenamiento.

La arquitectura de la aplicación se basa en torno a los microservicios. Este enfoque consiste en construir una aplicación como un conjunto de pequeños servicios, los cuales se ejecutan en su propio proceso y se comunican con mecanismos ligeros, como una API basada en HTTP. Cada microservicio se encarga de implementar una funcionalidad individual, completa e independiente [Services (2018a)]. Esto se diferencia de las arquitecturas monolíticas en las que las funcionalidades están interconectadas y son menos tolerantes a fallos, como se puede observar a continuación:

Una vez finalizado el desarrollo de los microservicios correspondientes, estos se encapsularán en contenedores virtuales. Para ello, se utilizará Docker, que crea dichos contenedores, y Kubernetes, que se encargará de desplegar, detener y escalar dichos contenedores según la demanda de la aplicación [Atlassian (2017)].

Los objetivos del proyecto son los siguientes:

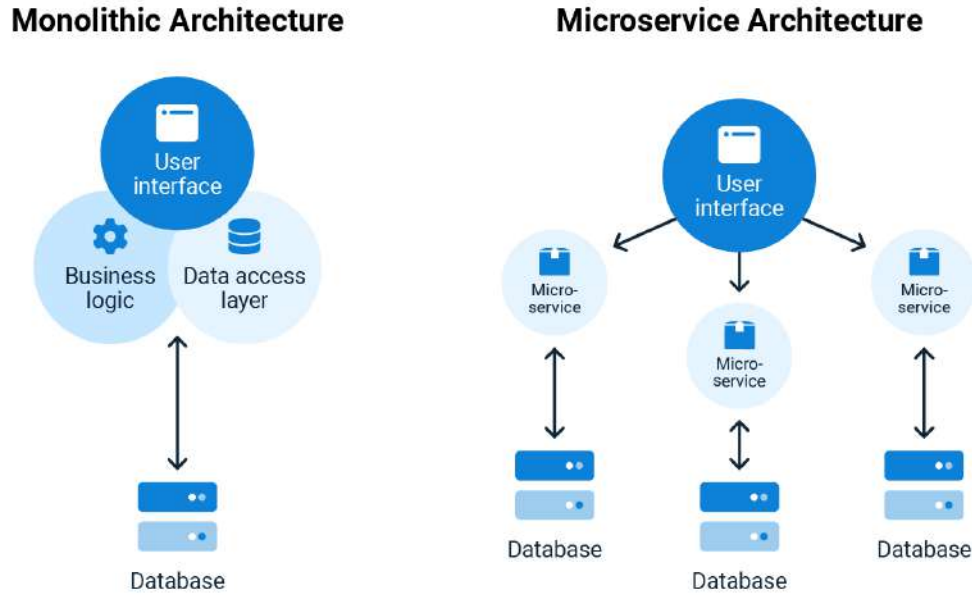


Figura 2.1: Arquitectura de Microservicios vs. Monolítica

- Crear una aplicación móvil con React Native, para ambas plataformas (iPhone y Android), facilitando así a los visitantes de los parques temáticos su experiencia, con especial énfasis en la posibilidad de unirse a las colas virtuales de todas las atracciones.
- Desarrollar un tablero de control para los gestores del parque, desde el cual puedan modificar en tiempo real las propiedades del parque y obtener análisis del uso para extraer información valiosa.

2.3. Descripción del sistema

Como hemos destacado previamente, lo más importante del sistema es la arquitectura de microservicios, los cuales se pueden observar en la siguiente figura. Cada uno de estos tiene una funcionalidad concreta e independiente, y se puede ver cuáles interactúan con qué aplicación.

Los microservicios y elementos de la arquitectura son los siguientes:

- **Aplicación Cliente:** Esta es la aplicación móvil que proporcionará a los usuarios todas las herramientas e información necesarias para disfrutar de una exitosa visita al parque.
- **Aplicación del Administrador Principal:** La interfaz web del administrador del parque de atracciones, el cual podrá administrar toda la información mostrada al usuario.
- **Dashboard de Analíticas:** Este microservicio tendrá acceso al servicio de colas y a toda la información de los servicios del parque (atracciones, restaurantes,

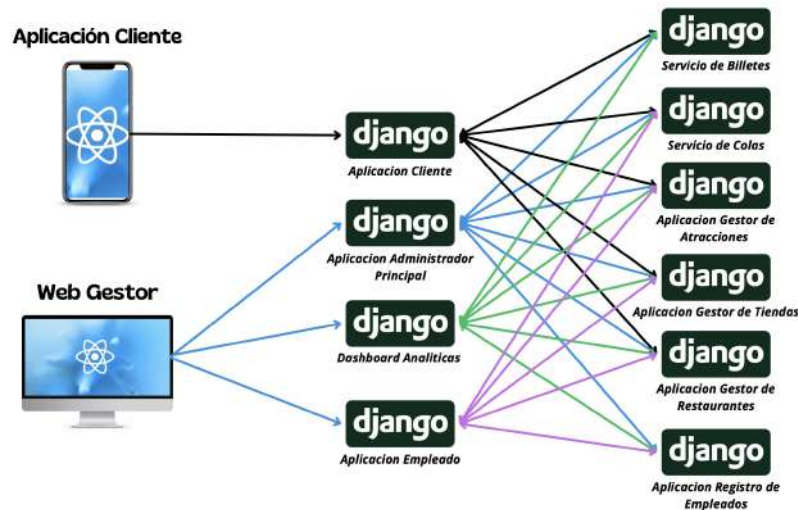


Figura 2.2: Figura 2: Arquitectura de Microservicios del proyecto

empleados, tiendas, etc.).

- **Aplicación para Empleados:** Los gestores del parque tendrán una base de datos en la cual podrán asignar a los empleados distintos roles dentro del parque. Dependiendo del rol que tengan, tendrán acceso para modificar u observar las diferentes propiedades de los servicios del parque.
- **Servicio de Boletos:** Plataforma en la que los clientes podrán comprar sus boletos o registrar a los miembros de su visita.
- **Servicio de Colas:** Servicio en el cual se administrarán las colas virtuales para cada atracción.
- **Servicio API de Información del Parque:** Servicio en el que se manejarán todas las propiedades del parque. Se puede modificar información sobre las tiendas, restaurantes, atracciones y empleados.

2.4. Resultados

Los resultados de este Trabajo de Fin de Grado han sido satisfactorios. Se ha logrado desarrollar e implementar una página web para los administradores que permite actualizar la información sobre el parque de atracciones en tiempo real, cambios que se reflejan instantáneamente en la aplicación móvil. Esto resulta en una aplicación eficiente y rápida para los administradores de parques de atracciones y eventos.

Gracias a la creación de una API REST y autenticación basada en tokens, se ha construido una forma eficiente y segura de comunicación entre ambas aplicaciones.

La aplicación móvil se ha desarrollado de forma efectiva y completa, proporcionando al usuario una interfaz fácil, interactiva e intuitiva para su visita al parque de atracciones. La aplicación cuenta con una variedad de vistas desde las cuales el usuario puede realizar diversas acciones útiles para su visita.

Además de los objetivos cumplidos de este trabajo, se ha tenido en cuenta constantemente las metodologías y buenas prácticas de programación en el desarrollo del proyecto, como DRY (Don't Repeat Yourself), KISS (Keep it Simple, Stupid) y la metodología SOLID [Domareski (2020)].

2.5. Conclusiones

El proyecto, Virtual Q, se originó a partir de mi insatisfacción inicial debido a la incapacidad de experimentar completamente un parque temático en días pico. Me pareció inaceptable que las corporaciones de gran escala pudieran someter a sus clientes a períodos de espera tan mundanos y estancados, especialmente en contraste con la emoción de sus atracciones y otras ofertas.

Este emprendimiento no solo me ha permitido idear una posible solución a este problema, sino que también me ha brindado profundos conocimientos en sistemas de teoría de colas, una multitud de marcos de trabajo para back-end, gestión de bases de datos y desarrollo de aplicaciones móviles front-end.

Debo admitir que el alcance potencial de este proyecto podría alcanzar proporciones astronómicas. Sin embargo, su estado actual sirve como una base sólida y un punto de partida para la aplicación. Además, ilustra la gama de habilidades que he adquirido durante su desarrollo.

Capítulo 3

Introducción

La revolución informática e internet han sido catalizadores de cambio para individuos, comunidades y empresas, mejorando la eficiencia, disponibilidad y globalización de procesos. Los sistemas informáticos contemporáneos buscan ser más eficientes, seguros y escalables.

La transformación digital ha impulsado a organizaciones a modernizar sus recursos tecnológicos para mantener la competitividad. Estos avances ofrecen mejoras en la disponibilidad y eficiencia de los servicios.

En la actualidad, la mayoría de las personas cuentan con un dispositivo móvil con capacidades tecnológicas superiores a las del cohete que llevó al hombre a la luna. Estos dispositivos, con sus diversas aplicaciones, forman parte de una compleja infraestructura interconectada. Sin embargo, hay ámbitos en nuestra sociedad donde aún no se ha aprovechado plenamente la tecnología.

La forma de consumir ocio ha evolucionado considerablemente en la última década. Aunque el entretenimiento al aire libre sigue siendo una parte importante de nuestras vidas, especialmente después de una pandemia global, eventos y parques de atracciones han sido lentos en incorporar nuevas tecnologías que mejoren la experiencia del usuario.

Un problema prevalente es la formación de largas colas en eventos y atracciones, lo cual es frustrante e ineficiente en un mundo acostumbrado al acceso instantáneo. Este Trabajo Fin de Grado abordará este problema a través del uso de tecnologías del Internet de las Cosas (IoT). Se analizarán diversas metodologías de colas e implementaciones tanto en una aplicación móvil para usuarios, como en una aplicación gestora para empleados de un parque de atracciones o evento correspondiente. Este objetivo se alcanzará utilizando una arquitectura de microservicios para administrar aplicaciones de software IoT.

La motivación detrás de este proyecto es la aplicación de conceptos y tecnologías innovadoras que lideran el desarrollo web y de software.

Este documento presentará las tecnologías utilizadas en el proyecto, discutirá el estado actual del arte y describirá el trabajo realizado. Finalmente, se presentarán las aplicaciones y sistemas desarrollados, se analizarán los resultados obtenidos y se discu-

tirán conclusiones y futuros trabajos. Se incluirá una guía de instalación y manual de usuario, junto con reflexiones sobre el estado actual de los parques de atracciones y la importancia de los Objetivos de Desarrollo Sostenible en relación con el proyecto.

Capítulo 4

Descripción de las tecnologías

En ese capítulo se mencionarán y explicarán las tecnologías y herramientas usadas para el desarrollo de este trabajo: Django, Python y React Native, entre otras.

4.1. Front-end

El *front-end* corresponde a la parte que el usuario final utiliza para interactuar con el servidor de la aplicación. En este proyecto, hay 2 secciones front-end:

- La aplicación cross-platform desarrollada con React Native, la cual permitirá a usuarios acceder desde cualquier dispositivo iPhone o Android a la aplicación del parque de atracciones, en la cual la cantidad de potenciales usuarios siendo todos los que posean una conexión a internet.
- La página web del administrador del parque de atracciones administrada completamente por el framework proporcionado por Django, en la cual se podrá manipular la información del parque de manera fácil, customizable e intuitiva.

Típicamente, las aplicaciones front-end constan de tres secciones: El contenido, el estilo y la funcionalidad. Aun así, se podrá observar más adelante en la documentación, que React Native permite el ahorro de tener que implementar el contenido y la funcionalidad de manera separada gracias a su base en JSX.

4.1.1. React.js

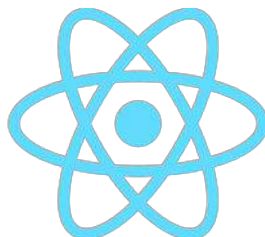


Figura 4.1: Logotipo de React.js

React.js [Meta (2015)], o simplemente React, es una biblioteca de JavaScript de código abierto utilizada para construir aplicaciones web e interfaces de usuario de una página. (SPA, Single Page Applications)¹.

Desarrollado y mantenido por Meta, React permite a los desarrolladores crear componentes reutilizables de la interfaz de usuario y gestionar el estado y renderización de los mismos.

React Native es un marco de desarrollo de aplicaciones móviles de código abierto basado en React [Meta (2017)]. La cual permite a los desarrolladores crear aplicaciones móviles nativas para plataformas como iOS y Android usando JSX y la sintaxis de React.

Al contrario que las aplicaciones web móviles o las aplicaciones híbridas, React nativo compila componentes de la interfaz de usuario en vistas nativas específicas de la plataforma, lo que permite un rendimiento más cercano al de las aplicaciones móviles nativas tradicionales. Una de sus principales ventajas es que los desarrolladores pueden reutilizar una gran cantidad del código entre ambas plataformas iOS y Android, lo cual reduce el tiempo y los recursos necesarios para desarrollar aplicaciones móviles en ambas plataformas.

```
1      import React from 'react';
2
3      class App extends React.Component {
4          render() {
5              return (
6                  <div>
7                      <h1>Hello, World!</h1>
8                  </div>
9              );
10         }
11     }
12     export default App;
```

Listing 4.1: Programa Hello World usando React.js

Entorno de desarrollo de React

El entorno de desarrollo para React.js y React Native consta de varias herramientas y tecnologías que facilitan el proceso de creación y mantenimiento de aplicaciones basadas en estas bibliotecas. Algunas de las herramientas y tecnologías clave en el entorno de desarrollo incluyen:

Node.js: React y React Native utilizan Node.js [Dahl (2010)], un entorno de tiempo de ejecución de JavaScript basado en el motor V8 de Chrome, para ejecutar tareas del lado del servidor y administrar dependencias.

Create React App (para React.js): Es una herramienta de línea de comandos que permite a los desarrolladores configurar rápidamente un entorno de desarrollo de React.js. Genera automáticamente una estructura de archivos y una configuración óptima para proyectos de React, permitiendo a los desarrolladores comenzar a trabajar en sus aplicaciones sin preocuparse por la configuración inicial.

¹A web application that dynamically updates the content on a single web page, providing a seamless user experience without the need for page reloads.

Expo (para React Native): Expo es un marco y una plataforma que simplifica el proceso de desarrollo de aplicaciones React Native [Expo (2018)]. Permite a los desarrolladores crear, implementar y probar aplicaciones en dispositivos reales sin tener que pasar por el proceso de compilación nativa. Expo también proporciona herramientas y servicios adicionales, como Over-the-Air (OTA) updates y una amplia gama de componentes listos para usar.

yarn packet manager: Yarn es un administrador de paquetes para JavaScript, que se utiliza para compartir y usar código de JavaScript con otros desarrolladores de todo el mundo. Fue desarrollado por Facebook, Exponent, Google y Tilde para solucionar una serie de problemas con npm, otro popular administrador de paquetes de JavaScript [Krishna (2023)]. El administrador de paquetes es una herramienta que permite a los desarrolladores automatizar la instalación, actualización, configuración y remoción de paquetes de software. Los paquetes son módulos de código que proporcionan funcionalidad adicional para su aplicación.

Editores de código: Los desarrolladores pueden utilizar una variedad de editores de código y entornos de desarrollo integrados (IDE) para trabajar con React y React Native. Algunos de los más populares incluyen Visual Studio Code, Sublime Text, Atom y WebStorm. Estos editores a menudo ofrecen extensiones y complementos específicos para React y React Native que mejoran la productividad y facilitan la detección de errores y el resaltado de sintaxis.

Herramientas de depuración: Las herramientas de depuración ayudan a los desarrolladores a identificar y solucionar problemas en sus aplicaciones. React Developer Tools es una extensión para navegadores como Chrome y Firefox que permite inspeccionar el árbol de componentes de React y observar el estado y las propiedades de los componentes. React Native Debugger es una herramienta similar para depurar aplicaciones React Native.

Bundlers y transpiladores: Herramientas como Webpack, Babel y Metro Bundler son fundamentales para el proceso de desarrollo de aplicaciones React y React Native. Webpack y Metro Bundler son utilizados para empaquetar y optimizar los archivos JavaScript y otros recursos, mientras que Babel permite utilizar características modernas de JavaScript y asegurar la compatibilidad con navegadores y plataformas más antiguas.

Hooks de React

Los hooks son una característica fundamental de React que permite utilizar el estado y otras características de React en componentes funcionales, en lugar de utilizar componentes de clase. Los hooks ofrecen una forma más sencilla y directa de trabajar con el estado y los efectos secundarios en los componentes funcionales, mejorando la legibilidad y reutilización del código. Es lo que ha hecho que la plataforma sea tan utilizada en el entorno de desarrollo web.

Aquí hay una descripción general de algunos de los hooks más comunes y útiles en React:

1. **useState:** Este hook permite a los componentes funcionales tener y manipular el estado [Meta (2016d)]. Devuelve un par de valores, el estado actual y una función

para actualizar ese estado. Por ejemplo:

```
1 const [count, setCount] = useState(0);
```

Listing 4.2: State hook de React.js

2. **useEffect**: Este hook se utiliza para realizar efectos secundarios en componentes funcionales, como solicitudes de red, suscripciones o manipulaciones del DOM^[2] **Meta (2016b)**. Puedes pensar en `useEffect` como una combinación de los métodos `componentDidMount`^[3], `componentDidUpdate`^[4] y `componentWillUnmount`^[5] en los componentes de clase. Por ejemplo:

```
1 useEffect(() => {  
2     //Codigo para ejecutar el efecto secundario  
3     return () =>{  
4         //Limpieza del efecto secundario  
5     };  
6 },[dependencias]); //Array de dependencias para controlar cuando se ejecuta y  
    limpia el efecto
```

Listing 4.3: Effect hook de React.js

3. **useContext**: Este hook permite a los componentes funcionales acceder al valor actual de un contexto, sin necesidad de utilizar un componente `Context.consumer`^[6] **Meta (2016a)**. Por ejemplo:

```
1 const theme = useContext(ThemeContext);
```

Listing 4.4: Context hook de React.js

4. **useRef**: Este hook permite a los componentes funcionales mantener referencias a elementos del DOM o a instancias de componentes. Devuelve un objeto mutable con una propiedad `.current`^[7] **Meta (2016c)**. Por ejemplo:

```
1 const inputRef = useRef(null);
```

Listing 4.5: Ref hook de React.js

Estos son los hooks principales de React usados en este proyecto. Se pueden crear propios hooks personalizados para extraer lógica de componentes y hacerla reutilizable en otros componentes o aplicaciones.



Figura 4.2: Logotipo del conjunto de tecnologías web HTML, CSS, JavaScript

4.1.2. Lenguajes Web: HTML, CSS, JavaScript

HTML (Hypertext Markup Language) es el lenguaje de marcado estándar utilizado en la creación y estructuración de contenido web. Permite definir la estructura básica de una página web, incluidos encabezados, párrafos, listas, tablas, imágenes y enlaces. En el proyecto de final de grado, HTML es fundamental para definir la estructura de la aplicación web y presentar la información de la cola virtual de manera organizada y accesible.

```

1      <!DOCTYPE html>
2      <html>
3      <head>
4          <title>Ejemplo</title>
5      </head>
6      <body>
7          <h1>Hello, world!</h1>
8      </body>
9      </html>

```

Listing 4.6: Ejemplo simple de archivo HTML

CSS (Cascading Style Sheets) es un lenguaje de hojas de estilo que se utiliza para describir la apariencia y el diseño de una página web. Con CSS, se pueden controlar aspectos como el color, el tamaño de fuente, el espaciado, las disposiciones y las animaciones, permitiendo una mayor flexibilidad y consistencia en el diseño de la aplicación web. En el contexto del proyecto, CSS es relevante para garantizar que la interfaz de usuario sea atractiva, fácil de usar y consistente en todos los dispositivos y navegadores.

```

1      body {
2          background-color: #f0f0f0;
3          font-family: Arial, sans-serif;
4          text-align: center;

```

²Es una interfaz de programación que representa documentos HTML y XML como una estructura de nodos y objetos, permitiendo que los lenguajes de programación interactúen con ellos

³Es un método de ciclo de vida en React que se ejecuta después de que la salida del componente ha sido renderizada en el DOM.

⁴Es un método de ciclo de vida en React que se ejecuta después de que un componente se actualiza, es decir, después de que se produce una re-renderización.

⁵Es un método de ciclo de vida en React que se ejecuta justo antes de que un componente sea destruido o eliminado del DOM.

⁶Es un componente React que se suscribe a cambios de contexto, usado con la API Context para acceder a los valores del contexto

⁷Es una propiedad mutable que mantiene una referencia persistente al valor actual a lo largo de los renderizados en React.

```
5 | }
```

Listing 4.7: Ejemplo simple de archivo CSS. Cambio de color de fondo, alineacion de texto y tipografia de letra

JavaScript es un lenguaje de programación de alto nivel, interpretado y orientado a objetos que se ejecuta en el navegador del cliente. JavaScript permite agregar interactividad, lógica y comportamiento dinámico a una página web, mejorando la experiencia del usuario. En el proyecto de final de grado, JavaScript es crucial para gestionar la interacción del usuario con la cola virtual, realizar actualizaciones en tiempo real y comunicarse con el servidor para obtener y enviar información.

```
1   var button = document.getElementById("myButton");  
2  
3   button.addEventListener("click", function() {  
4       alert("Button clicked!");  
5   });
```

Listing 4.8: Ejemplo simple de archivo JavaScript que causa una alerta cuando se hace clic sobre un boton

En conjunto, HTML, CSS y JavaScript colaboran para formar una página web completa y funcional. HTML proporciona la estructura y el contenido, CSS controla la presentación y el diseño, y JavaScript añade interactividad y comportamiento dinámico [Camps (2021)]. Estos tres lenguajes de programación web son fundamentales en el desarrollo de la aplicación web para el sistema de colas virtuales, ya que permiten crear una solución efectiva y atractiva para mejorar la experiencia del usuario en el parque de atracciones y reducir los tiempos de espera.

4.2. Back-end

El backend, o el lado del servidor, se refiere a la parte de un sistema informático que se encarga de la lógica y la funcionalidad del servidor, proporcionando la infraestructura necesaria para que una aplicación o sitio web funcione correctamente.

Es la capa responsable de manejar la lógica, la manipulación de datos y la comunicación con las partes del sistema, como las bases de datos y los servicios externos.

Se encarga de recibir las solicitudes del cliente (típicamente a través de una API o interfaz de usuario) y proporcionar las respuestas correspondientes. Lo cual implica realizar operaciones de procesamiento, la recuperación o el almacenamiento de datos, entre otras tareas necesarias para que la aplicación funcione de manera correcta.

4.2.1. Python

Python es un lenguaje de programación multipropósito. Relevante para el desarrollo de aplicaciones web debido a su facilidad de uso, versatilidad, ecosistema de bibliotecas y frameworks, portabilidad, comunidad y capacidad de integración con otras tecnologías.

En este proyecto, se usa Python v3 para el desarrollo back-end al igual que el módulo de microservicios. Mediante el framework back-end de Django se programará la sección



Figura 4.3: Logotipo de Python

de servidores y front-end. Una de las características de Python es el uso de hilos de ejecución simultáneos, por lo que podrá realizar distintos procesos al mismo tiempo.

```
1 print('Hello world!')
```

Listing 4.9: Ejemplo de sintaxis en Python

4.2.2. Django



Figura 4.4: Logotipo de django

Django es un marco de trabajo (*framework*) de desarrollo web de código abierto y alto nivel escrito en Python. Fue diseñado para facilitar y agilizar el proceso de desarrollo de aplicaciones web, permitiendo a los desarrolladores crear aplicaciones rápidamente y con menos esfuerzo, siguiendo el principio de “no reinventar la rueda”⁸ [Foundation (2018)]. Django sigue el patrón de diseño Model-View-Template (MVT)⁹ y se centra en la reutilización y la conexión de componentes.

Django ofrece muchas características integradas que facilitan tareas comunes en el desarrollo web, como:

1. Un sistema de mapeo objeto-relacional (ORM)¹⁰ que permite interactuar con bases de datos utilizando objetos y consultas de Python en lugar de SQL.
2. Soporte para la creación y administración de formularios.
3. Un sistema de autenticación y autorización que permite a los desarrolladores gestionar de manera sencilla el acceso y permisos de los usuarios.
4. Un potente sistema de plantillas para la creación de interfaces de usuario.
5. Soporte para la internacionalización y localización de aplicaciones.

⁸Es un principio de desarrollo de software que aconseja el uso de soluciones ya existentes y probadas en lugar de crear nuevas soluciones para problemas ya resueltos.

⁹Es una variación del patrón MVC en la que los Templates representan la capa de presentación, los Models definen la estructura de los datos y las Views procesan y preparan los datos para los Templates.

¹⁰Es una técnica de programación para convertir datos entre sistemas de tipos incompatibles en lenguajes de programación orientados a objetos y bases de datos relacionales.

6. Herramientas de administración automática que permiten generar interfaces de administración para las aplicaciones.
7. Capacidad para manejar múltiples aplicaciones modulares dentro de un proyecto Django.

El ecosistema de Django también incluye una gran cantidad de paquetes y extensiones desarrolladas por la comunidad que pueden ser utilizadas para ampliar aún más las capacidades y funcionalidades del framework.

Django es especialmente adecuado para aplicaciones web que requieren un manejo sólido de bases de datos, una estructura bien organizada y escalabilidad. Es utilizado por muchas organizaciones y proyectos de diferentes tamaños, incluidos sitios web de alto tráfico como Instagram, Mozilla, Pinterest y The Washington Times [Foundation](#) (2018).

4.2.3. SQLite



Figura 4.5: Logotipo de SQLite

SQLite es un motor de base de datos relacional¹¹, ligero, de archivos y de dominio público [\[SQLite \(2019\)\]](#). A diferencia de otros sistemas de gestión de bases de datos como PostgreSQL o MySQL, SQLite no es un servidor de base de datos separado; en su lugar, se integra directamente en la aplicación como una biblioteca. Esto significa que las bases de datos SQLite son archivos locales en el sistema de archivos, lo que facilita la configuración, distribución y gestión de la base de datos.

En el contexto de Virtual Q, SQLite ofrece varias ventajas relevantes al trabajar con Django [\[Javatpoint \(2019\)\]](#):

1. **1. Configuración sencilla:** Dado que SQLite es la base de datos predeterminada de Django y no requiere la instalación de software adicional, se puede configurar y utilizar rápidamente durante el desarrollo de su aplicación web. Esto le permite centrarse en la lógica y funcionalidad de la aplicación en lugar de preocuparse por la configuración y administración de la base de datos.
2. **2. Portabilidad:** Las bases de datos SQLite se almacenan en archivos locales, lo que facilita su copia, distribución y respaldo. Esto puede ser útil durante el desarrollo y pruebas, ya que puede compartir fácilmente la base de datos con

¹¹Es una base de datos organizada en tablas con datos que se pueden relacionar entre sí, basada en el modelo de datos relacional.

otros miembros del equipo o realizar copias de seguridad de los datos en diferentes puntos del desarrollo.

3. **Integración con Django:** SQLite funciona sin problemas con el ORM de Django, lo que permite utilizar todas las funciones y abstracciones proporcionadas por Django para trabajar con datos. Así, puede modelar fácilmente las entidades del sistema de colas virtuales (como usuarios, atracciones y tiempos de espera) como modelos de Django y aprovechar las capacidades del ORM para realizar consultas y manipulaciones de datos.

4.3. Otras tecnologías usadas

4.3.1. Git y GitHub



Figura 4.6: Logotipos de Git y GitHub

Git es un sistema de control de versiones distribuido y de código abierto, diseñado para gestionar y rastrear cambios en el código fuente de proyectos de software. Git permite a desarrolladores trabajar en ramas separadas del código, realizar cambios y fusionarlos en la rama principal cuando estén listos [Organization (2020)]. Esto facilita la colaboración entre múltiples desarrolladores y hace más sencillo el seguimiento de los cambios en el código a lo largo del tiempo.

GitHub, por otro lado, es un servicio web basado en Git que proporciona una plataforma para el alojamiento de repositorios de código, facilitando la colaboración y la administración de proyectos de software [Kinsta (2022)]. GitHub ofrece funciones adicionales como la gestión de incidencias, solicitudes de extracción, control de acceso y herramientas para la integración continua y la implementación.

Tanto Git como GitHub son relevantes en el desarrollo del proyecto por las siguientes razones [Kinsta (2022)]:

1. **Control de versiones:** Git permite mantener un historial detallado de los cambios en el código fuente, lo que facilita la identificación y corrección de errores, así como la comparación y restauración de versiones anteriores del proyecto si es necesario.

2. **Colaboración:** Con GitHub, puedo compartir mi proyecto con otros miembros del equipo, profesores y colaboradores, lo que facilita la colaboración y la revisión del código. Los colaboradores pueden trabajar en ramas separadas, proponer cambios y fusionarlos en la rama principal a través de solicitudes de extracción, garantizando que el código sea revisado antes de integrarse.
3. **Gestión de incidencias:** GitHub proporciona un sistema de seguimiento de incidencias que permite a los desarrolladores y colaboradores informar y asignar problemas, errores o mejoras en el proyecto. Esto ayuda a mantener una lista organizada de tareas pendientes y a garantizar que se aborden de manera eficiente.
4. **Integración continua y despliegue:** GitHub es compatible con diversas herramientas y servicios de integración continua y despliegue, lo que permite automatizar la construcción, prueba e implementación del proyecto. Esto asegura que el código sea probado y validado antes de su lanzamiento, reduciendo la probabilidad de errores en la aplicación.

4.3.2. Visual Studio Code



Figura 4.7: Logotipo de Visual Studio Code

Visual Studio Code (VS Code) es un editor de código fuente desarrollado por Microsoft [Microsoft (2021)]. Aunque técnicamente no es un IDE completo (Entorno de Desarrollo Integrado)¹², ofrece muchas características y funcionalidades que lo hacen altamente útil para el desarrollo de software.

- **Multiplataforma:** VS Code está disponible para Windows, macOS y Linux, lo cual permite a los desarrolladores utilizarlo en diferentes sistemas operativos.
- **Altamente personalizable:** Permite a los desarrolladores ajustar su apariencia, atajos de teclado, temas, extensiones y más para adaptarse a las preferencias y el flujo de trabajo de los usuarios.
- **Multi-lenguaje:** A la hora de desarrollar este proyecto, se ha tenido que usar Python, HTML, CSS, JavaScript, JSX y otros, los cuales están todos soportados extensamente por VS Code. Ofreciendo adicionalmente resaltado de sintaxis, sugerencias de código, autocompletado y otras características útiles para cada lenguaje.

¹²Es un software que proporciona servicios integrales a los programadores para el desarrollo de software, incluyendo edición de código, compilación, depuración y pruebas.

- **Extensiones y su ecosistema:** Probablemente, lo que más le distinga a VS Code es su ecosistema de extensiones muy amplio. Esto implica que los desarrolladores pueden agregar funcionalidades adicionales mediante la instalación de extensiones. Hay extensiones disponibles para tareas específicas, depuración y control de versiones con Git.
- **Depuración integrada** Proporciona una herramienta de depuración integrada que permite a los desarrolladores depurar sus aplicaciones directamente desde el editor. Lo cual facilita la identificación y corrección de errores durante el proceso de desarrollo.

A diferencia de algunos de los IDE más pesados, Visual Studio Code destaca por su rendimiento y ligereza. Es rápido en cuanto a tiempos de carga y respuesta, lo que permite a los desarrolladores trabajar de manera eficiente incluso en los proyectos más grandes.

Capítulo 5

Estado del arte

En el presente Trabajo de Final de Grado, se exploran tecnologías de vanguardia clave en el desarrollo de aplicaciones web y móviles. Existen una gran variedad de herramientas y conceptos a aplicar. Es por ello que se han tenido que escoger las más relevantes para el desarrollo del proyecto.

Entre ellas, destacan los microservicios, que permiten crear aplicaciones flexibles y escalables, descomponiéndolas en servicios independientes y cohesionados, y el Internet of Things (IoT), que conecta dispositivos y objetos físicos para la recopilación y análisis de datos en tiempo real.

Asimismo, se abordarán otras tecnologías relevantes como DevOps, arquitectura serverless ¹ y API, cuyas descripciones y aplicaciones se desarrollarán a lo largo del capítulo. Estas tecnologías innovadoras han demostrado ser eficaces en la creación de aplicaciones modernas y escalables, adaptándose a las necesidades del mercado y a las expectativas de los usuarios.

5.1. Microservicios

Los microservicios pueden ser analizados y explorados como una arquitectura de software innovadora que ha ganado popularidad en los últimos años [Services (2018a)]. Esta arquitectura se basa en la descomposición de una aplicación en pequeños servicios independientes y altamente cohesionados, cada uno enfocado en una funcionalidad específica.

Cada microservicio es responsable de una parte del negocio o de un dominio particular, y se desarrolla, despliega y escala de forma independiente. Esta independencia permite a los equipos de desarrollo trabajar en paralelo, acelerando el proceso de desarrollo y facilitando la implementación de cambios y actualizaciones en cada servicio sin afectar al resto del sistema.

¹Es un modelo de diseño y despliegue de aplicaciones en el que los desarrolladores no tienen que gestionar ni mantener servidores, ya que estos aspectos son manejados por proveedores de servicios en la nube.

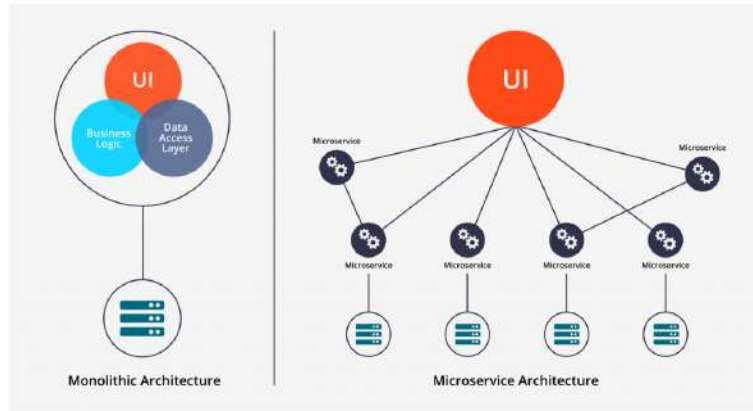


Figura 5.1: Diagrama Arquitectura monolítica vs. microservicios

Los microservicios se comunican entre sí a través de protocolos y formatos de intercambio de datos ligeros, como REST y JSON, lo que facilita la interoperabilidad entre diferentes tecnologías y plataformas. La arquitectura de microservicios también promueve la resiliencia, ya que un fallo en un servicio no debería afectar al funcionamiento de los demás.

Por el contrario, una arquitectura tradicional como la monolítica [Awati and Wigmore (2022)] tiene como enfoque de desarrollo una en la que todos los componentes de una aplicación se combinan en una única unidad, llamada monolito. En este enfoque, las diferentes funcionalidades de la aplicación, como la interfaz, lógica y gestión de datos, están estrechamente interrelacionadas y se ejecutan en un solo proceso. Aunque sea más simple de entender y desarrollar, puede presentar desafíos a medida que la aplicación crece en tamaño y complejidad.

La principal desventaja de las aplicaciones monolíticas es el volumen de código a la hora de escalar la aplicación. Se requiere más trabajo en el desarrollo, la realización de pruebas y la depuración de errores, lo que puede afectar a nuevas funcionalidades o las implementadas anteriormente.

Igualmente, un cambio en la funcionalidad de un servicio puede afectar negativamente a otro. El entorno es mucho más rígido, ya que solo se puede utilizar un único lenguaje de programación. Por lo tanto, a la hora de estabilización vertical, no es muy apropiado su uso.

5.1.1. Análisis de ventajas y desventajas de ambas arquitecturas

La arquitectura de microservicios y la arquitectura monolítica son dos enfoques distintos para el desarrollo de aplicaciones de software [Atlassian (2014)]. A continuación, se presenta una comparación extensiva de ambos enfoques.

Característica	Arquitectura	
	Monolítica	de Microservicios
Estructura	Única unidad	Servicios independientes
Desarrollo	Más simple y rápido	Más desafiante y lento
Comunicación	Llamadas internas	Protocolos ligeros
Despliegue	Única unidad	Independiente por servicio
Escalabilidad	A nivel de toda la aplicación	Independiente por servicio
Mantenimiento	Dificultad en mantenimiento y actualizaciones	Facilita el mantenimiento y actualizaciones
Resiliencia	Fallo en un componente puede afectar toda la aplicación	Fallo en un servicio no afecta a los demás
Independencia tecnológica	Dependencia tecnológica única	Diferentes tecnologías por servicio

Tabla 5.1: Diferencias entre arquitectura monolítica y arquitectura de microservicios

5.2. *Internet of Things* (IoT)



Figura 5.2: Internet of Things

IoT (Internet de las cosas) es una tecnología que permite la conexión y comunicación entre objetos físicos y dispositivos electrónicos a través de internet [Oracle (2017)]. Es decir, se trata de la interconexión de dispositivos cotidianos que se comunican entre sí, recopilan y transmiten datos para ser analizados y utilizados con diversos fines.

En el contexto de una aplicación que promueve colas virtuales en un parque de atracciones, IoT puede ser muy útil para recopilar información en tiempo real sobre la afluencia de personas en las atracciones y las zonas del parque. Se pueden utilizar dispositivos IoT para recopilar datos de sensores en las atracciones, como el número de personas que entran y salen, y la velocidad a la que se mueve la fila.

5.3. API, Metodología REST y CRUD

Las API (Interfaces de programación de aplicaciones) son conjuntos de herramientas y protocolos que permiten la interacción y comunicación entre diferentes aplicaciones y sistemas informáticos [Hat (2018)]. En otras palabras, permiten a diferentes aplicaciones comunicarse y compartir datos entre sí. Son ampliamente utilizadas en el desarrollo de aplicaciones modernas y en la integración de diferentes sistemas.



Figura 5.3: Explicación básica de API

La metodología REST (Representational State Transfer) es un estilo arquitectónico que se utiliza para diseñar sistemas de software que se comunican a través de la web. La metodología REST se basa en el protocolo HTTP² (Hypertext Transfer Protocol) y utiliza verbos HTTP como GET³, POST⁴, PUT⁵ y DELETE⁶ para realizar operaciones en los recursos del sistema. La metodología REST se enfoca en la creación de sistemas escalables y flexibles que pueden ser utilizados por diferentes aplicaciones y dispositivos.

CRUD (Create, Read, Update, Delete) es una metodología que se utiliza para describir las operaciones básicas de un sistema de gestión de bases de datos. CRUD describe las operaciones que se pueden realizar en los datos almacenados en una base de datos, como la creación de nuevos datos, la lectura de los datos existentes, la actualización de los datos existentes y la eliminación de los datos existentes. La metodología CRUD se utiliza comúnmente en la creación de aplicaciones web y móviles para realizar operaciones en los datos almacenados en la base de datos [LogicMonitor (2022)].

La metodología CRUD y las API están estrechamente relacionadas en el desarrollo de aplicaciones web y móviles. Las API permiten que diferentes aplicaciones y sistemas se comuniquen entre sí y compartan datos, y la metodología CRUD describe las operaciones básicas que se pueden realizar en los datos almacenados en una base de datos.

²Es un protocolo de comunicación que permite las transferencias de información en la web entre clientes y servidores.

³Es un método de solicitud utilizado en las APIs para obtener o recuperar datos de un recurso específico.

⁴Es un método de solicitud utilizado en las APIs para actualizar o reemplazar un recurso existente con los datos proporcionados

⁵Es un método de solicitud utilizado en las APIs para enviar datos y crear un nuevo recurso en el servidor.

⁶Es un método de solicitud utilizado en las APIs para eliminar un recurso específico del servidor.

5.4. Desarrollo Cross-Platform

El desarrollo cross-platform, también conocido como desarrollo multiplataforma, se refiere a la creación de aplicaciones que pueden funcionar en múltiples plataformas. Tanto en sistemas operativos móviles (iOS, Android) o de escritorio (Windows, macOS, Linux). En lugar de desarrollar aplicaciones nativas separadas para cada plataforma, el enfoque cross-platform permite compartir código y recursos entre diferentes plataformas, lo que puede ahorrar tiempo y esfuerzo para los desarrolladores [Kotlin (2017)].

El enfoque que se ha usado en este proyecto es el siguiente:

Frameworks multiplataforma: Estos frameworks permiten escribir el código de la aplicación una vez y luego compilarlo para que se ejecute en varias plataformas. Los ejemplos más notables siendo React Native de Meta y Flutter de Google. Estos frameworks suelen utilizar tecnologías web como HTML, CSS y JavaScript, y proporcionan acceso a características nativas a través de API.

Los beneficios del desarrollo cross-platform incluyen la capacidad de llegar a un público más amplio con una sola base de código, mayor eficiencia en términos de tiempo y recursos, y la posibilidad de mantener y actualizar una aplicación en múltiples plataformas de manera más sencilla.

Sin embargo, también hay consideraciones a tener en cuenta. Algunas características y funcionalidades nativas pueden ser más difíciles de acceder o pueden no estar disponibles en todas las plataformas. Además, el rendimiento y la apariencia pueden variar entre plataformas debido a las diferencias en la implementación y el diseño nativo [Marcak (2023)].

En última instancia, la elección de utilizar el desarrollo cross-platform o el desarrollo nativo dependerá de las necesidades específicas del proyecto, los recursos disponibles y las preferencias del equipo de desarrollo. En el caso de este proyecto, al no haber ninguna tarea que haga alto uso de grandes recursos, no se ha optado por darle gran valor a la eficiencia interna de la aplicación, y el trade-off que sea multiplataforma es mayor.

5.5. Cultura DevOps

DevOps es una metodología de desarrollo de software que busca unir y mejorar la colaboración entre los equipos de desarrollo (Dev) y operaciones (Ops). Esta práctica surge como respuesta a la necesidad de aumentar la eficiencia, la calidad y la agilidad en el proceso de entrega de software, permitiendo una integración y entrega continuas (CI/CD) y una mejora en la comunicación entre los equipos involucrados [Ashtari (2022)].

Esta metodología se centra en la automatización y monitorización de todo el ciclo de vida del desarrollo de software, desde la integración, las pruebas, el despliegue hasta la gestión de infraestructura y la supervisión del rendimiento. Al adoptar un enfoque DevOps, se busca reducir el tiempo de lanzamiento de nuevas características y soluciones de errores, así como mejorar la estabilidad y la seguridad del software en producción.

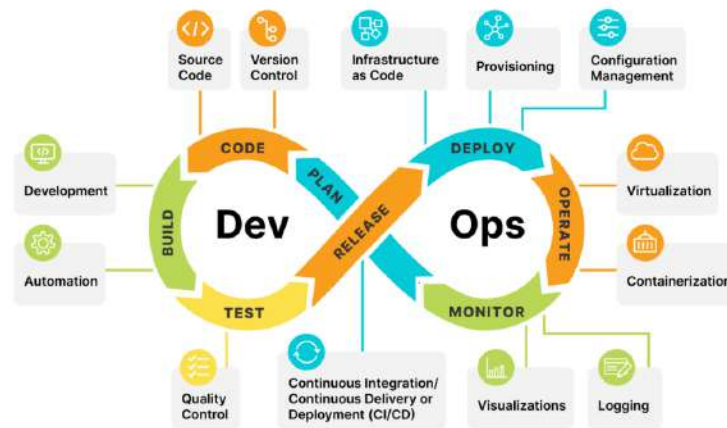


Figura 5.4: Figura que resume DevOps y sus estructuras

En el contexto del Trabajo de Final de Grado, la adopción de la metodología DevOps puede contribuir significativamente al éxito del proyecto al facilitar la entrega rápida y confiable de las actualizaciones y mejoras del sistema. Además, al implementar prácticas DevOps, se promueve una mayor colaboración entre los equipos, se optimizan los recursos y se reduce el riesgo asociado con la implementación de cambios en el entorno de producción.

5.6. Teoría de Colas

5.6.1. Introducción

La teoría de colas se ocupa de los problemas relacionados con la formación de colas (o tiempo de espera). Antes de adentrarnos en la teoría de colas, es importante aclarar algunos conceptos, tanto el del servicio como el del elemento. Un elemento puede ser una persona, una máquina u otro objeto que necesita recibir un servicio en un punto determinado [Manoj (2020)] [Ramamurthy (2007)]. El servicio, por su parte, engloba cualquier tipo de atención al consumidor para satisfacer sus necesidades.

Por ejemplo:

- Una persona acudiendo al hospital para obtener consejo médico del doctor es un elemento.
- Una persona que va a la estación de tren a comprar un billete para su viaje es un elemento.
- Una persona que va al supermercado a comprar productos consumibles es un elemento.

5.6.2. Notación y Terminología

Para entender los conceptos a continuación se debe de tener en cuenta la siguiente terminología:

- $n \rightarrow$ Número de elementos en el sistema.
- $p_n(t) \rightarrow$ La probabilidad de que haya n elementos en una cola en el instante de tiempo t .
- $P_n \rightarrow$ La probabilidad de que haya n elementos en el sistema.
- $\lambda_n \rightarrow$ Número medio de elementos llegando por unidad de tiempo, cuando ya haya n elementos en el sistema.
- $\mu_n \rightarrow$ Número medio de elementos siendo servidos por unidad de tiempo, cuando ya hay n unidades en el sistema.
- $\mu \rightarrow$ Tasa de servicio o la tasa de procesamiento del servidor en un sistema de colas. La cantidad media de clientes que pueden ser atendidos por unidad de tiempo en el servidor.
- $s \rightarrow$ Número de canales de servicios paralelos en el sistema.
- $\frac{1}{\lambda} \rightarrow$ La tasa de llegada de los clientes al sistema de colas. La cantidad media de clientes que llegan al sistema por unidad de tiempo.
- $\frac{1}{\mu} \rightarrow$ El tiempo promedio que tarda un cliente en completar el servicio.
- $\rho \rightarrow$ Intensidad de tráfico. La fracción de tiempo que se espera que estén ocupados los servidores.
- $N \rightarrow$ Número máximo de elementos permitidos en un sistema.
- $L_s \rightarrow$ Número medio de elementos en el sistema.
- $L_q \rightarrow$ Número medio de elementos en la cola.
- $W_s \rightarrow$ Tiempo medio de espera en el sistema.
- $W_q \rightarrow$ Tiempo medio de espera en la cola.
- $P_w \rightarrow$ Probabilidad de que un consumidor tenga que esperar por un servicio.

5.6.3. Modelos y clasificación

Los modelos de cola más simples asumen que las llegadas/entradas y salidas/resultados sigan un proceso de nacimiento y muerte de tareas. Cualquier modelo de colas se caracteriza por situaciones en las que llegas y salidas ocurren de manera simultánea. Dependiendo de la naturaleza de las entradas y las facultades de servicios, puede haber una variedad de modelos de colas:

- **Modelo de cola probabilístico:** La tasa de llegada y servicio son variables aleatorias desconocidas.
- **Modelo de cola determinístico:** Ambas la tasa de llegada y de servicio se conocen y son fijas.
- **Modelo de cola mixta:** Una de las tasas tanto la de llegada o la de servicio será una variable aleatoria y desconocida, mientras que la otra se conocerá y será fija.

Las cuales se representarán mediante el siguiente sistema: Secuencia de llegada/Secuencia de Servicio/Número de canales/(Capacidad/Orden de Servicio)

$$A/B/S/(d/f)$$

M se usa para indicar una distribución Poisson (Markoviana) de llegadas y salidas. D se utiliza para indicar una constante o distribución determinística. E_k se emplea para representar una distribución de probabilidad erlangiana. G se usa para mostrar una distribución de probabilidad general.

Los modelos de colas se emplean generalmente para describir el comportamiento de un sistema de colas, permitiendo cuantificar el impacto de las variables de decisión en los tiempos de espera pronosticados, así como en la generación de costos de espera y de la información de costo del servicio. Estos sistemas se pueden evaluar por los aspectos y el sistema, con el objetivo de obtener el que traiga un coste mínimo.

