ParkPal:Mobile Parking Application

by
ZEYNEP DÜKKANCIOĞLU

Engineering Project Report

Yeditepe University
Faculty of Engineering
Department of Computer Engineering
2024

ParkPal:Mobile Parking Application

APPROVED BY:

Associate Prof. Dr. Mert Özkaya (Supervisor) .....

Prof. Dr. Şebnem Baydere .....

Assist. Prof. Dr. Tacha Serif .....

DATE OF APPROVAL: .../.../2024

ACKNOWLEDGEMENTS

First of all I would like to thank my advisor Associate Professor Dr. Mert Özkaya for his guidance and support throughout my project. Finally, I must give thanks to my family for their,love, support and continuous encouragement for all my life and education.

ABSTRACT

ParkPal:Mobile Parking Application

This project is a mobile application aimed at optimizing parking managment in urban areas. The objective is to develop a user-friendly platform that streamlines the process of finding available parking spaces, thereby reducing time wastage and improving overall efficiency. ParkPal caters to two types of users: drivers and attendants. Drivers utilize the app to navigate streets supported by ParkPal, checking for vacant parking spots on the map and making reservations when available. The system employs a coded reservation mechanism for booking convenience. Upon reservation, users receive directions to the designated spot, each equipped with a unique, refreshed code for identification. Attendants, on the other hand, mark parking spots as occupied or vacant. By matching codes with those used by drivers, attendants ensure seamless communication and accurate spot status updates. ParkPal aims to alleviate the parking challenges in bustling cities like Istanbul, offering a solution that enhances time management and minimizes frustration for both drivers and attendants.

ÖZET

ParkPal: Mobil Park Uygulaması

Bu proje, şehirlerdeki otopark yönetimini optimize etmeyi hedefleyen ParkPal adlı bir mobil uygulamayı sunmaktadır. Amaç, mevcut park yeri bulma sürecini kolaylaştıran, böylelikle zaman kaybını azaltarak genel verimliliği artıran kullanıcı dostu bir platform geliştirmektir. ParkPal, şoförler ve görevliler olmak üzere iki tür kullanıcıya hitap etmektedir. Şoförler, ParkPal tarafından desteklenen sokaklarda boş otopark yerlerini harita üzerinde kontrol ederek rezervasyon yaparlar. Rezervasyon için kodlu bir sistem kullanılmaktadır. Rezervasyon yapıldıktan sonra, kullanıcılara belirlenen noktaya yönlendirme sağlanır, her nokta kendine özgü, yenilenen bir kod ile donatılmıştır. Görevliler ise park yerlerini dolu veya boş olarak işaretlerler. Şoförlerin kullandığı kodlarla eşleştirme yaparak, görevliler kesintisiz iletişim sağlar ve park yeri durumu güncellemelerini doğrulukla yaparlar. ParkPal, kalabalık şehirlerdeki otopark zorluklarını hafifletmeyi amaçlamaktadır, böylelikle hem şoförler hem de görevliler için zaman yönetimini geliştiren ve stresi en aza indiren bir çözüm sunar.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Urbanization and the increasing number of vehicles in cities have led to significant challenges in parking management. As cities grow, the demand for parking spaces escalates, exacerbating issues such as traffic congestion, air pollution, and inefficient land use. In response to these challenges, advancements in technology have paved the way for innovative solutions aimed at optimizing parking management systems. One such solution is the development of mobile applications tailored to facilitate the process of finding available parking spaces in urban areas. This thesis presents the design and implementation of ParkPal, a mobile application specifically designed to address the parking woes encountered in bustling city environments. ParkPal aims to revolutionize the traditional approach to parking management by harnessing the power of technology to streamline the process of locating, reserving, and managing parking spaces. By leveraging real-time data and user-friendly interfaces, ParkPal endeavors to enhance the overall efficiency of parking management systems, thereby alleviating the burden on both drivers and parking attendants.