Para generar una solución

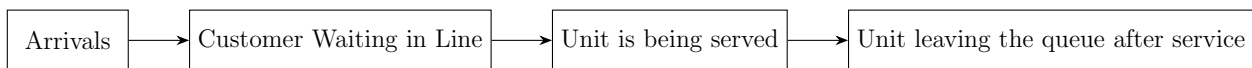
1. Indicar el sistema de colas alternativo.
2. Evaluar el sistema en distintas condiciones, como tiempo, longitud y costes.
3. Seleccionar el mejor sistema.

5.6.4. Sistema de colas o procesos

Un sistema de colas se puede describir completamente por las siguientes características:

- Entradas (Secuencias de llegada).
- El mecanismo o el servicio.
- La disciplina de la cola.
- El comportamiento del elemento.

Los componentes del sistema de colas son llegadas, el elemento que espera en la cola, la unidad siendo servida, las facilidades del servicio y la unidad que sale de la cola tras haber sido servida. [Ver figura 2]



5.6.5. Proceso de Llegada

La llegada se refiere a la forma en que los consumidores llegan y se unen al sistema. En general, las llegadas son aleatorias y no pueden predecirse, ya que cada elemento es independiente y la organización encargada del servicio no tiene control sobre el cliente. Estas características de las llegadas se pueden observar en diversos escenarios, como gasolineras o taquillas de entradas, donde individuos o grupos llegan de manera aleatoria. Para describir el proceso de llegadas, se consideran las siguientes características: la naturaleza de las llegadas, la capacidad del sistema y el comportamiento de los elementos. Estas características se muestran en la figura 3.

1. **Tamaño de las llegadas:** El tamaño de las llegadas al sistema de servicio dependen principalmente en la naturaleza del tamaño de la población de referencia, la cual puede ser finita o infinita. La secuencia de llegada se puede describir claramente en términos de probabilidades y consecuentemente mediante una distribución que representa el intervalo de tiempo entre dos llegadas consecutivas.
2. **Tiempo entre llegada:** Este periodo entre la llegada de elementos individuales puede ser constante o puede estar distribuida de alguna manera. La mayoría de los modelos de cola suponen que se aplica alguna distracción de tiempo entre llegadas para todos los clientes durante el período de estudio. Ciertamente, en la mayoría de las situaciones el tiempo de servicio es una variable aleatoria con la misma distribución para todas las llegas, pero hay casos en los cuales se distinguen dos o más tipos de elementos.
3. **Capacidad del sistema de servicio:** En el contexto de las colas, la capacidad se refiere al espacio disponible para que las llegadas esperen antes de ser atendidas. El espacio disponible puede ser limitado o ilimitado. Cuando el espacio es limitado, la longitud de la cola supera un cierto límite; no se permite la entrada de más unidades o llegadas al sistema hasta que se desocupe algún espacio. Este tipo de sistema se conoce como sistema con capacidad finita y tiene su efecto en el patrón de llegada del sistema. Por ejemplo: Un ejemplo de una cola con capacidad finita podría ser una sala de espera en un consultorio médico. La sala de espera tiene un número limitado de asientos disponibles para los pacientes que esperan ser atendidos. Una vez que todos los asientos están ocupados, no se permite la entrada de más pacientes hasta que se desocupe algún asiento.
4. **Comportamiento del elemento:** La longitud de la cola, el tiempo de espera de un elemento, el tiempo de espera de las facilidades de un servicio dependen del comportamiento del cliente. Aquí el comportamiento se refiere al nivel de paciencia del elemento durante su espera en la cola. La cual se puede clasificar de la siguiente manera:
 - **Abandono:** En este caso el elemento se une a la cola, y tras esperar un tiempo pierde su paciencia y se va de la cola. Este tipo de comportamiento también puede suponer la pérdida de un cliente para la organización. Que siempre haya una cola larga indica un servicio insuficiente.
 - **Rechazo:** Este elemento directamente no se une a la cola tras observar la gran longitud. Igualmente, puede suponer una pérdida de cliente para la organización.
 - **Colusión:** En este caso, varios elementos colaboran entre sí tal que uno de ellos se quede en la cola. Un elemento representa un grupo de elementos. En este caso la longitud de la cola será pequeña, pero el tiempo de servicio para cada individuo será mayor. El cual puede agotar la paciencia de otros elementos en la cola y llevar a una situación peor.
 - **Maniobra:** En caso de haber un número plural de colas para un servicio. Como en las taquillas de cine, gasolineras, etc. Un elemento en una de las colas tras inspeccionar la longitud de una cola alternativa más corta, con la

esperanza de ser atendido con más rapidez. Pero cabe la posibilidad de que se puede encontrar con una cola con elementos de colusión. En todo caso, la probabilidad de ser atendido será muy variable.

5.6.6. Mecanismo o Facultad de Servicio

El tiempo que requiere servir a un elemento no se puede estimar, hasta que se sepan sus demandas. Muchas veces se suele determinar por una variable estadística y no se puede determinar de ninguna manera hasta que se haya completado el servicio del elemento. El diseño de la facultad de un servicio suele ser demostrado por la siguiente figura:

- a) **Diseño de la cola:** Los clientes que llegan puede que se les pida esperar en una cola individual o múltiple dependiendo en el tipo de servicio. Cuando esperan en una cola individual se conoce como un diseño de canal individual. Cuando esperan en una cola múltiple se denomina diseño multicanal.
 - a) **Canal Individual:** Si la organización solo entrega una línea, solo se puede servir una unidad en un instante de tiempo. Es por ello, que se generara una cola. El siguiente elemento será servido únicamente en el instante en el que el servicio del elemento anterior se haya completado. En este tipo de canal se puede subdividir en colas de fase singular o múltiple.
 - b) **Multicanal:** Cuando la tasa de entrada y la demanda de servicios incrementa, la organización añadirá servicios adicionales para reducir el flujo de clientes o su tiempo de espera. De esa manera, colas de longitud distinta se formarán. Si el servicio se le otorga en un centro único, se conoce como un multicanal de fase individual.
- b) **Disciplina de Cola:** Cuando los elementos esperan en cola, se les llama de un orden específico dependiendo de la naturaleza o el tipo de cola [Giri \(2020\)](#). Los tipos principales son:
 - a) **First In First Out (FIFO):** Un principio de organización de una estructura de datos donde la entrada más antigua (primera) o “cabeza” de la cola se procesa primero. Esto significa que la solicitud (como un cliente en una tienda o un trabajo de impresión enviado a una impresora) se procesa en el orden en que llega.
 - b) **Last In First Out (LIFO):** Es un método para organizar la manipulación de una estructura de datos (a menudo, específicamente un búfer de datos) donde la entrada más reciente (última) o “cima” de la pila se procesa primero.
 - c) **Service in Random Order (SIRO):** Una estructura en la que el cliente es elegido para el servicio de manera aleatoria, lo que significa que todos los clientes tienen la misma probabilidad de ser seleccionados. El tiempo de llegada del cliente no tiene consecuencias en la selección del cliente.
 - d) **Servicio por nivel de prioridad:** Un método de organización de colas en el que los clientes son atendidos en función de su nivel de prioridad. Esto significa que los clientes con un nivel de prioridad más alto son atendidos antes que los clientes con un nivel más bajo.

5.6.7. Definición de estados transitorios y estacionarios

La distribución de la llegada de un elemento y su tiempo de servicio son dos componentes que forman el estudio de esperar en cola. Bajo condiciones fijas de llegadas de elementos y diseños de cola, su longitud se convierte en una función de tiempo. Es por ello por lo que un sistema de colas se puede considerar un experimento aleatorio, y los eventos varios pueden ocurrir cambio en el sistema en cualquier instante de tiempo. Se identifican tres estados de llegada a un sistema, los cuales se denominan estado de transición y estado estacionario [Giri (2020)].

Estado transitorio: Un sistema se denomina ‘en estado de transición’ cuando sus características operativas o comportamiento dependen del tiempo. Suele ocurrir en las fases iniciales de la operación del sistema, donde su comportamiento sigue dependiendo de las condiciones iniciales. Por lo tanto, cuando la distribución de llegadas, tiempo de espera y tiempos de servicio dependen del tiempo, un sistema se dice que está ‘en estado de transición’.

Estado estacionario: Un sistema se establece como estacionario cuando la tasa de llegadas de elementos es menor que la tasa de servicio, y cuando ambas son constantes. El sistema no solo se vuelve estacionario, sino que también será independiente del estado inicial de la cola. La probabilidad de encontrar la longitud de la cola a cualquier hora será la misma. Aunque esta longitud fluctúe, su comportamiento estadístico quedará estable. Por lo tanto, se podrá confirmar que un estado estacionario ocurre cuando el comportamiento de la cola es independiente del tiempo.

Una condición necesaria para que se alcance el estado estacionario, sea que el tiempo desde el comienzo de la operación sea lo suficientemente grande. Es decir: $t \rightarrow \infty$. Esta condición es necesaria, pero no suficiente. La existencia de un estado estacionario también depende del comportamiento del sistema. Si la tasa de llegada es mayor que la de servicio, un estado estacionario no se puede alcanzar.

$$\lim_{t \rightarrow \infty} P_n(t) \rightarrow P_n$$

Por lo tanto, en un sistema estacionario, la distribución de probabilidad de llegadas, tiempo de espera, y tiempo de servicio no es dependiente en el tiempo.

5.6.8. Notación de Kendall y la clasificación de los modelos.

Los diferentes modelos en la teoría de colas son clasificados usando una notación definida por D.G Kendall en 1953 [Kendall (1953)] con el formato $(a/b/c)$. Tras esto, A.M. Lee in 1966 decidió añadir los símbolos d y e a la notación de Kendall. Tal que el formato de una cola estándar quedaría expresado como:

$$(a/b/c) : (d/e)$$

Donde:

- $a \rightarrow$ Distribución de llegadas.
- $b \rightarrow$ Distribución de tiempo de servicio (o salidas).

- $c \rightarrow$ Número de canales de servicio.
- $d \rightarrow$ Número máximo de elementos permitidos en el sistema.
- $e \rightarrow$ Tipo de cola

Hay algunas notaciones que son usadas para la distribución de llegada y salida:

- $M \rightarrow$ Tiempos de llegada o de servicios exponenciales (Markovianos/Poisson)
- $D \rightarrow$ Intervalos de llegadas/salidas constantes o determinísticos
- $G \rightarrow$ Distribución del tiempo de servicio general (No se supone ninguna característica del tipo de distribución)
- $GI \rightarrow$ Tiempo entre llegadas teniendo una distribución de probabilidad general como normal, uniforme u otra empírica.
- $E_k \rightarrow$ Distribución Erlang- k para los tiempos entre llegada y salida con un parámetro k (Es decir, si $k = 1$, Erlang es equivalente a una exponencial, y si $k = 0$, Erlang será determinístico)

Por ejemplo, en un sistema de colas en el cual el número de llegadas se describa por una distribución de probabilidad Poisson, el tiempo de servicio se describe mediante una distribución exponencial, mientras haya un solo servidor. Es decir: M/M/1.

5.6.9. Distribuciones en sistemas de colas

Los modelos básicos de colas han sido desarrollados bajo la suposición que la tasa de llegada sigue una distribución Poisson y que el tiempo de servicio sigue una distribución exponencial negativa. Esta situación se denomina llegada de Poisson y tiempo de espera exponencial. Estas suposiciones suelen ser validas en situaciones operativas. Al menos que se indique lo contrario, siempre se debe de suponer este modelo.

La distribución Poisson

Este modelo se justifica al mencionar que una distribución de Poisson corresponde a intervalos completamente aleatorios e independientes. El símbolo usado que representa la tasa de llegada media es la letra griega Lambda (λ), la cual representa las llegadas por unidad de tiempo [Asthana (2015)]. Se puede demostrar que cuando la tasa de llegada sigue una distribución Poisson λ , el tiempo entre llegadas sigue una distribución negativa $\frac{1}{\lambda}$. Esta relación entre tiempo medio de llegada y tiempo medio entre llegadas no ocurre en otras distribuciones.

Modelos útiles de colas vistas en este proyecto

Una vez que ya se ha acabado de definir toda la terminología necesaria para comprender los parámetros involucrados en la teoría de colas, se puede permitir pasar a analizar los tipos de colas que se van a investigar en nuestro sistema.

Modelo I – (M/M/1: ∞ / FCFS)

La derivación de este modelo viene bajo las siguientes suposiciones del sistema de colas:

1. La distribución exponencial de tiempos entre llegadas.

$$\int_0^\infty \frac{\lambda}{\mu} (\mu - \lambda) e^{-2(\mu - \lambda)} d\lambda$$

2. Cola individual sin restricción en la longitud de la cola, con elementos de comportamiento estándar.
3. Tipo de Cola FIFO (FCFS). Primero en llegar, primero en ser servido.
4. Único punto de servicio con una distribución exponencial en el tiempo a ser servido.

Métricas de rendimiento

$$p_0 = 1 - \frac{\lambda}{\mu} = 1 - \rho; \quad \rho = \frac{\lambda}{\mu}$$

$$p_n = \left(\frac{\lambda}{\mu}\right)^n (1 - \frac{\lambda}{\mu}) = \rho^n (1 - \rho); \quad \rho < 1, \quad n = 0, 1, 2, \dots$$

P_w = Probabilidad de que el servidor esté ocupado (*i.e. cada elemento tiene que esperar* $= 1 - \rho_{(0 = \frac{\lambda}{\mu})}$)

1. Número esperado de elementos en el sistema (elementos en la cola y los elementos siendo servidos)

$$\begin{aligned} L_s &= \sum_{n=1}^{\infty} n P_n = \sum_{n=1}^{\infty} n (1 - \rho) \rho^n, \quad 0 < \rho < 1 \\ &= \frac{\rho}{1 - \rho} = \frac{\lambda}{\mu - \lambda}; \quad \rho = \frac{\lambda}{\mu} \end{aligned}$$

2. La longitud esperada de la cola o el número de elementos esperando en la cola.

$$L_q = \sum_{n=1}^{\infty} (n - 1) P_n = \sum_{n=1}^{\infty} n P_n - \sum_{n=1}^{\infty} P_n = L_s - (1 - P_0) = L_s - \frac{\lambda}{\mu} = \frac{\lambda^2}{\mu(\mu - \lambda)}$$

3. Tiempo de espera medio de un cliente en la cola.
4. Tiempo de espera medio de un cliente en la cola.

$$W_q = \frac{\lambda(1 - \frac{\lambda}{\mu})}{(\mu - \lambda)^2} = \frac{\lambda}{\mu(\mu - \lambda)} = \frac{L_q}{\lambda}$$

5. Tiempo medio de un elemento en el sistema (espera y servicio)

$$W_s = W_q + \frac{1}{\mu} = \frac{\lambda}{\mu(\mu - \lambda)} + \frac{1}{\mu} = \frac{1}{\mu - \lambda} = \frac{L_s}{\lambda}$$

6. Tiempo de espera medio en un sistema ocupado

$$W_b = \frac{\text{Tiempo de espera de un elemento en la cola}}{\text{Probabilidad de que un sistema esté ocupado}} = \frac{1}{\mu - \lambda}$$

7. Probabilidad de k o más elementos en un sistema (Poisson)

$$P(n \geq k) = \left(\frac{\lambda}{\mu}\right)^k; \quad P(n > k) = \left(\frac{\lambda}{\mu}\right)^{k+1}$$

8. La variación de la longitud de la cola

$$\frac{p}{(1-p)^2} = \frac{\lambda\mu}{(\mu - \lambda)^2}$$

9. Probabilidad de que la espera sea más de $P(x > t) = \begin{cases} (\mu - \lambda)e^{-(\mu - \lambda)t} & \text{(Sistema)} \\ \frac{\lambda}{\mu}(\mu - \lambda)e^{-(\mu - \lambda)t} & \text{(Cola)} \end{cases}$

Relación entre L_q , L_s , W_q , y W_s

Se puede demostrar bajo las condiciones generales de llegada, salida y el tipo de cola que las fórmulas:

$$L_s = \lambda W_s$$

$$L_q = \lambda W_q$$

Estas fórmulas actúan como puntos clave en establecer las relaciones fuertes entre W_s , W_q , L_s , L_q , las cuales se definen como:

$$W_q = W_s - \frac{1}{\mu} \Rightarrow (\text{Multiplicando ambos lados por } \lambda) \Rightarrow L_q = L_s - \frac{\lambda}{\mu}$$

5.6.10. Modelo Markoviano M/M/1 – Cola actual en parques de atracciones

Una cola M/M/1 representa la longitud de la cola en un sistema con un único servidor, en el que las llegadas están determinadas por un proceso de Poisson y los tiempos de servicio de los trabajos tienen una distribución exponencial. El nombre del modelo se escribe en notación de Kendall [Asmussen (2003)]. El modelo es el más elemental de los modelos de colas y un objeto de estudio atractivo, ya que pueden obtenerse expresiones de forma cerrada para muchas métricas de interés en este ámbito. para

muchas métricas de interés en este modelo. Una extensión de este modelo con más de un servidor es el modelo M/M/c. Una cola M/M/1 es un proceso estocástico cuyo estado está en el set $0, 1, 2, 3, \dots$ en el cual el valor corresponde al número de elementos en el sistema, incluyendo cualquiera que ya esté siendo servido.

1. Las llegadas ocurren a una tasa λ en un proceso de Poisson para mover el proceso del estado i al $i+1$.
2. Los tiempos de servicio tienen una distribución exponencial con el parámetro de tasa λ en la cola M/M/1, donde $1/\mu$ es el tiempo medio de servicio.
3. Un servidor único les sirve a los elementos uno a la vez desde el frente de una cola, basándose en un tipo FIFO. Cuando el servicio está completo, el elemento deja la cola y el número de elementos decrece por uno.
4. El buffer es de tamaño infinito, por lo tanto, no hay límite en el número de elementos que pueda contener.

El modelo se puede describir como una cadena continua de tiempo Markov con una matriz de tasa de transición:

$$Q = \begin{bmatrix} -\lambda & \lambda & 0 & 0 & \dots \\ \mu & -(\mu + \lambda) & \lambda & 0 & \dots \\ 0 & \mu & -(\mu + \lambda) & \lambda & \dots \\ 0 & 0 & \mu & -(\mu + \lambda) & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

Con el espacio de estados $0, 1, 2, 3, \dots$. Es la misma cadena de tiempo Markov que con el proceso de nacimiento y muerte. El diagrama state-space para la siguiente cadena, se muestra a continuación:

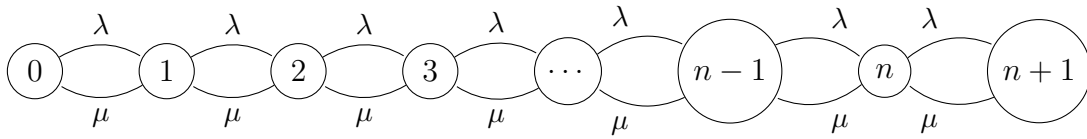


Figura 5.5: Diagrama State Space de una cola M/M/1. El cual anota el número de elementos en el sistema de colas. El valor de μ y λ representan la llegada y servicio de los elementos.

5.6.11. Modelo de cola multicanal: M/M/c: (α /FCFS)

Los símbolos de arriba indican un sistema con entradas y salidas distribuidas mediante un proceso Poisson con un número de canales $= c$, donde $c \geq 1$. La capacidad de la línea es finita y de tipo FIFO o FCFS. Aquí la longitud de la cola depende en el número de canales que están usando. En el caso de que el número de elementos sea menor que el número de canales (es decir: $n \leq c$) no habrá problema de espera y la tasa de servicio serán μ , ya que solo habrán n canales ocupados, cada uno sirviendo con tasa μ . En caso de que $n > c$, todos los canales estarán ocupados, y en el caso donde $n \geq c$, entonces $n-c$

elementos estarán esperando en cola con tasa de servicio $c\mu$, ya que todos los canales c estarán ocupados.

$$p_o = \frac{1}{\sum_{n=0}^{c-1} \frac{(\lambda/\mu)^n}{n!} + \frac{(\lambda/\mu)^c}{c!} \cdot \frac{c\mu}{c\mu - \lambda}}$$

$$\text{OR} = \frac{1}{\sum_{n=0}^{c-1} \frac{(c\rho)^n}{n!} + \frac{(c\rho)^c}{c!(1-\rho)}}$$

$$p_n = \begin{cases} \frac{(\lambda/\mu)^n}{n!} p_o, & 1 \leq n \leq c; \\ \frac{1}{(c^n - c c!)} (\lambda/\mu)^n p_o, & n \geq c \end{cases}$$

Número medio de elementos esperando en la cola del sistema:

$$E(n) = \frac{\rho p_c}{(1 - \rho)^2}$$

Número medio de la cola:

$$E(L) = \frac{p_c \rho}{(1 - \rho)^2} + c\mu$$

Longitud media de la cola = Número medio de elementos esperando + número medio de elementos siendo servidos

$$E(w) = \frac{\text{Número medio de unidades en espera}}{\lambda} = \frac{p_c \rho}{\lambda(1 - \rho)^2} = \frac{E(L)}{\lambda} = \frac{\mu(\lambda/\mu)^c}{(c - 1)!(c\mu - \lambda)^2} p_o$$

Tiempo de espera medio =

$$E(v) = \frac{E(n)}{\lambda}$$

$$\text{Probabilidad que todos los canales estén ocupados} = p(n \geq c) = \frac{1}{(1 - \rho)} p_c$$

Esta breve introducción al mundo de teoría de colas tiene como propósito que, en caso de que el lector analice el código presentado adjunto en el proyecto, pueda analizar el archivo de QueueingSim.py, en el cual se demuestra de manera teórica, porque un sistema de colas como el de Virtual Q ahorraría una cantidad significativamente grande de tiempo, y lo entienda con facilidad.

Capítulo 6

Definición del Trabajo

El propósito de este capítulo es describir y justificar el desarrollo de este trabajo de fin de grado. Se presentarán a continuación los objetivos que se desean alcanzar, incluida la metodología usada para su desarrollo, así como la estimación económica y la planificación del mismo.

6.1. Motivación

Los parques de atracciones se conocen como los sitios que más ilusión y emoción le puede traer a un público universal. Con un sentido increíble de comunidad y felicidad que es muy difícil replicar a cualquier otra escala.

En cambio, los parques de atracciones de hoy en día no evolucionan, solo estrenan nuevas experiencias o atracciones para las cuales siempre hay que hacer cola; Cambia la experiencia de la atracción, pero no la de la visita al parque. Hacer cola durante la gran mayoría de la visita es tiempo muerto, el cual se podría invertir explorando distintas áreas del parque, comiendo, comprando o divirtiéndose, lo cual son las cosas que hacen una visita memorable.

El hecho que en días de alta demanda se tenga que esperar múltiples horas para atracciones de corta duracion son motivo suficiente para dañar la experiencia de visitar el parque y no querer volver.

Desmotiva futuras visitas y desilusiona al publico que haya decidido comprar una entrada, invirtiendo su tiempo y dinero con el objetivo de llevarse un bonito recuerdo y pasar su día de manera agradable y divertida.

El propósito de este proyecto es doble. Se busca tanto revolucionar la experiencia de tiempo muerto en los parques de atracciones y desarrollar una plataforma plantilla que administre la aplicación móvil de los usuarios el día de su visita para los gestores del parque de manera simple a través de una página web.

La aplicación cliente proporcionará todas las funcionalidades e información que pueda requerir un usuario en su visita. El objetivo es que nuestros clientes visiten todo lo posible en el parque de atracciones, no solo al proporcionar información de antemano,

pero minimizando el tiempo de espera; aprovechando todos los puntos de interés que se ofrecen.

6.2. Objetivos del proyecto

El fin con el que se va a desarrollar el presente trabajo es diseñar una aplicación dual (web y móvil), la cual permita mejorar el manejo de un parque de atracciones. Introduciendo nuevas características como las colas virtuales, aparte de centralizar toda la experiencia del parque en una única aplicación personalizable. Los objetivos principales del proyecto son:

- **Creación de un Sistema de Colas Virtuales:** Se creará un sistema de gestión para asignar a usuarios del sistema a horarios particulares en las atracciones. En ella se tendrá que equilibrar capacidad, horario y miembros ya presentes de manera eficiente y dinámica. Este sistema no solo funcionará por sí mismo, pero también se usará para la recomendación de itinerarios.
- **Desarrollo de una aplicación web para la gestión de parques de atracciones:** Aplicación web a la que solo tendrán acceso los administradores del parque. En ella se podrán añadir regiones de parques, atracciones individuales, junto con todas sus características (Título, Accesibilidad, Descripción, Ubicación, Horario para las colas virtuales, etc.) en la cual modificará los contenidos de una REST API cuya información se le mostrará al público en tiempo real en la aplicación móvil. Igualmente, esta aplicación obtendrá analíticas y datos en tiempo real por partes de los usuarios para poder hacer análisis útiles y usarlos para mejorar la experiencia del parque (tal como mapas de flujo).
- **Desarrollo de una aplicación móvil para el usuario:** Aplicación móvil que aprovecharan los usuarios los días antes y el día de su visita al parque de atracciones. En ella se podrá crear un perfil y proceder con las siguientes funcionalidades: Gestión de reserva del parque, creación y recomendación de itinerarios basado en grupos, cola virtual para atracciones, pedidos online para comidas o tiendas del parque, mapa virtual del parque con puntos de interés pop-up.

Toda la información vendrá de la REST API, la cual se obtiene por parte del gestor de atracciones en tiempo real.

6.3. Metodología

El desarrollo del proyecto seguirá la metodología AGILE [BBVA (2018)], la cual tiene un enfoque para el desarrollo de software, la entrega incremental y colaborativa de proyectos, en contraste con los métodos de desarrollo tradicionales y en cascada. Agile se basa en un enfoque iterativo e incremental, donde se trabaja en colaboración y se adapta a medida que avanza en el proyecto.

Las tareas a realizar se definen en distintos bloques, denominados *sprints*, que son periodos cortos con objetivos a cumplir claramente definidos.

Trabajar con esta metodología es obtener resultados concretos en cada iteración, permitiendo una mayor flexibilidad y adaptación a medida que se avanza en el proyecto.

La razón por la que se ha elegido la metodología AGILE es debido a la amplia gama flexible de ventajas que ofrece en comparación con otros enfoques.

6.4. Planificación y estimación económica

	<i>Octubre</i>	<i>Noviembre</i>	<i>Diciembre</i>	<i>Enero</i>	<i>Febrero</i>	<i>Marzo</i>	<i>Abril</i>	<i>Mayo</i>
Familiarización con el Framework Django								
Busqueda y investigacion de teoria de colas y los sistemas integrados actualmente								
Desarrollo del Back-End del proyecto usando Django								
Customizacion de la interfaz de la vista del administrador								
Familiarización con React Native								
Testing y validacion de los metodos back-end								
Integracion del sistema IoT para los empleados del parque								
Desarrollo de aplicación cliente - React Native								
Implementar la aplicacion en la nube usando Docker y Kubernetes								
Documentacion del Proyecto								

Tabla 6.1: Tabla de Gantt mostrando el desarrollo del TFG

El desarrollo de las tareas que se exponen se han descrito y detallado en el Capítulo 5.

Las tareas se dividieron en prioridad usando herramientas como Notion para su seguimiento.

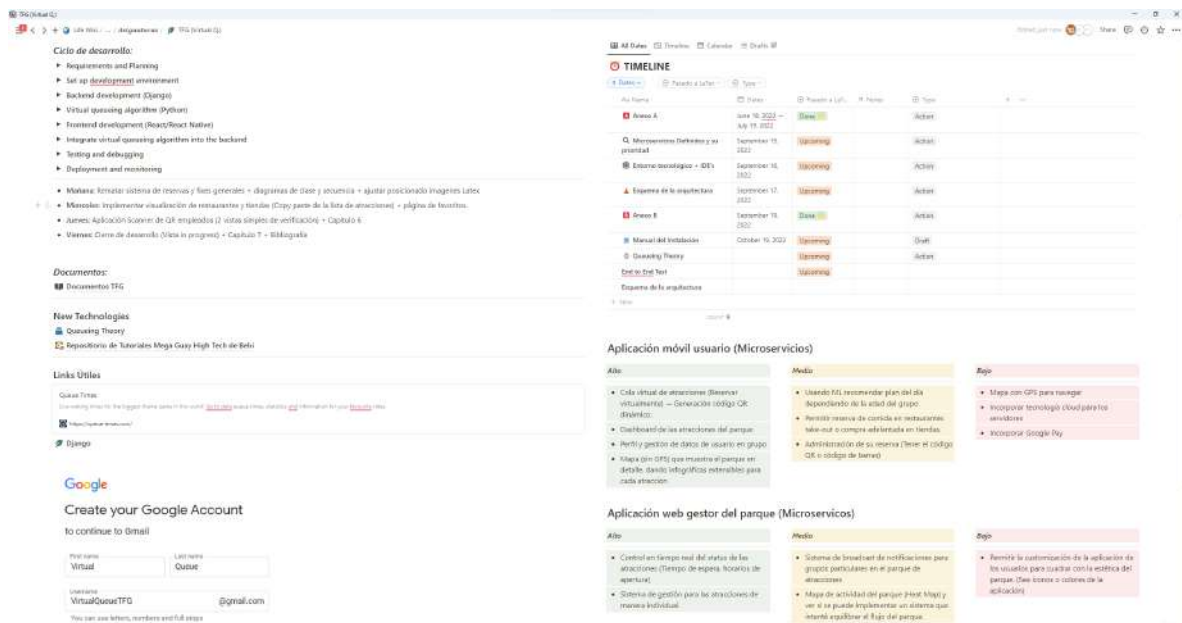


Figura 6.1: Caption

De esta forma, se pudieron dividir las tareas en tres grandes grupos: tareas por hacer, tareas en proceso y en tareas terminadas, permitiendo tener presente en todo momento el progreso del proyecto.

Estimación Económica

Con respecto a la estimación económica se han incluido estimaciones de salarios de ingenieros de software y el equipo necesario para desarrollarlo.

En cuanto a los recursos materiales que han sido utilizados en el desarrollo de este trabajo, ha sido un ordenador portátil y un teléfono móvil.

El desglose de precios de los siguientes se puede apreciar a continuación **Computers (2013)]** **Xiaomi (2021)]**:

<i>Ordenador Portátil Lenovo Ideapad 710S Plus</i>	
Procesador	Intel Core i6 Processor
Sistema Operativo	Windows 10
Tarjeta Grafica	NVIDIA GeForce 940MX
RAM	8 GB DDR4
Storage	512 GB PCIe SSD
Precio	899,99 €

Tabla 6.2: Especificaciones del ordenador portatil

<i>Xiaomi Redmi Note 11 - Dispositivo Movil</i>	
Procesador	Snapdragon 680 @ 2.4GHz
Sistema Operativo	MIUI 13 basado en Android 12
Pila	5000 mAh (typ)
RAM	4 GB
Storage	216 GB SSD
<i>Precio</i>	199,99 €

Tabla 6.3: Especificaciones del dispositivo movil

En términos de personal, para llevar al cabo el desarrollo de este proyecto, solo se ha necesitado un desarrollador de software.

El salario medio de un desarrollador de software en España cobra de media uno 2.121 € al mes [es.talent.com (2023)]. Considerando que el proyecto ha llevado en torno a los 8 meses, el coste estimado es el siguiente:

<i>Concepto</i>	<i>Coste</i>	<i>Tiempo trabajado</i>	<i>Coste Total</i>
Desarrollador de full-stack	2.958 €/mes	8 meses	23.666 €

Tabla 6.4: Coste humano del desarrollo

Lo cual nos lleva que el precio base total estimado del proyecto ha sido **24.766 €**

6.4.1. Estudio económico a 5 años vista

Para realizar el estudio económico a 5 años, vamos a considerar los costos adicionales de mantenimiento del software y hosting, así como los ingresos esperados.

Costes recurrentes

El servicio de mantenimiento del software y los gastos de hosting cubren la siguiente serie de gastos principales:

- **Cuotas de App Store y Google Play:** Ambas tiendas de aplicaciones cobran una cuota de inscripción. Para Google Play es un pago único de 25 USD, mientras que para la App Store de Apple es una cuota anual de 99 USD [Apple (2023)].
- **Alojamiento en la nube:** Los costos de alojamiento pueden variar enormemente dependiendo del volumen de tráfico y la cantidad de almacenamiento necesarios. Por ejemplo, una pequeña instancia en AWS puede costar alrededor de 20 USD al mes [Services (2023)], pero esto puede aumentar rápidamente con el tráfico y las necesidades de almacenamiento. Si asumimos un costo promedio de 50 USD al mes, serían unos 600 USD al año.
- **Mantenimiento del software:** Este costo cubre el tiempo y los recursos necesarios para actualizar y mantener la aplicación en funcionamiento. Esto puede incluir el tiempo del desarrollador para corregir bugs y agregar nuevas características, así como el costo de cualquier software o servicio de terceros que se utilice. Un

estimado conservador podría ser alrededor del 20 % del costo inicial de desarrollo al año.

Por lo tanto, un estimado aproximado de los costos recurrentes anuales podría ser:

- Cuota de App Store: 99 USD
- Alojamiento en la nube: 600 USD
- Mantenimiento del software: 20 % del costo inicial de desarrollo [4,955.2 €]
- Otros costos: Variable

Concepto	Coste anual
Mantenimiento del software (20 % del coste inicial de desarrollo)	4.733,2 €
Hosting (Suponiendo un coste de 50 €/mes)	600 €
Coste total recurrente	5.333,2 €

Tabla 6.5: Costes recurrentes anuales

Ingresos esperados

Los ingresos se pueden proyectar considerando tres escenarios: negativo, realista y positivo. Para este ejemplo, vamos a asumir que los ingresos provienen de las suscripciones de los usuarios a la aplicación. Los precios de las suscripciones pueden variar, por lo que en este caso vamos a suponer un precio medio de suscripción de 10 € al mes.

Escenario	Usuarios	Ingreso por usuario (anual)	Ingresos totales (anuales)	Ingresos totales (5 años)
Negativo	500	120 €	60.000 €	300.000 €
Realista	1000	120 €	120.000 €	600.000 €
Positivo	2000	120 €	240.000 €	1.200.000 €

Tabla 6.6: Ingresos esperados

Análisis del ROI

El ROI se calcula como la diferencia entre la ganancia y la inversión, dividido por la inversión. Vamos a calcularlo para cada escenario.

Escenario	Ingresos totales (5 años)	Coste total (5 años)	ROI
Negativo	300.000 €	51.432,0 €	4,82
Realista	600.000 €	51.432,0 €	10,65
Positivo	1.200.000 €	51.432,0 €	22,31

Tabla 6.7: Análisis del ROI

Como puede verse, incluso en el escenario negativo, el proyecto es rentable después de 5 años. Por supuesto, estos números son aproximados y los ingresos reales pueden variar.

Capítulo 7

Sistema Desarrollado

En el siguiente capítulo, se describirá detalladamente el sistema desarrollado. Se comenzará exponiendo el proceso de conceptualización de los microservicios y los pasos que se han tomado para implementarlos en el servidor de Django. A su vez, se explicará el desarrollo simultáneo de la vista del administrador y como es capaz de editar los datos de manera eficiente e intuitiva. Se incluirá el procedimiento realizado para el desarrollo de la API REST, la cual es la encargada de comunicar el lado del servidor con la aplicación del cliente en el día de su visita. La aplicación mencionada tiene una gran variedad de utilidades, tales como una lista detallada de los puntos de interés del parque, detalles sobre el usuario y sus entradas, y, finalmente, un sistema de colas virtuales interactivo con el objetivo de cambiar la frecuencia de espera en los parques de atracciones, entre otros.

Para documentar el sistema desarrollado para este proyecto, se va a dividir en las siguientes 3 secciones imperativas:

1. Se detallarán los desarrollos e implementaciones de los microservicios y sus backends correspondientes, garantizando un rendimiento óptimo y no redundante para el manejo de datos en la aplicación. Se evaluarán los modelos de cada microservicio y los algoritmos empleados para asegurar una disponibilidad rápida y segura de los datos.
2. Se analizará el diseño de la API (Interfaz de Programación de Aplicaciones), mostrando cómo permite interconectar los datos del servidor con la plataforma administrativa, la aplicación IoT para los empleados y la aplicación móvil para los usuarios. Todo esto se realiza con el objetivo de gestionar la información de manera eficiente y accesible en cualquier situación que pueda surgir en el parque de atracciones.
3. Finalmente, se explorará el diseño de la aplicación de usuario implementada en React Native, discutiendo los pasos y directrices seguidas para no solo interconectar la aplicación con el servidor, sino también para proporcionar a los clientes del parque una experiencia cómoda, intuitiva y rápida. Incluirá la funcionalidad de un sistema de colas virtual para minimizar el tiempo de espera en las colas.

7.1. Simulación del sistema de colas

Antes de embarcar en el desarrollo pleno de este proyecto, era imperativo comprobar la viabilidad y la precisión de la idea principal. Esta consiste en la implementación de un sistema de colas virtuales y/o reservas con la intención de ahorrar una cantidad significativa de tiempo a los visitantes del parque. Si bien esta propuesta puede funcionar bien en teoría, su aplicación práctica sigue siendo desconocida hasta que se implemente en un entorno real.

Por lo tanto, se optó por realizar una simulación de colas, aplicando los conocimientos adquiridos de la teoría de colas que se detallaron en el Capítulo 3. La simulación se desarrolló utilizando el lenguaje de programación Python, y permitió la comparación de dos tipos distintos de sistemas de colas.

El primer sistema es el método tradicional de espera en una cola. En este sistema, cuando se libera un espacio, entra un nuevo usuario. A pesar de servir a un número finito de usuarios en intervalos determinados -típicamente la duración de la atracción-, frecuentemente se observan asientos vacíos, lo que repercute en la eficiencia del sistema.

El segundo sistema es la cola virtual, la cual otorga un tiempo de servicio preestablecido e instantáneo, aunque no necesariamente ocupe toda la capacidad de la atracción en todo momento. Este aspecto se diseñó de esta manera, ya que una parte importante de la experiencia en los parques de atracciones, y de la vida en general, es mantener cierto grado de espontaneidad y no ser completamente rígidos en la estructura y planificación.

Desarrollo del simulador

En el simulador se tuvieron en cuenta dos tipos de colas. La primera es el sistema tradicional de colas acompañado por un sistema “Fast-Pass” y la segunda es el sistema de colas virtuales implementado en la aplicación, las cuales se comparan de la siguiente manera:

	<i>FastPass</i>	<i>Sistema de Colas Virtuales</i>
<i>Gestión de la espera</i>	Asigna una ventana de tiempo para el acceso rápido a la atracción, pero la entrada inmediata no está garantizada.	Asigna una hora específica para el acceso, garantizando la entrada a la atracción basándonos en la capacidad de la misma.
<i>Disponibilidad</i>	Los pases pueden agotarse para atracciones populares, limitando el acceso rápido.	Las reservas pueden agotarse, pero los visitantes tienen garantizado el acceso durante su hora asignada.
<i>Conveniencia</i>	Puede requerir planificación y adaptación del itinerario del visitante.	Permite a los visitantes saber exactamente cuando necesitan estar en cada atracción, permitiendo una planificación más eficiente.

Tabla 7.1: Comparación de sistemas de colas

El simulador se divide en 3 clases principales, que son:

- **El cliente** con un tiempo de llegada, y la posibilidad de tener un fast-pass establecida al 20 %
- **Un evento**, el cual tiene definido el tipo de evento, el cliente y el tiempo.
- **El parque de atracciones:** Que maneja las llegadas, salidas y tiempos de servicio.

Para el cálculo de aforo y tiempos se realizó la siguiente estimación: Teniendo en cuenta que el vagón medio de una atracción, se puede meter en torno a 10 personas por turno, y que cada atracción dura una media de 3 minutos. En una sesión de 30 minutos, se montarán alrededor de 100 personas.

Como se ha detallado en el capítulo 3, el tiempo de llegada de persona se modela por una distribución exponencial, y el de servicio (el tiempo que tarda una atracción en completarse) se modela por una distribución uniforme. En el cual un nuevo usuario llega cada 2 minutos, y una atracción dura de media entre 2 y 5 minutos (sin contar espectáculos o atracciones audiovisuales).

La simulación tiene en cuenta 4 colas para hacer una comparación efectiva:

1. Cola normal
2. Cola fast pass
3. Cola normal - sistema virtual
4. Cola virtual

Teniendo en cuenta la espontaneidad de una visita al parque de atracciones, para el sistema de colas virtual se han establecido los siguientes parámetros:

	<i>Porcentaje de clientela para una visita</i>
<i>Limitación de pre-reserva Virtual Q.</i>	75 %
<i>Limitación de cola física - Virtual Q.</i>	15 %
<i>Limitación de reserva Virtual Q.</i>	85 %

Tabla 7.2: Porcentajes de clientela en Virtual Q.

Esta tabla indica que usando Virtual Q. los usuarios podrán reservar, con hasta 24 horas de antelación, hasta el 75 % de las plazas por cada sesión de 30 minutos. El día de la visita, este porcentaje subirá al 85 % del aforo, manteniendo un 15 % para poder hacer una fila normal en caso de no tener la oportunidad de reservar mediante la aplicación.

7.2. Diagrama de casos de usos

Los diagramas de casos de uso son una herramienta crucial en el análisis y diseño de sistemas de software, parte integral del Lenguaje Unificado de Modelado (UML). Estos diagramas representan las interacciones entre los actores (usuarios, sistemas externos, componentes de hardware) y el sistema bajo desarrollo, identificando las funciones o “casos de uso” que el sistema puede realizar. Elementos clave en estos diagramas son los actores, los casos de uso, las relaciones entre ellos y el sistema en sí. A continuación, se observará el diagrama de caso de usos de este proyecto, centrándose en los actores principales:

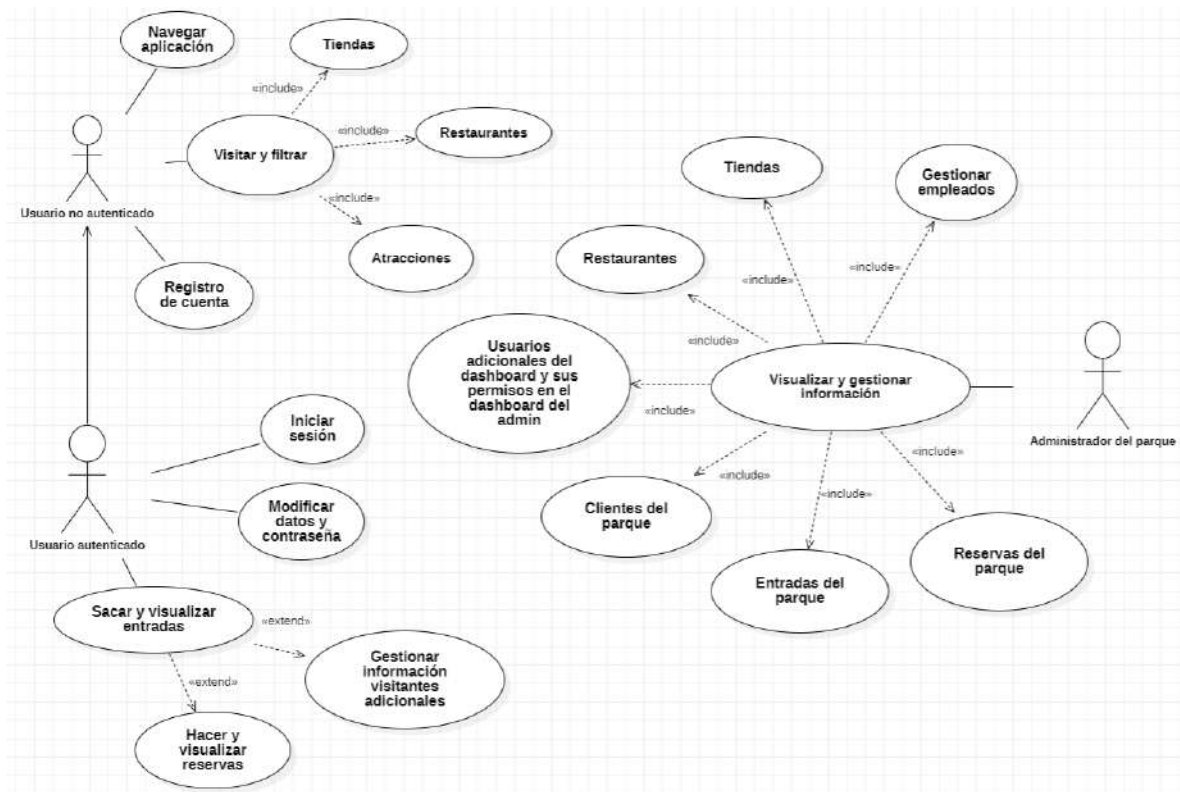


Figura 7.1: Caso de usos de la aplicación

7.3. Diseño de microservicios

Antes de comenzar a desarrollar una aplicación plantilla full-stack para el manejo de información en un parque de atracciones, se debe tener en cuenta todos los factores importantes de un parque de atracciones. En el contexto de la arquitectura de microservicios, cada componente de la aplicación ha sido concebido como un microservicio autónomo, diseñado para llevar a cabo una tarea específica.

La elección de esa metodología está dictada por la necesidad de garantizar la escalabilidad, la alta disponibilidad y la facilidad de mantenimiento. Además, esta estructura permite un desarrollo más ágil, al permitir que diferentes equipos o aplicaciones trabajen en diferentes microservicios de forma simultánea.