1.1. Problem Definition

The existing parking infrastructure in urban areas, including campuses and city centers, often fails to meet the growing demand for parking spaces, resulting in a myriad of challenges for both drivers and parking administrators. One of the primary issues plaguing conventional parking systems is the inefficiency in locating available parking spaces, leading to prolonged search times and increased traffic congestion. Moreover, the lack of real-time information on parking availability exacerbates the frustration experienced by drivers, contributing to a suboptimal parking experience. Furthermore, the manual nature of parking management systems poses additional challenges, including inaccuracies in parking space allocation, difficulty in monitoring parking occupancy, and limited means of communication between drivers and parking attendants. These shortcomings underscore the pressing need for innovative solutions that not only enhance the accessibility and availability of parking spaces but also streamline the administrative processes associated with parking management. In light of these challenges, ParkPal seeks to redefine the paradigm of parking management by offering a comprehensive solution that leverages technology to optimize the utilization of parking spaces, improve the user experience for drivers, and streamline the administrative tasks for parking attendants. Through the development and implementation of ParkPal, this thesis endeavors to address the aforementioned challenges and contribute to the advancement of parking management systems in urban environments.

1.2. Requirements

The parking management application aims to revolutionize the way users find, reserve, and manage parking spaces in urban environments. By offering a seamless digital platform, the application intends to alleviate common parking challenges such as spot availability, reservation reliability, and attendant coordination. With a user-centric approach, the application seeks to enhance user experience through intuitive interfaces and real-time updates. Leveraging modern technology, the app will provide users with spot information and efficient reservation processes. This set of requirements outlines the specific functionalities, software components, and hardware infrastructure necessary to realize the application's vision of simplifying urban parking.

- **User Authentication:** The system must provide robust user authentication functionality to ensure secure access to the application. Users should be able to register, log in, and manage their accounts securely. This includes features such as password encryption, account recovery options, and protection against common security threats like brute-force attacks and SQL injection.

- **Map Interface and Real-time Data:** The application must incorporate a map interface using a reliable mapping SDK (Software Development Kit). It should fetch real-time parking spot data from a backend server or API and display it on the map interface. This data should include information such as available parking spots, their locations, and any relevant details like pricing or restrictions.

- **Navigation to Empty Parking Spots:** The system should enable users to navigate to empty parking spots using GPS (Global Positioning System) and routing algorithms. Upon selecting a parking spot from the map interface, the application should generate an optimal route for the user to follow. The navigation feature should provide turn-by-turn directions, estimated time of arrival, and any relevant traffic information to ensure a smooth parking experience.

- **Park Spot Code System:** Implement a unique code system for each parking spot that is refreshed whenever the spot becomes available. Attendants can check these codes to verify the availability of parking spots. This feature eliminates the need for payment for reservations and simplifies the process for both users and attendants.

Software Requirements:

Operating System: Android-based mobile devices.

Development Environment: Android Studio for application development.

Mapping and Navigation: Integration with Mapbox API for mapping, navigation, and location services.

Database Management: Utilization of SQLite Database for storing parking spot information, user data, and reservation details.

Security: Implementation of SHA-256 encryption for securing user passwords and sensitive data.

Hardware Requirements:

Mobile Devices: The application should run on a variety of mobile devices, including smartphones and tablets, with different screen sizes and resolutions.

GPS Capability: Devices must have GPS capability or access to location services for accurate positioning and navigation functionalities.

Sufficient Storage: Adequate storage space on devices to store the application and associated data, including maps, user preferences, and cached information.

Internet Connectivity: Reliable internet connectivity (Wi-Fi or mobile data) for accessing backend services, downloading map data, and performing real-time updates.

Sensors: Devices should have necessary sensors like accelerometer, gyroscope, and magnetometer for enhanced navigation and augmented reality features, if applicable.

2. RELATED WORK

2.1. Parking Management Systems

Parking management systems have become increasingly important in urban environments to address the challenges of congestion, limited parking availability, and inefficient utilization of parking spaces .These systems encompass a range of technologies and strategies aimed at optimizing parking operations, enhancing user experience, and improving traffic flow. Previous studies have explored various aspects of parking management systems, including real-time parking data collection, parking space reservation systems, routing optimization, and user navigation solutions.[1] By leveraging advanced technologies such as sensors, mobile applications, and cloud computing, parking management systems aim to alleviate the stress associated with parking in densely populated areas and promote more sustainable transportation practices.

2.2. Map API's in Parking Management

Mapping APIs play a crucial role in enhancing the functionality and effectiveness of parking management systems by providing geospatial data visualization, routing services, and location-based analytics. Mapbox, a leading provider of mapping and location-based services, offers a comprehensive set of APIs and SDKs that enable developers to integrate dynamic maps, navigation capabilities, and geospatial analysis tools into their applications. In their paper titled "Evaluation of Mobile Map APIs." [2]Aichner and colleagues delve into a comprehensive assessment of various mobile map APIs. Published in the 12th International Conference on Design Science Research in Information Systems and Technology (DESRIST) in 2017, the paper provides a thorough exploration of the capabilities, features, and performance of different mobile map application programming interfaces (APIs). By rigorously evaluating these APIs, the authors contribute valuable insights into the strengths and limitations of each, aiding developers and researchers in selecting the most suitable mapping solution for their applications. This research serves as a valuable resource for understanding the landscape of mobile map APIs and informs decision-making processes in the development of location-based services and navigation applications. Access to the paper can be found on IEEE Xplore, offering readers in-depth analysis and comparisons essential for optimizing map integration within mobile applications. The integration of Mapbox API into parking management systems facilitates real-time visualization of parking availability, dynamic routing to available parking spots, and geofencing-based notifications for users. By harnessing

the power of Mapbox API, parking management systems can offer seamless navigation experiences, optimize parking resource utilization, and improve overall accessibility for drivers in urban areas. In the subsequent subsections, we explore the literature and case studies related to parking management systems and the integration of Map APIs, focusing on the utilization of Mapbox API in enhancing parking operations and user experience.

2.3. Anxiety and Urban Stress for Parking Spots

The research on " Anxiety and Urban Stress for Parking Spots" (Bajçinovci, 2019)[3] provides valuable insights into the significant challenges posed by parking in urban environments. The absence of a sustainable urban mobility plan and inadequate spatial planning exacerbate the problem, leading to a critical impact on residents' quality of life. The lack of a systematic approach to city planning results in a chaotic parking landscape, characterized by a mix of legal and illegal parking options. This situation not only increases urban congestion but also leads to social tensions, with incidents of disputes and violence over parking spaces becoming more common. The study emphasizes the urgent need for comprehensive parking regulation strategies rooted in data-driven analysis to address the immediate need for parking spaces and alleviate the growing urban stress associated with parking.

2.4. Parking Management Applications That Uses Different Technologies

The paper from UTAR describes an innovative smart parking management system that employs QR code technology, which is a distinguishing feature compared to other projects. This system leverages QR codes for a variety of functions, such as granting access to parking facilities, managing reservations, and tracking usage in real time. Users scan the QR codes to receive detailed parking information, streamline their parking experience, and ensure secure access. This method stands out from other systems that might use RFID, license plate recognition, or mobile apps without QR code integration, offering a robust and user-friendly solution for modern parking challenges. The use of QR codes enhances the security and efficiency of the parking management process, providing a straightforward and scalable solution suitable for various parking environments. This approach is designed to reduce congestion, improve user convenience, and simplify the overall management of parking facilities. (Weng)[4] Also the other paper about parking management is "Design and Development of Smart Parking Management System" [5] presents an innovative method for session management, focusing on the validation of user identity (UID) to set timers for specific activities. The system employs a robust protocol for UID verification, ensuring that only authorized users can initiate and manage sessions. This process involves multiple layers of security to authenticate the