A continuación, se observan los microservicios identificados y como interactúan entre sí. Tras eso, se justifica su existencia y relevancia en el sistema:

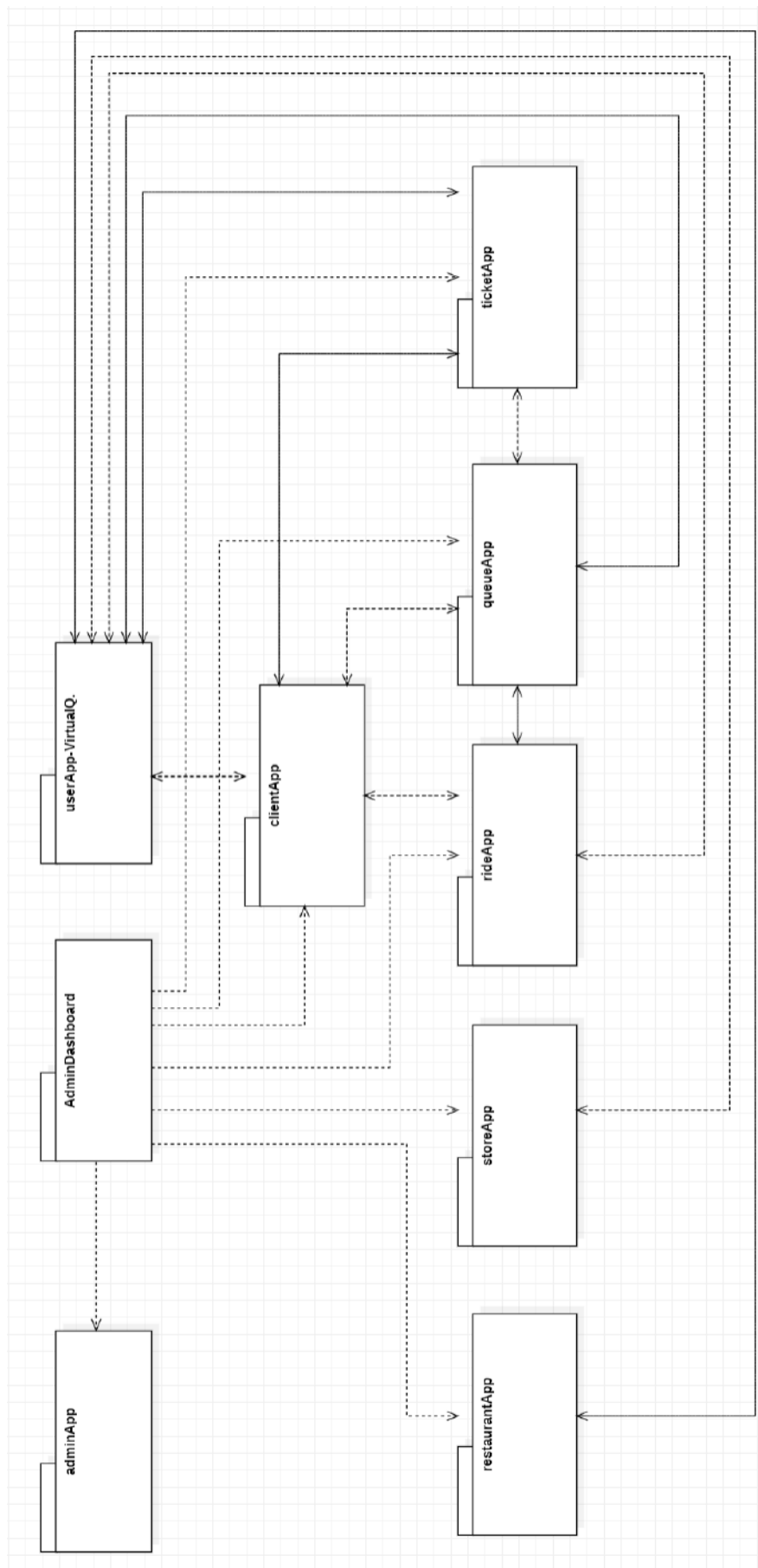


Figura 7.2: Diagrama de clases de la aplicación

1. **Virtual Q. (userApp):** Esta es la aplicación front-end que utilizarán los usuarios durante su visita al parque. Proporciona toda la información relevante a la que los usuarios tendrán acceso durante su visita, facilitando una experiencia cómoda y eficiente.
2. **Dashboard del administrador:** A pesar de ser un servicio principalmente automatizado por el framework de Django, puede ser personalizado según las preferencias del desarrollador. Este tablero constituye el medio mediante el cual un gerente del parque puede modificar la información y datos del parque de atracciones.
3. **Microservicio del Administrador (adminApp):** Este microservicio es fundamental para la gestión general del parque. Permite a los administradores gestionar la accesibilidad y roles de los empleados, garantizando que cada función del parque se lleve a cabo correctamente y que cada empleado tenga el nivel de acceso apropiado a la información.
4. **Aplicación de Clientes (clientApp):** Este componente se encarga de registrar a los usuarios del parque, lo que es esencial para poder ofrecerles una experiencia personalizada y accesibilidad a funcionalidades exclusivas. El registro también permite un seguimiento de los clientes para el análisis de comportamiento y mejora continua de los servicios ofrecidos.
5. **Aplicación de Colas (queueApp):** En un parque de atracciones, el manejo eficiente de las colas es fundamental para garantizar la satisfacción del cliente. Este microservicio permite a los usuarios ya registrados con una entrada reservar su lugar en las atracciones de forma virtual, ayudando a minimizar las largas esperas y gestionar mejor el flujo de personas.
6. **Aplicación de Manejo de Entradas (ticketApp):** La compra y gestión de entradas y reservas es un proceso esencial que debe de ser manejado eficientemente. Este microservicio ayuda a facilitar este proceso, asegurando una experiencia fluida para los clientes. Generando códigos QR correspondientes y cifrados de manera segura.
7. **Aplicación de Tiendas (storeApp):** Las tiendas son una parte integral de la experiencia en un parque de atracciones. Este microservicio facilita la gestión de las tiendas y proporciona a los clientes información relevante sobre productos y horarios.
8. **Aplicación de Restaurante (restaurantApp):** Este microservicio recoge y gestiona la información de los restaurantes y los puestos de comida esparcidos por el parque. Le permite a los clientes tomar decisiones informadas sobre donde y cuando comer. Además, ofrece a los administradores una herramienta para la gestión eficiente de los establecimientos de comida.

Cada uno de estos microservicios no solo cumple una función esencial en el funcionamiento del parque, sino que también contribuye la experiencia general del cliente, haciendo que su visita sea lo más agradable y eficiente posible.

7.3.1. Domain Driven Design y el diagrama de clases

Para identificar y diseñar los microservicios, se utilizó el método de Domain-Driven Design (DDD). El objetivo de Domain-Driven Design es simplificar el desarrollo de aplicaciones de microservicios.

Es una metodología que se basa en diseñar sistemas de software basados en el modelo del dominio de la aplicación o la empresa. Tiene una relación de organización con los microservicios y los mete en categorías para que su uso sea fácil de entender.

Un microservicio se documenta en un “bounded context”, implicando que su función está precisamente definida. Organizando microservicios ayuda a promover una estrategia colaborativa y reutilizable. Es esencial organizar cada microservicio en dominios de alto nivel (A quien le es útil) y dominios de bajo nivel (su funcionalidad).

Diagrama de clases

Un diagrama de clases es un tipo de diagrama de estructura estática que describe la estructura de un sistema mostrando las clases del sistema, sus atributos, operaciones (o métodos), y las relaciones entre los objetos. Es una herramienta fundamental de la ingeniería de software y forma parte de la metodología del Lenguaje Unificado de Modelado (UML).

Las clases se representan como rectángulos divididos en tres segmentos: el nombre de la clase en la parte superior, los atributos en el medio y las operaciones en la parte inferior. Las relaciones entre las clases se indican mediante líneas que conectan estas clases, y se utilizan diferentes tipos de líneas y puntas de flecha para indicar la naturaleza de la relación (asociación, herencia, dependencia, etc.)

A continuación, se observa el diagrama de clases del back-end de este proyecto. Demostrando las distintas relaciones que existen entre cada microservicio, y como se complementan para servir las distintas funcionalidades de la aplicación.

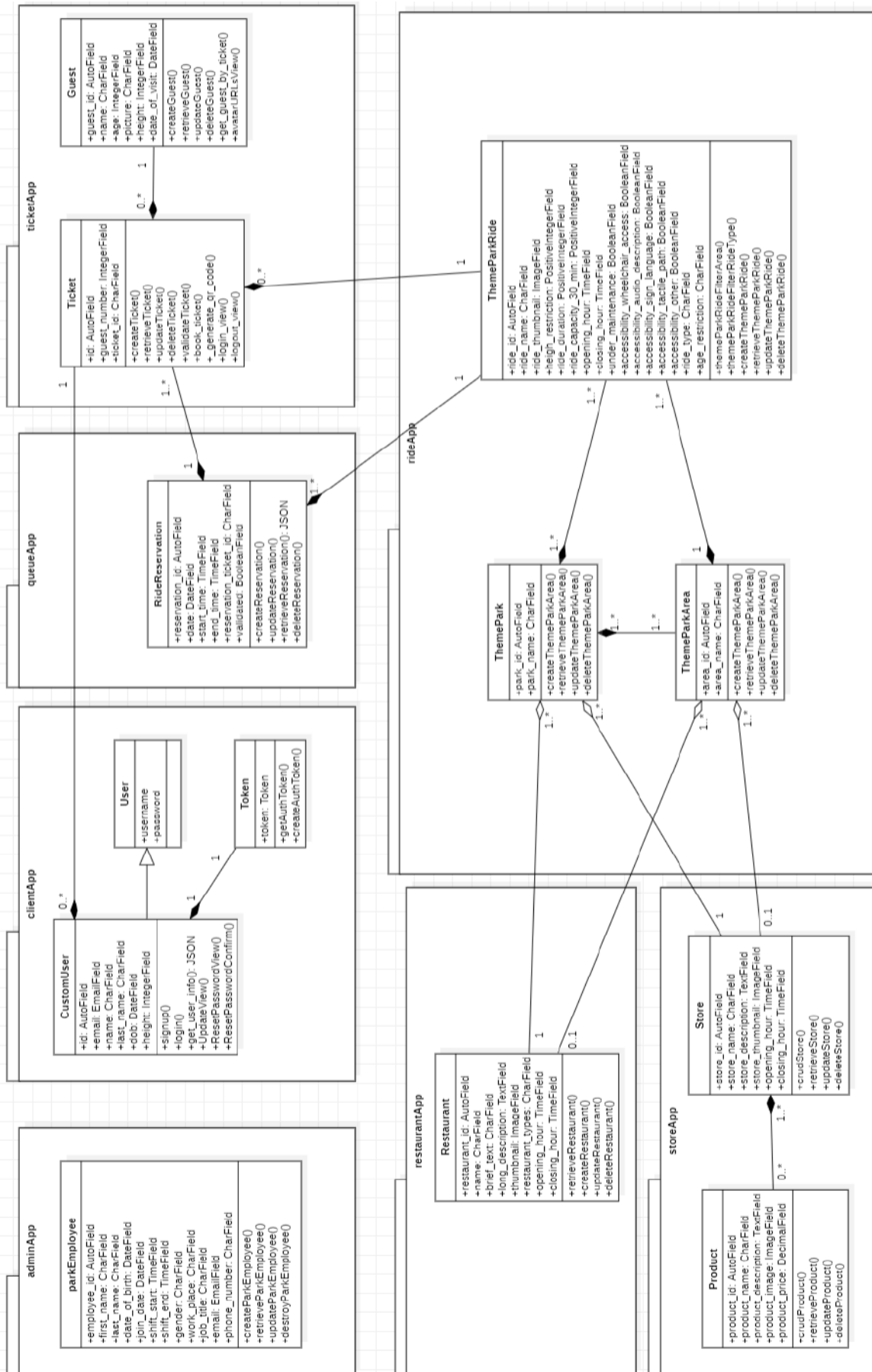


Figura 7.3: Diagrama de clases de la aplicación

7.3.2. Microservicio del Administrador

Este microservicio está encargado de gestionar a los empleados del parque, incluyendo el registro de sus datos personales, administrando las responsabilidades de control de acceso y administración de roles para los empleados del parque.

Campo	Tipo de información
<i>ID del empleado</i>	ID único de empleado de tipo numérico
<i>Nombre</i>	Character Field
<i>Apellido</i>	Character Field
<i>Fecha de nacimiento</i>	Date Field
<i>Día de comienzo (Empleabilidad)</i>	Date Field
<i>Comienzo de la jornada laboral</i>	Time Field
<i>Fin de la jornada laboral</i>	Time Field
<i>Género</i>	Masculino/Femenino/Otro
<i>Parque en el que se trabaja</i>	Character Field seleccionable
<i>Área del parque</i>	Character Field seleccionable
<i>Tipo de Empleo</i>	(Atracción/Tienda/Restaurante)
<i>Correo Electrónico</i>	Character Field regulado por RegEx
<i>Número de teléfono</i>	Character Field regulado por RegEx

Tabla 7.3: Modelo del Empleado

Funcionalidades

- **Gestión de Roles de Empleados:** Este microservicio permite a los administradores asignar y ajustar los roles y responsabilidades de los empleados. Esto asegura que cada empleado tenga una descripción clara de sus deberes, horarios y responsabilidades.
- **Control de Acceso:** El control de acceso se gestiona en este microservicio, el cual garantiza que los empleados solo puedan acceder a la información y a las funciones que son pertinentes en sus roles. Esto es crítico para mantener la seguridad de datos y la eficacia operativa.

7.3.3. Microservicio de Clientes

La aplicación de Clientes es uno de los microservicios críticos en el sistema de parque de atracciones. Esta aplicación es responsable de la gestión de los usuarios, desde su registro inicial hasta el seguimiento continuo de su interacción con el parque.

Campo	Tipo de información
Nombre	Character Field
Apellido	Character Field
Correo Electrónico	Email Field
Usuario	Character Field
Contraseña	Character Field
Fecha de Nacimiento	Date Field

Tabla 7.4: Modelo del Usuario de la aplicación (Cliente)

Funcionalidades

Este microservicio ofrece las siguientes funcionalidades:

- **Registro de Usuarios:** Permite a los visitantes del parque registrarse y crear un perfil de usuario único. Este perfil se puede utilizar para personalizar la experiencia del usuario, proporcionándole recomendaciones personalizadas y permitiéndole reservar atracciones y servicios.
- **Ingreso y Autenticación:** Los usuarios registrados pueden ingresar a la aplicación utilizando sus credenciales. Este proceso de autenticación es esencial para proteger la información del usuario y permitir funcionalidades exclusivas a los usuarios registrados. Con estos métodos el usuario podrá iniciar o cerrar sesión y editar sus datos, incluida su contraseña.

Debido a la importancia en el proyecto que tiene la autenticación de los usuarios al ingresar sesión, a continuación se detalla como se ha implementado la seguridad que caracteriza esta aplicación.

7.3.4. Autorización mediante Token en el Microservicio de Clientes

Una consideración crucial en la construcción de la Aplicación de Clientes fue la autenticación y autorización de los usuarios. Dada la naturaleza móvil de la aplicación y la necesidad de proteger los datos y las funcionalidades de los usuarios, se decidió implementar una Autorización mediante Token utilizando Django, el marco de trabajo del lado de servidor.

La Autorización mediante Token es un método que permite a los usuarios obtener acceso a los recursos de la aplicación presentando un token de autorización. Este token se obtiene al autenticarse con éxito en el sistema y es usado en las solicitudes subsecuentes para demostrar que el usuario está autorizado para acceder a los recursos solicitados.

Características y Razones para su Uso

Las características principales de la Autorización mediante Token en Django incluyen:

- **Seguridad:** Cada token es único para su usuario específico y tiene una vida útil limitada, lo que reduce la posibilidad de su uso malicioso. Django también

proporciona una serie de medidas de seguridad adicionales, como la protección contra ataques de tipo Cross-Site Request Forgery (CSRF)

- **Escalabilidad y Rendimiento:** Los tokens de autorización son una forma eficiente de autenticación porque no requieren de consultas constantes a la base de datos para verificar las credenciales del usuario. Una vez que se genera el token, este puede ser verificado independientemente.
- **Compatibilidad con Aplicaciones Móviles:** La Autorización mediante Token es especialmente adecuada para aplicaciones móviles. Los tokens son fácilmente transportables a través de diferentes dispositivos y plataformas, lo que facilita el uso de la Aplicación de Clientes en varios dispositivos móviles.

Dada su seguridad, escalabilidad, rendimiento y compatibilidad con aplicaciones móviles, la Autorización mediante Token en Django fue seleccionada como la solución óptima para manejar la autenticación y la autorización en la Aplicación de Clientes.

Funcionamiento de la Autorización mediante Token

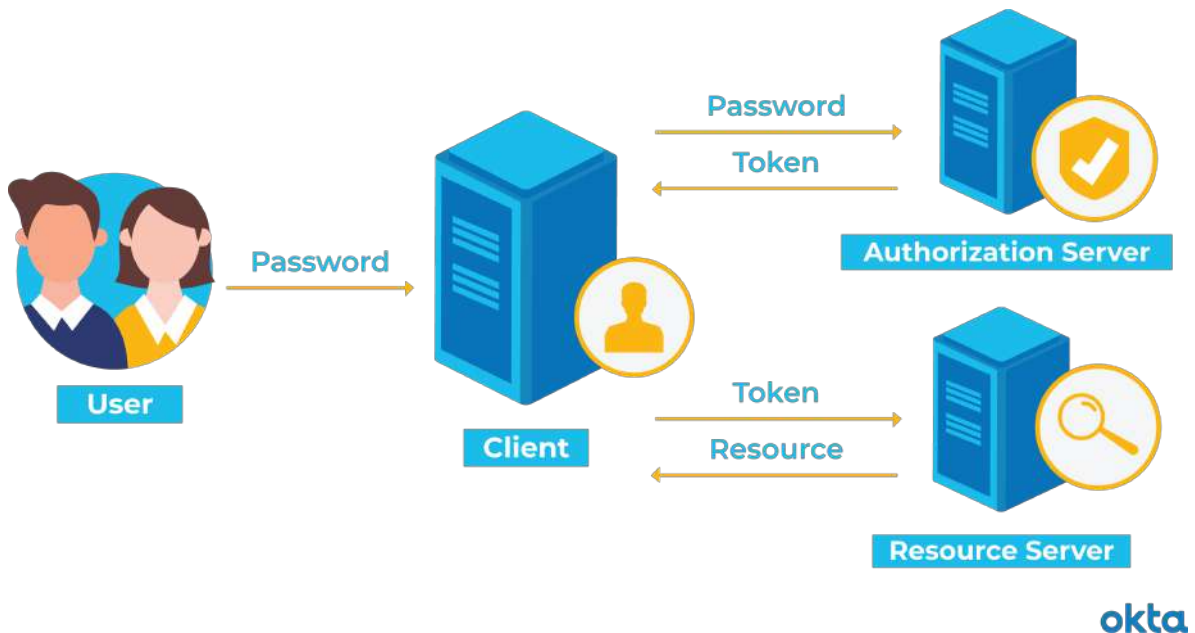


Figura 7.4: Autenticación mediante Token diagrama de flujo

La Autorización mediante Token en Django sigue un flujo de trabajo específico para garantizar que solo los usuarios autenticados y autorizados puedan acceder a los recursos pertinentes en la Aplicación de Clientes.

1. **Autenticación del Usuario:** Cuando un usuario intenta acceder por primera vez a la aplicación, se le solicita que proporcione sus credenciales (generalmente un nombre de usuario y una contraseña). Estas credenciales son enviadas a un endpoint de autenticación seguro en el servidor.
2. **Generación del Token:** Una vez que las credenciales del usuario se han verificado y confirmado que son válidas, el servidor genera un token único para el usuario.

Este token se asocia en la base de datos con el perfil del usuario correspondiente.

3. **Envío del Token al Usuario:** El servidor devuelve el token al cliente en la respuesta a la solicitud de autenticación.
4. **Uso del Token para Solicitudes Futuras:** Para futuras solicitudes al servidor, el cliente debe incluir este token en el encabezado de la solicitud. Este token sirve como prueba de que el usuario se ha autenticado previamente y está autorizado para realizar la solicitud.
5. **Verificación del Token:** Cuando el servidor recibe una solicitud con un token, verifica la validez del token y, si es válido, procesa la solicitud.
6. **Expiración y Renovación del Token:** Para mantener un alto nivel de seguridad, los tokens tienen un tiempo de vida limitado. Cuando un token expira, se debe solicitar uno nuevo a través del proceso de autenticación.

Mediante el siguiente diagrama se presenta como un usuario proporciona su usuario y clave al servidor (en este caso *admin* y *admin*) y el servidor le genera un token personal y único:

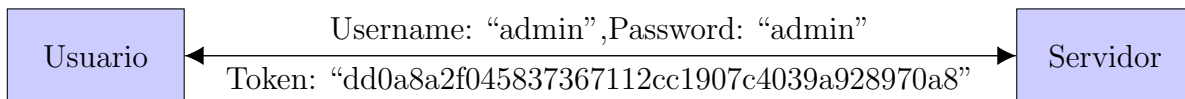


Figura 7.5: Demo de Token Based Authentication

Esta estrategia de autorización visualizada en la figura [7.4](#) garantiza que la identidad del usuario se comprueba de forma segura en cada interacción, protegiendo la información del usuario y limitando el acceso a las funciones de la aplicación únicamente a los usuarios autorizados. Además, la Autorización mediante Token proporciona una experiencia de usuario fluida, ya que los usuarios solo necesitan autenticarse una vez por sesión o hasta que su token expire.

7.3.5. Microservicio de Manejo de Entradas

La compra y gestión de entradas y reservas es un proceso esencial en la operación del parque de atracciones. Este microservicio facilita este proceso al proporcionar una interfaz accesible y eficiente para los clientes. Los usuarios pueden comprar entradas, hacer reservas e incluso realizar cambios en sus planes con facilidad. Además, para maximizar la seguridad y minimizar el fraude, cada entrada se asocia con un código QR único que se cifra de forma segura. Este código QR puede ser escaneado en los puntos de entrada y en las atracciones, asegurando que solo los titulares de entradas válidas tengan acceso.

Campo	Tipo de información
<i>Usuario</i>	Clave externa que se asocia a la base de datos del Cliente
<i>Día de la Visita</i>	Date Field
<i>Numero de visitante</i>	Integer Field (En caso de que un usuario saque más de una entrada en su visita)
<i>ID de entrada</i>	Character Field que se genera automáticamente basado en la información de los tres primeros campos.

Tabla 7.5: Modelo de entradas del parque.

Funcionalidades

- **Creación de entradas y reserva:** Esta funcionalidad permite la generación de entradas y reservas de manera eficiente y segura. Los usuarios pueden seleccionar la fecha de su visita y el número de visitantes adicionales. Una vez que la información se valida y se procesa la información y se crea su ID de entrada y registra en la aplicación lista para que la tenga a mano el usuario. Esto facilita una experiencia de usuario fluida y conveniente, y garantiza que los datos de las entradas se manejen de manera segura y confiable.
- **Validación de entradas:** Para garantizar que solo los titulares legítimos de entradas puedan acceder al parque y a sus atracciones, este microservicio incluye una funcionalidad de validación de entradas. Cada entrada generada se asocia con un código único y sé válida al momento de acceder al parque o a las atracciones. Este proceso incluye la verificación del código de la entrada, la cual verifica a su vez la coincidencia de los detalles del titular de la entrada y la comprobación de la validez temporal de la entrada.
- **Generación de códigos QR:** Con el objetivo de facilitar la validación y verificación de las entradas, este microservicio incluye una funcionalidad de generación de códigos QR. Cada entrada se asocia con un código QR único que contiene la información relevante de la entrada cifrada de manera segura. Este código QR puede ser escaneado rápidamente en los puntos de entrada y en las atracciones, facilitando un proceso de validación eficiente y minimizando la necesidad de interacción manual.
- **Registro completo de entradas:** Con el fin de mantener un registro detallado y preciso de las entradas, este microservicio proporciona una funcionalidad de registro de entradas. Cada entrada generada se registra en una base de datos segura, incluyendo detalles como la fecha de emisión y el titular de ella junto con el ID. Esto permite un seguimiento eficiente de las entradas y facilita la gestión y análisis de los datos de las entradas.

Debido a que una de las principales funcionalidades que se ofrece en este proyecto es la gestión de entradas, se ha garantizado la implementación de una metodología de códigos QR. Se verifica que la información que contiene el QR se ha cifrado correctamente. A continuación, se va a exponer como se ha realizado este proceso.

7.3.6. Cifrado para la generación de códigos QR como entradas

El modelo de entradas (Ticket) cuenta con un método privado `_generate_ticket()`, cuya finalidad es generar un identificador de entrada único y seguro. A este método se le llama cuando se guarda un nuevo objeto 'Ticket' y si este no tiene aún un identificador '`ticket_id`' asociado.

Para generar dicho identificador, el método '`_generate_ticket()`' primero construye una cadena de texto ('data') que consta del correo electrónico del usuario, la fecha de visita y el número de invitados, separadas por guiones bajos. Esta cadena sirve como la base de la información que se va a cifrar.

A continuación, se genera una cadena aleatoria ('`random_secret`') de 12 caracteres utilizando la función '`get_random_string()`'. El motivo por añadir esta cadena aleatoria es debido a que añade un nivel adicional de seguridad al proceso de cifrado, ya que hace que el `ticket_id` resultante sea impredecible, incluso si los datos de entrada son conocidos.

La cadena `data` y la `random_secret` se concatenan y se cifran utilizando el algoritmo SHA-256. Este algoritmo de cifrado de un solo sentido asegura que la información cifrada no puede ser revertida para obtener la información original, lo cual proporciona un grado muy elevado de seguridad.

El resultado del cifrado SHA-256 se codifica en Base64 para obtener una cadena que puede ser almacenada y transmitida de forma segura. Esta codificación es comúnmente utilizada para cuando hay necesidad de codificar datos binarios, especialmente cuando estos datos deben de ser almacenados y transferidos a través de sistemas que están diseñados para tratar con texto.

Finalmente, tras haber finalizado la generación del ID del ticket, este código se pasa a un generador de códigos QR en el front-end y se mostrará en la aplicación del usuario, tan pronto se haya creado una cuenta y haya comprado una entrada.

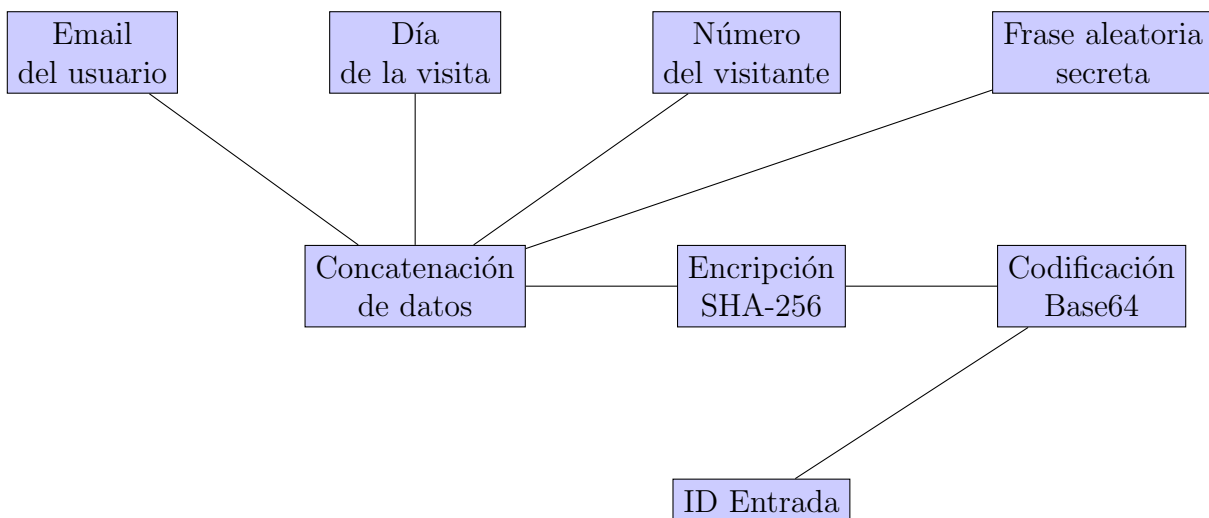


Figura 7.6: Diagrama de la metodología aplicada para la encriptación de entradas

El siguiente diagrama demuestra, usando una serie de valores ejemplares, el funciona-

miento del sistema existente y con ello la evolución del valor del ticket ID hasta su cifrado completo.

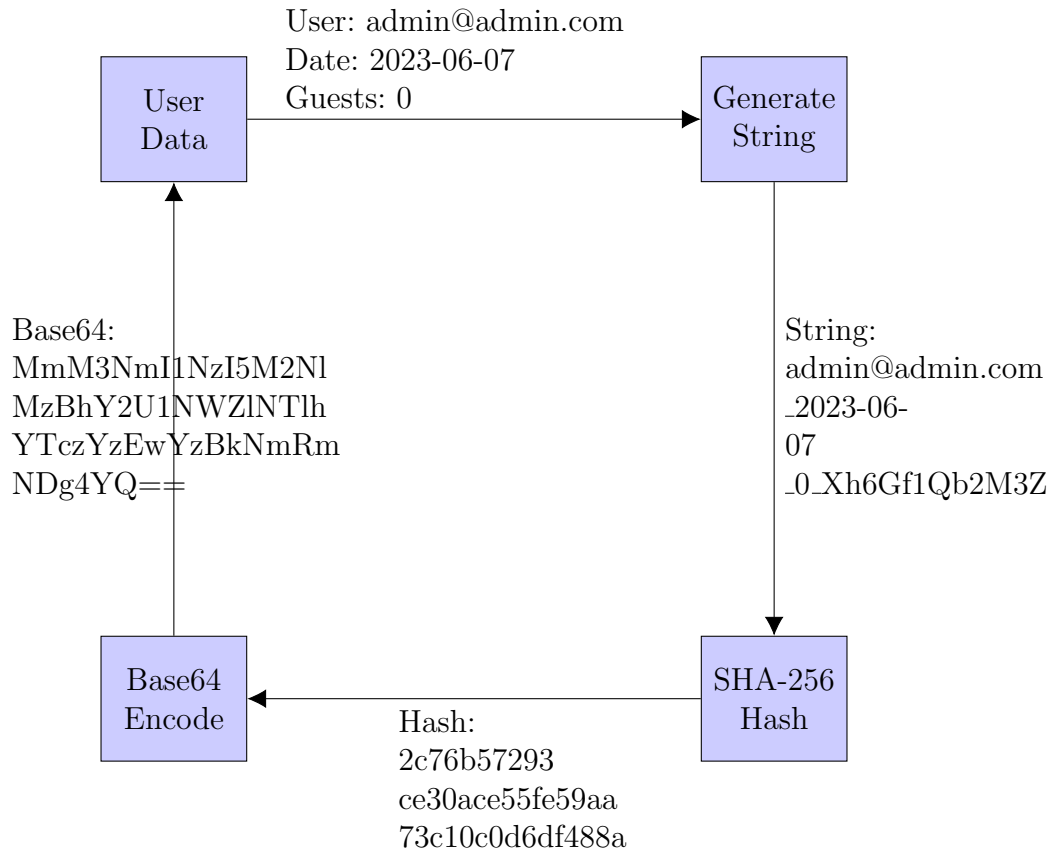


Figura 7.7: Demo de la codificación del Ticket ID

7.3.7. Microservicio de las Colas

En un entorno como un parque de atracciones, el manejo eficiente de las colas es crucial para la satisfacción del cliente. Este microservicio proporciona una solución robusta a este desafío, permitiendo a los usuarios registrados con una entrada válida reservar su lugar en las atracciones de forma virtual. Esta funcionalidad no solo reduce las largas esperas, sino que también ayuda a gestionar el flujo de visitantes de una manera más eficiente. Al considerar las variables como la capacidad de la atracción y los tiempos de espera estimados, este microservicio optimiza la distribución de los clientes y mejora su experiencia global. Además, se puede implementar un sistema de priorización para tratar de manera justa y eficiente las solicitudes de reserva de colas.

Este microservicio contiene los siguientes campos:

Las funcionalidades del microservicio son las siguientes:

- **Generación de Reservas:** Para poder crear una entrada en este modelo, son necesarios cumplir una serie de requisitos. Ante todo, para poder crear una reserva, dicho modelo requiere estar asignados a una entrada existente, (sea de un usuario

Campo	Tipo de información
<i>ID de la reserva</i>	AutoField
<i>Ticket</i>	Clave externa al modelo de Entradas
<i>Ride</i>	Clave externa al modelo de atracciones
<i>Fecha de la reserva</i>	DateField
<i>Comienzo de la reserva</i>	TimeField
<i>Fin de la reserva</i>	TimeField
<i>Código QR para la reserva</i>	CharField
<i>Validado</i>	BooleanField

Tabla 7.6: Modelo de reservas del parque

o de un acompañante suyo), una atracción, una fecha (la cual debe de coincidir con la fecha de la entrada para ser válida), y una hora de reserva.

Una vez obtenidos los siguientes datos, se deben de validar considerando las siguientes condiciones:

1. **Comprobar si la atracción está bajo mantenimiento:** No se le permitirá a ningún usuario, bajo ninguna condición, la habilidad de poder hacer una reserva mediante la aplicación móvil, en caso de que una atracción se encuentre bajo mantenimiento e inoperativa.
 2. **Comprobar que la fecha de la reserva coincide con la fecha de la entrada:** Esta condición se refuerza en el front-end, ya que le será a un usuario imposible seleccionar una fecha en la aplicación que no coincida con las fechas en las cuales ha sacado su entrada. Ya que si no, se podría perjudicar el sistema, y ocupar plazas que no se van a rellenar.
 3. **No permitirle a un usuario o a un acompañante reservar una plaza en la misma hora:** En caso de que un usuario o, por lo tanto, un miembro de su grupo intente hacer una reserva para un tiempo en el cual ya tiene otra, quedara expresamente prohibido y se le notificara mediante un error.
- **Poder obtener y visualizar reservas individuales:** Es imperativo que un usuario, una vez haya hecho y confirmado su reserva, que las pueda visualizar y gestionar de manera apropiada. En la aplicación front-end habría una vista dedicada para su visualización y, por lo tanto, es necesario incluir este método.

Desarrollando sobre este punto, es imperativo comentar que la reserva se mostrara la entrada de la atracción con un código QR. La generación de este código QR funciona de la misma manera que las entradas, con cambios hechos a la cadena de datos original. Mientras el algoritmo de cifrado SHA-256 y base64 se sigue utilizando, la cadena de verificación esta vez incluye:

el email del usuario encargado, el ID de la entrada asociada a la reserva, el tiempo de la reserva hecha.

Gracias a estos métodos se podrá gestionar las reservas tanto con el punto de perspectiva de los usuarios como los de los gerentes del parque.

7.3.8. Microservicio de tiendas

Las tiendas son una parte integral de la experiencia en un parque de atracciones, proporcionando a los visitantes la oportunidad de comprar recuerdos y suministros necesarios. Este microservicio gestiona la información relacionada con las tiendas del parque, incluyendo la ubicación, los productos disponibles, los precios y los horarios de apertura. Al hacerlo, permite a los clientes tomar decisiones informadas sobre sus compras. Además, esta aplicación ofrece a los administradores del parque una herramienta eficaz para monitorear y gestionar las operaciones de las tiendas, desde la gestión de inventario hasta la fijación de precios.

Campo	Tipo de información
<i>ID Parque</i>	Clave externa (Parque de atracciones en el cual se ubica la tienda)
<i>ID Area</i>	Clave externa (Área del parque de atracciones en la que se ubica la tienda)
<i>ID Tienda</i>	Automatic Field (ID numérico único que se crea de manera automática)
<i>Nombre Tienda</i>	Character Field
<i>Descripción Tienda</i>	Text Field
<i>Imagen de la Tienda</i>	Imagen que se muestra en la aplicación
<i>Hora de Apertura</i>	Time Field
<i>Hora de Cierre</i>	Time Field

Tabla 7.7: Modelo de la tienda

Campo	Tipo de información
<i>Tienda</i>	Clave Externa (Relacionar con el modelo de tienda)
<i>ID del Producto</i>	Clave automática
<i>Nombre del Producto</i>	Character Field
<i>Descripción del Producto</i>	Text Field
<i>Precio del Producto</i>	Decimal Field

Tabla 7.8: Modelo de los productos

Funcionalidades del microservicio

- **Actualización de precios:** Los administradores pueden utilizar este microservicio para actualizar los precios de los productos de forma rápida y eficiente. Esto puede ser especialmente útil durante eventos especiales o promociones.
- **Información detallada de los productos:** Los visitantes del parque pueden acceder a descripciones detalladas de los productos, incluyendo el precio, a través de este microservicio. Esta funcionalidad puede ayudar a los visitantes a tomar decisiones informadas sobre sus compras.
- **Localización de tiendas:** Los visitantes pueden utilizar este microservicio para localizar las tiendas dentro del parque. Esta funcionalidad puede ayudar a los visitantes a planificar su visita y a localizar fácilmente las tiendas que desean visitar.
- **Horarios de apertura y cierre:** Los visitantes pueden utilizar este microservicio para obtener información sobre los horarios de apertura y cierre de las tiendas.

Esta información puede ser útil para planificar las visitas a las tiendas durante su visita al parque.

7.3.9. Microservicio de restaurantes

Este microservicio gestiona la información de los restaurantes y puestos de comida esparcidos por todo el parque. Proporciona a los clientes detalles como el tipo de cocina, horarios de apertura, menús y ubicación de cada restaurante o puesto de comida. Al permitir a los clientes tomar decisiones informadas, este microservicio mejora su experiencia de comida y maximiza la eficiencia de la operación de los restaurantes. Para los administradores, esta aplicación ofrece una herramienta eficaz para gestionar las operaciones de los restaurantes, desde la programación de los horarios de apertura hasta la gestión de los menús.

Campo	Tipo de información
<i>ID del restaurante</i>	Automatic Unique Field (Campo numérico con valores únicos que se genera automáticamente)
<i>Nombre del restaurante</i>	Character Field
<i>Brief Text</i>	Character Field (Slogan del restaurante)
<i>Long Description</i>	Character Field (Descripción del restaurante)
<i>Thumbnail</i>	Image Field (Imagen a mostrar en la aplicación)
<i>Tipo de restaurante</i>	Char Field (Valor seleccionable entre los tipos de restaurantes existentes)
<i>Opening Hour</i>	Time Field (Hora de apertura del restaurante)
<i>Closing Hour</i>	Time Field (Hora de cierre del restaurante)
<i>Park</i>	Clave externa que relaciona el parque en el cual se encuentra el restaurante
<i>Area</i>	Clave externa que relaciona el área adentro del parque en el cual se encuentra

Tabla 7.9: Modelo del restaurante

Funcionalidades

- **Gestión de menús:** Este microservicio permite a los administradores del restaurante actualizar y gestionar los menús de manera eficiente. Los cambios en el menú pueden ser reflejados en tiempo real, proporcionando a los visitantes información precisa y actualizada.
- **Detalles del restaurante:** Los visitantes del parque pueden obtener una descripción detallada de cada restaurante, incluyendo el tipo de cocina, el horario de apertura y cierre, y la ubicación. Esto les ayuda a tomar decisiones informadas sobre dónde y cuándo comer.
- **Localización de restaurantes:** Este microservicio proporciona a los visitantes la ubicación exacta de cada restaurante en el parque. Los visitantes pueden utilizar esta función para planificar su recorrido por el parque y asegurarse de que pueden encontrar fácilmente los restaurantes que desean visitar.
- **Gestión de horarios:** Los administradores del restaurante pueden utilizar este microservicio para gestionar los horarios de apertura y cierre de los restaurantes. Esto asegura que los visitantes tengan acceso a la información correcta y puedan planificar sus comidas en consecuencia.
- **Gestión de tipos de cocina:** Este microservicio permite a los administradores del restaurante definir y actualizar el tipo de cocina ofrecida en cada restaurante.

Esto ayuda a los visitantes a seleccionar restaurantes basados en sus preferencias culinarias.

7.4. Desarrollo de Back-End usando Django

Django es un framework de desarrollo web de código abierto en Python que sigue el patrón de diseño Model-View-Template (MVT). Su principal objetivo es simplificar la creación de sitios web complejos, proporcionando todas las herramientas necesarias para desarrollar aplicaciones web de alta calidad con un flujo de desarrollo simple.

Por defecto, usa un sistema de bases de datos SQLite. El cual peculiarmente centra toda la base de datos en un único archivo. A diferencia de otras bases de datos como MySQL o PostgreSQL, no hay un servidor separado que se comunica con la aplicación. En su lugar, toda la base de datos se almacena en un solo archivo en el disco.

Un sistema back-end en Django consta principalmente de cinco componentes: `models.py`, `serializers.py`, `admin.py`, `urls.py` y `views.py`. Cada uno de estos archivos desempeña un papel fundamental en la operación del sistema y el flujo de datos, que se observará a continuación:

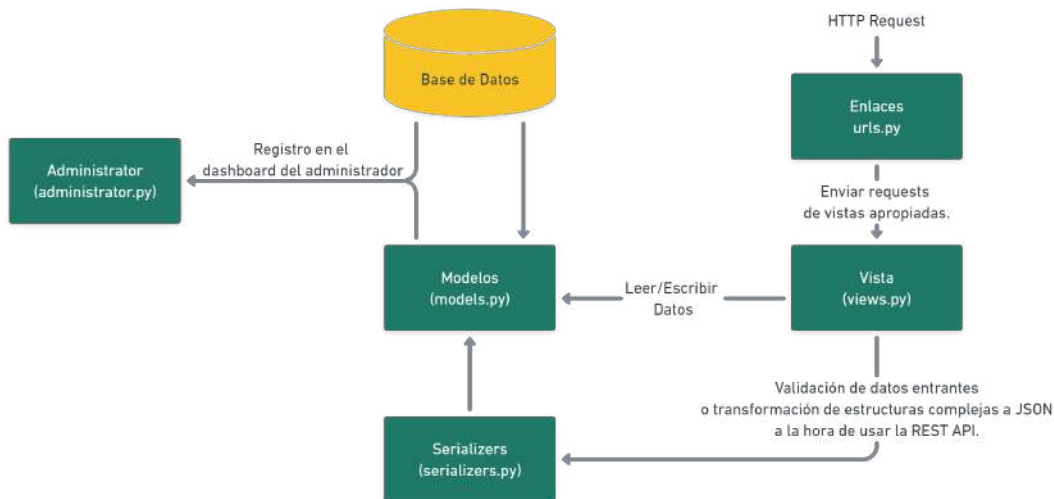


Figura 7.8: Gráfica de flujo de uso de archivos back-end Django

7.4.1. `models.py`

Models.py contiene las representaciones de las bases de datos utilizadas por la aplicación. Cada modelo en Django corresponde a una tabla en la base de datos y define los campos y comportamientos de los datos que se van a almacenar. Los modelos también pueden definir métodos para manipular los datos. Django se encarga de crear, acceder, modificar y eliminar registros en la base de datos a través de una API de alto nivel en Python.

El típico archivo `models.py` tiene la siguiente sintaxis:

```

1 from django.db import models
2

```

```

3 class ThemePark(models.Model):
4     park_id = models.AutoField(primary_key=True)
5     park_name = models.CharField(max_length=255)
6
7     def __str__(self):
8         return self.park_name

```

Listing 7.1: Definición de la estructura de la base de datos del usuario (models.py)

Como se puede observar, sigue una sintaxis simple y ordenada en el cual se le asigna una propiedad, su tipo de campo, y su longitud máxima, o el tipo de clave que es. Los campos usados a través del proyecto han sido los siguientes:

- **CharField:** Este tipo de campo se utiliza para almacenar cadenas de texto cortas. Debes especificar el argumento y su longitud máxima. Este tipo de campos se usan para registrar nombres, apellidos, frases cortas, etc...
- **IntegerField:** Como su nombre indica, este tipo de campo se utiliza para almacenar un número entero, útil a la hora de hacer cuenta de usuarios, registros, entre otros.
- **DateField:** Este campo se utiliza para almacenar fechas. Puede tomar varias opciones para manejar automáticamente las fechas de creación y modificación. Útil para fechas de nacimiento, registro de días de visita.
- **TimeField:** Este campo se utiliza para almacenar fechas. Puede tomar varias opciones para manejar automáticamente las fechas de creación y modificación. Especialmente útil para horas de reserva, al igual que los horarios del parque.
- **AutoField:** Es un tipo de Integer Field que se autoincrementa. Se utiliza generalmente para los campos de identificación principal (primary key) de los modelos, ya que proporciona un identificador único para cada instancia del modelo.
- **ForeignKey:** Este campo se utiliza para crear una relación de muchos a uno. Es decir, permite que muchas instancias de un modelo se relacionen con una sola instancia de otro modelo. Este campo debe recibir como primer argumento el modelo con el que se establece la relación.
- **BooleanField:** Como su nombre indica, este campo almacena un valor booleano (Verdadero o Falso). Es útil para indicar el estado de una instancia del modelo, como si está activa o inactiva, si está visible o no, entre otros.
- **DecimalField:** Este campo se utiliza para almacenar números decimales. Se debe especificar el número máximo de dígitos (*max_digits*) y el número máximo de dígitos decimales (*decimal_places*).

Sin embargo, dependiendo de los requisitos de cada microservicio, existen varias variaciones notables:

1. **En el caso de la generación del ID de entradas:** Previamente, se ha justificado la metodología para proporcionar un ID de entrada que sea seguro usando los algoritmos de codificación SHA-256 y Base64. Su implementación es la siguiente:

```

1 #Importacion de librerias necesarias
2 from django.db import models

```



```

3 from django.conf import settings
4 from django.utils.crypto import get_random_string
5 from hashlib import sha256
6 import base64
7
8
9 class Ticket(models.Model):
10     user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
11     date_of_visit = models.DateField()
12     guest_number = models.IntegerField(null=True, blank=True)
13     ticket_id = models.CharField(max_length=200, editable=False) # Relational
14         target
15
16     def _generate_ticket(self, guest_number):
17         guest_number_str = (str(guest_number) if guest_number is not 0 else "0")
18         # Convertimos el numero de visitantes adicionales a una cadena
19         data = f"{self.user.email}_{self.date_of_visit}_{guest_number_str}" #
20         user_email_date_of_visit_1,2,3,4,5,
21         random_secret = get_random_string(12) # Incorporamos la cadena aleatoria
22         al final de los datos
23
24         return base64.urlsafe_b64encode( sha256(f"{data}_{random_secret}".encode
25             ()).digest()).decode()
26
27     def save(self, *args, **kwargs):
28         if not self.ticket_id: # Comprobamos si existe ya el ticket ID
29             previamente a generarlo, para prevenir duplicados
30             self.ticket_id = self._generate_ticket(self.guest_number)
31         super().save(*args, **kwargs)

```

Listing 7.2: Generación de datos de un campo al crear una nueva entrada (models.py)

2. **En el caso de solo poder seleccionar un rango de valores fijos para un CharField:** Este caso es aplicable en el caso de tener que seleccionar el tipo de comida que sirve un restaurante, a la edad limite a la cual está destinada una atracción, o a un tipo de atracción existente:

```

1 class ThemeParkRide(models.Model):
2     ride_id = models.AutoField(primary_key=True)
3     ride_name = models.CharField(max_length=255)
4     ride_description = models.TextField()
5     ride_thumbnail = models.ImageField(upload_to="media/ride_thumbnails/")
6     area_id = models.ForeignKey(ThemeParkArea, on_delete=models.CASCADE)
7     park_id = models.ForeignKey(ThemePark, on_delete=models.CASCADE)
8     height_restriction = models.PositiveIntegerField()
9     ride_capacity = models.PositiveIntegerField()
10    ride_duration = models.PositiveIntegerField()
11    opening_hour = models.TimeField()
12    closing_hour = models.TimeField()
13    under_maintenance = models.BooleanField()
14
15    # Accessibility features
16    accessibility_wheelchair_access = models.BooleanField(default=False)
17    accessibility_audio_description = models.BooleanField(default=False)
18    accessibility_braille = models.BooleanField(default=False)
19    accessibility_sign_language = models.BooleanField(default=False)
20    accessibility_closed_captioning = models.BooleanField(default=False)
21    accessibility_tactile_path = models.BooleanField(default=False)
22    accessibility_other = models.BooleanField(default=False)
23
24    #Indicamos aqui el tipo de atraccion unica a escoger
25    RIDE_TYPE_CHOICES = [
26        ("ROLLER_COASTER", "Roller Coaster"),
27        ("BIG_DROPS", "Big Drops"),
28        ("SMALL_DROPS", "Small Drops"),
29        ("THRILL_RIDES", "Thrill Rides"),
30        ("SLOW_RIDES", "Slow Rides"),
31        ("STAGE_SHOWS", "Stage Shows"),
32        ("FIREWORKS", "Fireworks"),

```

```

33         ("CHARACTER_EXPERIENCES", "Character Experience"),
34         ("PARADES", "Parades"),
35         ("WATER_RIDES", "Water Rides"),
36         ("SPINNING", "Spinning"),
37         ("LIVE", "Live Entertainment"),
38         ("DARK", "Dark"),
39         ("LOUD", "Loud"),
40         ("SCARY", "Scary"),
41     ]
42
43     #Indicamos aqui la edad limite a poder escoger
44     AGE_CHOICES = [
45         ("PREESCHOOLERS", "Preeschoolers"),
46         ("KIDS", "Kids"),
47         ("TEENS", "Teens"),
48         ("ADULTS", "Adults"),
49         ("TWEENS", "Tweens"),
50     ]
51
52     ride_type = models.CharField(max_length=255, choices=RIDE_TYPE_CHOICES)
53     age_restriction = models.CharField(max_length=255, choices=AGE_CHOICES)
54
55     #Metodos en caso de tener que definir en texto varias propiedades de la base
    de datos.
56     def __str__(self):
57         return self.ride_name
58
59     def park_open_hours(self):
60         """
61         This method returns the ride is open
62         """
63         return self.closing_hour - self.opening_hour
64
65     def get_area_name(self):
66         return self.area_id.area_name

```

Listing 7.3: Codificación de la base de datos en la cual un campo tiene un rango determinado de valores (models.py)

Aparte de estas excepciones, la registración de elementos adentro de un modelo de Django se desarrolla de manera metódica y formulada como se ha explicado.

7.4.2. serializers.py

Serializers.py se encarga de la transformación de los datos complejos de los modelos en tipos nativos de Python para que puedan ser fácilmente renderizados en JSON, XML u otros formatos de contenido. De igual forma, los serializers validan los datos entrantes al sistema para mantener la integridad de los datos. Aunque no suelen ser parte de un proyecto normal de Django, suelen ser esenciales si se decide desarrollar una base en torno a un framework REST API, el cual es nuestro caso.

Típicamente, un serializador de Django sigue la siguiente estructura:

```

1 #Librerias
2 from rest_framework import serializers
3 from .models import Ticket
4
5 # serializers.py
6 class TicketSerializer(serializers.ModelSerializer):
7     user = serializers.SerializerMethodField() # Add this line
8
9     #Registro del modelo
10    class Meta:
11        model = Ticket

```

```

12     fields = ("user", "date_of_visit", "ticket_id")
13     read_only_fields = ("ticket_id",)
14
15     def get_user(self, obj): # Add this method
16         return obj.user.username # Or obj.user.email

```

Listing 7.4: Transformación de modelos de datos complejos en tipos de datos Python nativos y viceversa, facilitando la creación de API REST (serializers.py)

Este código utiliza el Django REST framework, que es un módulo potente y flexible para construir APIs en Django. En este caso, se define un serializador para el modelo ‘Ticket’ que se utiliza para convertir los datos del modelo a un formato JSON para su manipulación o visualización.

1. **Importación de librerías:** Aquí se importan las bibliotecas necesarias para el código. El módulo ‘serializers’ del Django REST framework se utiliza para convertir los objetos de modelo a JSON y viceversa.
2. **Clase Serializer:** Esta clase es un serializador del modelo definido (en este ejemplo ‘Ticket’) que hereda de ‘serializers.ModelSerializer’. Los ModelSerializers son una forma corta de crear un serializador que maneje los campos más comunes de un objeto de modelo.
3. **‘class Meta:’** Aquí se definen metadatos para el serializador. En concreto, se especifica el modelo que se va a serializar (‘Ticket’) y los campos del modelo que se incluirán en la representación serializada. Los ‘read_only_fields’ son campos que solo serán utilizados para la representación de la salida y no serán utilizados para la deserialización de los datos de entrada.
4. **def get_user(self, obj):** Este método se utiliza para obtener el valor del campo ‘user’ en la representación serializada. Aquí, se devuelve el nombre de usuario del usuario asociado al ticket. Este método es necesario porque se ha definido ‘user’ como ‘SerializerMethodField’.

Este código define un serializador que se puede utilizar para convertir los objetos de un modelo definido en los `models.py` a un JSON y viceversa, con algunas personalizaciones para cómo se maneja el campo ‘user’.

La única excepción que cabe resaltar en el proyecto con los serializers.py es la autenticación de los clientes del parque de atracciones, en las cuales se ha tenido que crear serializadores únicos con el fin de poder generar los usuarios en las bases de datos, y actualizar su información en caso de que se desee.

```

1 from rest_framework import serializers
2 from django.contrib.auth import get_user_model
3
4
5 CustomUser = get_user_model()
6
7 class UserUpdateSerializer(serializers.ModelSerializer):
8     class Meta:
9         model = CustomUser
10        fields = (
11            "first_name",
12            "last_name",
13            "username",
14            "email",

```

```

15         "dob",
16         "additional_guests",
17     )
18
19     def update(self, instance, validated_data):
20         instance.first_name = validated_data.get("first_name", instance.first_name)
21         instance.last_name = validated_data.get("last_name", instance.last_name)
22         instance.username = validated_data.get("username", instance.username)
23         instance.email = validated_data.get("email", instance.email)
24         instance.dob = validated_data.get("dob", instance.dob)
25         instance.additional_guests = validated_data.get(
26             "additional_guests", instance.additional_guests
27         )
28         instance.save()
29         return instance
30
31     def validate_username(self, value):
32         if self.instance:
33             if (
34                 CustomUser.objects.filter(username=value)
35                 .exclude(pk=self.instance.pk)
36                 .exists()
37             ):
38                 raise serializers.ValidationError("Username already exists")
39             else:
40                 if CustomUser.objects.filter(username=value).exists():
41                     raise serializers.ValidationError("Username already exists")
42         return value
43
44     def validate_email(self, value):
45         if self.instance:
46             if (
47                 CustomUser.objects.filter(email=value)
48                 .exclude(pk=self.instance.pk)
49                 .exists()
50             ):
51                 raise serializers.ValidationError("Email already exists")
52             else:
53                 if CustomUser.objects.filter(email=value).exists():
54                     raise serializers.ValidationError("Email already exists")
55         return value

```

Listing 7.5: Definición de las rutas URL de una aplicación, vinculando cada ruta con su correspondiente función de vista (urls.py)

En la clase *UserSerializer* se crea un nuevo usuario. Se puede observar el uso de *extra_kwargs*, el cual se utiliza para proporcionar argumentos adicionales a un campo específico. En este caso se establece que el campo *"password"* es de escritura. Indicando que se utilizará en la creación de un usuario, pero no se presentará en el JSON de salida por términos de seguridad. De la misma manera, analizando el método *create* para crear un nuevo objeto de usuario a partir de los datos validados. Es en donde se asignan los campos y se guarda el nuevo usuario en la base de datos.

Lo último por resaltar son los métodos *validate_username* y *validate_email* los cuales son validadores personalizados. En estos, cada vez que se actualiza el nombre de usuario o el correo electrónico de un cliente, estos verifican que el nuevo valor no esté ya en uso por otro usuario para hacer usuarios y correos duplicados una imposibilidad. Si ocurre, se lanza una excepción de validación.

7.4.3. admin.py

Admin.py proporciona una interfaz de administración web lista para usar. Django incluye un sistema de administración integrado que permite a los desarrolladores y administradores del sitio web gestionar el contenido y las cuentas de usuario. La interfaz de administración se genera dinámicamente a partir de los modelos registrados en el archivo admin.py.

Django incluye una interfaz de administración generada automáticamente y lista para usar. Es una herramienta muy potente y es una de las características que distingue a Django de muchos otros frameworks web. Aunque está principalmente destinada a uso interno y administrativo, se puede utilizar para todo tipo de tareas, lo cual conviene para la gestión de información de la aplicación del cliente y para los empleados que manejen tiendas, restaurantes o atracciones.

El administrador de Django se encuentra en el módulo `django.contrib.admin` y su configuración se encuentra en el archivo `admin.py` de cada aplicación.

Para hacer que un model esté disponible en la interfaz de administración, se debe de registrar dicho modelo en el archivo `admin.py` de la aplicación. Una vez que un modelo está registrado, Django automáticamente generará una interfaz de administrador para él.

Por ejemplo, en el caso de los clientes del parque de atracciones, el modelo `CustomUser` se registraría de la siguiente manera:

1. Se navega al microservicio correspondiente, en este caso `clientApp`
2. Se navega al archivo `restaurantApp.py`
3. Se abre dicho archivo y se registra de la siguiente manera:

```

1 from django.contrib import admin
2 from .models import Restaurant
3
4
5 class RestaurantAdmin(admin.ModelAdmin):
6     list_display = ('restaurant_id', 'name', 'park',
7                     'area', 'opening_hour', 'closing_hour')
8     search_fields = ('name', 'restaurant_types')
9     list_filter = ('park', 'area', 'restaurant_types',
10                   'opening_hour', 'closing_hour')
11
12
13 admin.site.register(Restaurant, RestaurantAdmin)
14
15
16 class RestaurantEmployeeArea(admin.AdminSite):
17     site_header = 'Restaurant Employee Area'
18
19
20 restaurant_employee_admin_site = RestaurantEmployeeArea(
21     name="Restaurant Admin")
22 restaurant_employee_admin_site.register(Restaurant, RestaurantAdmin)

```

Listing 7.6: Registración de usuario en el dashboard de Administrador Django

7.4.4. urls.py

Urls.py define las rutas URL para la aplicación. En otras palabras, determina cómo se asignan las URL a las vistas. Django extrae la URL solicitada por el usuario y la dirige a la vista correspondiente a través de las expresiones regulares definidas en urls.py.

En una aplicación web desarrollada con Django, el archivo ‘urls.py’ desempeña un papel crucial en la definición de la interfaz de usuario. Este archivo es responsable de mapear las distintas URL a su correspondiente funcionalidad en la aplicación, conectando de esta forma la interfaz web con el back-end de la aplicación. Esencialmente, ‘urls.py’ define el esquema de URL de la aplicación, conectando cada ruta de URL con su correspondiente vista.

Las URL se definen en Django a través de patrones de expresiones regulares o, a partir de la versión 2.0, a través de un sistema de ‘rutas’ más sencillo y fácil de leer. Cada patrón de URL se asocia con una función de vista, que es un procedimiento de Python que toma una solicitud web y produce una respuesta. Esta respuesta puede ser la entrega de un documento HTML, un redireccionamiento, un documento XML, una imagen, o prácticamente cualquier cosa.

Tienen la sintaxis que se observa a continuación:

```

1 from django.urls import path
2 from .views import (
3     TicketCreateView,
4     TicketValidationView,
5     book_visit,
6     login_view,
7     logout_view,
8     UserTicketsListAPIView,
9 )
10
11 urlpatterns = [
12     path("tickets/", TicketCreateView.as_view(), name="create_ticket"),
13     path("tickets/validate/", TicketValidationView.as_view(), name="validate_ticket"),
14     path("book_visit/", book_visit, name="book_visit"),
15     path("login/", login_view, name="login"),
16     path("logout/", logout_view, name="logout"),
17     path("tickets-view/", UserTicketsListAPIView.as_view(), name="user-tickets"),
18 ]

```

Listing 7.7: Archivo urls.py

Este archivo proporciona la configuración de las rutas de la API y asocia cada ruta a la vista correspondiente de Django, la cual se encuentra en el archivo *views.py*. Adentro de cada *urlpatterns*, se define una ruta y la vista asociada a ella.

La arquitectura back-end de esta aplicación está diseñada primordialmente para proporcionar funcionalidades de API en lugar de servir como una aplicación web completa. Aunque hay alguna funcionalidad que se encarga de la venta de billetes, las vistas basadas en funciones (FBV) son casi inexistentes. En su lugar, se hace un uso predominante de las Vistas Basadas en Clases (CBV). Las CBV, en este contexto, se utilizan principalmente para realizar operaciones en la base de datos y responder con información relevante a las peticiones del cliente. Por lo tanto, la esencia de este back-end reside en su capacidad para procesar y gestionar solicitudes de API a través de estas CBV.

7.4.5. views.py

Views.py define las vistas que responden a las solicitudes de los usuarios. Una vista puede ser cualquier función de Python que tome un objeto de solicitud web y devuelva una respuesta web. Las vistas pueden renderizar plantillas, redirigir, mostrar errores, procesar formularios y realizar otras tareas.

En términos del flujo de datos, cuando se realiza una solicitud a la aplicación Django, Django primero consulta `urls.py` para determinar a qué vista debe dirigirse la solicitud. La vista, a su vez, interactúa con el modelo correspondiente para recuperar o manipular los datos necesarios. Los datos del modelo se pasan a un serializer para convertirlos en un formato apto para la renderización o transmisión. La vista puede entonces usar estos datos para renderizar una plantilla o formar una respuesta API. La respuesta es entonces enviada de vuelta al usuario.

views.py tiene indicio a ser el documento más extenso del back-end, ya que es en el cual se ejecutan todas las funciones para tratar con los datos recibidos por el usuario en la parte del servidor.

En cuanto a la vinculación con las vistas, Django puede trabajar con funciones de vista basadas en vistas de función (FBVs) y vistas basadas en clases (CBVs).

1. **Vistas de función (FBVs):** Son funciones de Python que toman una solicitud web y devuelven una respuesta web. Las vistas de función son fáciles de leer y entender. Un programador que sepa un poco de Python y entienda el protocolo HTTP puede entender qué hace una vista de función.
2. **Vistas basadas en clases (CBVs):** Las vistas basadas en clases son una forma alternativa de implementar vistas como objetos Python en lugar de funciones. Proporcionan una forma de reutilizar el código común y la funcionalidad, permitiendo la herencia y la mezcla (mixins) de clases. Por lo tanto, son especialmente útiles cuando se necesita compartir código entre varias vistas.

Se observan y se explican los siguientes ejemplos para construir una imagen sobre como se desarrollan los siguientes archivos:

1. Vistas para obtener la información de un modelo usando la REST API

```

1 from rest_framework import generics
2 from .models import ParkEmployee
3 from .serializers import ParkEmployeeSerializer
4
5 class ParkEmployeeListCreateView(generics.ListCreateAPIView):
6     queryset = ParkEmployee.objects.all()
7     serializer_class = ParkEmployeeSerializer
8
9 class ParkEmployeeRetrieveUpdateDestroyView(generics.RetrieveUpdateDestroyAPIView):
10     queryset = ParkEmployee.objects.all()
11     serializer_class = ParkEmployeeSerializer

```

Listing 7.8: Definición de las vistas o funciones que reciben en una solicitud web, y devolviendo una respuesta, sirviendo como un puente entre los modelos y las plantillas en el framework

El archivo tiene la siguiente estructura:

- a) **Importación de librerías, modelos y serializadores:** En el archivo se detalla el uso del módulo *generics* del framework REST de django. Este módulo contiene un número de CBV's que se usan rápidamente para crear vistas API para varias operaciones, como la creación y el listado de objetos. Igualmente, se ha importado el modelo relevante *ParkEmployee* con el cual va a tratar el archivo junto con su serializador correspondiente, cuya estructura se observa en la sección anterior.
- b) **Clases CreateView y UpdateDestroyView:** Se observan que estas dos clases hacen uso de vista que heredan de `ListCreateAPIView` y `RetrieveUpdateDestroyAPIView` respectivamente. Estas vistas proporcionan de manera automática controladores para solicitudes GET, PUT, PATCH y DELETE. Los cuales recuperan, actualizan y eliminan una instancia del modelo respectivamente.

2. Tratamiento de datos recibidos por el usuario mediante la API:

```

1 CustomUser = get_user_model()
2
3 # Sign Up
4 @api_view(["POST"])
5 @permission_classes([AllowAny])
6 def signup(request):
7     serializer = UserSerializer(data=request.data)
8     if serializer.is_valid():
9         user = serializer.save()
10        token = Token.objects.get(user=user)
11        return Response({"token": token.key}, status=status.HTTP_201_CREATED)
12    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

```

Listing 7.9: Menjo de vistas para las llamadas API y autenticación (views.py)

El archivo sigue la siguiente estructura:

- a) En primer lugar, se obtiene el modelo del usuario a través de la función `get_user_model()`. Esto es útil porque Django permite personalizar el modelo del usuario, por lo que es una buena práctica usar esta función en vez de referirse directamente al modelo `User`.
- b) A continuación, se define una función de vista llamada `signup`, diseñada para manejar las solicitudes POST, que se usan para enviar datos a un servidor para su creación.
- c) La decoración `@api_view(["POST"])` es proporcionada por la framework REST de Django (DRF), indicando que esta vista solo manejará las solicitudes POST. En caso de recibir otro tipo de solicitud (como GET o PUT), DRF responderá automáticamente con un error 405 (Método no permitido).
- d) La anotación `@permission_classes([AllowAny])`, suministrada por DRF, establece las políticas de permisos para esta vista. El uso de `AllowAny` indica que cualquier usuario, ya sea autenticado o no, tiene permiso para realizar una solicitud a esta vista. En ausencia de este decorador, al efectuar la llamada a esta vista, sería obligatorio incluir un Token en la cabecera de la solicitud para garantizar el acceso a la vista en cuestión.

- e) Finalmente, una vez se tiene acceso a la vista, se puede ver que se hace uso del serializador importado para poder verificar la validez de los datos nuevos del usuario y registrarlos correspondientemente en nuestra base de datos, pasando el formato JSON a un modelo Python. Si es exitoso en el procedimiento se devolverá un `HTTP_201_Created` y en caso contrario un `HTTP_400_BAD_REQUEST`

Aunque exista una gran variedad de métodos de vista en los archivos `views.py` del proyecto, se ha optado únicamente por desarrollar estos dos ejemplos para darle una imagen al lector sobre el desarrollo de estos archivos. En caso de que se desee conocer más, se puede ver la funcionalidad completa de cada microservicio en los diagramas de flujo de la sección anterior, o en el código fuente del proyecto en [GitHub](#).

7.4.6. SQLite y Django Admin

Django se integra fácilmente con SQLite a través de su Object-Relational Mapping (ORM), una técnica de programación para convertir datos entre el sistema de tipos utilizados en un lenguaje de programación orientado a objetos y la base de datos relacional. El ORM de Django maneja la conversión de modelos de Django en tablas de base de datos y viceversa, permitiendo la manipulación de los datos de la base de datos a través de objetos de Python.

Cuando se crea un nuevo proyecto de Django, se configura automáticamente para usar SQLite como base de datos por defecto. Esta configuración se puede encontrar en el archivo `settings.py` del proyecto. Si bien SQLite es muy útil para el desarrollo y las pruebas debido a su simplicidad y facilidad de configuración, puede no ideal para aplicaciones en producción que requieren un alto rendimiento y la capacidad de manejar muchas conexiones simultáneas.

El Administrador de Django

El dashboard de administrador de Django, también conocido como Django Admin, es una de las características más poderosas y notables de Django. Este sistema administrativo es capaz de manejar las operaciones CRUD (crear, leer, actualizar y eliminar) sobre las bases de datos definidas en la aplicación.

Cuando se definen los modelos en Django, se puede automáticamente construir una interfaz de administrador a partir de estos modelos. Django Admin es altamente personalizable y se puede especificar, por ejemplo, qué campos mostrar en el listado, qué campos utilizar como filtros, cómo ordenar los resultados, entre otros, tal y como se ha visto en la subsección de archivos `admin.py`.

Las funcionalidades esenciales del dashboard de administración son las siguientes:

1. **Administración de datos:** Se puede crear, leer, actualizar y eliminar los registros de tus modelos directamente desde la interfaz de administrador sin necesidad de escribir una línea de código.

¹<https://github.com/SaltyYagi123/VirtualQ>

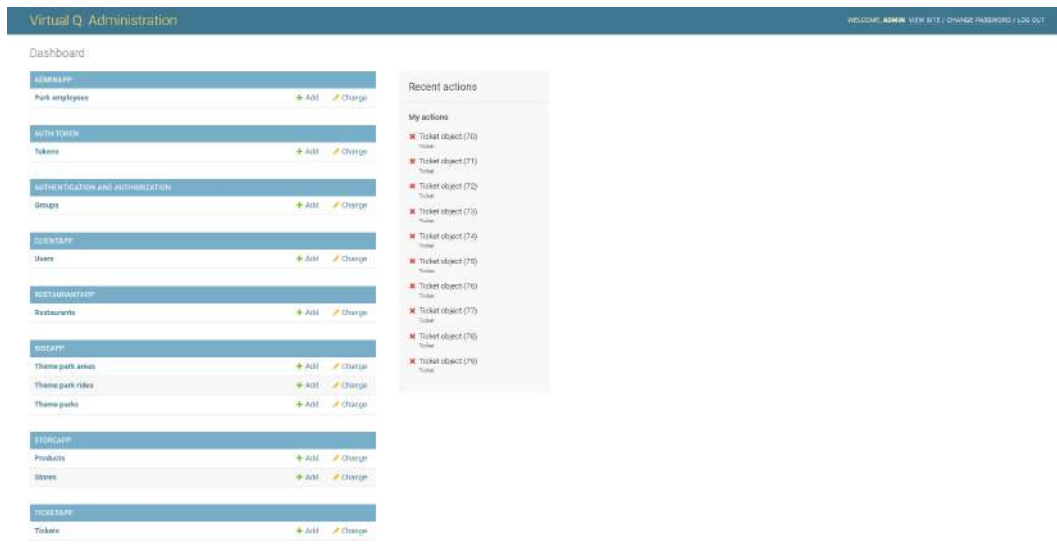


Figura 7.9: Homescreen del administrador

2. **Personalización:** El panel de administrador es altamente personalizable. Puedes definir cómo se deben mostrar los datos, qué campos deben ser editables, qué campos deben ser mostrados en el listado, entre otras cosas.
3. **Autenticación y autorización:** Django Admin cuenta con un sistema de autenticación y autorización integrado. Esto permite que solo los usuarios autorizados puedan acceder al panel de administrador y que estos tengan diferentes niveles de acceso según sus permisos.
4. **Búsqueda:** Django Admin incluye una funcionalidad de búsqueda que te permite buscar registros de manera rápida y eficiente.
5. **Filtrado:** Puedes agregar filtros que ayudan a encontrar y organizar los registros de manera más eficiente.
6. **Relaciones de datos:** Django Admin maneja automáticamente las relaciones entre diferentes modelos. Si tienes modelos que están relacionados entre sí (por ejemplo, un modelo de Autor y un modelo de Libro), puedes agregar, editar y eliminar libros desde la página de detalles del autor.
7. **Exportación de datos:** Django Admin te permite exportar los datos de tus modelos a diferentes formatos como CSV, Excel, entre otros.

7.5. Vista del Administrador

A continuación se observarán dos ejemplos que administra únicamente el administrador.

7.5.1. Añadir empleados al parque

1. Se navega a la base de datos correspondiente:

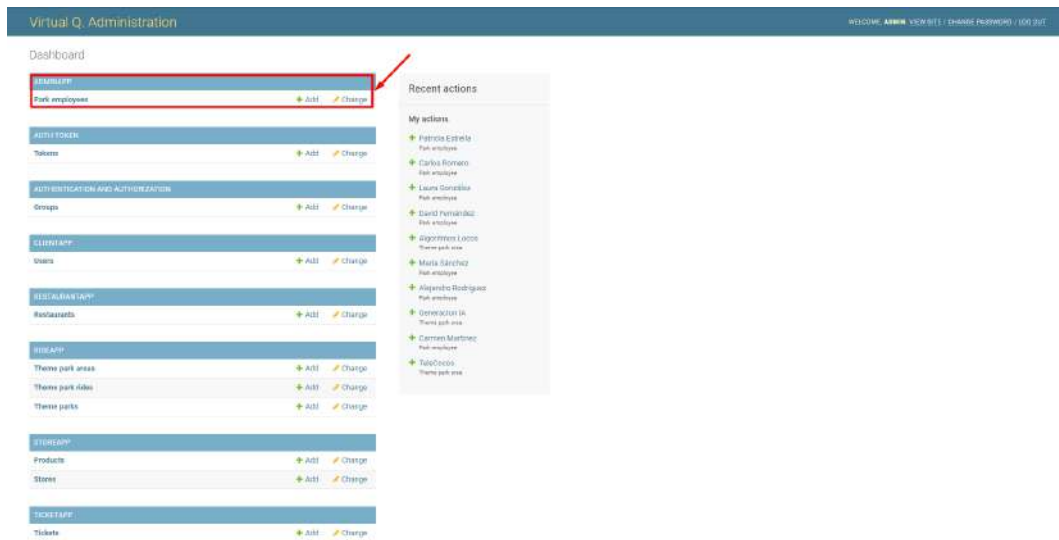


Figura 7.10: Selección de base de datos a modificar

2. Se observan los registros actuales que se tienen en la base de datos, organizados por columnas (Nombre, Apellido, Título, en que parque y en que área trabajan. . .) y usando el panel de filtro se pueden filtrar a los empleados por sus propiedades, o se puede usar la barra de búsqueda para encontrarlos de manera directa:

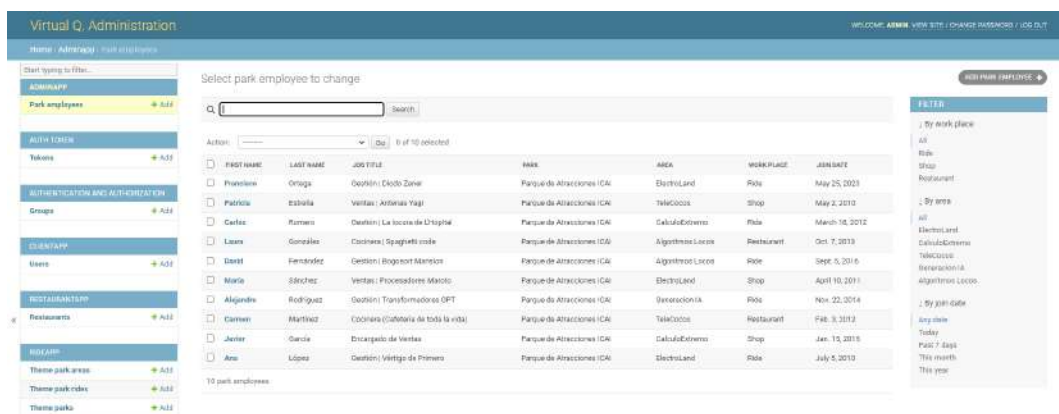


Figura 7.11: Visualización de la base de datos de empleados

3. Tras hacer clic en añadir a un nuevo empleado aparece la siguiente vista en la que se pueden rellenar los campos de la siguiente manera:

The screenshot shows the 'Virtual Q. Administration' interface. On the left is a sidebar with a search bar and a list of categories: ADMINSTAFF, AUTH-TICKET, TICKETS, AUTHENTICATION AND AUTHORIZATION, CLIENTSTAFF, RESTAURANTSTAFF, BGCSTAFF, STORESTAFF, and TICKETSTAFF. The main area is titled 'Add park employee'. The form contains the following fields: First name (Francisco), Last name (Ortega), Date of birth (1985-02-05), Join date (2023-05-25), Shift start (Now), Shift end (Now), Gender (Male), Park (Parque de Atracciones), Area (Electroland), Work place (RdA), Job title (Cajón / Diodo Zener), Email (francisco.ortega@q1a.es), and Phone number (+34 905 123 456). A calendar and time picker are open over the 'Date of birth' field.

Figura 7.12: Relleno de datos de empleados mediante un formulario

- Se observa que rellenar estos campos también contiene una validación de relleno, tal que el administrador rellene los campos de la manera correcta.

The screenshot shows the 'Virtual Q. Administration' interface with the 'Add park employee' form. The form is filled with the same data as in Figure 7.12, but with validation errors. Red boxes highlight the following errors: 'Please correct the errors below' at the top, 'Enter a valid date' for the birth date, 'Enter a valid date' for the join date, 'Enter a valid time' for the shift start, 'Enter a valid time' for the shift end, 'This field is required' for the gender, 'This field is required' for the park, 'This field is required' for the area, and 'This field is required' for the work place.

Figura 7.13: Validación de datos insertados

Una vez los datos se hayan insertado correctamente, al hacer clic en el botón **SAVE**, estos cambios se verán reflejados tanto en la base de datos, como en la página web del administrador.

7.5.2. Añadir atracciones al parque

- Se navega al microservicio correspondiente:

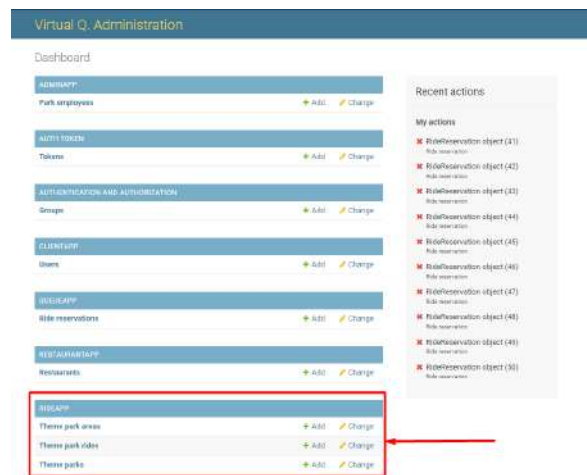


Figura 7.14: Selección de la base de datos a modificar

- Tras hacer clic en **Theme Park Rides**, se obtiene el siguiente dashboard, el cual contiene todas las atracciones que se tienen registradas hasta ahora, la cual se pueden editar en tiempo real. Se puede observar a mano derecha los siguientes filtros que se solicitó en el archivo `admin.py`

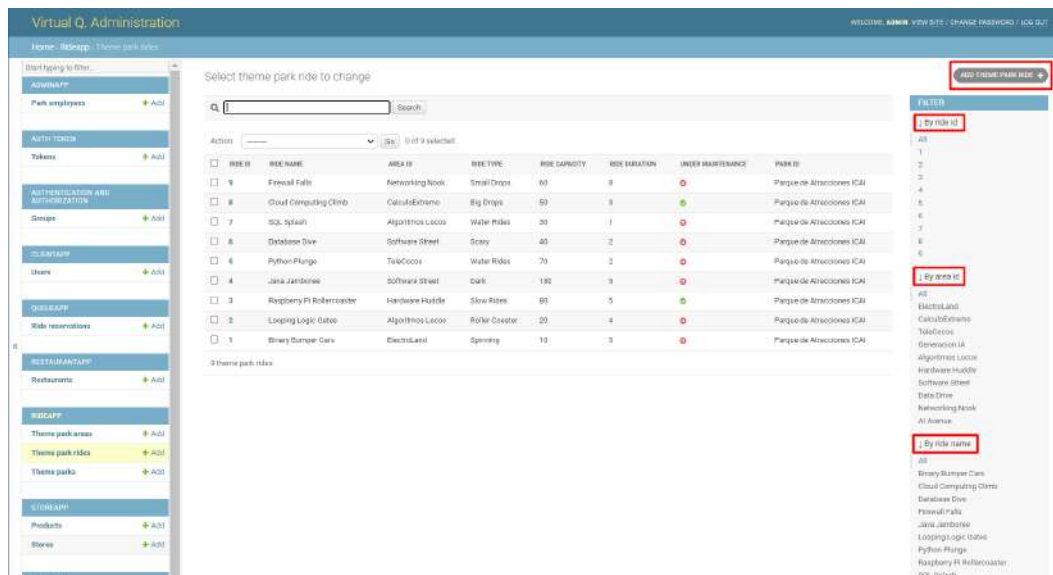


Figura 7.15: Dashboard Atracciones

- Si se hace clic en una de las atracciones existentes, se observa que todos los campos son editables y de varios tipos. Una vez se hayan hecho los cambios deseados, estos se verán reflejados en la aplicación front-end correspondiente.

The screenshot shows the 'Virtual Q. Administration' interface. On the left is a sidebar menu with categories like ADMINAPP, AUTH TOKEN, AUTHENTICATION AND AUTHORIZATION, CLIENTAPP, QUEUEAPP, RESTAURANTAPP, RIDEAPP, STOREAPP, and TICKETAPP. The 'RIDEAPP' section is expanded, showing 'Theme park rides' as the active item. The main content area is titled 'Change theme park ride' and shows the details for 'Firewall Falls'. The form includes fields for 'Ride name', 'Ride description', 'Ride thumbnail', 'Area id' (set to 'Networking Nook'), 'Park id' (set to 'Parque de Atracciones ICAI'), 'Height restriction' (35), 'Ride capacity' (60), 'Ride duration' (8), 'Opening hour' (10:30:00), and 'Closing hour' (19:30:00). There are also checkboxes for 'Under maintenance', 'Accessibility wheelchair access', and 'Accessibility audio description'.

Figura 7.16: Edición de datos del dashboard de atracciones

En el caso de campos relacionales, como lo son en esta base de datos el ID del Área y el ID del Parque, se puede observar tres iconos a la derecha de las pestañas de valores:



Figura 7.17: 3 iconos para bases de datos relacionadas

Las cuales tienen la siguiente función:

- *Icono lápiz*: Se abre una nueva pestaña para modificar los valores actuales que tiene la entrada de la base de datos seleccionada.
- *Icono suma*: Se abre una nueva pestaña en la que se puede añadir una nueva entrada en la base de datos correspondiente. Guardarla, y tras eso asignarla al campo actual que se estaba rellenando.
- *Icono ojo*: Se redirige el navegador a la vista de la base de datos correspondiente.

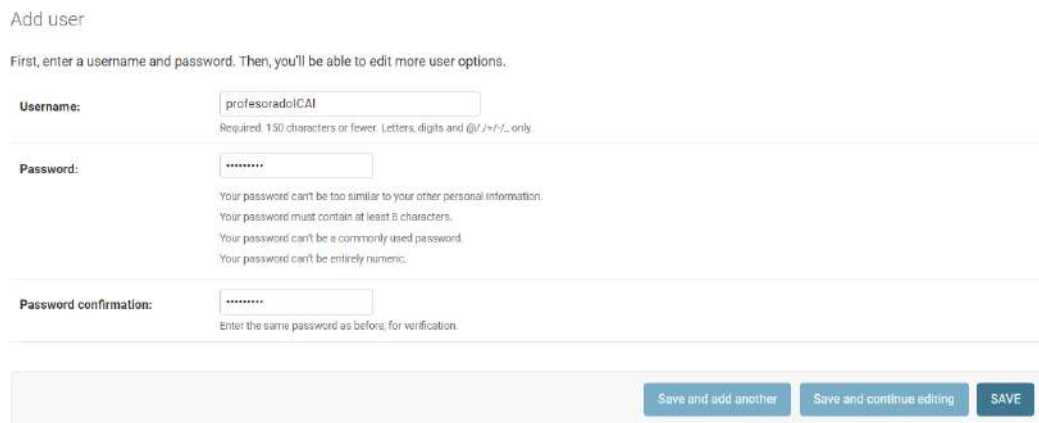
Estos dos ejemplos destacan la gran importancia de la página web del administrador para gestionar la entrada, edición y visualización de la base de datos completa del proyecto. Todos los cambios hechos en la base de datos por el administrador se reflejarán en el sistema, y mediante llamadas de la REST API, se podrá visualizar estos cambios en tiempo real.

Añadir usuarios, empleados y administradores

Una de las funciones más cruciales de esta interfaz es la gestión de usuarios, lo que permite la creación, edición y manejo de permisos de los usuarios.

Para crear un nuevo usuario en Django a través de la interfaz de administración, debes seguir los siguientes pasos:

1. *Navegación al Panel de Usuarios:* Una vez se haya iniciado sesión, será redirigido a la página principal del administrador de Django. Aquí, busca y haz clic en la opción 'Users' bajo el encabezado 'clientApp'.
2. *Creación de un Nuevo Usuario:* En la página de Usuarios, encontrarás un botón llamado 'Add User' en la parte superior derecha de la página. Al hacer clic en este botón, serás llevado a la página de creación de usuarios.
3. *Introducción de Datos:* Aquí, podrás introducir el nombre de usuario y la contraseña para el nuevo usuario. Es importante que la contraseña sea segura y única. Tras proporcionar estos datos, haz clic en 'SAVE'.



The screenshot shows the 'Add user' form in the Django Admin interface. At the top, it says 'Add user' and 'First, enter a username and password. Then, you'll be able to edit more user options.' The form has three main sections: 'Username:' with a text input containing 'profesoradoICA' and a note 'Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.'; 'Password:' with a password input, a strength indicator, and several error messages: 'Your password can't be too similar to your other personal information.', 'Your password must contain at least 8 characters.', 'Your password can't be a commonly used password.', and 'Your password can't be entirely numeric.'; and 'Password confirmation:' with another password input and the note 'Enter the same password as before, for verification.' At the bottom right, there are three buttons: 'Save and add another', 'Save and continue editing', and 'SAVE'.

Figura 7.18: Creación de un nuevo usuario

Edición de Datos de Usuario y Gestión de Permisos

Tras la creación de un usuario, es posible que se necesite editar los detalles o administrar los permisos. Esta gestión está rápidamente facilitada por Django con las siguientes funciones:

- **Edición de Datos de Usuario:** Para editar los detalles de un usuario, navega a la página de Usuarios como antes, y haz clic en el nombre de usuario del usuario que deseas editar. Se abrirá una nueva página con el formulario de edición de

usuarios, donde podrás modificar los detalles del usuario. No olvides hacer clic en 'SAVE' una vez hayas hecho tus cambios.

The screenshot shows the Django user editing interface. At the top, a green message states: "The user 'profesoradoICAI' was added successfully. You may edit it again below." Below this is the "Change user" section for the user "profesoradoICAI". It includes fields for "Username" (profesoradoICAI) and "Password" (algorithm: pbkdf2_sha256 iterations: 390000 salt: wNITZg***** hash: oOcyrm*****). A "Personal info" section contains fields for "First name" (Profesorado), "Last name" (ICAI), and "Email address" (profesorado@icai.com). A "HISTORY" button is visible in the top right.

Figura 7.19: Edición de datos del usuario

- **Gestión de Permisos de Usuario:** En el mismo formulario de edición de usuarios, puedes administrar los permisos del usuario. Django permite asignar permisos específicos a cada usuario, además de la capacidad de asignar grupos de permisos predefinidos. Puedes añadir o eliminar permisos individuales en la sección 'Permisos de usuario' o asignar al usuario a uno o varios grupos en la sección 'Grupos'.

The screenshot shows the "Permissions" section of the Django user editing form. It includes three checkboxes: "Active" (checked), "Staff status" (unchecked), and "Superuser status" (unchecked). Below these is the "User permissions:" section, which contains two panels: "Available user permissions" and "Chosen user permissions". The "Available user permissions" panel lists various permissions such as "admin | log entry | Can add log entry", "admin | log entry | Can change log entry", "admin | log entry | Can delete log entry", "admin | log entry | Can view log entry", "adminApp | park employee | Can add park employee", "adminApp | park employee | Can change park employee", "adminApp | park employee | Can delete park employee", "adminApp | park employee | Can view park employee", "auth | group | Can add group", "auth | group | Can change group", "auth | group | Can delete group", "auth | group | Can view group", and "auth | permission | Can add permission". The "Chosen user permissions" panel is currently empty. A "Choose all" button is at the bottom left, and a "Remove all" button is at the bottom right. A red box highlights the "Add" button in the "Available user permissions" panel.

Figura 7.20: Manejo de permisos del usuario

Finalmente, debido a la configuración de nuestra aplicación y el uso de Autenticación mediante tokens, por cada usuario nuevo creado, tanto visitante como administrador, se les otorgará un **Token**, el cual se puede observar en la siguiente pestaña de la página.



Figura 7.21: Vista de Tokens de los usuarios generados

Como se puede observar en esta descripción extensa, mediante una interfaz intuitiva, el administrador puede acceder de forma sencilla y práctica a todas las bases de datos de los microservicios relevantes, tanto para el sistema como para los usuarios durante su visita al parque de atracciones.

7.6. API

Django es reconocido por su robusta capacidad para construir interfaces de programación de aplicaciones (API) eficientes y seguras. La creación de una API en Django implica una serie de pasos que incluyen la configuración de modelos, vistas, serializadores y rutas. A continuación, se detalla la funcionalidad de la API de Django y cómo se desarrolla.

Modelos y la Base de datos

El primer paso en la creación de una API en Django implica la definición de los modelos. Los modelos son la representación de la estructura de la base de datos y definen la naturaleza de los datos que se almacenarán. Cada modelo corresponde a una tabla en la base de datos y cada atributo del modelo se traduce en un campo en la tabla de la base de datos.

Una vez definidos los modelos, es necesario realizar las migraciones. Las migraciones son la manera en la cual Django propaga los cambios que haces en tus modelos (como agregar un campo, eliminar un modelo, ...) en la base de datos.

Una vez definidos los modelos, es necesario realizar las migraciones. Las migraciones son cómo Django propaga los cambios que haces en tus modelos (como agregar un campo, eliminar un modelo, etc.) en tu base de datos. El comando `python manage.py makemigrations` genera los scripts de migración y el comando `python manage.py migrate` aplica estos cambios en la base de datos.

```

1 class CustomUser(AbstractUser):
2     id = models.AutoField(primary_key=True)
3     email = models.EmailField("email address", unique=True, blank=False, null=False)
4     name = models.CharField(max_length=30, blank=False, null=False, default="Undefined")
5     last_name = models.CharField(max_length=150, blank=False, null=False)
6     dob = models.DateField(null=True, blank=True)
7     height = models.IntegerField(null=True, blank=True)
8
9

```

```

10 @receiver(post_save, sender=settings.AUTH_USER_MODEL)
11 def create_auth_token(sender, instance=None, created=False, **kwargs):
12     if created:
13         Token.objects.create(user=instance)

```

Listing 7.10: Archivo models.py ejemplar de un usuario

Serializadores

Los serializadores en Django permiten transformar los datos complejos de los modelos en tipos de datos Python nativos que pueden ser fácilmente renderizados en formatos como JSON o XML para su uso en la API. Por otro lado, los serializadores también validan los datos recibidos en una petición para su conversión en un formato compatible con el modelo antes de almacenarlo en la base de datos.

```

1 from rest_framework import serializers
2 from django.contrib.auth import get_user_model
3
4 CustomUser = get_user_model()
5
6 class UserSerializer(serializers.ModelSerializer):
7     class Meta:
8         model = CustomUser
9         fields = ("id", "username", "email", "password", "first_name", "last_name", "
10             dob", "additional_guests") # Include first_name and last_name here
11         extra_kwargs = {"password": {"write_only": True}}
12
13     def create(self, validated_data):
14         user = CustomUser.objects.create(
15             username=validated_data["username"],
16             email=validated_data["email"],
17             first_name=validated_data.get("first_name", ""),
18             last_name=validated_data.get("last_name", ""),
19             dob=validated_data["dob"],
20             additional_guests=validated_data["additional_guests"],
21         )
22         user.set_password(validated_data["password"])
23         user.save()
24         return user

```

Listing 7.11: Serializers.py, convierte los datos complejos en datos nativos de Python que pueden ser renderizados en JSON

Vistas y Controladores

Django maneja las solicitudes a la API a través de vistas. En cada vista, se define qué acción se tomará en respuesta a una solicitud HTTP particular (GET, POST, PUT, DELETE, etc.). Aquí es donde los serializadores se utilizan para transformar los datos de los modelos en una respuesta que se pueda enviar al cliente, o para validar los datos entrantes en una petición antes de almacenarla en la base de datos.

```

1 CustomUser = get_user_model()
2
3 # Sign Up
4 @api_view(["POST"])
5 @permission_classes([AllowAny])
6 def signup(request):
7     serializer = UserSerializer(data=request.data)
8     if serializer.is_valid():
9         user = serializer.save()

```

```

10     token = Token.objects.get(user=user)
11     return Response({"token": token.key}, status=status.HTTP_201_CREATED)
12     return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
13
14 # Get User Info
15 @api_view(["GET"])
16 @permission_classes([IsAuthenticated])
17 def get_user_info(request):
18     user = request.user
19     data = {
20         "id": user.id,
21         "email": user.email,
22         "name": user.name,
23         "last_name": user.last_name,
24         "username": user.username,
25         "dob": user.dob, # uncomment this if your user model has a date_of_birth
                           field
26     }
27     return Response(data)

```

Listing 7.12: Archivo views.py encargado de la manipulación de datos para cada microservicio.

Enrutamiento

Finalmente, las rutas en Django definen cómo se mapean las URL a las vistas. En un archivo de rutas, se define una URL para cada vista que se puede utilizar para realizar solicitudes a la API. El cual se puede ver que tras el `urls.py` del proyecto y los cuales se encuentran en nuestros microservicios construyen el link completo de acceso.

Para demostrar como se construye un url para acceder a cualquier función del back-end, se analizará el archivo base `urls.py` y el archivo correspondiente de un microservicio:

- **Archivo `urls.py` del proyecto:** En la raíz del proyecto Django, se encuentra un archivo llamado `urls.py`. Este es el archivo de enrutamiento principal del proyecto. Cuando Django recibe una solicitud HTTP, primero mira este archivo para determinar a dónde dirigir la solicitud.

Este archivo `urls.py` principal debería contener una lista de `urlpatterns`. Cada `urlpatterns` es una instancia de `path()` o `re_path()`, ambas funcionan para conectar una URL específica con la vista correspondiente. Sin embargo, en lugar de conectar directamente con todas las vistas de todas las aplicaciones, este archivo `urls.py` principal suele incluir los `urls.py` de las aplicaciones individuales.

Para conectar los archivos `urls.py` del proyecto y las aplicaciones, Django utiliza la función `include()`. Dentro de los `urlpatterns` en el archivo `urls.py` del proyecto, se puede usar `include()` en una instancia de `path()` para referenciar al archivo `urls.py` de una aplicación.

Por ejemplo, el microservicio de atracciones llamado `'rideApp'`, incluye una línea en el archivo `urls.py` base del proyecto:

```

1 from django.contrib import admin
2 from django.urls import path, include
3 from rideApp.admin import ride_employee_admin_site
4 from restaurantApp.admin import restaurant_employee_admin_site
5 from django.conf import settings

```

```

6 | from django.conf.urls.static import static
7 |
8 | urlpatterns = [
9 |     path("admin/", admin.site.urls),
10 |     path("api/restaurantEmployee/", restaurant_employee_admin_site.urls),
11 |     path("api/rideEmployee/", ride_employee_admin_site.urls),
12 |     path("api/parkRides/", include("rideApp.urls")),
13 |     path("api/employees/", include("adminApp.urls")),
14 |     path("api/stores/", include("storeApp.urls")),
15 |     path("api/restaurants/", include("restaurantApp.urls")),
16 |     path("api/clients/", include("clientApp.urls")),
17 |     path("api/tickets/", include("ticketApp.urls")),
18 |     path("api/queue/", include("queueApp.urls")),
19 | ]
20 |
21 | admin.site.index_title = "Dashboard"
22 | admin.site.site_header = "Virtual Q. Administration"
23 | admin.site.site_title = "Administration Dashboard"
24 |
25 | if settings.DEBUG:
26 |     urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)

```

Listing 7.13: Archivo base del proyecto urls.py que interconecta todos los microservicios

De tal manera que ahora se tiene contruidos los URL para acceder a los enlaces relacionados con los distintos microservicios entrelazados, tal que:

127.0.0.1:8000/api/{nombre-del-microservicio} dará acceso a los URL correspondientes.

■ Archivo urls.py de las Aplicaciones

Cada microservicio tiene su propio archivo urls.py. Esto permite que las URL de cada aplicación se manejen de forma aislada, manteniendo la organización y facilitando el mantenimiento del código.

Dentro de este archivo urls.py de la aplicación, también tendrás una lista de urlpatterns. Estas son instancias de `path()` o `re_path()`, que mapean las URL específicas de la aplicación a las vistas correspondientes. Se puede observar a continuación el archivo del microservicio de clientes `clientApp`:

```

1 | from django.urls import path
2 | from django.contrib.auth import views as auth_views
3 | from .views import (
4 |     signup,
5 |     login,
6 |     ResetPasswordView,
7 |     UserUpdateView,
8 |     get_user_info,
9 | )
10 |
11 | urlpatterns = [
12 |     path("signup/", signup, name="signup"),
13 |     path("login/", login, name="login"),
14 |     path("user/", get_user_info, name="user_info"),
15 |     path("update/", UserUpdateView.as_view(), name="user_update"),
16 |     path("reset-password/", ResetPasswordView.as_view(), name="resetpassword"),
17 |     path(
18 |         "reset-password/<uidb64>/<token>/",
19 |         auth_views.PasswordResetConfirmView.as_view(
20 |             template_name="clientApp/registration/password_reset_confirm.html"
21 |         ),

```

```

22         name="password_reset_confirm",
23     ),
24     path(
25         "reset-password/done/",
26         auth_views.PasswordResetCompleteView.as_view(
27             template_name="clientApp/registration/password_reset_done.html"
28         ),
29         name="password_reset_complete",
30     ),
31 ]

```

Listing 7.14: urls.py del microservicio de clientes

- **Generación de URL Completa de una API** En el contexto de una API, esta estructura de enrutamiento permite definir una URL completa y específica para cada punto final de la API.

En el ejemplo expuesto, se tiene en la API un punto final que permite manejar propiedades del usuario, como iniciar sesión, actualizar información, obtener la información, etc. Para coger la información del usuario actual, se accederá mediante `127.0.0.1:8000/api/clients/user/`. Donde `api/clients/` viene por el `urls.py` base y `user/` del microservicio. Este último incluye una referencia a la vista que se debe de acceder para ejecutar el proceso deseado.

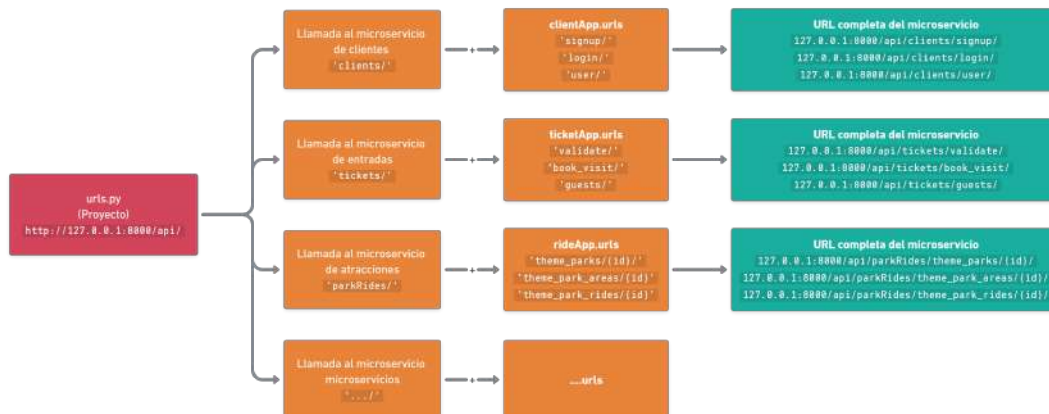


Figura 7.22: Construcción de URL para la API diseñada

7.6.1. Diagramas de Secuencia

Una forma efectiva de representar y comprender las interacciones entre microservicios, así como entre el front-end y el back-end, es a través de los diagramas de secuencia. Estos diagramas son una herramienta de modelado que describe cómo las entidades en un sistema interactúan en un orden específico para lograr una funcionalidad determinada. En nuestro caso, las entidades serían nuestros microservicios y componentes de front-end y back-end.

Los diagramas de secuencia pueden aportar varios beneficios. Por un lado, permiten visualizar la secuencia de interacciones entre los diferentes microservicios y componentes de la aplicación, proporcionando una visión clara y detallada del flujo de control y la lógica de la funcionalidad que se está implementando. Por otro lado, ayudan a identificar posibles problemas o ineficiencias en el diseño, facilitando así su optimización.

7.7. Desarrollo de front-end móvil usando React Native

Virtual Q. es la aplicación del parque de atracciones. Una plataforma digital diseñada con el objetivo de proporcionar a los usuarios una experiencia inigualable en la organización y el disfrute de su visita al parque temático.

La aplicación se ha creado con la premisa de facilitar cada aspecto de la visita. Desde la comodidad del dispositivo móvil, los usuarios pueden planificar su visita de manera eficiente y personalizada, adaptando cada detalle a sus preferencias y necesidades particulares.

Las funcionalidades clave de nuestra aplicación incluyen la posibilidad de iniciar sesión para personalizar la experiencia, adquirir entradas para el parque de atracciones y huéspedes adicionales, visualizar y seleccionar atracciones específicas basándonos en sus preferencias, y hacer reservas para dichas atracciones. Cada una de estas funcionalidades se ha desarrollado con el objetivo de proporcionar una experiencia de usuario fluida e intuitiva.

La siguiente documentación proporcionará una visión detallada de cada una de estas funcionalidades, explicando cómo los usuarios pueden interactuar con ellas y beneficiarse de su uso. Los procesos y características se explicarán en detalle, y se proporcionarán imágenes ilustrativas para facilitar la comprensión.

7.7.1. Desarrollo en React Native

React Native es un marco de trabajo de JavaScript desarrollado por Meta que permite crear aplicaciones móviles nativas para iOS y Android utilizando una base de código común. Se basa en React, una biblioteca de JavaScript para construir interfaces de usuario, y permite a los desarrolladores crear componentes de UI en JavaScript que se renderizan como vistas nativas en las plataformas móviles.

React Native ofrece varias ventajas clave. Primero, permite la reutilización de código, lo que puede reducir significativamente el tiempo y los recursos necesarios para desarrollar aplicaciones para múltiples plataformas. En segundo lugar, cuenta con una gran comunidad de desarrolladores y una abundancia de paquetes de terceros, lo que facilita la incorporación de funcionalidades avanzadas. En tercer lugar, React Native permite el recargado en caliente, lo que significa que los desarrolladores pueden ver los cambios en tiempo real sin necesidad de recompilar la aplicación.

A continuación se detallarán los fundamentos del desarrollo de esta aplicación móvil, incluidas sus funcionalidades. Al final de explicar cada funcionalidad se encontrará un diagrama de secuencias.

Sintaxis de React Native

La sintaxis de React Native es muy similar a la de React. Se basa en componentes, que son bloques de construcción reutilizables que definen cómo una parte de la interfaz de usuario debería renderizarse y comportarse.

Cada componente se define como una función o una clase en JavaScript, y retorna un elemento de React que describe cómo debería verse el componente. Este elemento de React se suele escribir utilizando JSX, una sintaxis que permite escribir HTML en JavaScript.

```
1 import React from 'react';
2 import { Text, View } from 'react-native';
3
4 const MiComponente = () => {
5   return (
6     <View>
7       <Text>Hola, mundo!</Text>
8     </View>
9   );
10 };
11
12 export default MiComponente;
```

Listing 7.15: Ejemplo de sintaxis React Native

Hooks de React Native

Los Hooks son una característica de React (y por ende de React Native) que permiten utilizar el estado y otras características de React sin tener que escribir una clase. Esto puede hacer que el código sea más legible y fácil de mantener.

Existen varios Hooks predefinidos en React, como `useState`, que permite añadir estado local a un componente funcional, y `useEffect`, que permite ejecutar efectos secundarios en el componente.

Por ejemplo, se observa un componente que utiliza el hook `useState` para manejar su propio estado:

```
1 import React, { useState } from 'react';
2 import { Button, Text, View } from 'react-native';
3
4 const MiComponente = () => {
5   const [contador, setContador] = useState(0);
6
7   return (
8     <View>
9       <Text>Tienes {contador} clics</Text>
10      <Button
11        title="Haz clic"
12        onPress={() => setContador(contador + 1)}
13      />
14    </View>
15  );
16 };
17
18 export default MiComponente;
```

Listing 7.16: Código React Native que usa Hooks para actualizar un contador.

Creación e Inicio de una Aplicación en React Native utilizando Yarn

Es importante destacar que para trabajar con React Native, se necesita tener Node.js y Yarn instalados en la máquina. Node.js es un entorno de tiempo de ejecución que permite ejecutar JavaScript en el servidor, mientras que Yarn es un gestor de paquetes rápido,

fiable y seguro que permitirá instalar, actualizar y administrar nuestras dependencias de una manera eficiente.

Una vez que has instalado Node.js y Yarn, se puede comenzar a crear tu nueva aplicación React Native. Para ello, se puede utilizar la herramienta npx, que es un ejecutor de paquetes que viene con Node.js. Para arrancar la creación de una nueva aplicación React Native, se ejecuta el siguiente comando en tu terminal ubicado en el directorio de preferencia:

```
npx react-native init userApp
```

Este comando creará una nueva carpeta con el nombre de la aplicación que contendrá todos los archivos y directorios necesarios para comenzar a desarrollar la aplicación React Native.

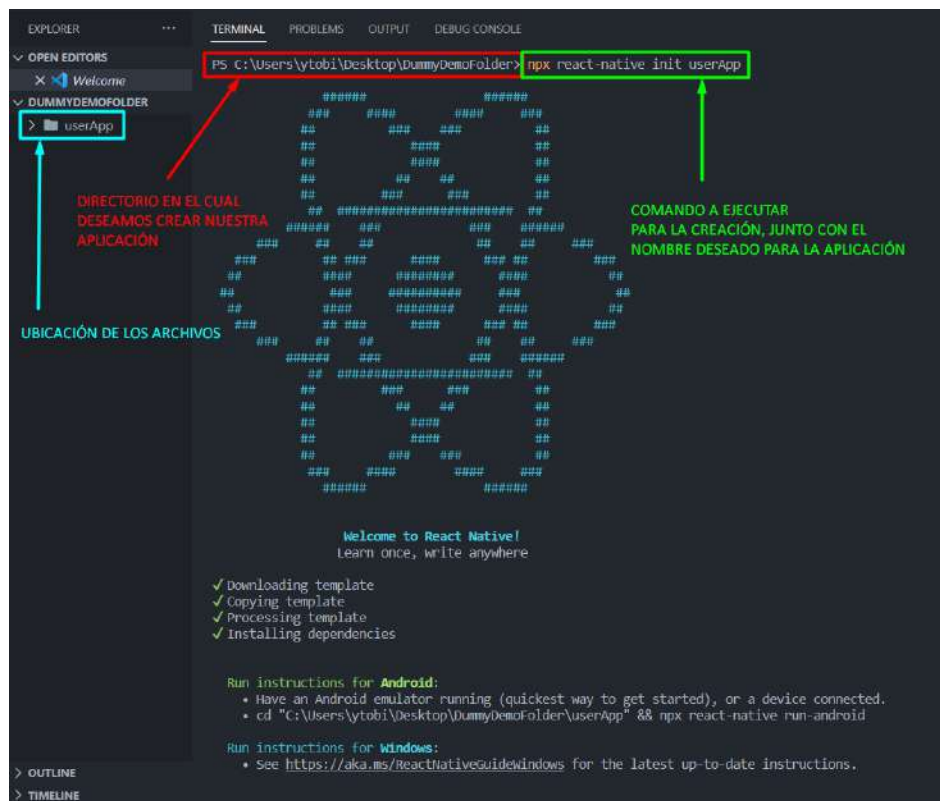


Figura 7.23: Creación de una aplicación React Native

Una vez que el proceso se haya completado, navega a la carpeta de tu aplicación utilizando el siguiente comando:

```
cd userApp
```

Para iniciar tu aplicación, puedes utilizar el comando `yarn start`. Esto iniciará el servidor Metro Bundler, que es el servidor de JavaScript de React Native y se podrá visualizar nuestra aplicación en nuestro dispositivo o emulador móvil.

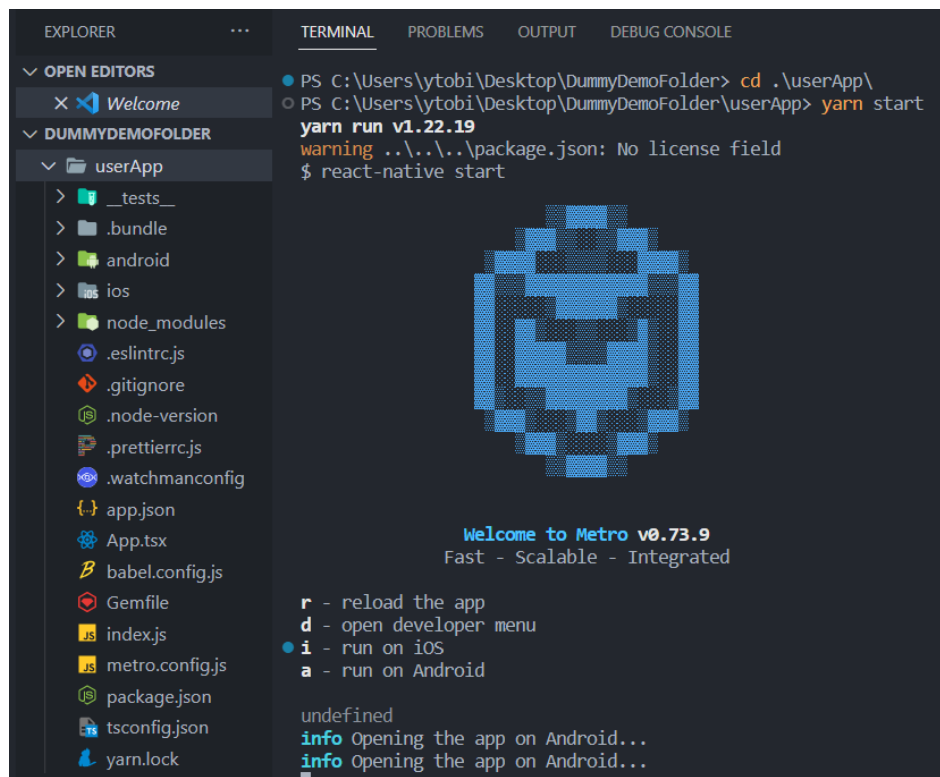


Figura 7.24: Arrancar una aplicación React Native

7.7.2. Inicio de sesión y gestión de la cuenta

Este apartado se centra en los detalles de cómo los usuarios pueden iniciar y cerrar sesión en nuestra aplicación de Parque de Atracciones. Adicionalmente, se discutirá cómo pueden actualizar su información personal y restablecer su contraseña en caso de ser necesario. Se proporcionará información detallada sobre los datos personales necesarios para el registro, así como la forma en que dicha información se protege y se utiliza.

Creación de la Cuenta:

Para disfrutar de todas las funcionalidades que ofrece nuestra aplicación, el usuario deberá crear una cuenta. Este proceso puede iniciarse haciendo clic en el botón de Registro.º "Sign Up", o bien, al intentar acceder a una funcionalidad exclusiva para usuarios registrados, el sistema automáticamente redirigirá al usuario a la pantalla de inicio de sesión, donde encontrarán la opción para crear una cuenta.

Para crear una cuenta, es necesario que el usuario proporcione la siguiente información: nombre, apellidos, fecha de nacimiento, correo electrónico, nombre de usuario, contraseña y verificación de contraseña. Una vez ingresados estos datos y al finalizar el proceso de registro, se generará automáticamente una cuenta de usuario, permitiéndole acceder a todas las funcionalidades de la aplicación sin necesidad de iniciar sesión nuevamente.

Es importante destacar que en caso de que se intente crear una cuenta con un usuario o un correo electrónico que ya exista, la creación del usuario se declarará como nulo con

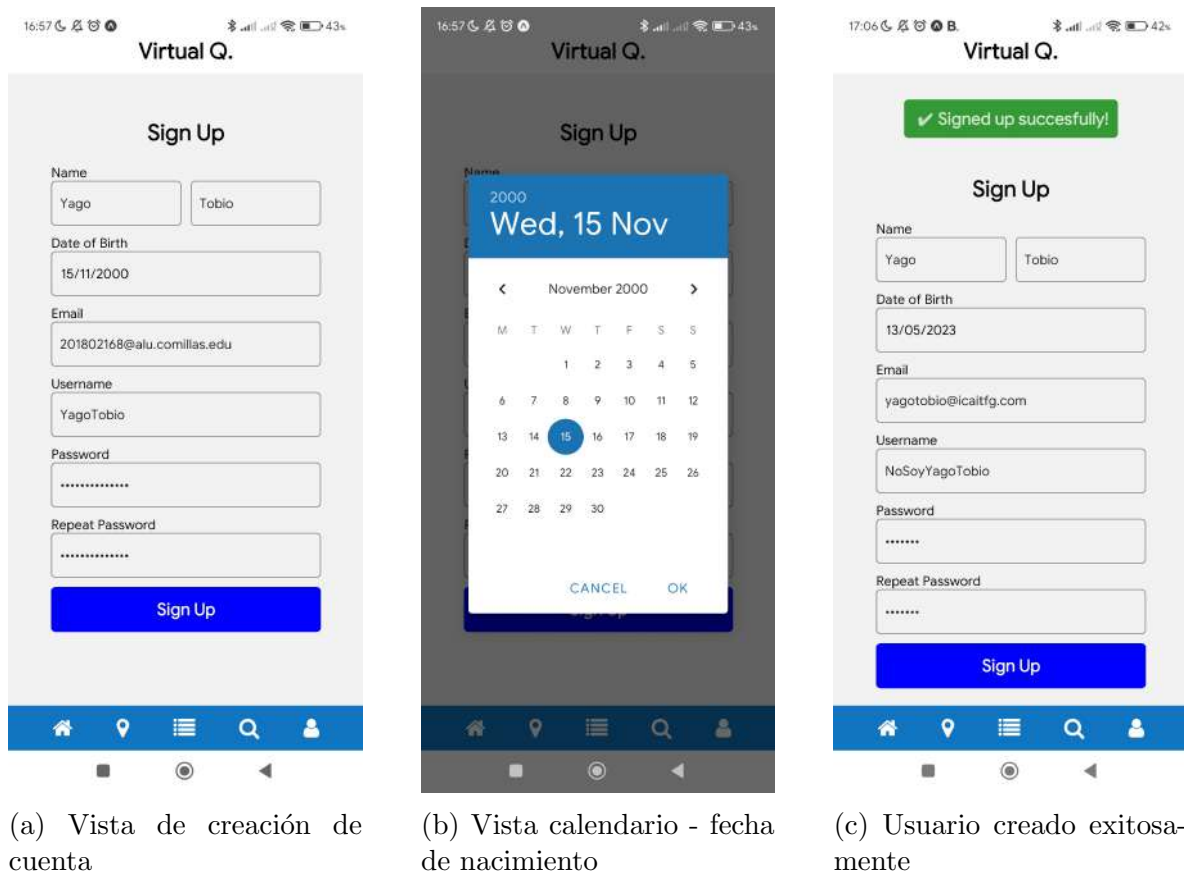


Figura 7.25: Flujo de creación de un usuario

una alerta avisando que ya existe dicho usuario.

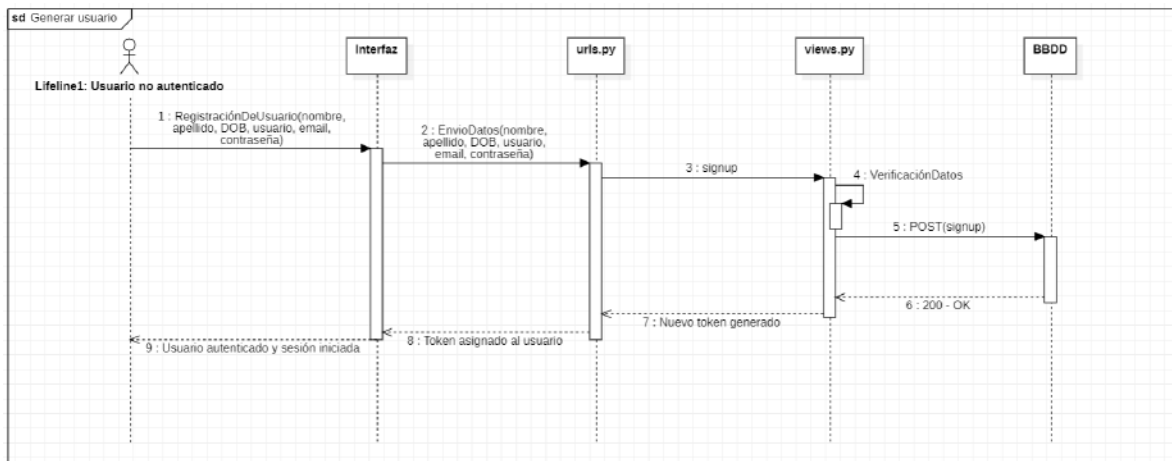


Figura 7.26: Diagrama de secuencia del Registro de un nuevo usuario

Inicio de Sesión

Si el usuario ya cuenta con una cuenta registrada, podrá iniciar sesión introduciendo su nombre de usuario y contraseña. Tal como se mencionó en secciones anteriores de la

documentación, este proceso asignará un nuevo token de sesión, permitiendo al usuario disfrutar de todas las funcionalidades exclusivas de la aplicación.

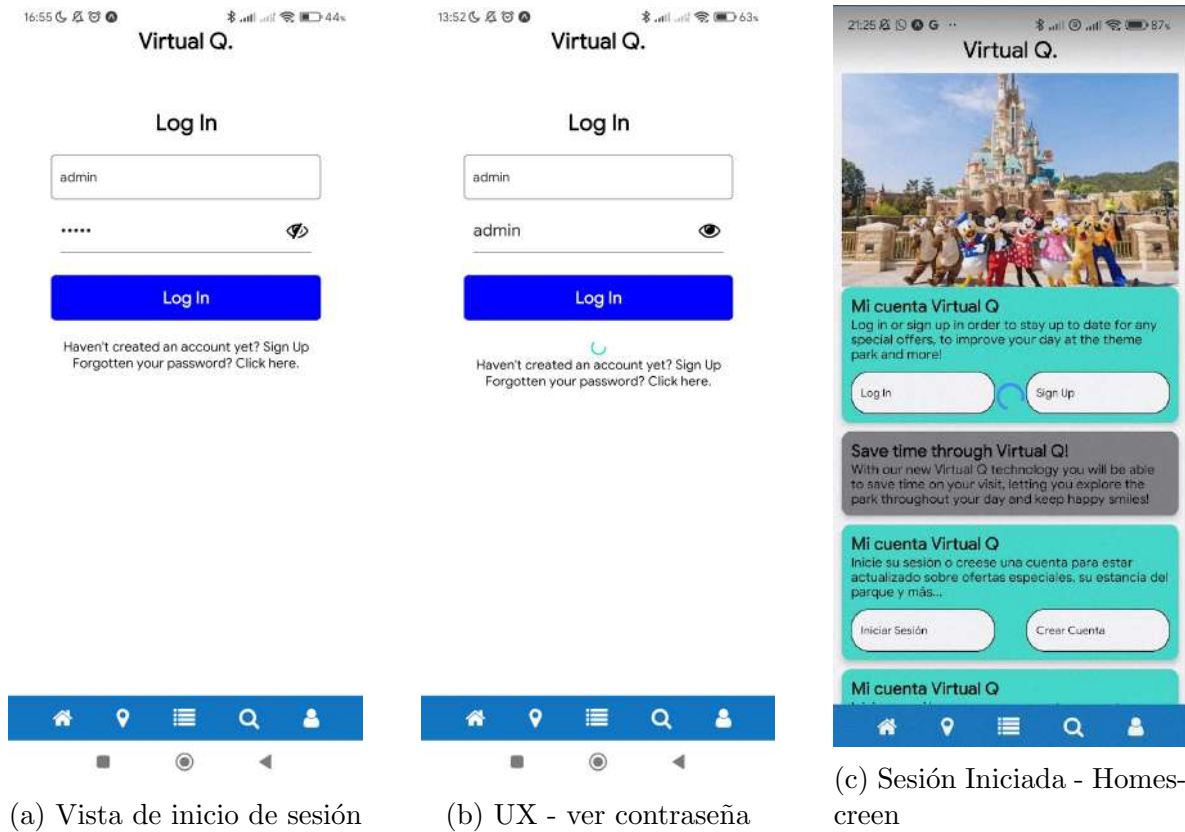


Figura 7.27: Flujo inicio sesión

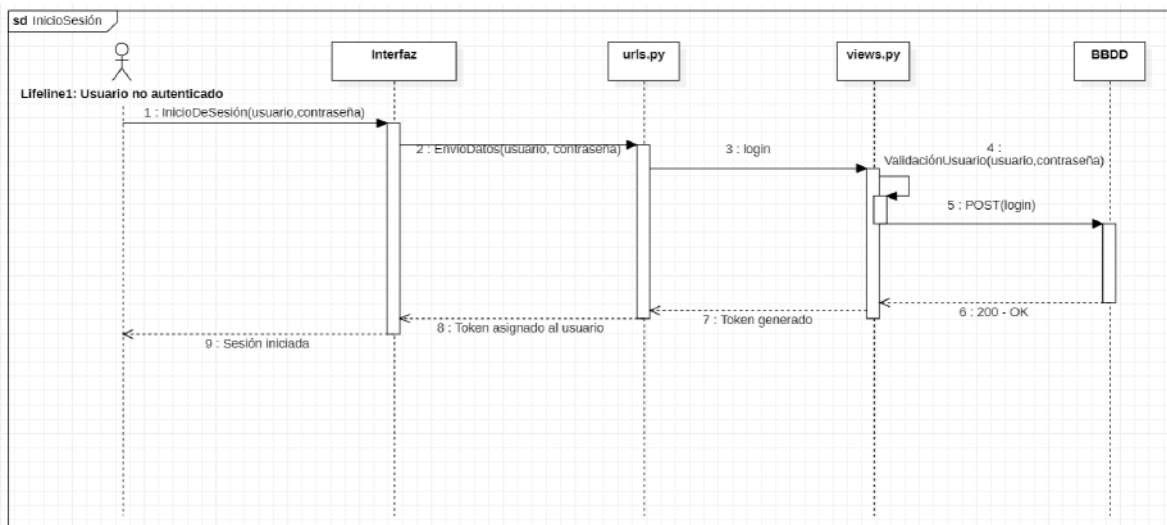


Figura 7.28: Diagrama de secuencia del inicio de sesión

Restablecimiento de Contraseña

Se entiende que puede haber ocasiones en las que los usuarios olviden su contraseña. En tales casos, se ha proporcionado una opción en la página de inicio de sesión para restablecer la contraseña. Al hacer clic en esta opción, se redirigirá al usuario a una nueva pantalla donde podrá introducir su correo electrónico.

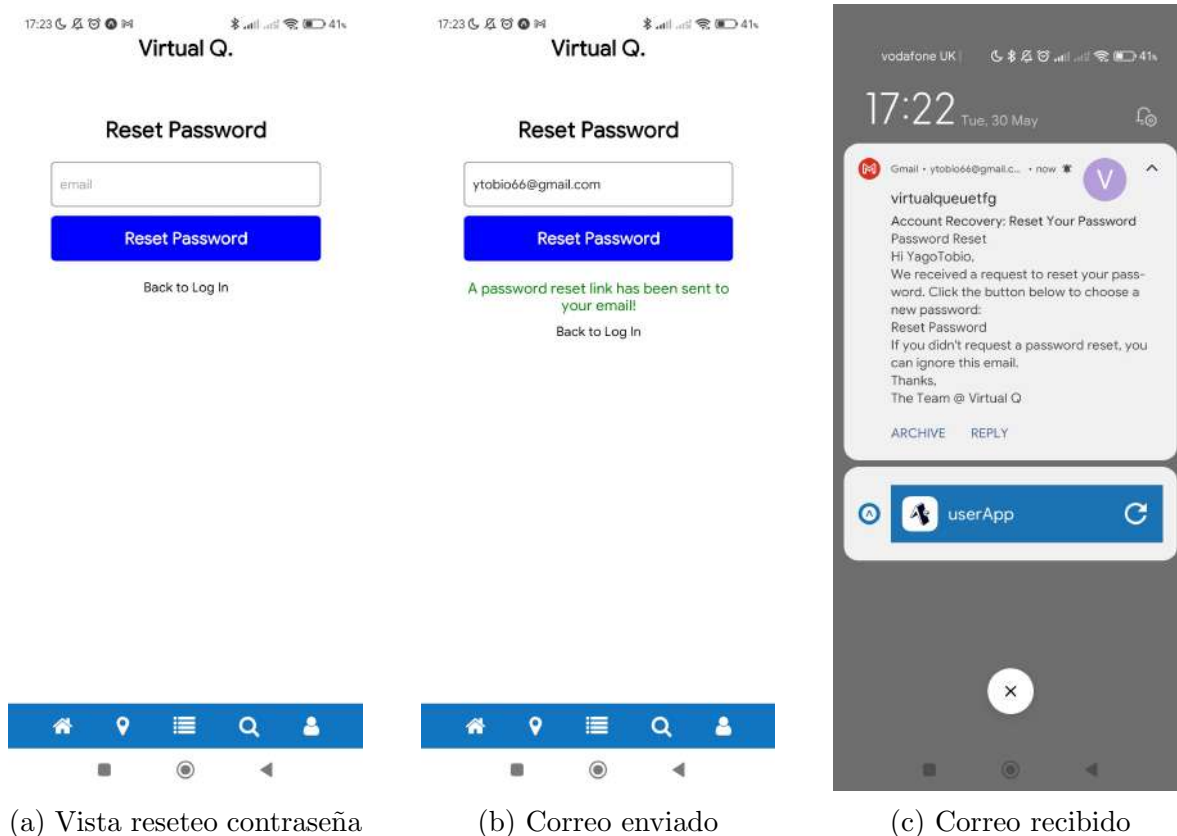


Figura 7.29: Flujo de peticiones para solicitar el reseteo de contraseña

Si el correo electrónico introducido está asociado a una cuenta existente, el sistema enviará un correo electrónico al usuario con un enlace para restablecer su contraseña, en caso contrario se mostrará una alerta por pantalla. El email es customizado incluyendo el nombre del usuario e incluyendo un botón con un enlace. Al hacer clic en este enlace, se les dirigirá a una página segura donde podrán crear una nueva contraseña. Una vez que la contraseña ha sido restablecida, los usuarios podrán utilizarla para iniciar sesión y continuar disfrutando de las funcionalidades de nuestra aplicación.

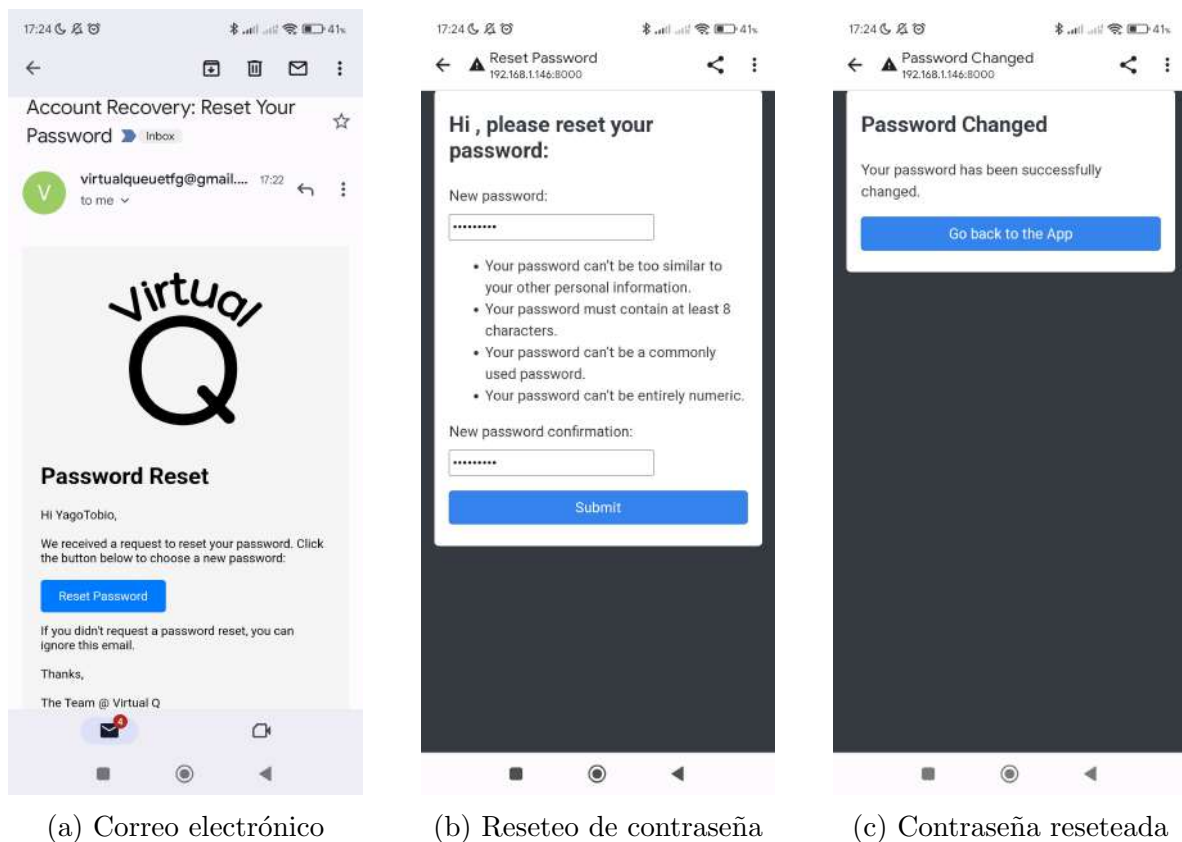


Figura 7.30: Activación del correo - reseteo de contraseña

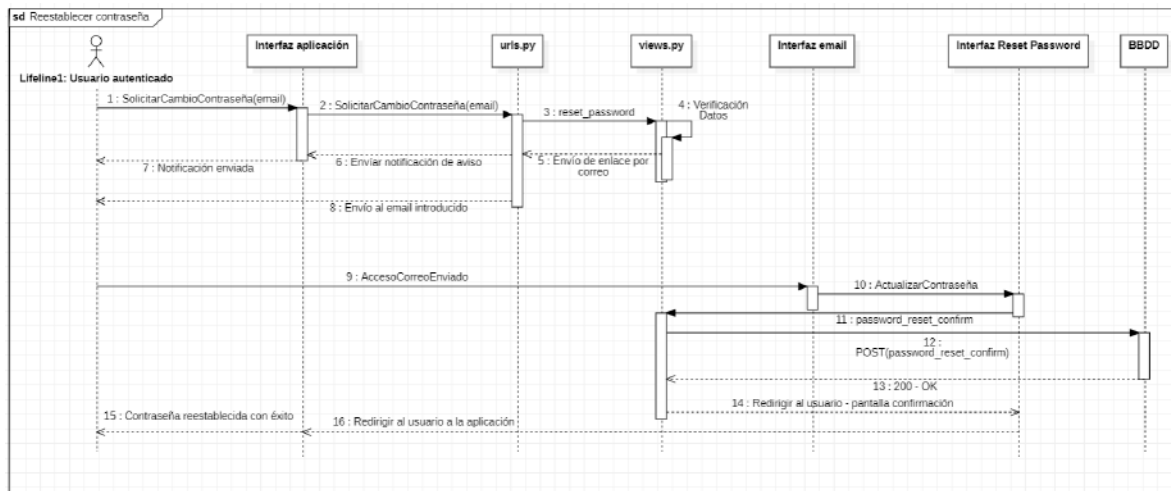


Figura 7.31: Diagrama de secuencia del reseteo de contraseña.

Edición de Datos del Usuario

Una vez registrados, los usuarios tienen la flexibilidad de modificar su información personal, permitiendo un control total sobre sus perfiles. Este proceso se realiza de manera sencilla y rápida desde la sección de "Perfil" de la aplicación. Aquí, los usuarios

pueden actualizar su correo electrónico, nombre, apellido, nombre de usuario, fecha de nacimiento y contraseña según sus necesidades.

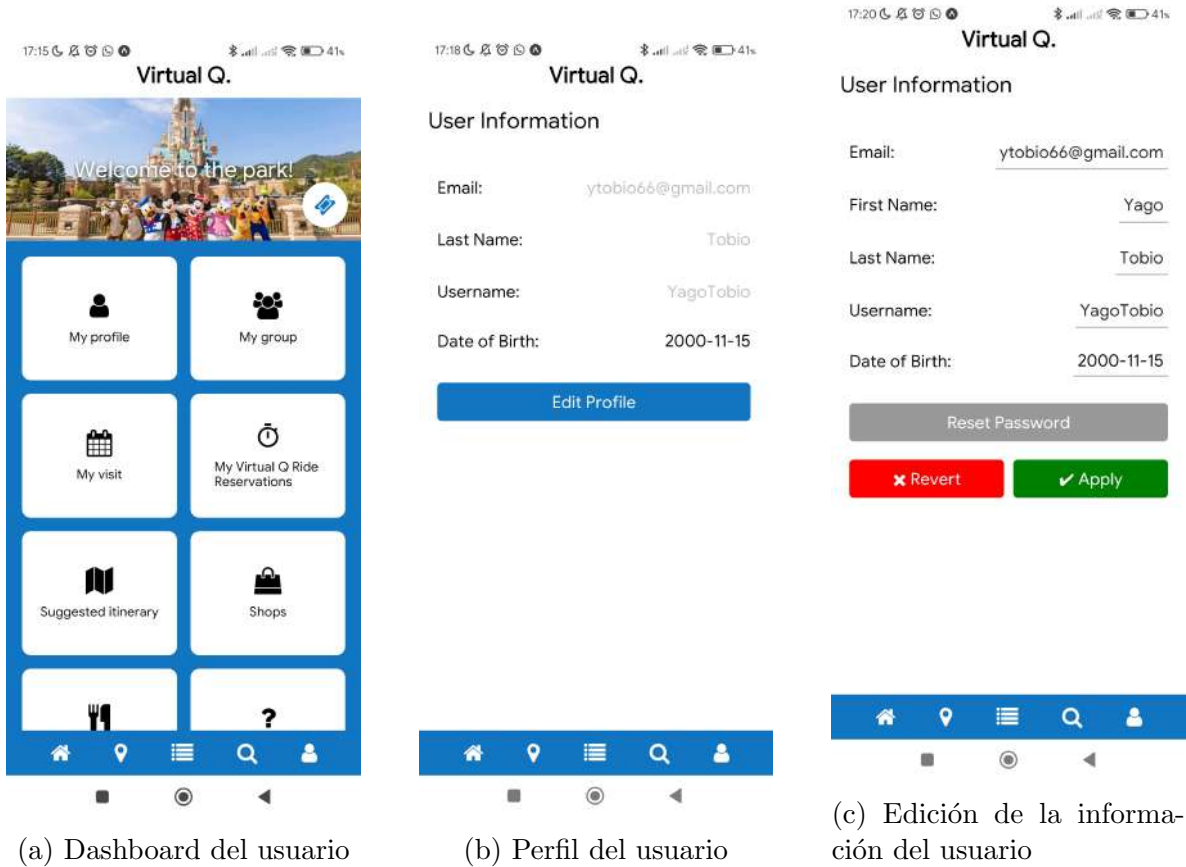


Figura 7.32: Flujo para editar los datos del usuario.

Es importante destacar que la aplicación garantiza la unicidad de ciertos datos como el correo electrónico y el nombre de usuario. En otras palabras, en caso de que el usuario intente actualizar su correo electrónico o nombre de usuario a uno que ya exista en el sistema, la aplicación no permitirá dicha actualización y se mostrará un mensaje de error informando al usuario sobre la existencia previa de dichos datos. Este mecanismo ayuda a mantener la integridad de la información de los usuarios y evita conflictos entre las cuentas.



Figura 7.33: Flujo para editar los datos del usuario.

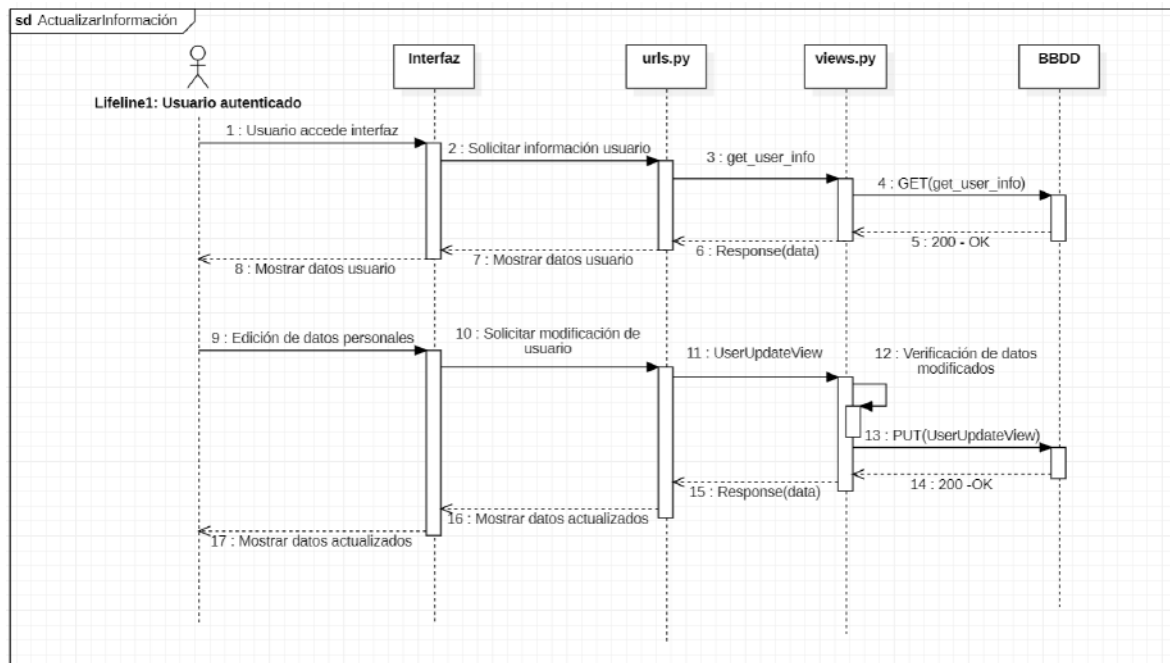


Figura 7.34: Diagrama de secuencia - actualizar datos usuario

7.7.3. Compra de entradas y gestión de huéspedes

La aplicación de Parque de Atracciones facilita la adquisición de entradas para los usuarios. Desde el momento en que los usuarios inician sesión en la aplicación, tienen la posibilidad de comprar entradas de manera sencilla y rápida.

Desde el "Dashboard" del usuario, se puede visualizar un botón identificado con un logotipo de unas entradas. Al seleccionarlo, el sistema realiza una consulta GET a la

API del servidor para determinar si el usuario ya posee entradas. En caso contrario, se presentará un botón “Comprar Entradas” que redirige al usuario a una página web segura.

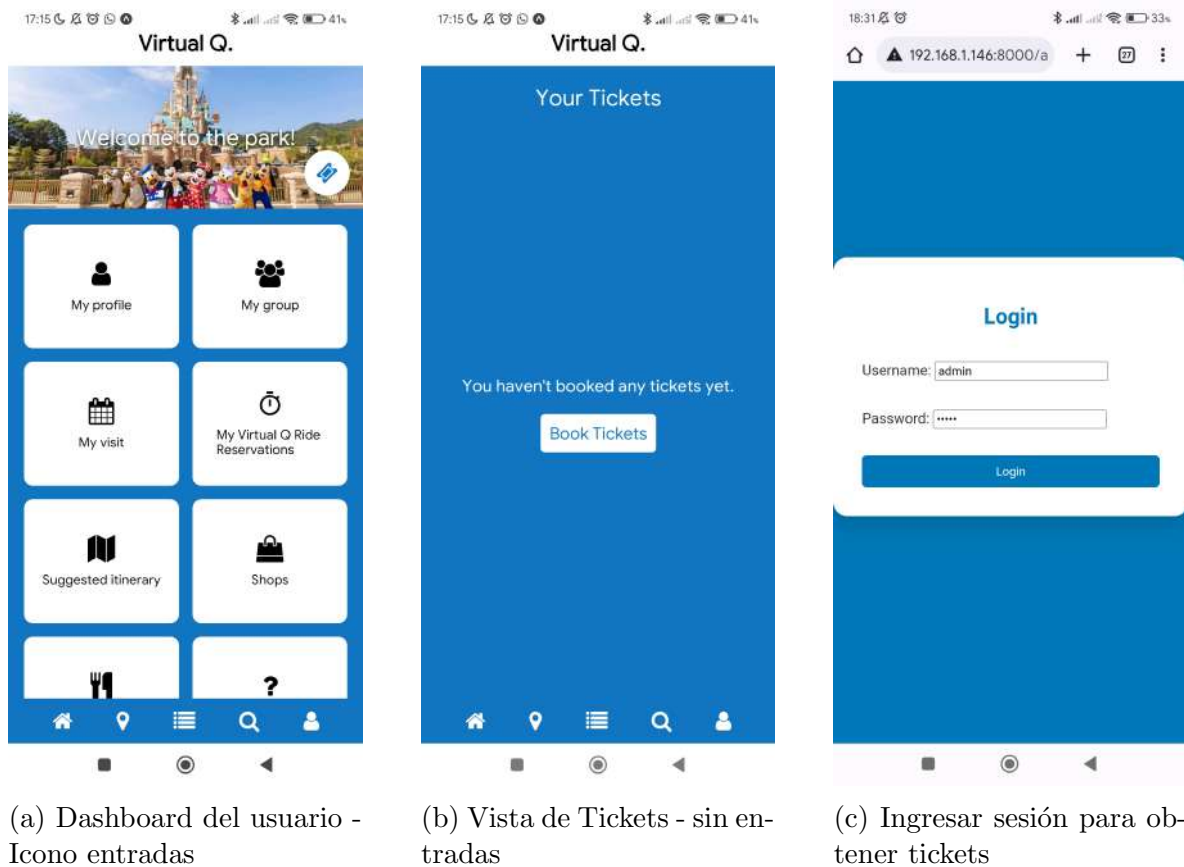


Figura 7.35: Flujo para solicitar entradas.

Tras entrar en la página web, se observa que se pueda iniciar sesión y una vez se haya ingresado con éxito, se abrirá una vista en la cual se puede seleccionar la fecha y el número de visitantes adicionales que acompañarán al usuario el día de su visita.

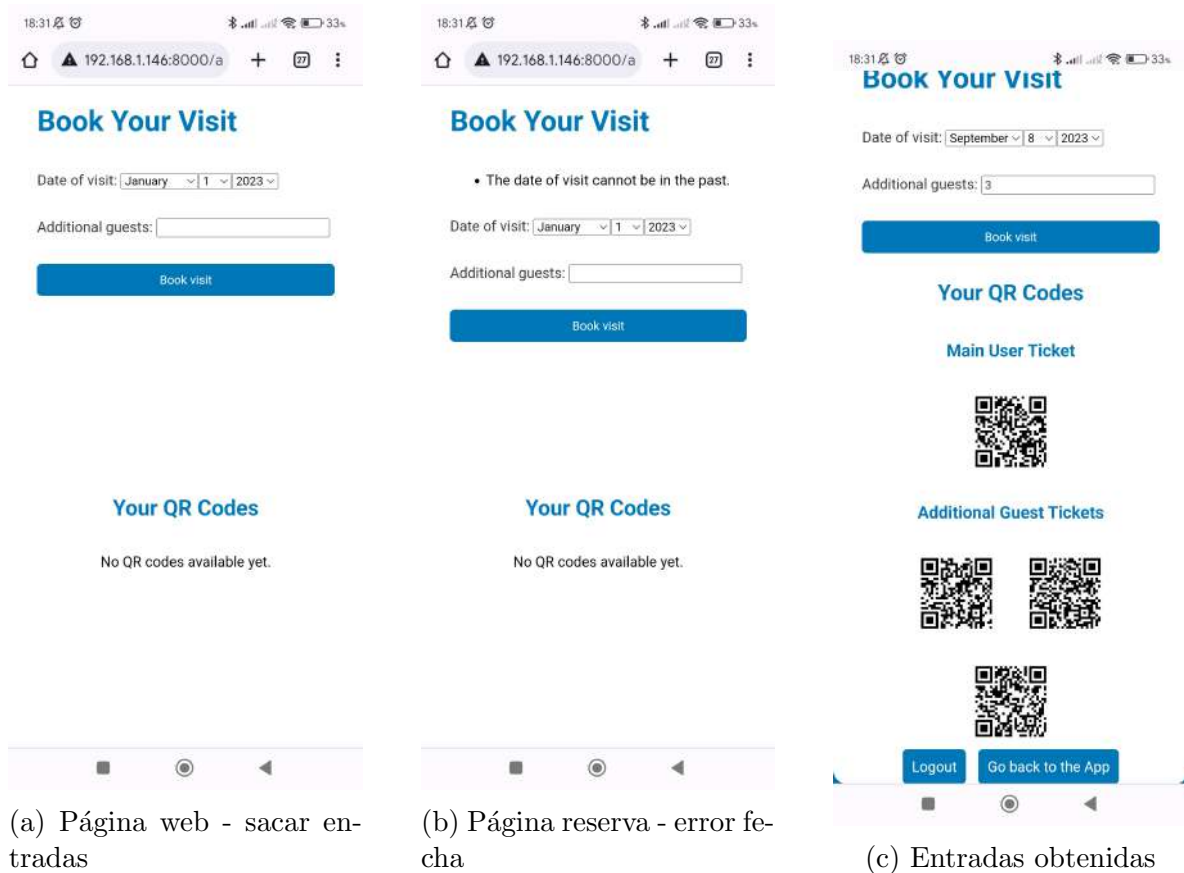


Figura 7.36: Flujo para obtener entradas [Fecha reservada - 8/8/23]

Una vez que el usuario ha seleccionado la fecha y el número de visitantes adicionales, se generan automáticamente códigos QR individuales para cada visitante, incluido el usuario. Estos códigos se presentan de forma intuitiva dentro de la aplicación para su fácil acceso el día de la visita. Al hacer clic sobre cualquiera de las entradas, se abrirá una vista magnificada, en la cual se podrá deslizar por las entradas para ese día. [La generación de estos códigos QR se detallan en el backend]



Figura 7.37: Visualización en la vista de entradas para ver las reservas hechas - 8/8/23

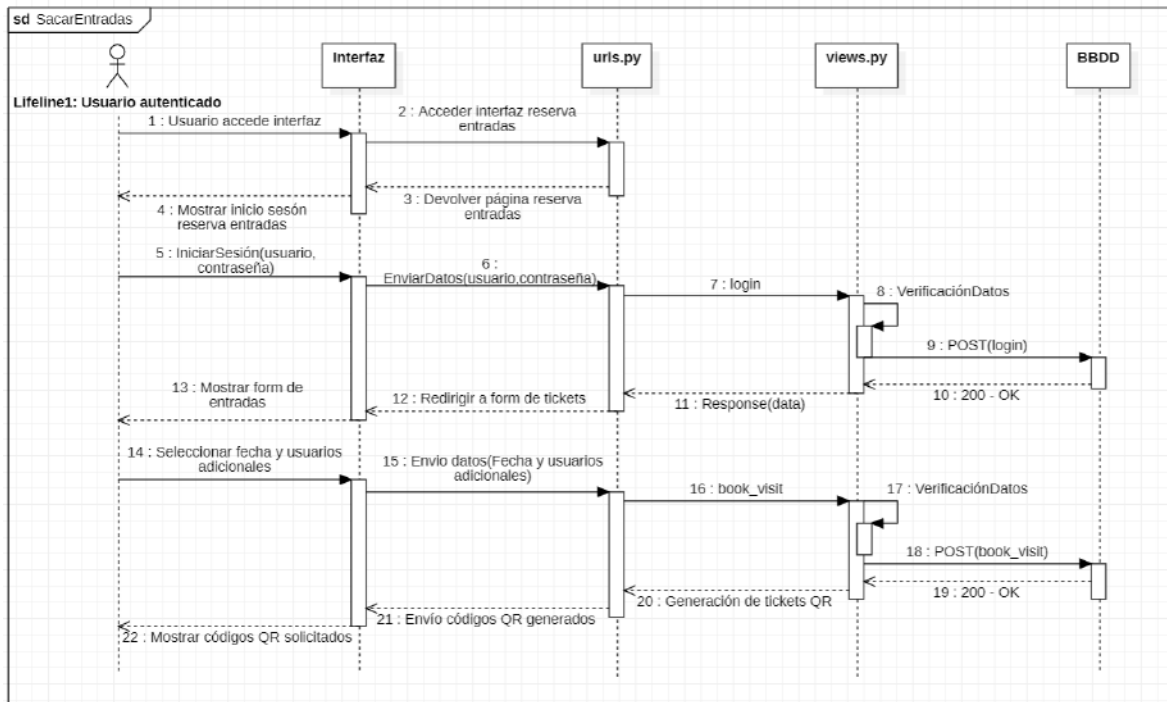


Figura 7.38: Diagrama de secuencia de la compra de entradas

Además de la compra de entradas, la aplicación ofrece la opción de personalizar la información de los visitantes adicionales. Desde el "Dashboard" del usuario, se puede acceder a la opción "Mi Grupo", donde se muestran perfiles vacíos para cada visitante adicional. Cada perfil puede ser personalizado con un nombre, edad, altura y un icono de perfil.



Figura 7.39: Visualización de la edición de los visitantes adicionales

Esta funcionalidad tiene dos propósitos principales:

1. Proporcionar un toque personalizado a cada visitante, permitiéndole personalizar su avatar y mejorar su experiencia en la aplicación.
2. Recoger información útil que puede ser utilizada para mejorar la experiencia en el parque. Por ejemplo, la aplicación puede realizar recomendaciones basadas en la edad promedio del grupo, emitir advertencias si un visitante no cumple con la altura mínima para una atracción, o gestionar reservas considerando las preferencias individuales de cada visitante.



Figura 7.40: Información de acompañantes rellena

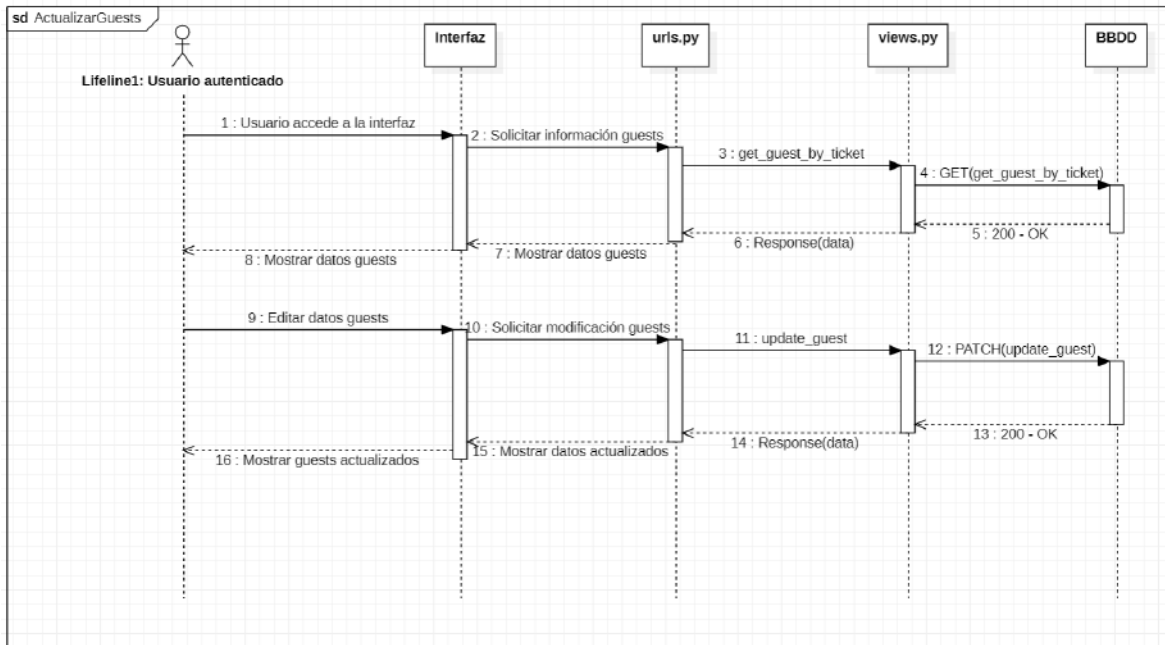


Figura 7.41: Diagrama de secuencia de la gestión de guests

7.7.4. Exploración de los parques de atracciones

Nuestra aplicación proporciona una experiencia completa de exploración de los parques de atracciones disponibles, las cuales han sido subidas por el administrador a través del back-end. Los usuarios pueden visualizar un listado de todas las atracciones disponibles en el parque haciendo clic en el botón del footer central, con la opción de filtrar los resultados según el tipo de atracción y el área del parque en donde se ubica.

En la vista de lista de atracciones, cada atracción está presentada con una breve descripción que incluye su título, una imagen representativa, el área en la que se encuentra

dentro del parque, los iconos de accesibilidad y la opción de añadirla a la lista de atracciones favoritas del usuario.

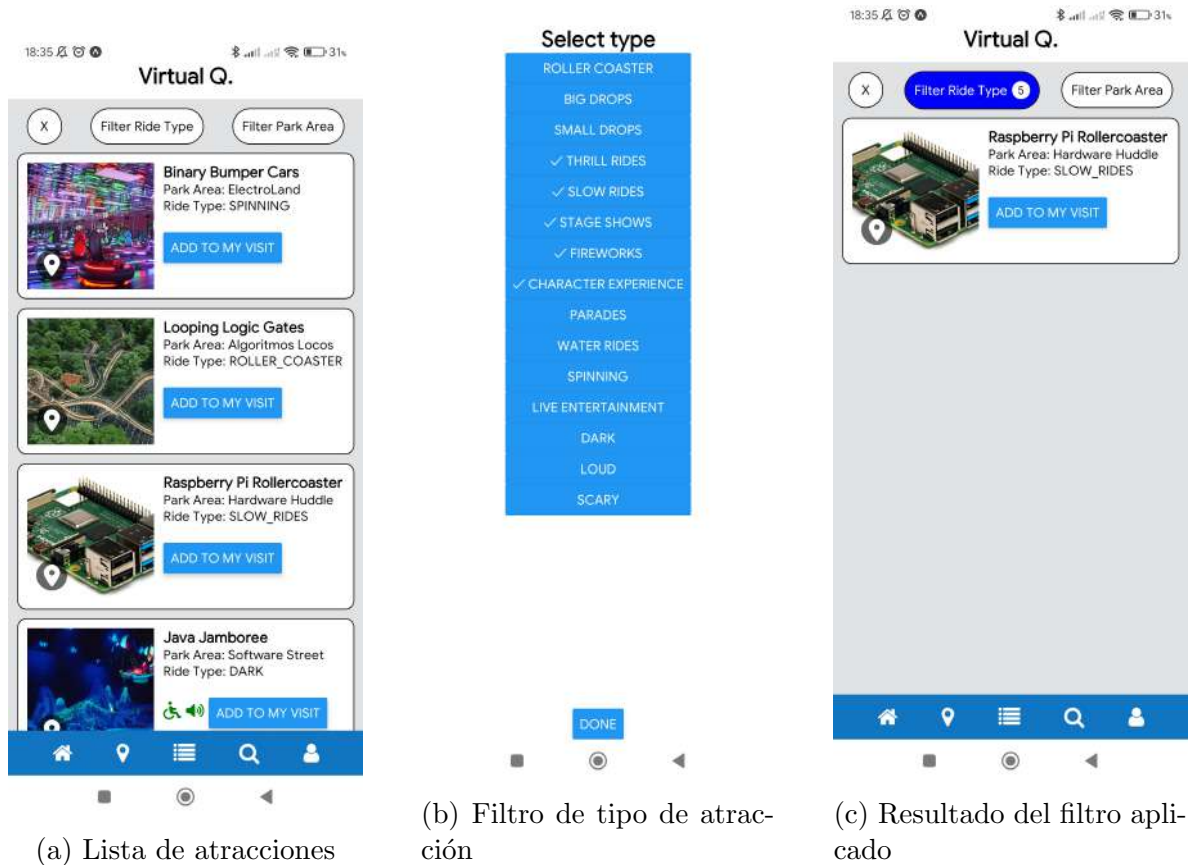


Figura 7.42: Visualización del listado de atracciones y sus filtros

Si el usuario desea más detalles sobre una atracción en particular, simplemente tiene que hacer clic en la misma para acceder a una versión expandida con información detallada. Esta información adicional incluye una descripción completa de la atracción, indicador de altura mínima requerida, la edad recomendada para los visitantes, todas las opciones de accesibilidad disponibles y los horarios de apertura.

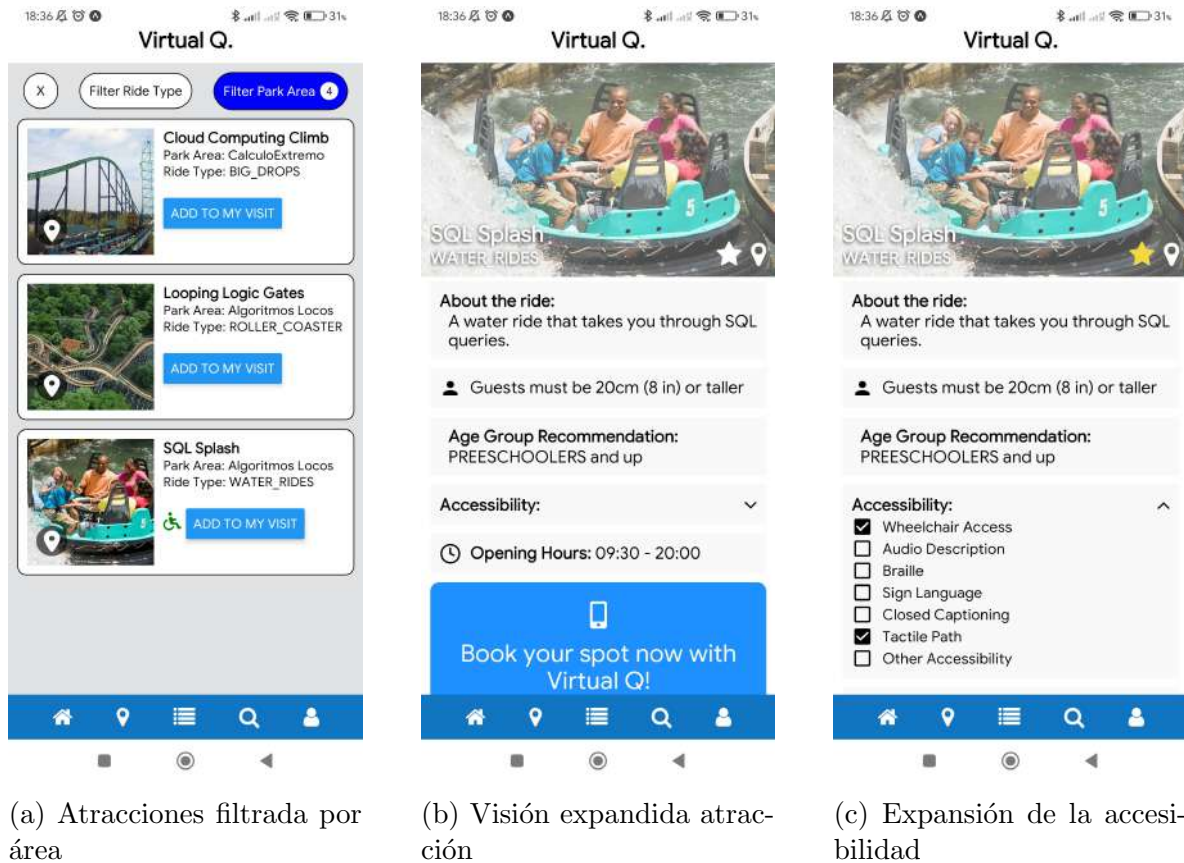


Figura 7.43: Visualización de la atracción completa

Esta funcionalidad está disponible para todos los usuarios, incluso para aquellos que no tienen una cuenta. Se requiere una cuenta de usuario para agregar atracciones a la lista de favoritos y acceder al sistema de reservas del parque de atracciones. Con esta estructura, la aplicación fomenta la exploración y proporciona a los usuarios todas las herramientas necesarias para personalizar y planificar su visita al parque de atracciones de manera eficiente y agradable.

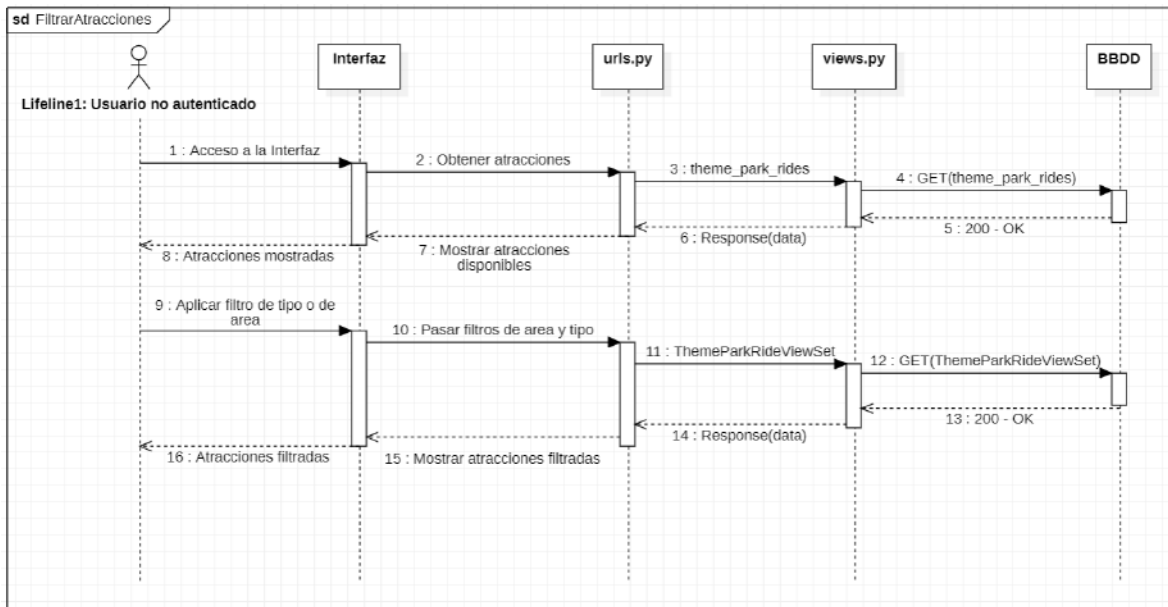


Figura 7.44: Diagrama de secuencia de la compra de entradas

7.7.5. Reservas de atracciones

La aplicación ofrece un sistema de reservas robusto y eficiente, permitiendo a los usuarios reservar sus atracciones favoritas para evitar largas colas y aprovechar al máximo su visita. Sin embargo, para utilizar este servicio, el usuario debe cumplir tres condiciones:

1. Debe de tener una cuenta registrada en la aplicación. En caso de no tenerla, se le redirigirá a la vista de inicio de sesión.
2. Debe haber adquirido entradas para el parque de atracciones.
3. Debe realizar la reserva con menos de 2 días de antelación a su visita.

El proceso de reserva es sencillo. Desde la vista detallada de la atracción, los usuarios pueden seleccionar la opción "Haz tu reserva con Virtual Q.". Esto les redirige a una pantalla donde pueden seleccionar la fecha de su visita (si tienen entradas para más de un día) y elegir un intervalo de 30 minutos para su reserva.

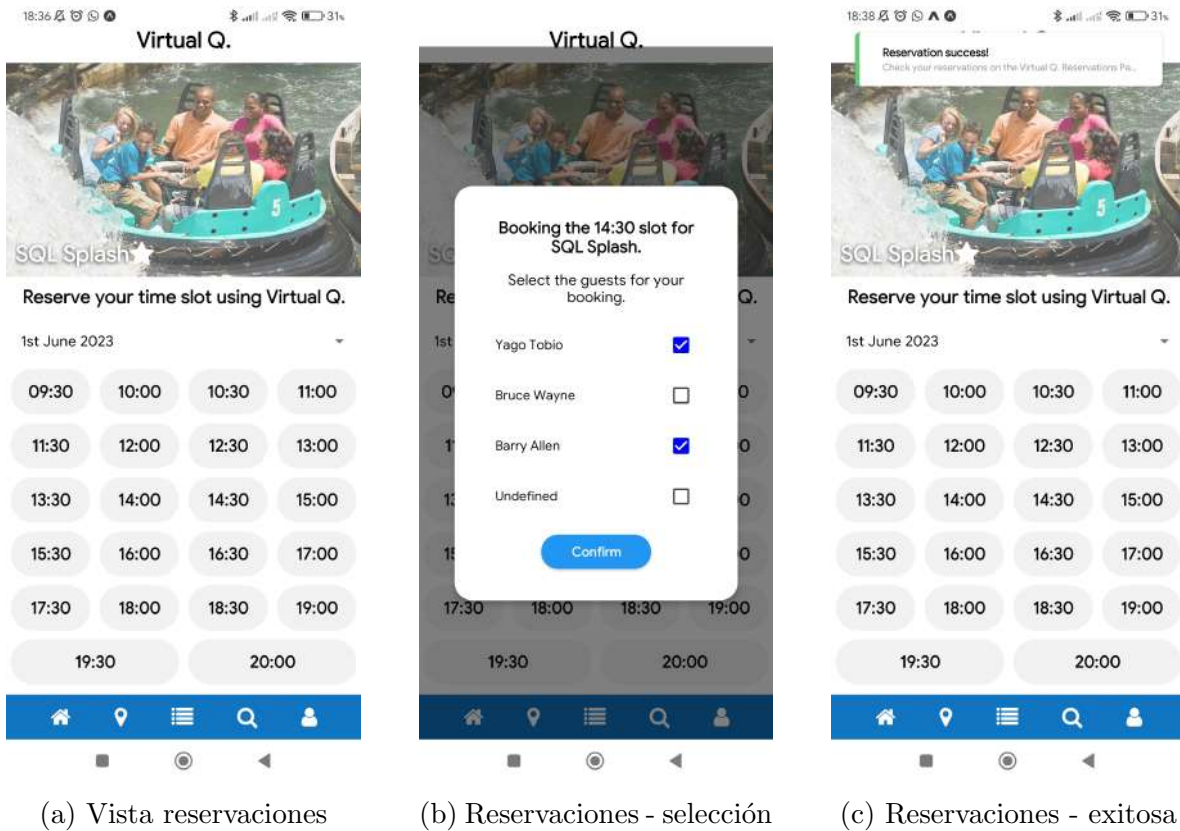


Figura 7.45: Visualización del sistema de reservas

Una vez seleccionado el intervalo de tiempo, se presenta al usuario una vista detallada de su reserva propuesta, que incluye la atracción seleccionada, la hora de la reserva y los miembros del grupo que asistirán a la atracción. Los usuarios tienen la libertad de seleccionar quiénes de su grupo participarán en la reserva, lo que permite mayor flexibilidad y evita la asignación innecesaria de plazas.

Sin embargo, hay varias restricciones para hacer una reserva. No se puede hacer una reserva si la atracción está en mantenimiento, la reserva debe coincidir con la fecha de la entrada adquirida, no debe existir otra reserva para el usuario en el mismo intervalo de tiempo, y la reserva no debe exceder la capacidad de la atracción durante ese intervalo de tiempo.

Una vez confirmada la reserva, los usuarios recibirán una notificación y podrán ver sus reservas en la vista "My Virtual Q. Reservations". Esta vista muestra todas las reservas del usuario de manera clara y concisa, incluyendo los detalles de la atracción, el horario y los miembros del grupo que asisten. Al seleccionar una reserva, los usuarios pueden ver y presentar los códigos QR correspondientes para cada miembro del grupo.

Para mantener la vista de reservas relevante y ordenada, todas las reservas y entradas de días pasados se eliminan automáticamente, evitando la saturación de información innecesaria.

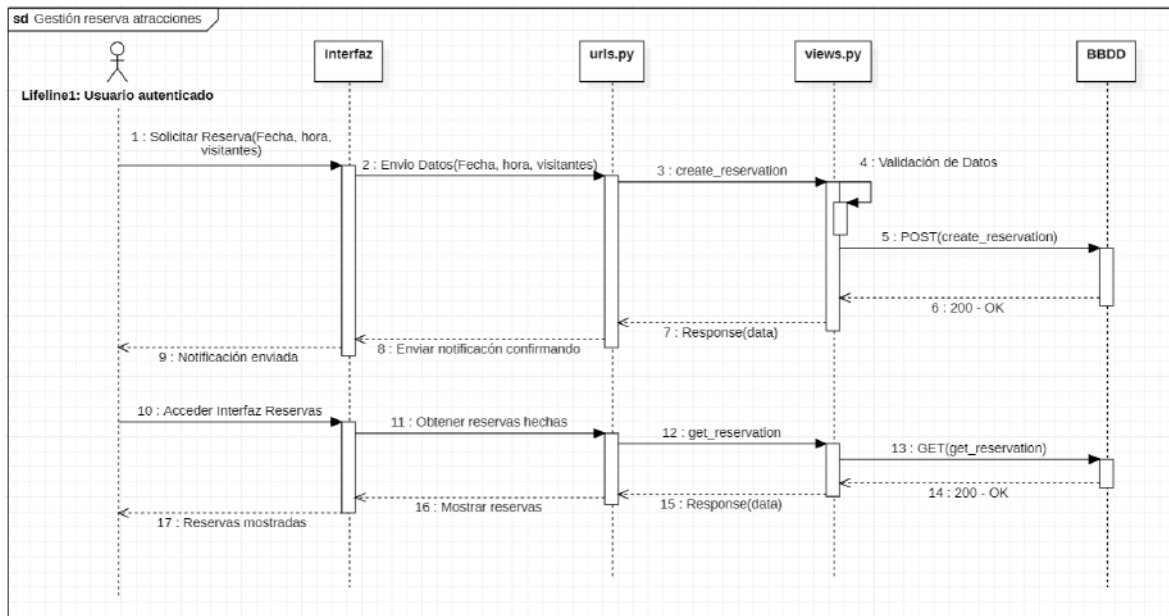


Figura 7.46: Diagrama de secuencia de la compra de reservas.

Capítulo 8

Resultados del Trabajo

Este capítulo presenta y analiza detalladamente los resultados obtenidos en el Trabajo de Fin de Grado, tras implementar las funcionalidades expuestas en el Capítulo 5.

Virtual Q. ofrece una variedad de funcionalidades que brindan a nuestros usuarios una experiencia rica y personalizada. Desde la creación y gestión de cuentas hasta la compra de entradas, la exploración de parques de atracciones y la reserva de atracciones, se ha prestado especial atención por hacer que la visita al parque sea lo más eficiente y agradable posible.

Como se indicó a principios del capítulo 4, la intención de esta aplicación es dual. Tal que no solo fuese fácil para el usuario acceder a la información, sino que también fuese intuitivo para el administrador poder insertar y gestionar la información al back-end.

Cuenta con el desarrollo de múltiples pruebas unitarias con el fin de verificar el correcto funcionamiento de todos los componentes de la aplicación y que las funcionalidades actúen de la manera correcta y esperada.

Una característica distintiva de la aplicación es la atención meticulosa que se ha prestado a su diseño estético y la experiencia del usuario. El propósito se ha centrado en la creación de una interfaz de usuario que sea intuitiva, fácil de usar y que se adhiera a las mejores prácticas de la experiencia de usuario en el desarrollo de aplicaciones móviles. Cada pantalla, cada botón y cada flujo de trabajo han sido diseñados teniendo en cuenta la facilidad de uso y la eficiencia, lo que permite a los usuarios navegar por la aplicación con facilidad y realizar acciones con un mínimo esfuerzo.

Además, esta aplicación incorpora la innovadora funcionalidad “Virtual Q.”, un avanzado sistema de reserva de atracciones. Este sistema permite a los usuarios organizar su visita y maximizar su tiempo en el parque, minimizando el tiempo de espera y transformando así la experiencia en el parque en algo aún más satisfactorio. Esta característica, aunque se ha comenzado a implementar en algunos de los parques de atracciones más grandes, todavía es escasa y a menudo se presenta de forma limitada.

A continuación, se analizará los resultados obtenidos del proyecto y verificar la funcionalidad del mismo.

8.1. Análisis de los resultados de la simulación

Se van a analizar los resultados obtenidos tras la simulación vista en el capítulo 5. El objetivo de la simulación es comprobar que el nuevo sistema que se desea implementar en esta aplicación proporciona una ventaja significativa en comparación con el sistema actual de colas.

Para ello, en la simulación se han tenido en cuenta 2 escenarios distintos, compuestos de 2 colas cada uno:

1. Cola normal
2. Cola fast pass
3. Cola normal - sistema virtual
4. Cola virtual

Teniendo en cuenta la espontaneidad de una visita al parque de atracciones, para el sistema de colas virtual se han establecido los siguientes parámetros:

	<i>Porcentaje de clientela para una visita</i>
<i>Limitación de pre-reserva Virtual Q.</i>	75 %
<i>Limitación de cola física - Virtual Q.</i>	15 %
<i>Limitación de reserva Virtual Q.</i>	85 %

Tabla 8.1: Porcentajes de clientela en Virtual Q.

Partiendo de estos datos ejecutamos la simulación desarrollada implementando ambos sistemas de colas y obtuvimos los siguientes resultados:

	<i>Tiempo de espera de media (minutos) - sistema de colas estandar</i>	<i>Tiempo de espera (minutos) - sistema de colas virtuales</i>
Simulación 1	135.52	50.75
Simulación 2	153.22	50.33
Simulación 3	121.86	48.25
Media	136.86	49.78

Tabla 8.2: Tabla de resultados de la simulación

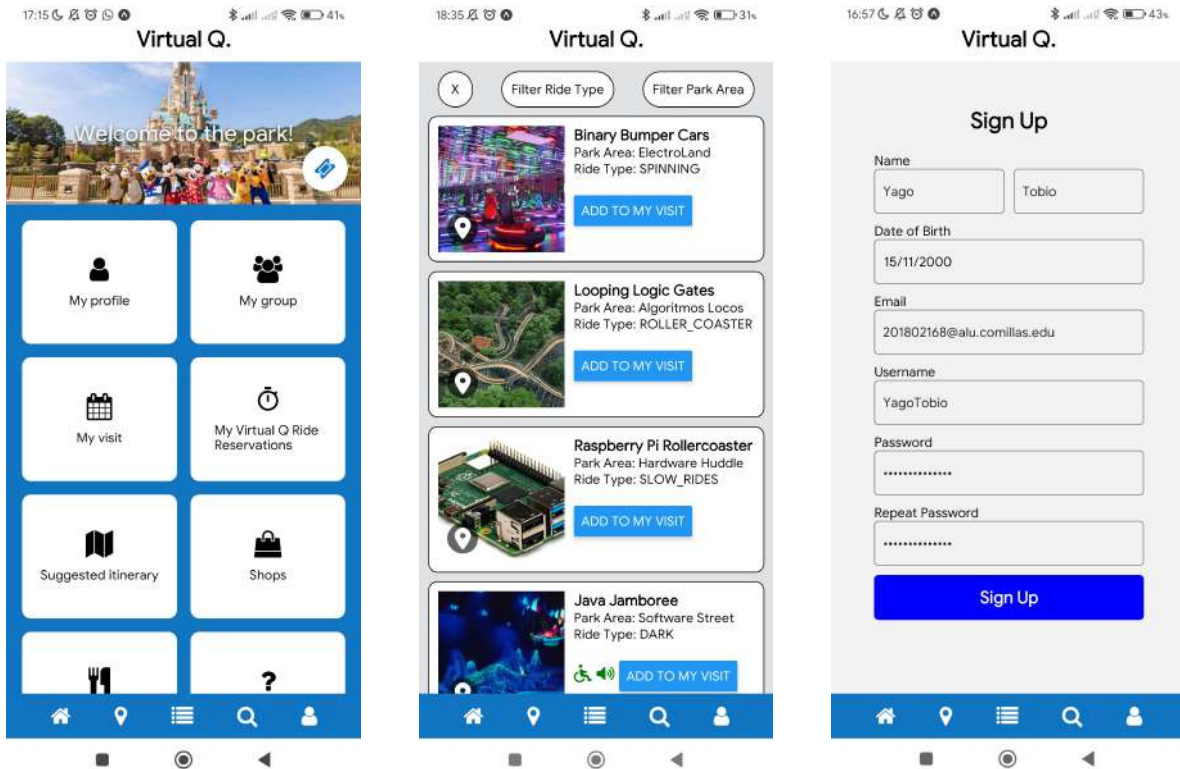
Como se puede observar, esta simulación ha simulado un escenario ideal en el cual se tiene en cuenta el tiempo de espera para las colas físicas para ambos sistemas, ya que con el sistema Virtual Q. el tiempo de espera es mínimo. Con este sistema de colas, se observa una mejora del 36 % en tiempos de espera físicos. Lo cual es una mejora notable, indicando que en días de alta demanda este servicio podría llegar a ser imperativo para mejorar la experiencia del usuario.

Aunque en este trabajo no se ha desarrollado un nuevo sistema de colas en sí, se ha implementado una plataforma que sirve como base para su incorporación.

8.2. Análisis de la aplicación móvil front-end

La aplicación se ha diseñado con el objetivo de proporcionar una experiencia útil, accesible, informativa y personalizable para los usuarios en el día de su visita al parque temático. A continuación, discutimos las características clave y los aspectos de diseño de nuestra aplicación.

Primero, la aplicación proporciona todas las vistas que se creían necesarias para maximizar la experiencia del usuario. Estas vistas abarcan desde la información general del parque hasta los detalles específicos de las atracciones y servicios disponibles. Hemos tenido cuidado de incluir información relevante y actualizada para que los usuarios puedan planificar y optimizar su visita de manera eficaz. Aseguramos que estas vistas sean intuitivas y de fácil navegación para proporcionar una experiencia de usuario sin problemas.



(a) Dashboard del usuario

(b) Vista de atracciones

(c) Vista creación de cuenta

Figura 8.1: Showcase de vistas de la aplicación

Además, antes de aceptar cualquier registro de nueva información de los usuarios, se ha implementado una verificación extensa de los datos. Este proceso de validación de datos asegura que la información introducida por los usuarios tiene el formato correcto y evita entradas duplicadas cuando sea necesario. Este paso es esencial para mantener la integridad de los datos y proporcionar resultados consistentes a los usuarios.

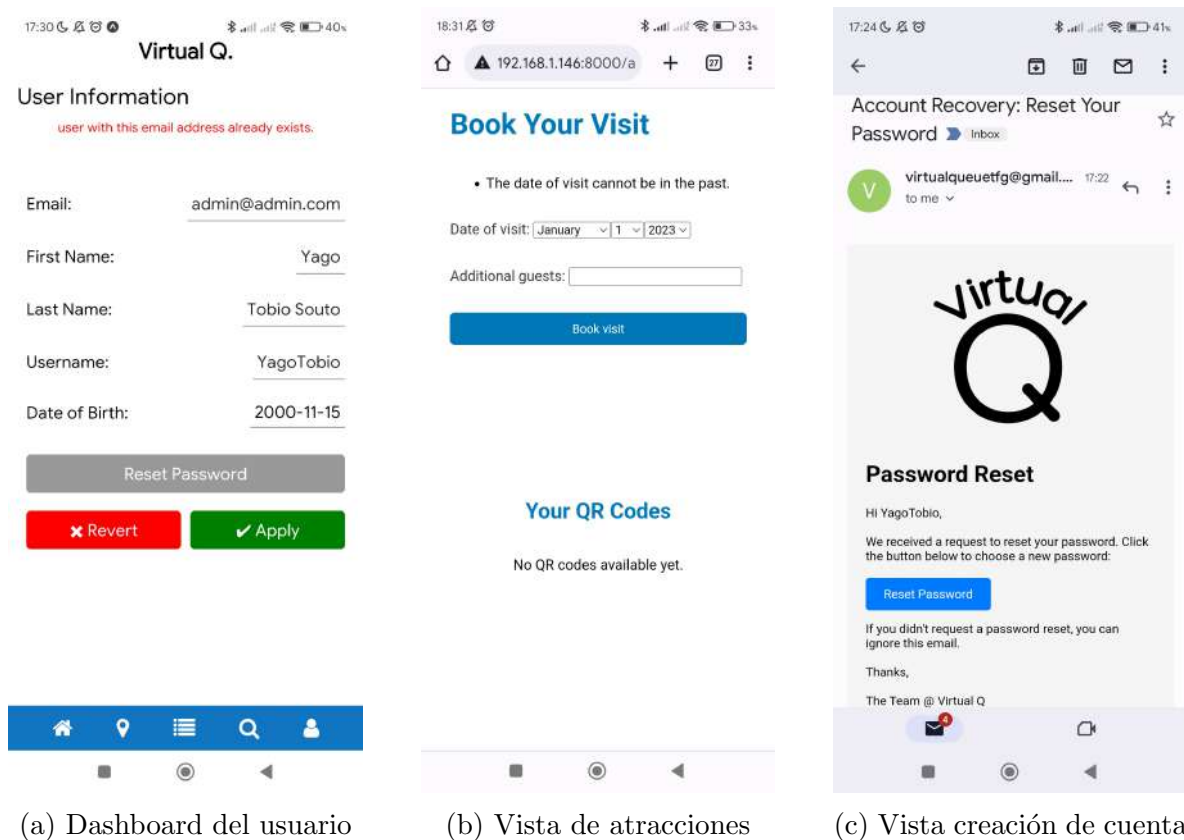


Figura 8.2: Verificación de Datos

Por último, Virtual Q. se diseñó teniendo en cuenta el estado actual de la tecnología y las expectativas de los usuarios. A través de un análisis del estado del arte, hemos incorporado todas las características que consideramos esenciales para los visitantes del parque. Esto incluye, entre otras cosas, información en tiempo real sobre las atracciones, el manejo de usuarios, sus entradas y sus itinerarios.

En conclusión, nuestra aplicación React Native está diseñada para ser una guía integral y fácil de usar para los visitantes del parque temático. Con una amplia gama de vistas útiles, verificación rigurosa de datos y características basadas en análisis de vanguardia, nuestra aplicación está equipada para mejorar la experiencia de los visitantes de manera significativa.

8.3. Análisis del back-end y el dashboard del administrador

En esta sección, se explora el desarrollo y diseño del back-end, construido sobre el robusto framework Django, y el panel de administración asociado, que permite una gestión eficaz y una visión clara de las operaciones del parque temático.

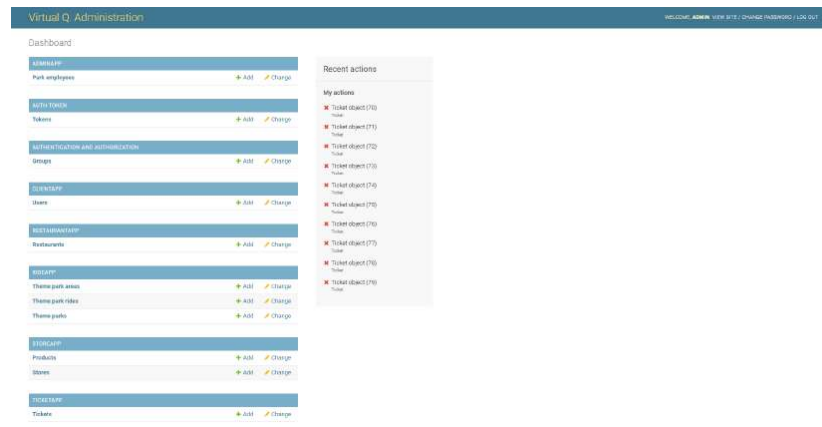


Figura 8.3: Dashboard Administrador

Comenzamos con un enfoque de microservicios para el desarrollo de nuestro back-end. Esta arquitectura nos ha permitido dividir las bases de datos y los métodos de manera intuitiva, clara y organizada, lo que permite una escalabilidad y mantenimiento más sencillos. Cada microservicio es responsable de una funcionalidad específica y tiene su propia base de datos, lo que evita el acoplamiento estrecho y mejora la coherencia y la confiabilidad del sistema.

Imágenes de cómo se dividen los microservicios y cómo se estructuran las bases de datos podrían insertarse aquí para dar una idea más clara de cómo funciona esto

Para garantizar que nuestro back-end funciona correctamente y proporciona los resultados esperados, hemos implementado tests unitarios para los métodos CRUD (Create, Read, Update, Delete) de selectos modelos desarrollado dentro de los microservicios. Estos tests nos permiten identificar y corregir errores temprano en el ciclo de desarrollo y asegurar la calidad y confiabilidad del sistema.

```
PS C:\Users\ytobi\Desktop\OkayForRealThisTime\VirtualQ-Test-2> python manage.py test
Found 5 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....
-----
Ran 5 tests in 6.016s

OK
Destroying test database for alias 'default'...
```

Figura 8.4: Resultado de tests unitarios exitosos

Además, hemos desarrollado una API que facilita el acceso y la manipulación de datos. A través de esta API, somos capaces de realizar operaciones complejas sin perjudicar el rendimiento de la aplicación. Esto también proporciona una capa adicional de abstracción entre el front-end y el back-end, lo que facilita la adaptabilidad y la escalabilidad del sistema.

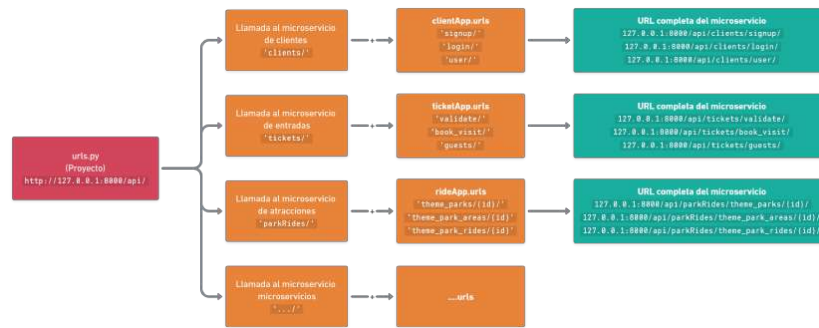


Figura 8.5: Construcción de la API

Todos los principios aplicados en el desarrollo de nuestro back-end se ven reflejados en el dashboard del administrador construido por Django, ya que una vez que todos los campos de los modelos y las bases de datos estén completos, estarán listos para su gestión a través de él.

En conclusión, nuestro back-end Django y el dashboard del administrador se han desarrollado con un enfoque en la claridad, la organización y la eficiencia. Con un diseño basado en microservicios, pruebas unitarias exhaustivas, una API robusta y la eliminación de redundancias, hemos creado un sistema sólido y confiable que respalda nuestra aplicación de front-end y mejora la gestión de las operaciones del parque temático.

Capítulo 9

Trabajo Futuros

En el próximo capítulo, se presentarán las conclusiones y se identificarán las tareas futuras necesarias para alcanzar plenamente los objetivos de este proyecto. Se realizará una evaluación exhaustiva de los objetivos trazados al comienzo del proyecto, y se medirán en relación con los logros obtenidos hasta la fecha. De igual manera, se explorarán las perspectivas y posibilidades de expansión futura del proyecto, esbozando las áreas potenciales de crecimiento y desarrollo continuo. Nuestro objetivo es perpetuar el crecimiento y evolución de este proyecto, consolidando sus fortalezas actuales y trascendiendo cualquier obstáculo que se haya presentado durante su ejecución.

9.1. Conclusiones y resultados principales

El motivo principal por el desarrollo del Trabajo de Final de Grado se ha logrado. Observando el Capítulo 6, se puede observar que se ha llegado a generar una aplicación dual tanto para el administrador de los parques de atracciones, como para el usuario. En el cual se tienen actualizaciones en tiempo real y un sistema de reservas implementado.

Analizando los objetivos expuestos en el capítulo 4.2 se puede verificar lo siguiente:

- **Creación de un Sistema de Colas Virtuales:** Se ha logrado crear un sistema de gestión para asignar a usuarios del sistema a horarios particulares en las atracciones. En dicho sistema, se ha tenido en cuenta la capacidad de la atracción, el horario y los miembros ya presentes de manera eficiente y dinámica.
- **Desarrollo de una aplicación web para la gestión de parques de atracciones:** Se ha desarrollado la aplicación web a la que solo tendrán acceso los administradores del parque. En ella se pueden añadir regiones de parques, atracciones individuales, junto con todas sus características (Título, Accesibilidad, Descripción, Ubicación, Horario para las colas virtuales, etc.) en la cual se modifican los contenidos de la REST API cuya información se le muestra al público en tiempo real en la aplicación móvil.
- **Desarrollo de una aplicación móvil para el usuario:** Se ha desarrollado una aplicación móvil con React Native. Es especialmente útil para los usuarios los días antes y el día de su visita al parque de atracciones. En ella se puede crear

un perfil y proceder con las siguientes funcionalidades: gestionar sus entradas del parque, crear itinerarios basados en grupos, reservar plazas en las atracciones, y gestionar el resto de su visita y su información, entre otros... , observando todo lo que tiene que ofrecer el parque.

9.2. Trabajos Futuros

9.2.1. Desarrollo de mapa virtual - GPS

La futura expansión de nuestra aplicación incorporará un sofisticado mapa virtual interactivo, impulsado por la avanzada tecnología GPS. Este mapa revolucionará la forma en que los usuarios visualizan y navegan por el parque de atracciones, proporcionándoles una comprensión clara e intuitiva de su distribución y permitiéndoles ubicar de manera eficiente las diversas atracciones, instalaciones y servicios que ofrece el parque.

Los distintos puntos de interés estarán representados por iconos personalizados, diseñados para permitir a los usuarios identificar de manera instantánea y sencilla lo que cada uno representa en el mapa. En respuesta a la necesidad de una visualización más clara, se implementarán filtros que simplifiquen y clarifiquen la vista del mapa.

Además, la aplicación permitirá a los usuarios personalizar su experiencia mediante la función de favoritos. Al agregar atracciones, tiendas o restaurantes a sus listas de favoritos, estos puntos de interés se resaltarán de manera constante en el mapa, asegurando que estén siempre al alcance.

Este módulo mejorará drásticamente la experiencia del usuario en el parque. Al facilitar la orientación y minimizar el tiempo dedicado a la búsqueda de atracciones y servicios, permitirá a los visitantes aprovechar al máximo su tiempo, haciendo que cada visita sea aún más memorable.

9.2.2. Visualización de tiendas y restaurantes e implementación Pay2Go

El siguiente escalón en nuestro plan de expansión implica una mejora sustancial en las funcionalidades relativas a tiendas y restaurantes dentro de nuestra aplicación. La aplicación permitirá a los usuarios explorar el abanico completo de tiendas y restaurantes disponibles en el parque de atracciones, brindando detalles como menús, horarios de servicio y ubicaciones físicas.

Una adición crítica en esta fase será la integración de la funcionalidad Pay2Go, una solución de pago integral que permitirá a los usuarios realizar transacciones directamente desde la aplicación. Esta funcionalidad agilizará la experiencia de compra, desde la adquisición de entradas hasta la reserva y pago de comidas en restaurantes o la compra de productos en tiendas, todo ello sin tener que hacer largas colas o manipular dinero físico.

Para asegurar una interfaz de usuario rápida e intuitiva, Pay2Go se integrará con los sistemas de pago más comunes y eficientes del mercado, como Apple Pay, Google Pay y la tecnología NFC de los dispositivos. Este nivel de integración permite a los usuarios

disfrutar de una experiencia de transacción simplificada, segura y sin complicaciones, enriqueciendo de manera significativa su experiencia general en el parque.

9.2.3. Aplicación Empleado - Scanner QR y gestión particular

Para potenciar la eficiencia operativa, nuestro plan incluye la implementación de una aplicación específicamente diseñada para el personal del parque de atracciones. Esta aplicación diferenciada de la de los usuarios, ofrecerá funcionalidades que agilizarán las tareas diarias, incluyendo un lector de códigos QR para autenticar entradas y reservas de atracciones de manera rápida y segura.

Además, cada empleado podrá gestionar aspectos específicos de su área de responsabilidad, ya sea una atracción, restaurante o tienda. Esta herramienta les permitirá manejar sus tareas de una forma más autónoma y eficiente, asegurando un flujo operativo suave que redundará en una experiencia óptima para el visitante.

Otro elemento clave en este desarrollo será la inclusión de un canal de comunicación directa con los supervisores o gerentes del parque. Mediante una función de chat instantáneo, los empleados podrán reportar incidencias o resolver dudas en tiempo real, lo que permitirá tomar acciones correctivas rápidas, minimizando los posibles contratiempos y mejorando el desempeño general del parque.

9.2.4. Desarrollo de generador de itinerarios usando Machine Learning basado en las características del grupo

Una de las funciones futuras que se planifica introducir es un generador de itinerarios impulsado por Machine Learning. Este algoritmo tomará en cuenta las preferencias y características del grupo del usuario generado al comprar las entradas y sugerir el itinerario óptimo para su visita al parque de atracciones. Esta función de personalización mejorará la experiencia del usuario, permitiendo a los visitantes maximizar su tiempo y disfrutar de lo mejor que el parque tiene para ofrecer,

9.2.5. Dashboard de Analíticas para los administradores del parque de atracciones

Para los administradores del parque, se desarrollará un panel de análisis de datos altamente robusto y versátil. Este instrumento proporcionará a los administradores un acceso inmediato a una diversidad de métricas y a indicadores clave de rendimiento (KPI, por sus siglas en inglés), que servirán como herramientas fundamentales para la toma de decisiones informadas y para la planificación estratégica.

Este panel de análisis de datos se alimentará tanto de información recolectada por la interacción de los clientes con la aplicación, como de los datos generados por la gestión interna del parque. Este conjunto de información incluirá datos recolectados cuando los clientes utilicen la aplicación, cuando escaneen códigos QR en los puntos de entrada y reserva, y mediante la funcionalidad GPS integrada, que rastrea el movimiento de los usuarios cuando tienen la aplicación abierta en el parque.

A través de este seguimiento detallado y continuo, se podrá obtener una visión clara del flujo y comportamiento general de los clientes durante el día. Esta valiosa analítica permitirá a los administradores optimizar la operación y planificación del parque de manera proactiva y basada en datos concretos.

Los administradores tendrán la capacidad de monitorear el rendimiento de las atracciones, la eficacia de las campañas de marketing, los patrones de tráfico de visitantes, entre otros aspectos críticos. De esta manera, se transformará la manera en que los administradores perciben y responden a las dinámicas del parque, con un enfoque basado en datos y enfocado en la mejora continua.

9.2.6. Deployment de la aplicación en la nube y en los mercados de aplicaciones móviles

Para optimizar la accesibilidad y la eficacia de nuestra aplicación, se planifica su despliegue tanto en la nube como en las principales plataformas de aplicaciones móviles. Este enfoque estratégico permitirá a los usuarios acceder a la aplicación independientemente de su ubicación y en cualquier momento, potenciando la flexibilidad y comodidad para el usuario. Más allá de esto, el alojamiento en la nube asegurará la escalabilidad, accesibilidad y seguridad de la aplicación, fundamentales para su éxito y crecimiento.

Ahora bien, en cuanto al despliegue de una aplicación con back-end de Django y Front-end de React Native, el proceso sería el siguiente:

1. **Preparación del back-end de Django para la implementación:** Antes de desplegar la aplicación, es necesario asegurar que el back-end de Django esté correctamente configurado para producción. Esto incluye la configuración de los ajustes de seguridad, la base de datos, y los archivos estáticos y multimedia.
2. **Despliegue del back-end de Django en un servidor en la nube:** Una vez que el back-end de Django está listo para producción, se puede desplegar en un servidor de AWS o Google Cloud, usando de antemano Docker y Kubernetes para manejar los distintos microservicios. Este proceso implica la creación y configuración de una instancia de servidor, la instalación de dependencias necesarias, la transferencia del código de la aplicación al servidor, y la configuración del servidor para ejecutar la aplicación.
3. **Preparación del front-end de React Native para la implementación:** De manera similar, antes de desplegar la aplicación de React Native, es necesario prepararla para la producción. Esto incluye la configuración de los ajustes de seguridad, la generación de los archivos de construcción, y la preparación de los íconos y las pantallas de inicio.
4. **Despliegue de la aplicación de React Native en Google Play y App Store:** Una vez que la aplicación de React Native está lista para la producción, se puede enviar a Google Play para Android y a la App Store para iOS. Este proceso implica la creación de una cuenta de desarrollador, la preparación de los materiales de la tienda de aplicaciones (como las capturas de pantalla, la descripción, y los términos de servicio), la carga de la aplicación, y la solicitud de revisión y publicación.

9.2.7. Desarrollo del sistema de colas virtual y refinamiento basado en modelos reales

Por último, se planifica desarrollar un nuevo sistema de colas y trabajar en su constante refinamiento para reflejar modelos de datos reales una vez se implemente la aplicación en el mundo real. Este sistema permitirá a los usuarios reservar su lugar en la fila de una atracción de manera virtual, mejorando la eficiencia del flujo de visitantes y reduciendo los tiempos de espera. Se continuará perfeccionando este sistema basándonos en los comentarios de los usuarios y en los patrones observados para garantizar que proporciona la mayor comodidad posible. Pudiendo generar modelos de Machine Learning reforzados para manejar la demanda del parque en horas clave.

Capítulo 10

Bibliografia

Bibliografía

- Apple (2023). Submit your ios apps to the app store.
- Ashtari, H. (2022). Key differences between ci/cd and devops.
- Asmussen, S. R. (2003). *Markov Additive Models*, volume 51 of *Stochastic Modelling and Applied Probability*, pages 302–339.
- Asthana, A. (2015). Poisson queueing system.
- Atlassian (2014). Microservices vs. monolithic architecture.
- Atlassian (2017). Kubernetes vs. docker.
- Awati, R. and Wigmore, I. (2022). What is monolithic architecture in software?
- BBVA (2018). Que es la metodologia agile.
- Camps, U. A. B. (2021). Html, css, and javascript.
- Computers, L. (2013). Lenovo ideapad 110s.
- Dahl, R. (2010). Node.js: About.
- Domareski, H. S. (2020). Dry, kiss & yagni principles.
- Engineering, S. (2013). System engineering 3(3+0).
- es.talent.com (2023). Salario medio para desarrollador full stack en españa, 2023.
- Expo (2018). Expo documentation: Create amazing apps that run everywhere.
- Foundation, D. S. (2018). Django overview.
- Giri, B. C. (2020). Chapter 6: Queueing theory.
- Gittins, J. C. (1989). *Multi-armed Bandit Allocation Indices*. John Wiley & Sons, New York.
- Hat, R. (2018). What is an api?
- Javatpoint (2019). Sqlite advantages and disadvantages.
- Kendall, D. G. (1953). Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded markov chain. *The Annals of Mathematical Statistics*, 24(3):338–354.
- Kinsta (2022). What is github? a beginner’s introduction to github.

- Kotlin (2017). What is cross-platform mobile development?
- Krishna, A. (2023). Yarn vs npm: Which one is best to choose?
- LogicMonitor (2022). Rest vs crud.
- Manoj, K. (2020). Operations research: Unit 5 (queuing theory) lecture notes.
- Marcak, M. (2023). React native pros and cons - 2023 updated.
- Meta, F. (2015). React.
- Meta, F. (2016a). React: usecontext.
- Meta, F. (2016b). React: useeffect.
- Meta, F. (2016c). React: useref.
- Meta, F. (2016d). React: usestate.
- Meta, F. (2017). React native: Learn once, write anywhere.
- Microsoft (2021). Documentation for visual studio code.
- Oracle (2017). What is the internet of things (iot)?
- Oracle (2021). Theme parks get back on track.
- Organization, G. (2020). About git.
- Ramamurthy, P. (2007). *Operations Research*. New Age International.
- Services, A. W. (2018a). What are microservices?
- Services, A. W. (2018b). What is virtualization?
- Services, A. W. (2023). Aws pricing.
- SQLite (2019). About sqlite.
- Xiaomi (2021). Xiaomi redmi note 11.

Capítulo 11

Anexo - Manual de Instalación

Este manual está destinado a guiar a los usuarios en el proceso de instalación y configuración del proyecto de final de grado, permitiendo su visualización y funcionamiento correcto en el entorno local de cada usuario. Por favor, siga las instrucciones cuidadosamente para garantizar la correcta implementación del proyecto.

11.1. Instalación del Back-end y del entorno de desarrollo

11.1.1. Instalación de Visual Studio Code

Comenzaremos por instalar Visual Studio Code (VS Code). Este es un editor de código fuente desarrollado por Microsoft, el cual será el medio principal para interactuar con el código del proyecto.

- Para descargar VS Code, visite la página oficial: <https://code.visualstudio.com/>
- Elija la versión apropiada para su sistema operativo y siga las instrucciones de instalación proporcionadas.

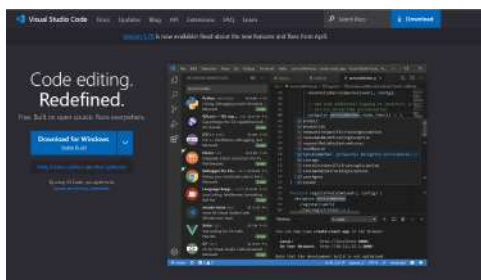


Figura 11.1: Pantalla de descarga Visual Studio Code



Figura 11.2: Vista de Visual Studio Code recién arrancado

11.1.2. Instalación de Python y pip

El siguiente paso consiste en instalar Python, el lenguaje de programación principal de este proyecto, y pip, el sistema de gestión de paquetes utilizado para instalar y administrar paquetes de software escritos en Python.

- Descargue e instale la última versión de Python desde la página oficial: <https://www.python.org/downloads/>
- En caso de estar utilizando el sistema operativo de Windows, el cual se ha utilizado para el desarrollo de este proyecto, también se puede navegar a la Microsoft Store y descargarse la última versión de Python ahí. La cual tiene como ventaja que se irá actualizando de manera automática sin la necesidad de hacerlo manualmente por el usuario.

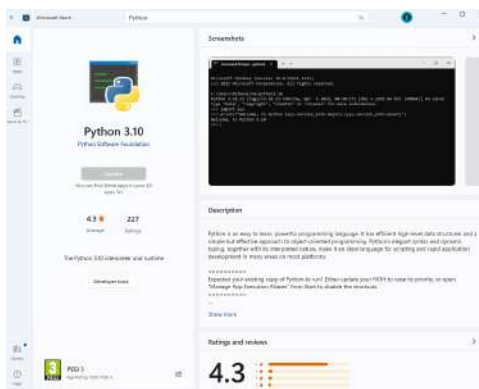


Figura 11.3: Microsoft Store Python instalación

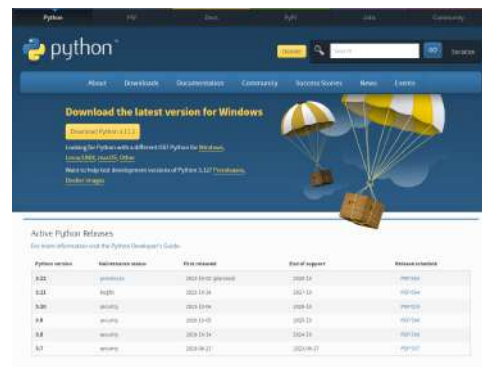


Figura 11.4: Python instalación desde su página web

- Asegúrese de que pip esté instalado junto con Python. Por defecto, pip se instala con Python. Para ello se debe de navegar a la terminal y ejecutar los siguientes dos comandos:

1. `python --version`
2. `pip --version`



Figura 11.5: Verificación de la instalación de Python y pip

- Compruebe la correcta sincronización entre Python, pip y VS Code. Para ello, al abrir Visual Studio Code, debe de presionar las teclas `Ctrl+Shift+P` y escribir `Python: Select code interpreter`

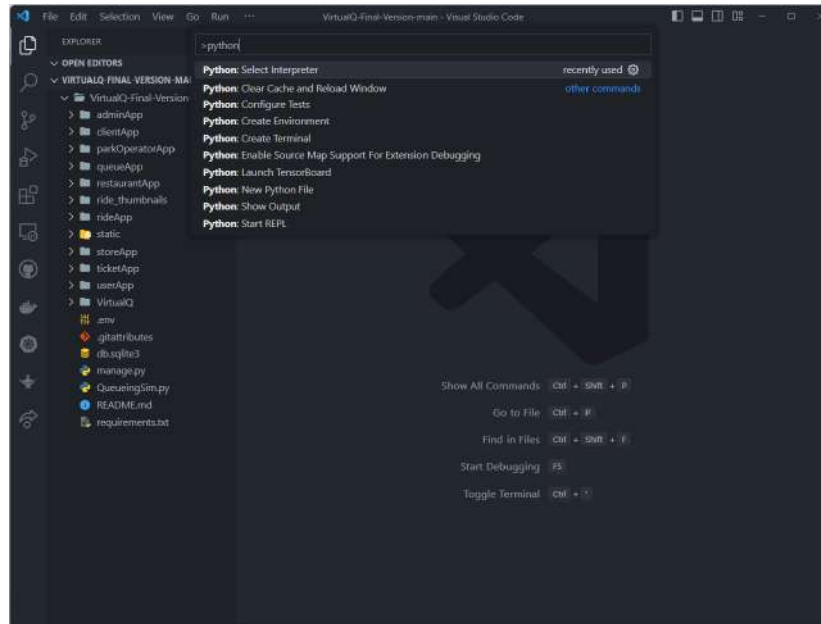


Figura 11.6: Python instalación desde su página web

11.1.3. Descarga del proyecto

Ahora debemos obtener una copia del proyecto. Para esto, utilizaremos GitHub, una plataforma de desarrollo colaborativo que almacena repositorios de código fuente.

- Visite el repositorio oficial del proyecto: <https://github.com/SaltyYagi123/VirtualQ-Final-Version>
- Haga clic en el botón 'Code' y luego en 'Download ZIP'.
- Una vez descargado, extraiga los archivos del repositorio en la ubicación de su preferencia.

11.1.4. Abriendo el proyecto en Visual Studio Code

Con el proyecto descargado y descomprimido, es hora de abrirlo en Visual Studio Code.

- Abra VS Code.
- Haga clic en "File» .°pen Folder» navegue hasta la carpeta donde ha extraído los archivos del repositorio.
- Seleccione la carpeta y haga clic en ".°pen".

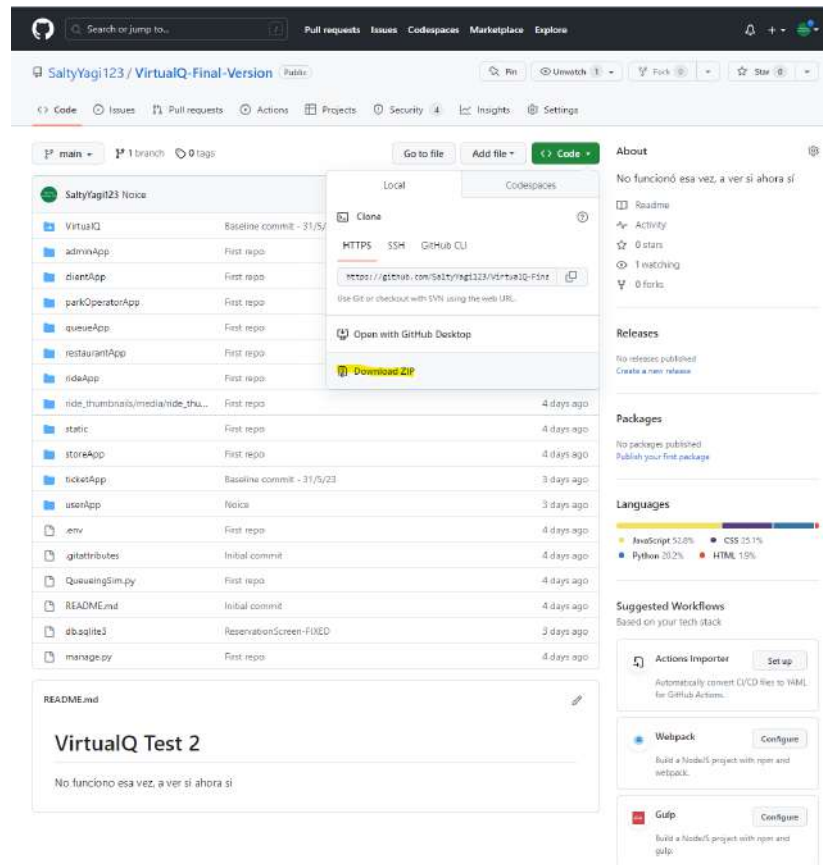


Figura 11.7: Descarga del proyecto desde GitHub



Figura 11.8: Abrir el proyecto a través de Visual Studio Code

11.1.5. Abriendo una terminal con permisos de administrador

Para la instalación de paquetes necesarios, requeriremos una terminal con permisos de administrador. Abrirlo como administrador nos permitirá instalar los paquetes necesarios para ejecutar la instalación.

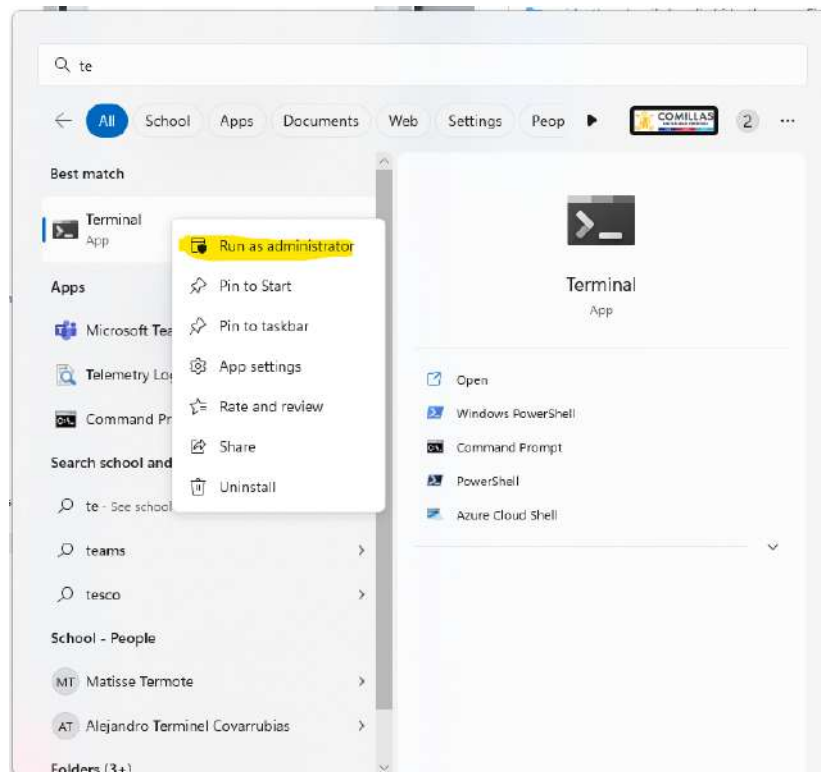


Figura 11.9: Abrir la terminal con permisos de administrador para poder instalar paquetes

11.1.6. Navegación al directorio del proyecto

Una vez abierta la terminal, es necesario navegar hasta el directorio base del proyecto. En la terminal, utilice el comando `cd` seguido de la ruta al directorio del proyecto. Por ejemplo, si el proyecto se encuentra en la carpeta 'Proyecto' en el escritorio, el comando

sería `cd Desktop/Proyecto`. Tras esto deberá de navegar al directorio tal que se ubique aquí:

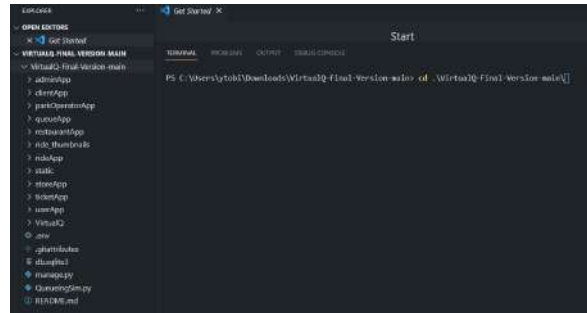


Figura 11.10: Abrir el directorio correspondiente para poder instalar el back-end

11.1.7. Instalación de los requisitos del proyecto

Finalmente, debemos instalar los paquetes de software necesarios para que el proyecto funcione correctamente.

- En la terminal, ejecute el comando `pip install -r requirements.txt`. Este comando instala todos los paquetes necesarios que están enumerados en el archivo `'requirements.txt'`.
- Una vez completada la instalación, estará listo para arrancar el servidor.

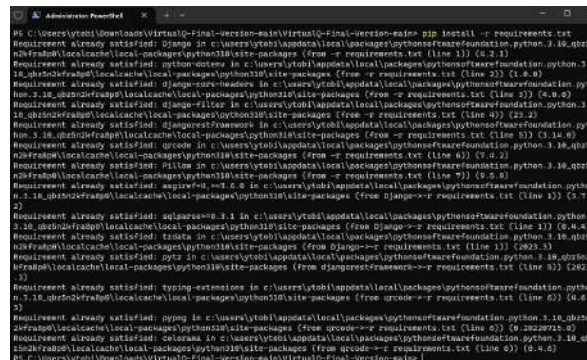


Figura 11.11: Instalar los requisitos para poder ejecutar el back-end de manera correcta.

11.1.8. Arranque del servidor

Una vez que estos pasos se hayan ejecutado correctamente, ya tendrá todos los requisitos para arrancar el servidor Django, el cual se hará ejecutando el siguiente comando:

```
python manage.py runserver 0.0.0.0:8000
```

Tras esto, observaremos la siguiente respuesta por parte de la terminal, lo cual implica que se ha ejecutado correctamente:



```
PS C:\Users\ytobi\Downloads\VirtualQ-Final-Version-main\VirtualQ-Final-Version-main> python manage.py runserver 0.0.0.0:8000
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
Sun 03, 2023 - 15:49:32
Django version 3.2.3, using settings 'VirtualQ.settings'
Starting development server at http://0.0.0.0:8000/
Quit the server with Ctrl-C.
```

Figura 11.12: Comando para ejecutar el back-end correctamente.

Una vez hecho, se abrirá el navegador y se navegará al siguiente url: `127.0.0.0:8000/admin`. En la cual se observara la siguiente vista:

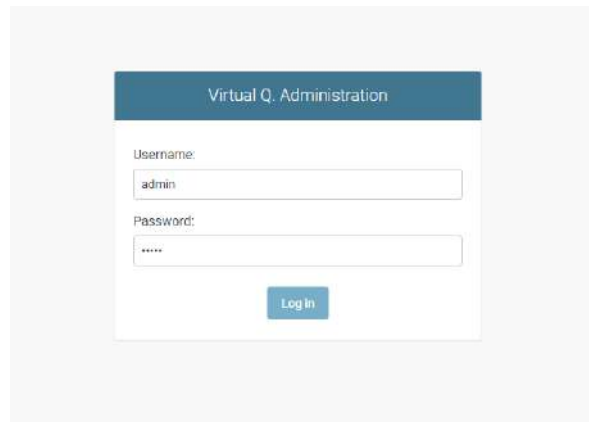
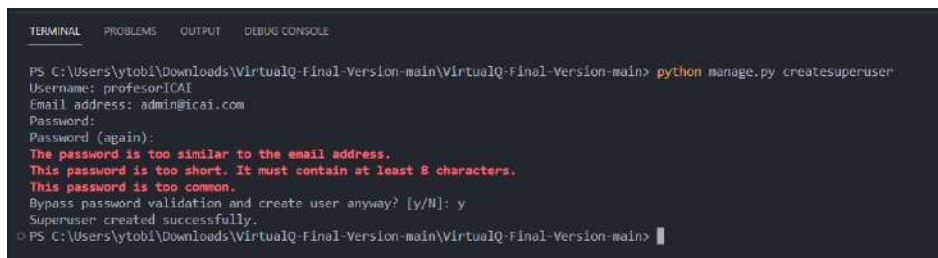


Figura 11.13: Vista inicio sesión para el administrador.

En este caso se para visualizar el dashboard del administrador se pueden insertar usuario y contraseña 'admin'. Pero en caso de querer crearse una cuenta nueva, puede cerrar el servidor en la terminal, usando `Ctrl+C`, y ejecutando el siguiente comando:

`python manage.py createsuperuser`

En el cual podrá insertar los datos de su cuenta de administrador de la siguiente manera:



```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE

PS C:\Users\ytobi\Downloads\VirtualQ-Final-Version-main\VirtualQ-Final-Version-main> python manage.py createsuperuser
Username: profesorICAI
Email address: admin@icai.com
Password:
Password (again):
The password is too similar to the email address.
This password is too short. It must contain at least 8 characters.
This password is too common.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
PS C:\Users\ytobi\Downloads\VirtualQ-Final-Version-main\VirtualQ-Final-Version-main>
```

Figura 11.14: Comando para crear un superuser mediante la terminal.

y después reiniciar el servidor para poder volver a iniciar la sesión en el dashboard.

11.2. React Native (front-end)

Este manual está destinado a proporcionar instrucciones detalladas para la configuración de una aplicación de React Native utilizando Yarn y Expo. Asegúrese de seguir cada paso cuidadosamente para una implementación correcta.

11.2.1. Instalación de Node.js y npm

Empezaremos instalando Node.js, una plataforma de código abierto que permite la ejecución de JavaScript en el servidor y la creación de aplicaciones en red escalables, y npm, su administrador de paquetes.

- Para descargar Node.js, visite la página oficial: <https://nodejs.org/es/download/>
- Elija la versión apropiada para su sistema operativo y siga las instrucciones de instalación proporcionadas.
- npm se instala automáticamente con Node.js.



Figura 11.15: Navegación a la página de Node.js



Figura 11.16: Instalación de Node.js

11.2.2. Verificación de la instalación

Es esencial verificar que tanto Node.js como npm se hayan instalado correctamente.

- Para comprobar la instalación de Node.js, abra una terminal y ejecute el comando `node -v`. Este comando debería mostrar la versión de Node.js que ha instalado.
- Para verificar la instalación de npm, ejecute en la terminal el comando `npm -v`. Este comando debería mostrar la versión de npm instalada.

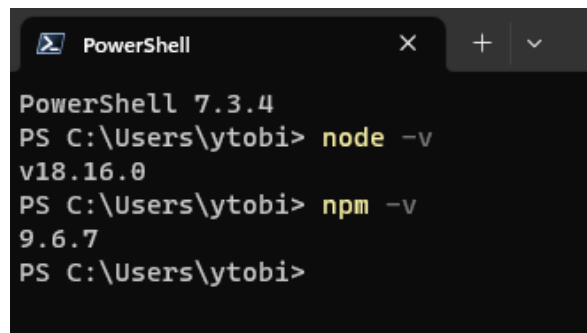


Figura 11.17: Verificación de instalación de Node.js y npm

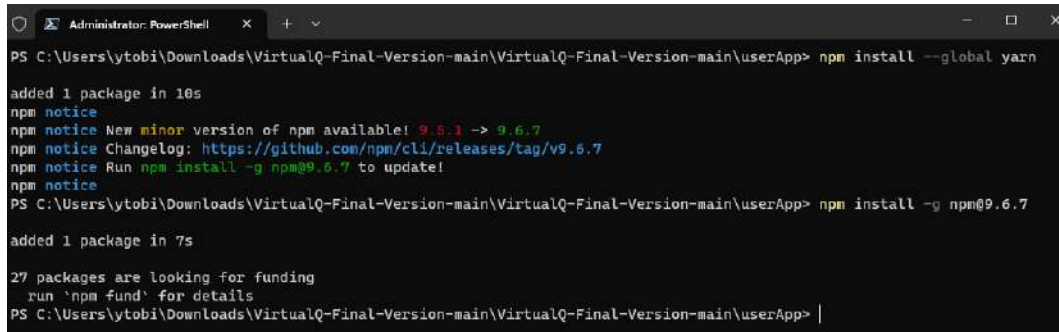
11.2.3. Reinicio del sistema

Después de la instalación de Node.js y npm, es recomendable reiniciar el sistema para asegurar que Node.js se integre correctamente en el PATH de su sistema.

11.2.4. Instalación de Yarn

Yarn es un gestor de paquetes que también funciona como gestor de proyectos. Utiliza el repositorio npm para su base de datos de paquetes.

- Para instalar Yarn, abra una terminal y ejecute el comando `npm install -g yarn`.



```

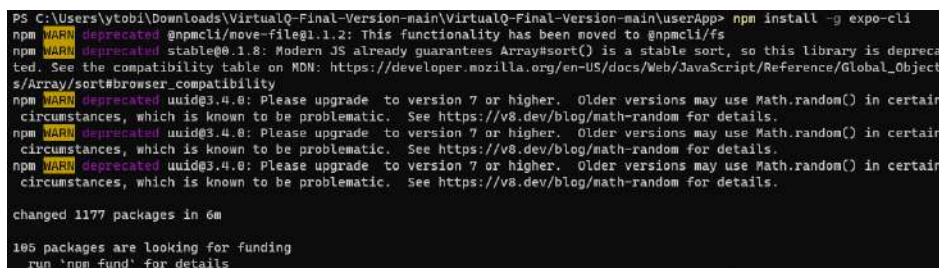
PS C:\Users\ytobi\Downloads\VirtualQ-Final-Version-main\VirtualQ-Final-Version-main\userApp> npm install --global yarn
added 1 package in 10s
npm notice
npm notice New minor version of npm available! 9.5.1 -> 9.6.7
npm notice Changelog: https://github.com/npm/cli/releases/tag/v9.6.7
npm notice Run npm install -g npm@9.6.7 to update!
npm notice
PS C:\Users\ytobi\Downloads\VirtualQ-Final-Version-main\VirtualQ-Final-Version-main\userApp> npm install -g npm@9.6.7
added 1 package in 7s
27 packages are looking for funding
  run 'npm fund' for details
PS C:\Users\ytobi\Downloads\VirtualQ-Final-Version-main\VirtualQ-Final-Version-main\userApp> |
  
```

Figura 11.18: Instalación de Yarn.

11.2.5. Instalación de Expo CLI

Expo es una herramienta construida sobre React Native que facilita la construcción de aplicaciones.

- Para instalar Expo CLI, en la misma terminal ejecuta el comando `npm install -g expo-cli`



```

PS C:\Users\ytobi\Downloads\VirtualQ-Final-Version-main\VirtualQ-Final-Version-main\userApp> npm install -g expo-cli
npm WARN deprecated @npmcli/move-file@1.1.2: This functionality has been moved to @npmcli/fs
npm WARN deprecated stable@0.1.8: Modern JS already guarantees Array#sort() is a stable sort, so this library is deprecated. See the compatibility table on MDN: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Array/sort#browser\_compatibility
npm WARN deprecated uuid@3.4.0: Please upgrade to version 7 or higher. Older versions may use Math.random() in certain circumstances, which is known to be problematic. See https://v8.dev/blog/math-random for details.
npm WARN deprecated uuid@3.4.0: Please upgrade to version 7 or higher. Older versions may use Math.random() in certain circumstances, which is known to be problematic. See https://v8.dev/blog/math-random for details.
npm WARN deprecated uuid@3.4.0: Please upgrade to version 7 or higher. Older versions may use Math.random() in certain circumstances, which is known to be problematic. See https://v8.dev/blog/math-random for details.
changed 1177 packages in 6m
105 packages are looking for funding
  run 'npm fund' for details
  
```

Figura 11.19: Instalación del Expo CLI - entorno de desarrollo front-end.

11.2.6. Instalación de las dependencias del proyecto

Ahora debemos instalar las dependencias necesarias para el proyecto front-end:

- En la terminal, navegue hasta el directorio de su proyecto utilizando el comando `cd`. En nuestro caso, se debe de ubicar en el directorio base del proyecto y meterese en la `userApp`
- Una vez en el directorio correcto, ejecute el comando `yarn add expo`.

```

PS C:\Users\ytobi\Downloads\VirtualQ-Final-Version-main\VirtualQ-Final-Version-main\userApp> yarn add expo
yarn add v1.22.19
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...
success Saved lockfile.
success Saved 2 new dependencies.
info Direct dependencies
├─ expo@48.0.19
└─ expo/cli@0.7.3
info All dependencies
├─ @expo/cli@0.7.3
└─ expo@48.0.19
Done in 168.84s.

```

Figura 11.20: Instalación de las dependencias de la aplicación react native mediante yarn.

11.2.7. Inicio de la aplicación

Con todas las dependencias correctamente instaladas, es hora de iniciar la aplicación.

- En la terminal, ejecute el comando `yarn expo start`. Esto iniciará el servidor de desarrollo de Expo y abrirá una nueva ventana en su navegador web con un código QR.

```

PS C:\Users\ytobi\Downloads\VirtualQ-Final-Version-main\VirtualQ-Final-Version-main\userApp> yarn start
yarn run v1.22.19
$ expo start
Starting project at C:\Users\ytobi\Downloads\VirtualQ-Final-Version-main\VirtualQ-Final-Version-main\userApp
Some dependencies are incompatible with the installed expo version:
  @react-native-picker/picker@2.4.10 - expected version: 2.4.8
Your project may not work correctly until you install the correct versions of the packages.
Install individual packages by running npx expo install @react-native-picker/picker@2.4.8
Starting Metro Bundler

```



```

> Metro waiting on exp://192.168.104.25:19089
> Scan the QR code above with Expo Go (Android) or the Camera app (iOS)

> Press a | open Android
> Press w | open web
> Press j | open debugger
> Press r | reload app
> Press m | toggle menu
> Press ? | show all commands

Logs for your project will appear below. Press Ctrl+C to exit.

```

Figura 11.21: Instalación automática de los módulos necesarios para React Native con Yarn.

11.3. Instalación de la aplicación Expo en su dispositivo móvil

Para interactuar con su aplicación de React Native en su dispositivo móvil, necesitará instalar la aplicación Expo.

- Dependiendo de su dispositivo, visite la App Store (para dispositivos iOS) o Google Play Store (para dispositivos Android) y busque Expo.
- Instale la aplicación en su dispositivo móvil.

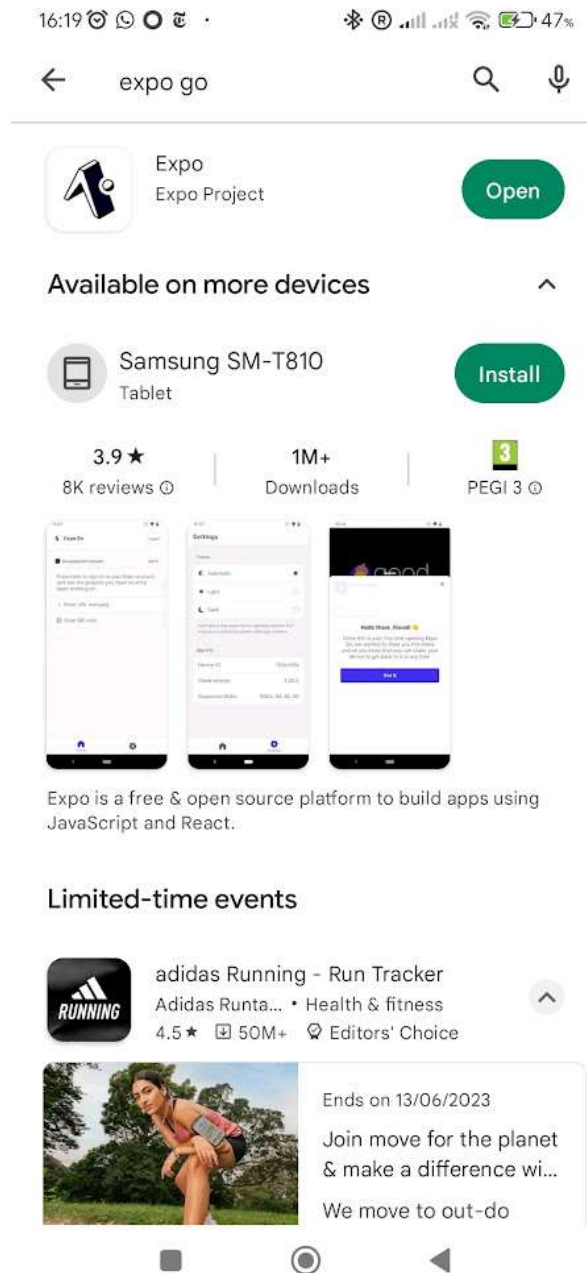


Figura 11.22: Página Google Play Store para instalarse Expo Go.

11.3.1. Escaneo del código QR

Con la aplicación Expo instalada en su dispositivo móvil, ya puede interactuar con su aplicación de React Native.

- Abra la aplicación Expo en su dispositivo móvil y utilice la opción de escaneo de

código QR.

- Apunte la cámara de su dispositivo al código QR que se muestra en la terminal.



Figura 11.23: Código QR a escanear para poder visualizar la aplicación en su móvil

Al escanear correctamente el código QR, su aplicación de React Native se cargará en su dispositivo móvil a través de la aplicación Expo. Ahora puede interactuar con su aplicación en su dispositivo móvil.

Capítulo 12

Anexo - Manual de Instalación

12.1. Introducción

Este manual ha sido desarrollado para facilitar el entendimiento y manejo de nuestra nueva Aplicación de Gestión y Reservas para Parque de Atracciones. Este software ha sido desarrollado como parte de un proyecto final de grado y tiene como objetivo brindar una experiencia fluida y eficiente tanto a los administradores del parque como a los clientes.

La aplicación posee dos componentes principales: el portal de administración y la interfaz de usuario del cliente. Ambos han sido diseñados con la máxima simplicidad y funcionalidad en mente, permitiendo una experiencia de usuario fácil e intuitiva.

12.2. Manual por parte del administrador

Este componente está destinado a los administradores del parque de atracciones, permitiendo un control y gestión completa de la información que los clientes ven en su aplicación. A través de este portal, los administradores pueden:

- Gestionar la información de las atracciones.
- Ver y editar las reservas de los clientes.
- Actualizar la disponibilidad de las atracciones.

A continuación, analizaremos las distintas vistas para poder administrar las siguientes tareas:

12.2.1. Acceso al dashboard del administrador

Una vez se ha arrancado de manera correcta el administrador, se debe de navegar a la página 127.0.0.0:8000/admin, en la cual observaremos la siguiente vista:

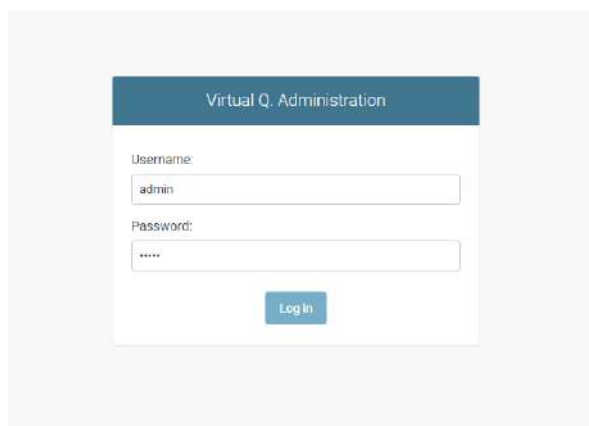


Figura 12.1: Vista inicio sesión para el administrador.

En dicha vista, el administrador debe de meter sus credenciales generadas por el sistema, para poder tener acceso al dashboard.

Una vez que haya ingresado sesión, se observará la siguiente vista con las bases de datos a manejar:

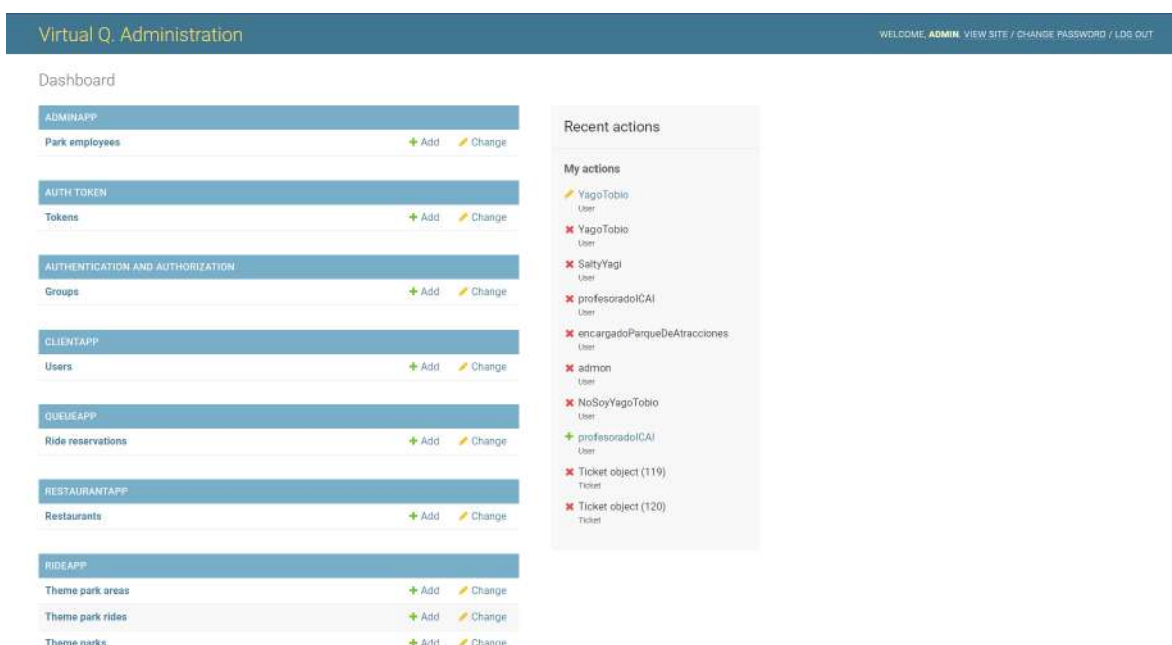


Figura 12.2: Vista inicio del administrador.

En la siguiente vista, se pueden observar en las cabeceras el nombre de los microservicios correspondientes, y bajo ellas los modelos que ha generado el sistema de Virtual Q., los cuales se han ido detallando en el capítulo 5.

En caso de desear analizar cualquiera de estos modelo, el administrador simplemente debe de hacer clic sobre el modelo deseado, el cual le abrirá una nueva vista con los detalles extensos.

Estructura de la página modelo



Figura 12.3: Vista de un modelo y sus características.

Una vez se haya abierto cualquiera de los modelos deseados, se podrán observar en la vista con todos los detalles relevantes de los componentes. En el centro de la vista ocupa las entradas de la base de datos. Cada entrada en la base de datos tiene su fila correspondiente con las columnas indicando la información de cada uno de sus atributos. En cada fila, la primera columna a mano izquierda indicará el ID particular de ese modelo, el cual es un campo seleccionable que permite editar esa fila en particular.

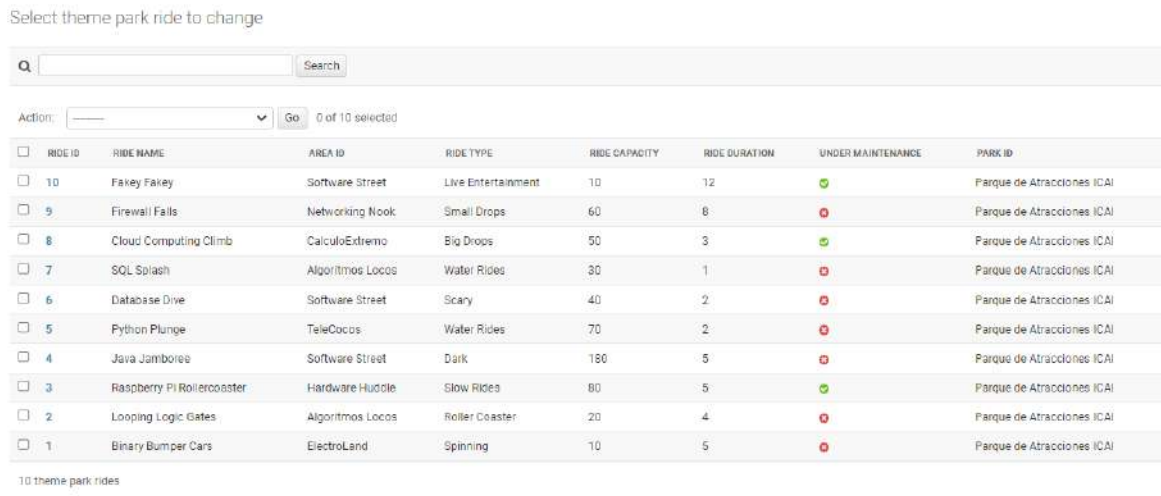


Figura 12.4: Entradas del modelo seleccionado.

Igualmente, en la primera columna, se pueden observar cajas para seleccionar varias entradas en particular, para hacer eliminaciones en grupo. Una vez se haya seleccionado las entradas queridas, podrá hacer clic sobre el rectángulo drop-down y seleccionar la opción de eliminar las entradas.

Una vez se haya ejecutado la instrucción, se le llevará a una pantalla de verificación para que le explique al administrador todos los cambios que causará borrar estas entradas. Una vez el administrador considere sus repercusiones, podrá decidir borrar definitivamente o volver.

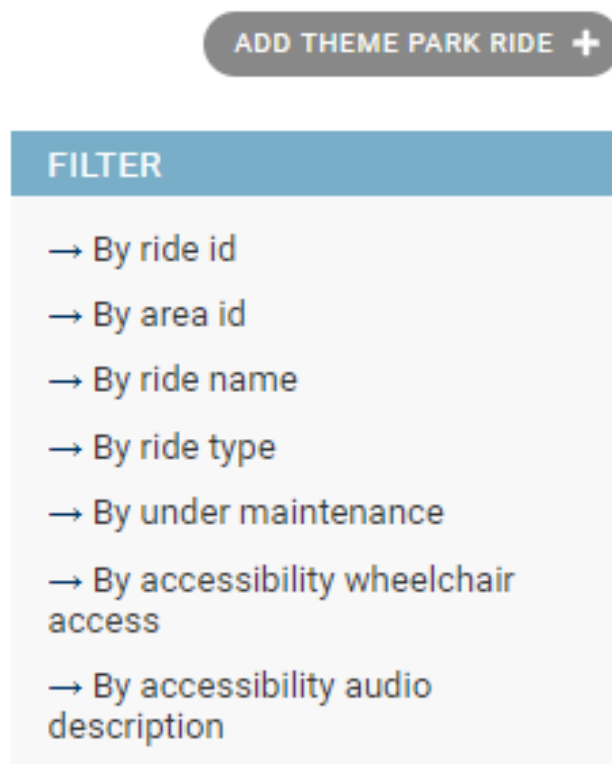


Figura 12.5: Sección para añadir nuevas entradas y filtros.

A mano derecha se observan dos campos. Por encima se ubica un botón negro con el texto **Add Theme Park Ride +**. Hacer clic a este botón permitirá la adición de una nueva entrada al modelo correspondiente y le llevará a la página de edición.

Debajo de ese botón, se puede observar una funcionalidad de filtración por campos extensa, en caso de poder visualizar tipos concretos de las entradas *En caso de las atracciones, los tipos, o en que área se ubica, etc.*

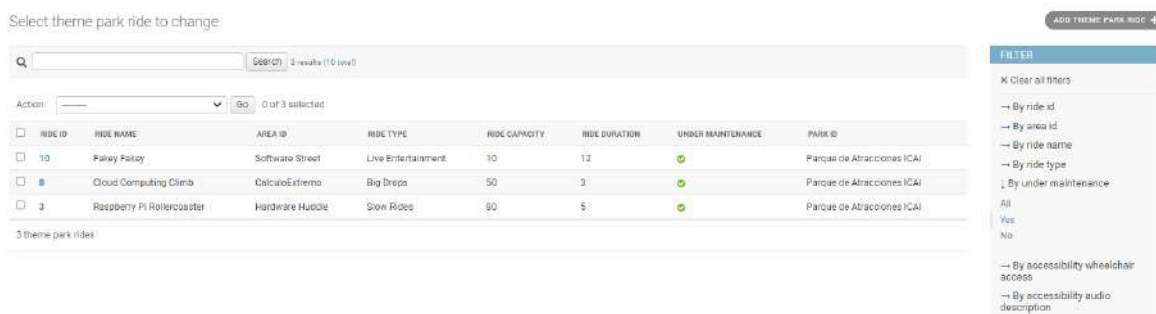


Figura 12.6: Visualización de un filtro aplicado.

A mano izquierda, se puede observar una vista condensada de los microservicios, junto a los demás modelos del proyecto, que abrirán una vista con la misma estructura a esta. Se puede observar que a la derecha del nombre del modelo, se puede observar un

+Add, el cual abrirá una nueva pestaña para poder añadir una nueva entrada al modelo correspondiente de manera rápida.

Tal y como hemos detallado a través del desarrollo del proyecto, se podrán visualizar y editar las siguientes propiedades del parque de atracciones:

- Empleados del parque.
- Clientes del parque.
- Reservas para las atracciones.
- Restaurantes del parque.
- Manejo de los parques de atracciones:
 - El nombre del parque.
 - Las áreas del parque.
 - Las atracciones del parque.
- Las tiendas del parque y sus respectivos productos.
- Las entradas y los visitantes del parque.

12.2.2. Manejo de modelos

Cuando se decide editar, o insertar una nueva entrada a un modelo, observaremos la siguiente vista:

Insertar imagen

Como se puede apreciar, cada atributo del modelo dispone de un campo editable o seleccionable, diseñado para incorporar toda la información relevante para el parque.

The screenshot shows a web form titled "Add store". It contains the following fields and controls:

- Park id:** A dropdown menu with a small icon to its right.
- Area id:** A dropdown menu with a small icon to its right.
- Store name:** A text input field.
- Store description:** A large text area.
- Store thumbnail:** A button labeled "Elegir archivo" and the text "No se ha seleccionado ningún archive".
- Opening hour:** A time input field showing "22:00:00" and a "Now" button with a clock icon. Below it, a note says "Note: You are 5 hour ahead of server time."
- Closing hour:** A time input field showing "22:00:00" and a "Now" button with a clock icon. Below it, a note says "Note: You are 5 hour ahead of server time."

At the bottom right of the form, there are three buttons: "Save and add another", "Save and continue editing", and "SAVE".

Figura 12.7: Visualización de una nueva entrada para un modelo.

Dependiendo del atributo, la entrada tendrá un formato particular, que se ha definido en el `models.py`. En el caso de tener que introducir horas, imágenes, o decimales, los formatos serán los apropiados, para asegurar formateo correcto.

Tras acabar de rellenar la información, se podrá:

- Guardar y añadir otra entrada.
- Guardar y continuar editando la misma entrada.
- Guardar y salir de la vista.

Durante el diseño de la aplicación, se ha prestado especial atención a la minimización de datos superfluos. Por este motivo, es imperativo completar todos los campos si se desea añadir una nueva entrada a la base de datos. En caso contrario, se mostrará un mensaje de error, indicando los datos que faltan por rellenar.

Add store

Please correct the errors below.

Park id:

Area id:

Store name:

Store description:

Store thumbnail: No se ha seleccionado ningún archivo

Opening hour: Now

Closing hour: Now

Buttons: Save and add another, Save and continue editing, SAVE

Figura 12.8: Errores al no introducir información en la edición.

Una vez se haya rellenado toda la información de manera correcta, se agregará esta entrada a la base de datos del modelo relevante.

Virtual Q. Administration

Home / Settings / Stores

Select store to change

ACTION	STORE NAME	STORE DESCRIPTION	OPENING HOUR	CLOSING HOUR	PARK ID	AREA ID
<input type="checkbox"/>	Midnight Welcome Gift Shop	Welcome to the store!	22:00	06:00	Parque de Atracciones (CAI)	Software Store!

1 store

Filters: By area id, By store name, All, By opening hour, All, 22:00:00

Figura 12.9: Nueva adición añadida exitosamente.

Como se ha podido ver, la función principal del administrador es gestionar y supervisar las bases de datos del parque de atracciones. Cada cambio realizado tiene un efecto directo sobre la aplicación del cliente, ya que la actualización se realiza a través de una API REST en tiempo real. Esto no solo proporciona actualizaciones inmediatas, sino que también hace que el intercambio de datos sea más informativo y eficaz.

12.3. Manual por parte del cliente

Los clientes tendrán acceso a la aplicación a través de su propia interfaz, la cual les permitirá:

- Crear su propia cuenta.
- Registrar a su grupo.
- Gestionar sus entradas.
- Visualizar las atracciones disponibles en el parque.
- Gestionar sus reservas.

Además, el cliente tiene la posibilidad de utilizar un sistema de reservas avanzado, que le permite reservar su plaza en el parque de atracciones para un día y fechas específicas.

Una vez se hayan arrancado tanto el servidor del back-end como la aplicación React-Native, se podrán llevar al cabo las siguientes secciones:

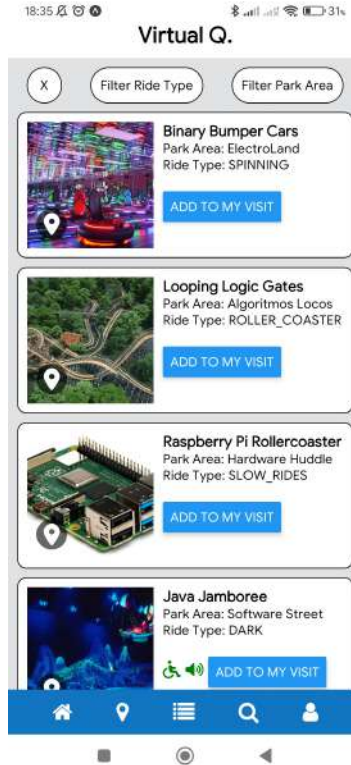
12.4. Vistas de un usuario no autenticado

Cuando un usuario abre la aplicación por la primera vez, tiene acceso a las siguientes vistas:

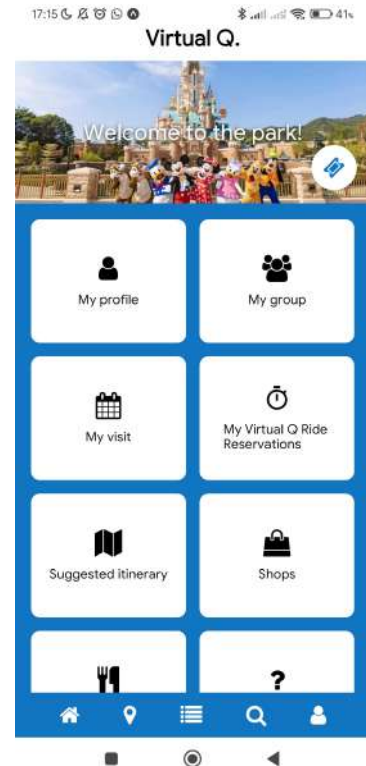
- Vista de inicio (*Crearse una cuenta o iniciar sesión*)
- Vista de atracciones (*Visualizar todas las atracciones en breve, y poder filtrar por área o tipo de atracción.*)
- Vista de atracciones en detalle (*Poder visualizar toda la información relevante sobre una atracción que le puede hacer falta a un cliente para informarse.*)
- Vista del dashboard para los usuarios (*En el cual se pueden ver todas las cosas que a las que puede acceder un usuario autenticado*)



(a) Vista de inicio



(b) Vista atracciones



(c) Vista del dashboard para los usuarios

En la aplicación, la navegación principal se maneja a través de un pie de vista en el cual se incluyen iconos para acceder rápidamente al HomeScreen, a la lista de atracciones, la sección de búsqueda para obtener información de rápido acceso y al dashboard del usuario.

12.4.1. Vista de las atracciones

Nuestra aplicación proporciona una experiencia completa de exploración de los parques de atracciones disponibles. Los usuarios pueden visualizar un listado de todas las atracciones disponibles en el parque haciendo clic en el botón del footer central, con la opción de filtrar los resultados según el tipo de atracción y el área del parque en donde se ubica.

En la vista de lista de atracciones, cada atracción está presentada con una breve descripción que incluye su título, una imagen representativa, el área en la que se encuentra dentro del parque, los iconos de accesibilidad y la opción de añadirla a la lista de atracciones favoritas del usuario.

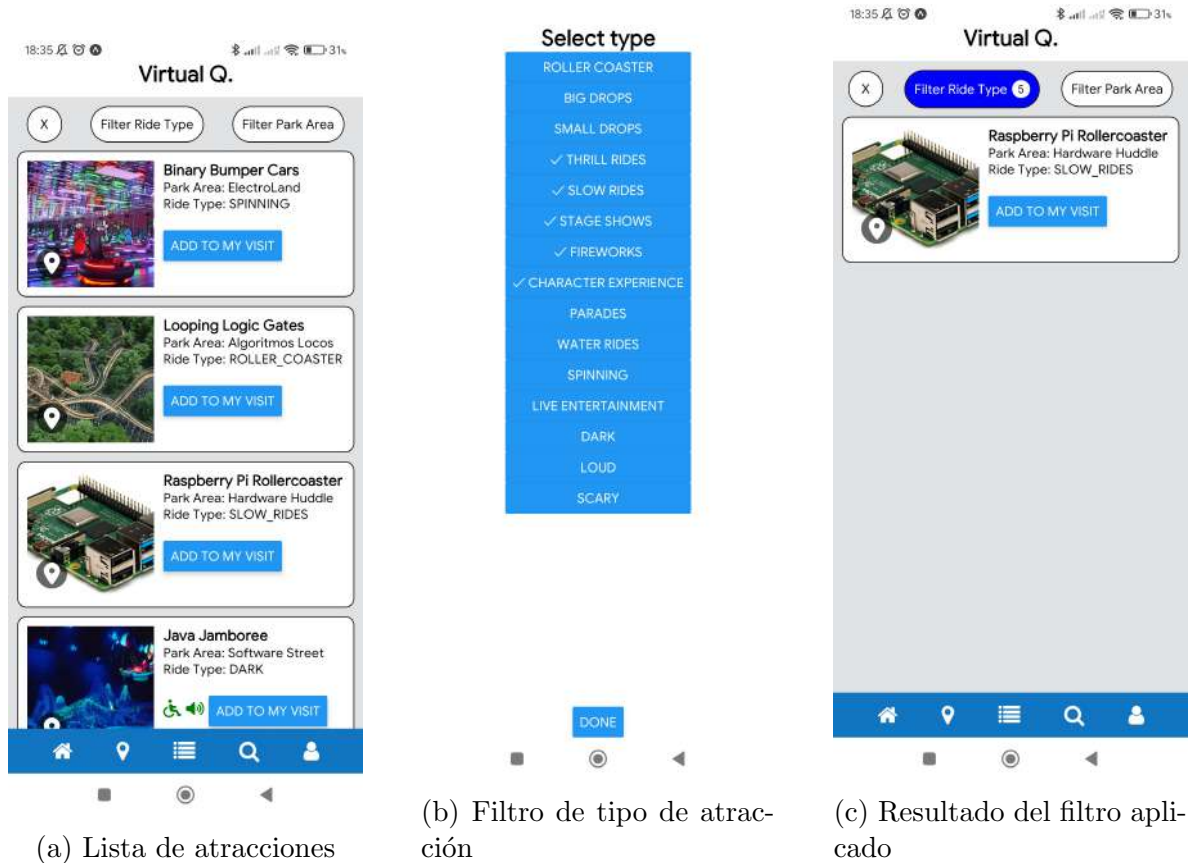
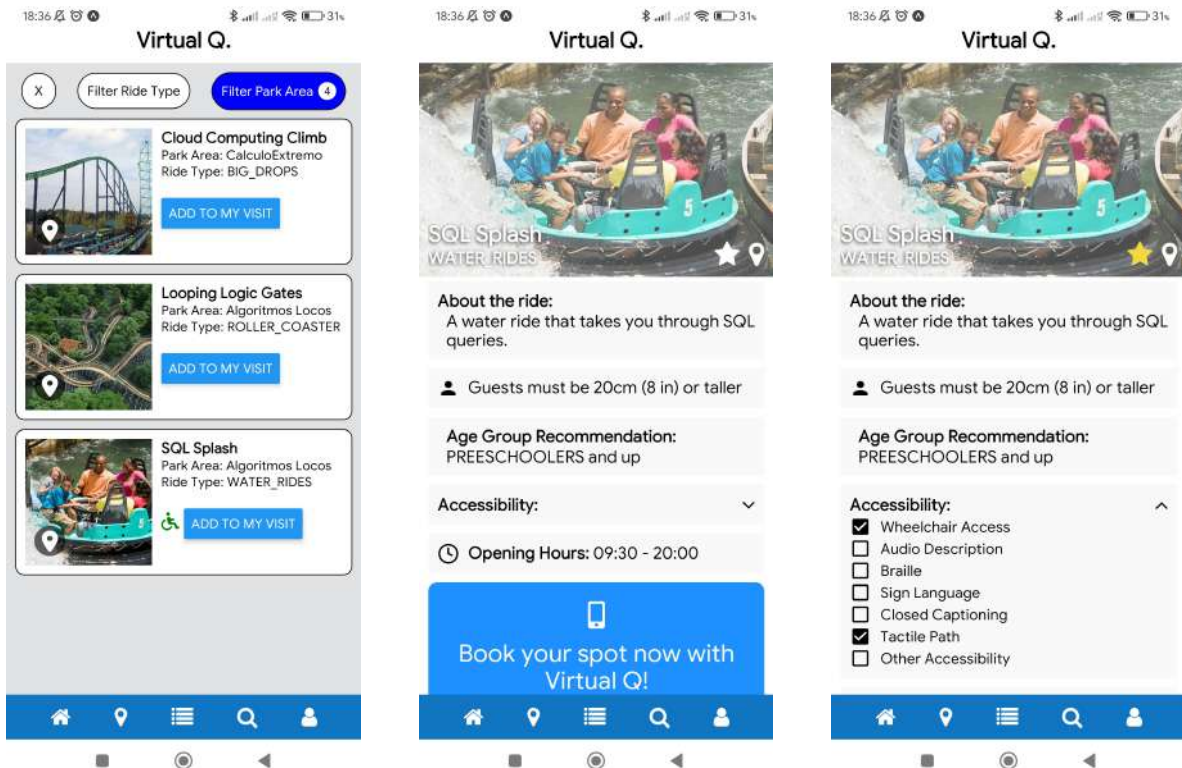


Figura 12.11: Visualización del listado de atracciones y sus filtros

Si el usuario desea más detalles sobre una atracción en particular, simplemente tiene que hacer clic en la misma para acceder a una versión expandida con información detallada. Esta información adicional incluye una descripción completa de la atracción, indicador de altura mínima requerida, la edad recomendada para los visitantes, todas las opciones de accesibilidad disponibles y los horarios de apertura.



(a) Atracciones filtradas por área

(b) Visión expandida atracción

(c) Expansión de la accesibilidad

Figura 12.12: Visualización de la atracción completa

En el caso del dashboard del usuario, se observan todas las distintas funcionalidades, si un usuario no autenticado intenta acceder a ellos, se le redirigirá automáticamente a la página de inicio de sesión.

12.5. Vistas de un usuario autenticado

Un usuario autenticado tiene todas las capacidades de un usuario no autenticado, pero se le suman numerosas habilidades particulares para la gestión de su visita:

1. Poder iniciar sesión
2. Poder editar su información y datos personales
3. Poder sacar entradas para su visita al parque (junto a sus acompañantes).
4. Poder editar la información de sus acompañantes para poder personalizar la experiencia del usuario.
5. Poder visualizar sus entradas en la aplicación (incluidas en múltiples días)
6. Hacer reservas para parques de atracciones a horas específicas.
7. Poder visualizar sus reservas en la aplicación.

Para que un usuario puede convertirse en un usuario autenticado, debe de registrarse como usuario. Para ello, puede navegar a la vista de inicio y hacer clic sobre el botón **Sign Up** e ingresar sus datos personales.

Una vez completos, se le ingresará sesión de manera automática en la aplicación.

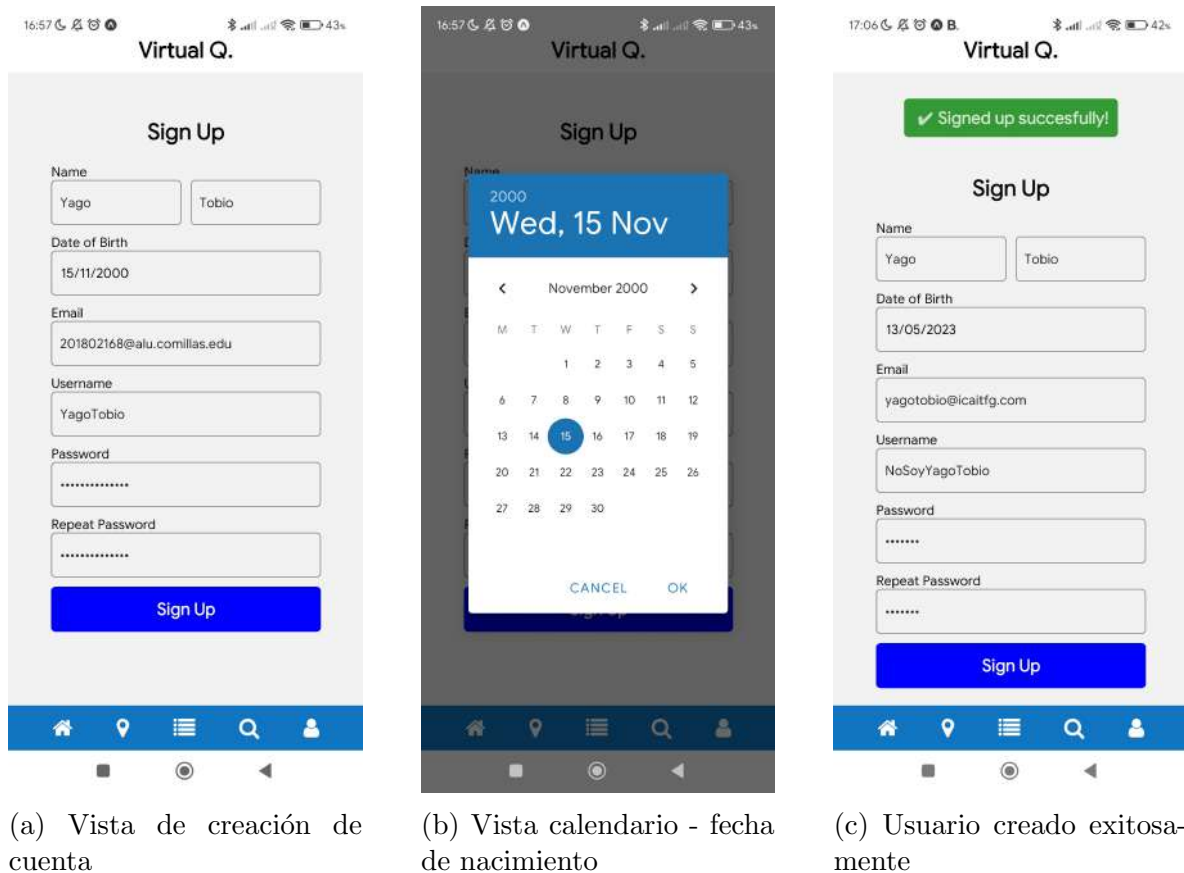


Figura 12.13: Flujo de creación de un usuario

12.5.1. Inicio de sesión

En caso de reiniciar el sistema (se puede presionar la letra 'r') en la terminal de React Native, o que se salga de la sesión inicial, un usuario nuevo puede iniciar su sesión. Es importante tener en cuenta que los datos se deben de insertar de manera correcta, o una alerta indicará los errores que se deben de corregir para que se registre el usuario correctamente.

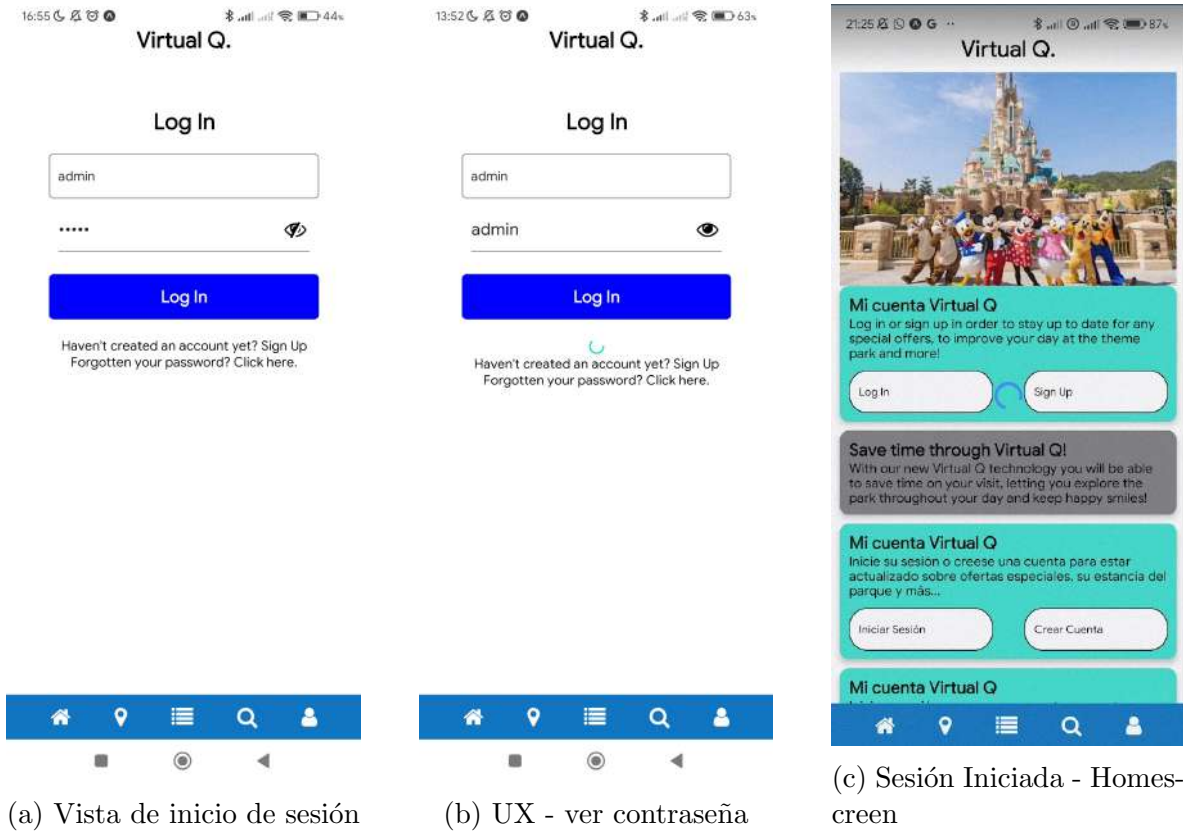


Figura 12.14: Flujo de inicio sesión

12.5.2. Gestión de datos personales

Una vez un usuario haya iniciado sesión, podrá navegar al dashboard de usuarios y hacer clic en el panel de My profile. Una vez haya hecho ingresado en la vista, el usuario podrá visualizar su información y poder editarla correctamente. En caso de que quiera re-establecer su contraseña, por motivos de seguridad, se le enviará un correo personal, con un enlace adjunto.

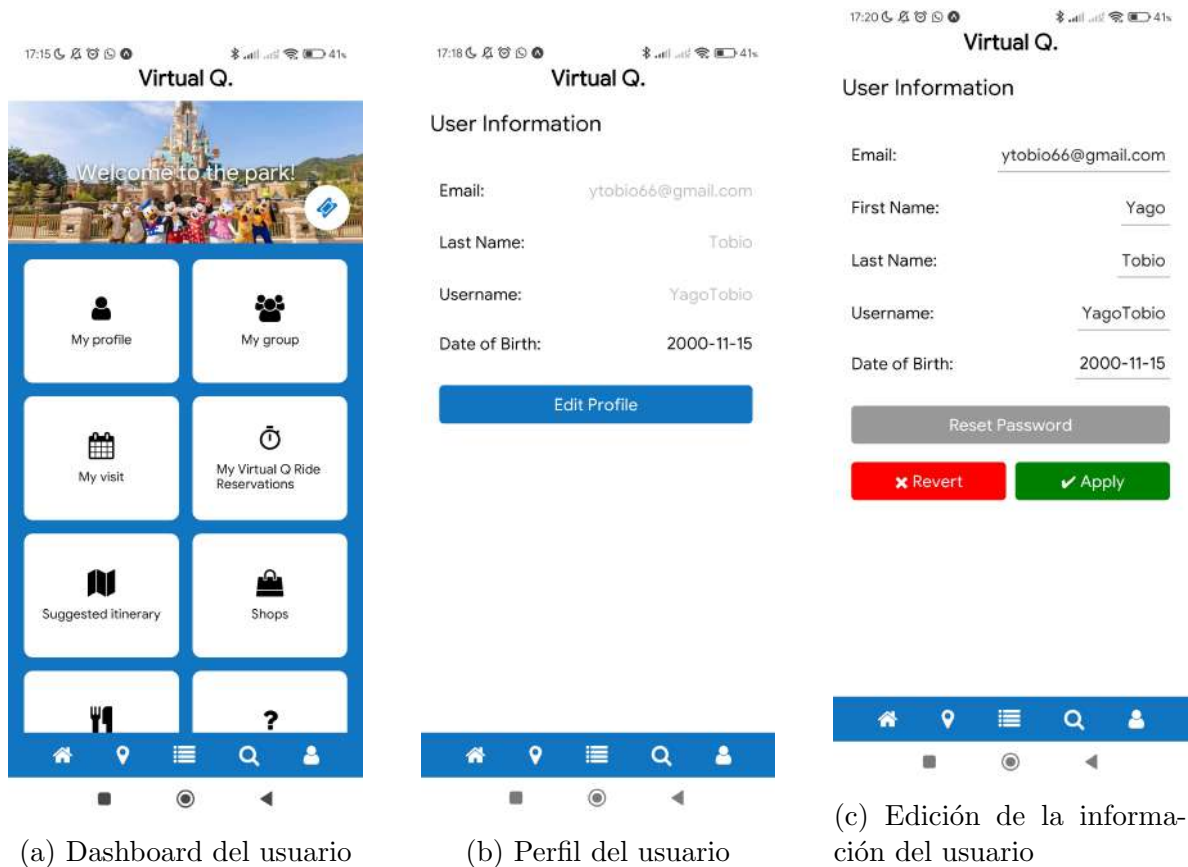


Figura 12.15: Flujo para editar los datos del usuario.

Para modificar su perfil, el usuario debe primero hacer clic en el botón denominado **Editar Perfil**. Al seleccionar esta opción, los campos del perfil se desbloquearán y se volverán editables, permitiendo al usuario hacer los cambios deseados.

Es importante destacar que, con el objetivo de evitar duplicidades, el sistema no autoriza la edición del nombre de usuario o correo electrónico en caso de que ya estén registrados. Esta medida es fundamental para mantener la unicidad de los perfiles en el sistema.

12.5.3. Sacar entradas para el parque de atracciones

La aplicación de Parque de Atracciones facilita la adquisición de entradas para los usuarios. Desde el momento en que los usuarios inician sesión en la aplicación, tienen la posibilidad de comprar entradas de manera sencilla y rápida.

Desde el "Dashboard" del usuario, se puede visualizar un botón identificado con un logotipo de unas entradas. Al seleccionarlo, el sistema consultará si el usuario ya posee entradas. En caso contrario, se presentará un botón **Comprar Entradas** que redirige al usuario a una página web segura.

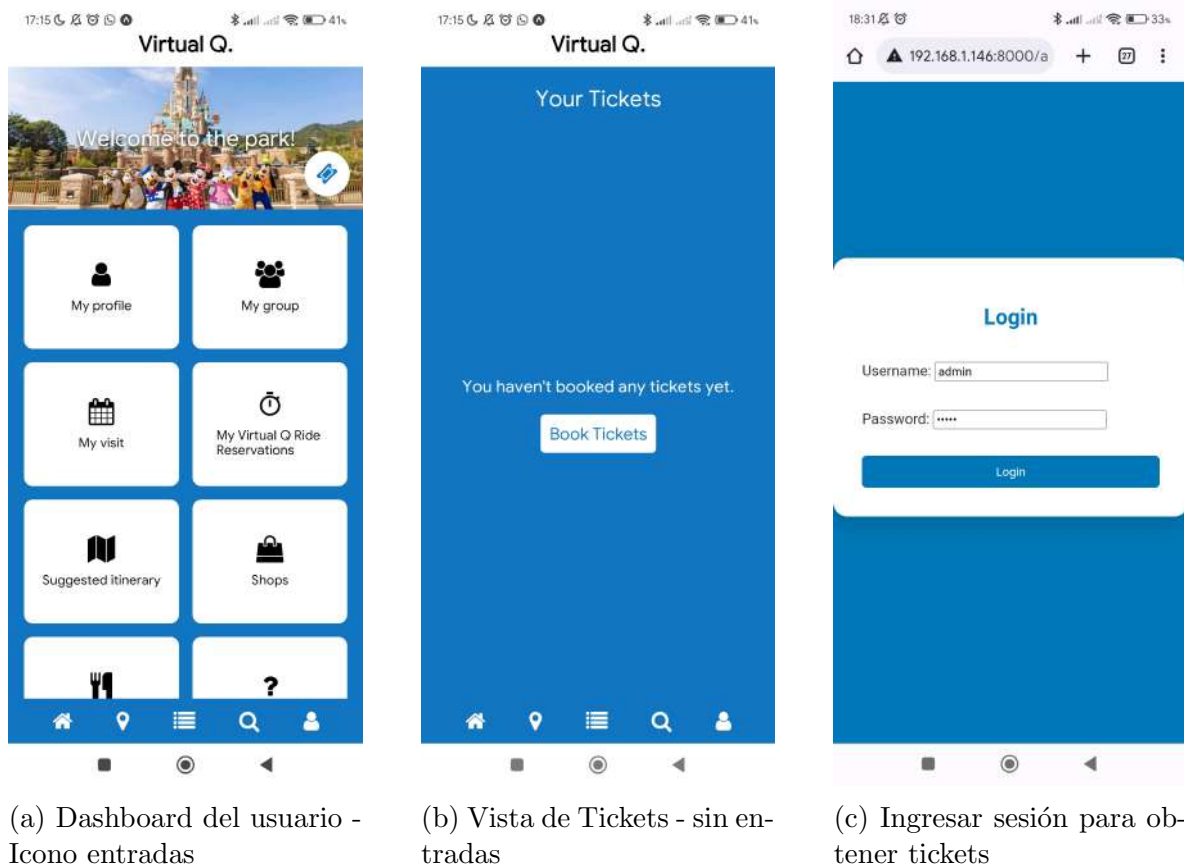


Figura 12.16: Flujo para solicitar entradas.

Tras entrar en la página web, se observa que se pueda iniciar sesión y una vez se haya ingresado con éxito, se abrirá una vista en la cual se puede seleccionar la fecha y el número de visitantes adicionales que acompañarán al usuario el día de su visita.

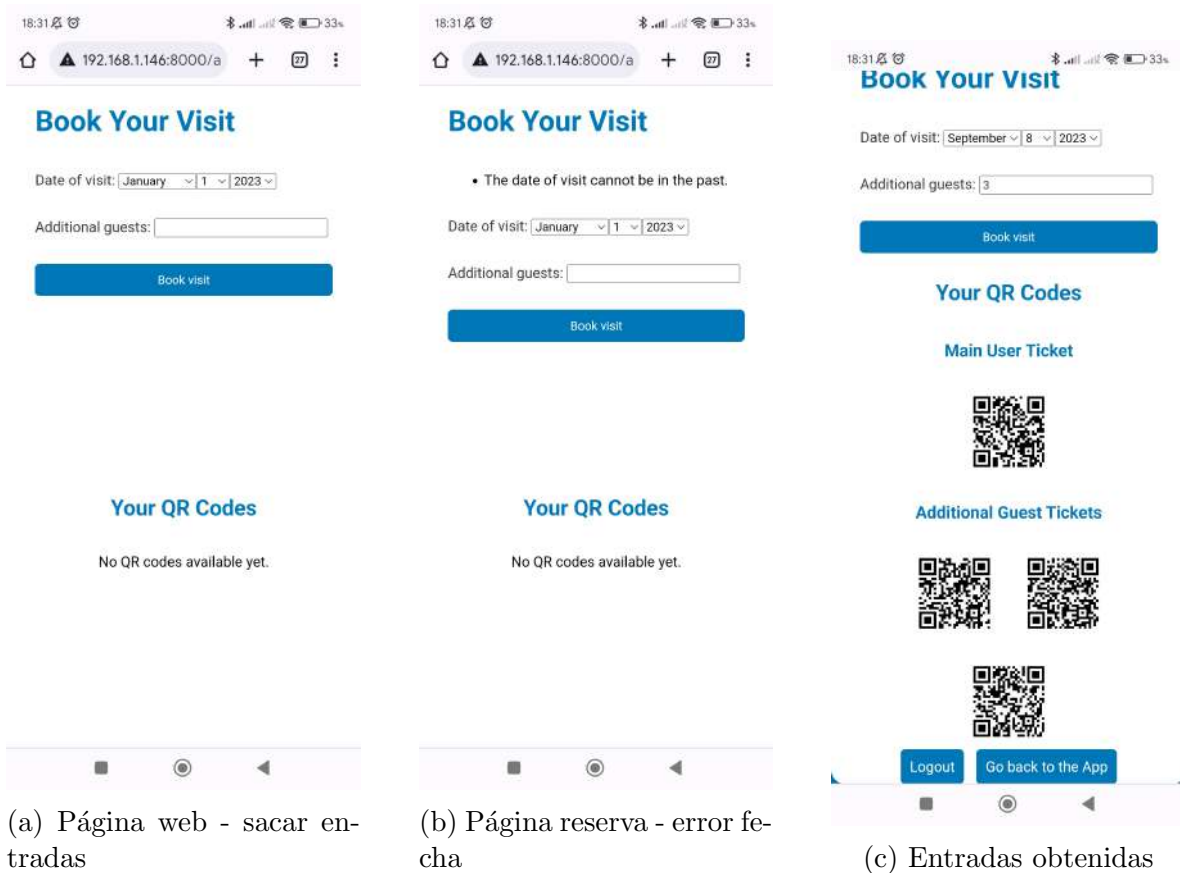


Figura 12.17: Flujo para obtener entradas [Fecha reservada - 8/8/23]

Una vez se haya ejecutado el proceso, el usuario tendrá la opción de salir de su sesión o de volver a la aplicación (*esta funcionalidad solo funciona mediante los dispositivos móviles*), en la cual ya podrá encontrar sus entradas registradas, listas para su visualización.

12.5.4. Visualizar entradas en la aplicación

Tras la selección por parte del usuario de la fecha deseada y la cuantificación de los visitantes adicionales, se generan códigos QR individuales de manera automática para cada visitante, incluyendo al propio usuario. Estos códigos son dispuestos de forma intuitiva dentro de la interfaz de la aplicación, asegurando un acceso sencillo el día que corresponda la visita. Para poder acceder a estos códigos, es necesario ingresar nuevamente al tablero de usuarios, específicamente en el ícono correspondiente a las entradas.

Al seleccionar cualquiera de las entradas, se desplegará una vista ampliada de las mismas, permitiendo el desplazamiento a través de las entradas correspondientes a la fecha seleccionada. Esta característica proporciona una visualización detallada y simplifica el proceso de revisión de las entradas.



Figura 12.18: Visualización en la vista de entradas para ver las reservas hechas - 8/8/23

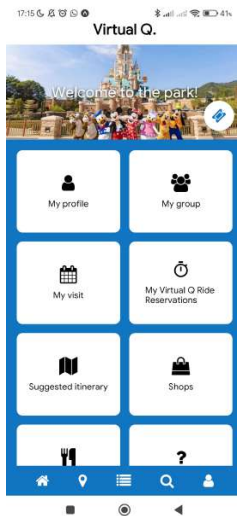
Esta funcionalidad no requerirá ninguna interacción más por parte del usuario, ya que estos códigos QR se escanearán por los empleados en la entrada del parque de atracciones.

12.5.5. Editar la información de sus acompañantes

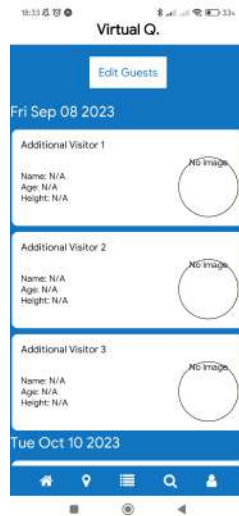
Además de facilitar la adquisición de entradas, la aplicación proporciona la funcionalidad de personalización de la información correspondiente a los visitantes adicionales. A través del tablero de control del usuario, denominado “Dashboard”, se puede acceder a la función **My Group**. En esta sección se presentan perfiles vacíos equivalentes al número de visitantes adicionales.

Una vez que el usuario ha ingresado en la sección **My Group** del tablero de control del usuario, observará que existen una serie de perfiles sin información, equivalentes al número de personas que lo acompañarán en su visita.

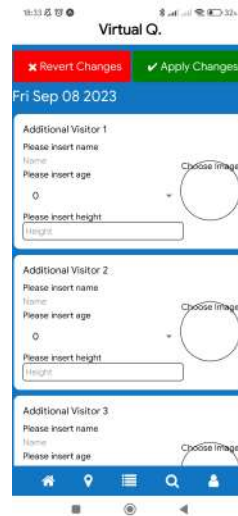
Cada uno de estos perfiles ofrece la posibilidad de personalización, permitiendo la inclusión de datos como el nombre, la edad, la altura y un icono de perfil, proporcionando así una experiencia más individualizada y personal. Esta característica busca enriquecer la interacción del usuario con la aplicación y mejorar su experiencia global durante la visita.



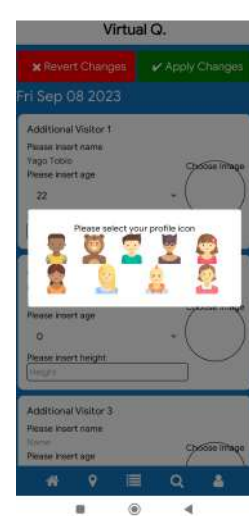
(a) Dashboard de usuario



(b) Vista de grupo - inicial



(c) Vista de grupo - edición



(d) Selección de iconos

Figura 12.19: Visualización de la edición de los visitantes adicionales

12.5.6. Hacer reservas para parques de atracciones a horas específicas.

La aplicación ofrece un sistema de reservas robusto y eficiente, permitiendo a los usuarios reservar sus atracciones favoritas para evitar largas colas y aprovechar al máximo su visita. Sin embargo, para utilizar este servicio, el usuario debe cumplir dos condiciones:

1. Debe haber adquirido entradas para el parque de atracciones.
2. Debe realizar la reserva con menos de 2 días de antelación a su visita.

El proceso de reserva es sencillo. Desde la vista detallada de la atracción, los usuarios pueden seleccionar la opción "Haz tu reserva con Virtual Q.". Esto les redirige a una pantalla donde pueden seleccionar la fecha de su visita (si tienen entradas para más de un día) y elegir un intervalo de 30 minutos para su reserva.

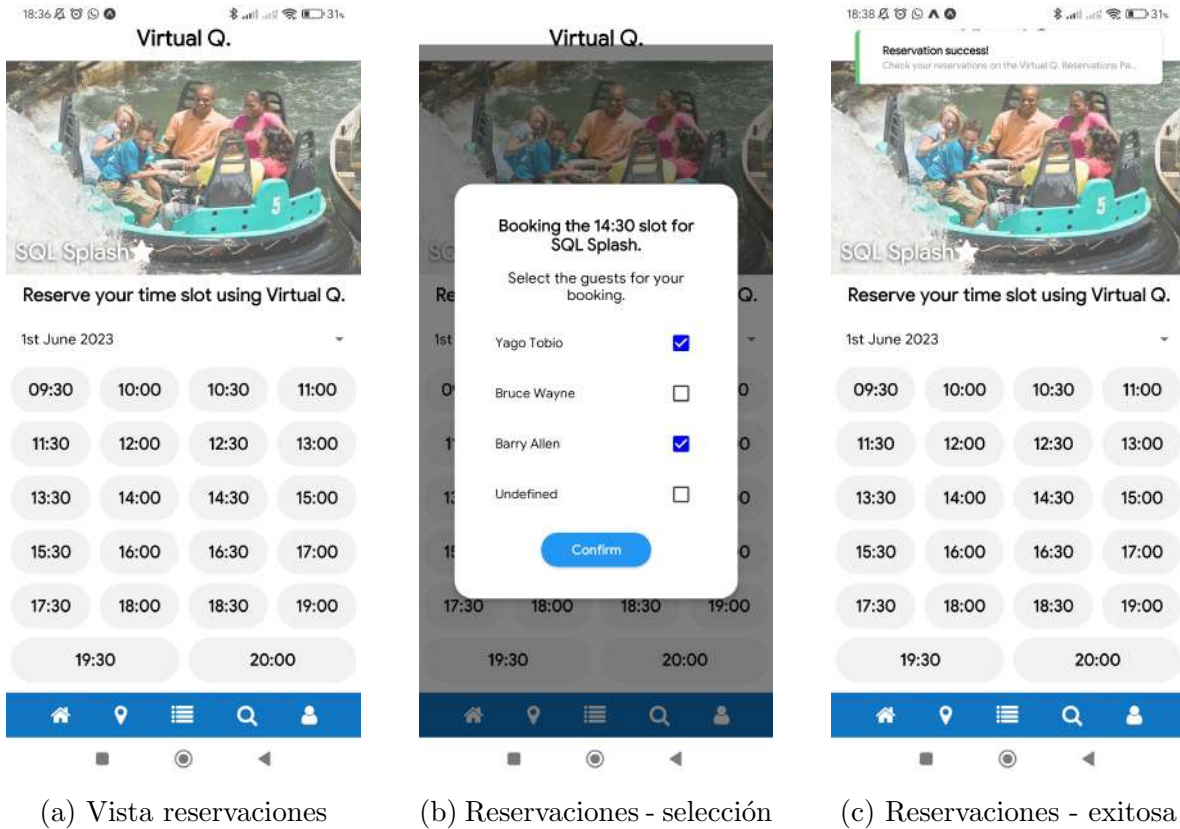


Figura 12.20: Visualización del sistema de reservas

Una vez seleccionado el intervalo de tiempo, se presenta al usuario una vista detallada de su reserva propuesta, que incluye la atracción seleccionada, la hora de la reserva y los miembros del grupo que asistirán a la atracción. Los usuarios tienen la libertad de seleccionar quiénes de su grupo participarán en la reserva, lo que permite mayor flexibilidad y evita la asignación innecesaria de plazas.

Es imperativo tener en cuenta que para poder proceder con una reserva exitosamente, se deben de cumplir las siguientes conclusiones:

- La atracción no debe de estar bajo mantenimiento.
- No debe de existir una reserva al usuario o guest asignado a esa misma hora.
- La reserva debe de coincidir con la fecha de la entrada que ha sacado el usuario.

12.5.7. Visualizar reservas en la aplicación

Tras que se haya ejecutado esto, el usuario podrá acceder a sus reservas mediante el dashboard de usuario, mediante el panel `My Virtual Q. reservations`

Capítulo 13

Anexo - Alineación con los Objetivos de Desarrollo Sostenible

En este capítulo se muestra la relación que tiene el presente Trabajo de Fin de Grado con los Objetivos de Desarrollo Sostenible (ODS), los cuales fueron aprobados por la ONU en 2015.

Cada uno de estos objetivos consiste en un conjunto de metas específicas que deben alcanzarse en los próximos 10 años. Seguir esta metodología cuenta con el fin de erradicar la pobreza, proteger el planeta y asegurar la prosperidad.

“La sostenibilidad es el desarrollo que satisface las necesidades del presente, sin comprometer la capacidad de las futuras generaciones, garantizando el equilibrio entre el crecimiento económico, el cuidado del medio ambiente y el bienestar social”

Estos 17 objetivos de Desarrollo Sostenible persiguen este concepto de sostenibilidad. Que se pueden clasificar según la dimensión con la que se relacionan. Las dimensiones involucran los aspectos económicos, sociales y medioambientales del proyecto, los cuales podemos observar en la siguiente figura como las ODS intervienen en cada una.

Se puede creer que, a la hora del desarrollo de una aplicación de software y nuevas tecnologías, no sea muy relevante el concepto de sostenibilidad, pero, aun así, hay objetivos relevantes dentro del ODS que destacan en particular: Consideramos que el ODS9 es el que más relación tiene con el proyecto desarrollado, debido a que las tecnologías que se usan son del estado del arte y contribuyen a un mejor uso del software y de las infraestructuras ya montadas.

Para el gran acceso de usuarios a la aplicación, el uso de Cloud Computing es imperativo, lo cual va a suponer un menor coste por parte de las empresas que contraten este servicio. Este contribuye a la sostenibilidad. La utilización de servicios Cloud en lugar de implementar servicios y centros de procesamiento de datos (CPD) propios hace que se reduzcan los recursos energéticos por parte de las empresas. Lo cual disminuye las



Figura 13.1: Objetivos de Desarrollo Sostenible

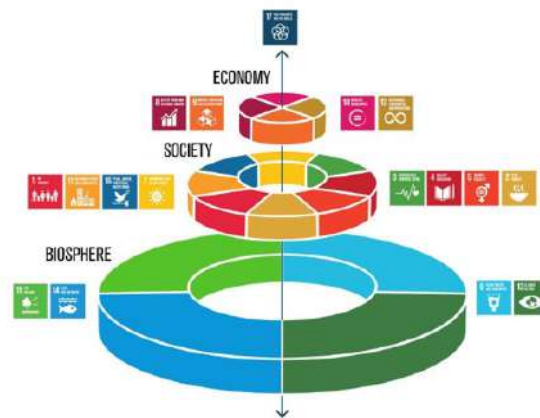


Figura 13.2: Las dimensiones de sostenibilidad y su desarrollo

emisiones de CO₂ en la atmósfera. Esta reducción de emisiones de gases de efecto invernadero supone entre un 30 % y un 90 % menos, en comparación con la implementación de un CPD propio.

Adicionalmente, adoptar los modelos de desarrollo de Arquitectura de Microservicios y DevOps, motiva las buenas prácticas que beneficia a las empresas al ser procesos de producción de software, automatizables, contenidos y no dependientes.

CAPÍTULO 13. ANEXO - ALINEACIÓN CON LOS OBJETIVOS DE DESARROLLO SOSTENIBLE

Analizando la descripción de cada uno de los diecisiete objetivos, se ha llegado a la conclusión de que los dos que presentan mayor relación con el propósito del desarrollo de este proyecto son:

Dimensión	ODS	Rol	Meta
Economía	ODS 9: Industria, innovación e infraestructura	Primario	9,4: De aquí a 2030, modernizar la infraestructura y reconvertir las industrias para que sean sostenibles, usando los recursos con mayor eficacia y promoviendo la adopción de tecnologías y procesos industriales limpios y ambientalmente racionales, logrando que todos los países tomen medidas de acuerdo con sus capacidades

En definitiva, aunque de antemano los proyectos de software y tecnología no parezcan tener una fuerte relación con los Objetivos de Desarrollo Sostenible, al analizar en detalle los beneficios que producen se ve más claro que sí que están. Lo cual implica la aportación de valor a la sostenibilidad perseguida por la ONU.