UID, thereby safeguarding against unauthorized access and ensuring accurate time tracking. The meticulous verification mechanism described in the paper highlights the importance of security and precision in managing user sessions, making it a significant contribution to the field of secure time-based applications. In contrast, my application employs a more manual approach, where the attendant checks a unique code provided by the driver to set the timer. This method relies on human verification and interaction, ensuring that the attendant manually confirms the driver's code before initiating the timing process. While less automated than the system described in the paper, this approach provides a straightforward and practical means of managing session times in a parking scenario. It simplifies the process and ensures that the driver's identity is verified in a user-friendly manner, balancing security with ease of use in the context of a parking management system.[6]

3. ANALYSIS & DESIGN

This chapter provides a comprehensive analysis and design overview for ParkPal. The objective is to gain a deep understanding of the system's functional requirements, non-functional requirements, hardware and software requirements, and platform requirements. To achieve this, we will use UML diagrams such as use cases, class diagrams, sequence diagrams, component diagrams, and state diagrams. Additionally, pseudo-codes and key algorithms will be outlined to facilitate the development of the application across various platforms.

3.1. Analysis

This section outlines the functional and non-functional requirements of ParkPal. At the end of this section, a use case diagram is provided to overview the functional requirements of the system and showcase the interactions between the user and the application.

3.1.1. Functional Requirements

User Interface: The initial user interface is the entry point for users into ParkPal. It should present a clean and intuitive layout for user interaction. This is where users can search for parking spots, view available spots, navigate to a selected spot, and access the application's ”Info” screen.

Parking Spot Detection: The application must support the real-time detection and display of available parking spots. Users should be able to view a map with markers indicating available spots and receive updates on spot availability.

Navigation: Integration with Mapbox API to provide turn-by-turn navigation to the selected parking spot. Users should be able to get directions from their current location to the available parking spot.

Data Security: The application must ensure the security and privacy of user data, implementing SHA-256 encryption for passwords and secure communication protocols to protect user information.

3.1.2. Non-Functional Requirements

ParkPal will be designed with the following non-functional requirements.

3.1.2.1. Performance. **Volere Template:** Performance-1

ID: NFREQ-001

Type: Non-Functional

Description: The application should provide responsive and efficient performance, ensuring smooth interaction and handling real-time updates without noticeable delays. , **Rationale:** The responsiveness and efficiency of the application are crucial to provide users with a seamless experience, especially when they need to find and navigate to available parking spots quickly.

Fit Criterion: The application should update the availability of parking spots in real-time, with a maximum delay of 1 second for updates to be reflected in the user interface.

Priority: High

Customer Satisfaction: 5

Customer Dissatisfaction: 2

3.1.2.2. Reliability. **Volere Template:** Reliability-1

ID: NFREQ-002

Type: Non-Functional

Description: The application should be reliable, ensuring the accuracy and integrity of parking spot availability data. It should handle errors gracefully and maintain consistent performance.

Rationale: Reliability is essential to maintain the trust of users. The application must accurately reflect the availability of parking spots to avoid user frustration and ensure they can find parking efficiently.

Fit Criterion: The application should have an error rate of less than 1% in reflecting parking spot availability and should recover gracefully from any data fetch or update errors.

Priority: High

Customer Satisfaction: 5

Customer Dissatisfaction: 1

3.1.2.3. Usability. **Volere Template:** Usability-1

ID: NFREQ-003

Type: Non-Functional

Description: The application should have an intuitive and user-friendly interface, guiding users through the process of finding and navigating to parking spots.

Rationale: Usability is crucial to ensure that users can easily navigate and use the Park-Pal application without extensive training or assistance. The interface should be intuitive, enabling users to find parking spots effortlessly.

Fit Criterion: 80% of users should be able to successfully find and navigate to a parking spot within 5 minutes of using the application for the first time.

Priority: Medium

Customer Satisfaction: 5

Customer Dissatisfaction: 1

3.1.2.4. Maintainability. **Volere Template:** Maintainability-1

ID: NFREQ-004

Type: Non-Functional

Description: The application should be designed in a modular and maintainable manner, allowing for easy enhancements, bug fixes, and future updates.

Rationale: Maintainability is important to ensure that the ParkPal application can adapt to evolving requirements, incorporate new features, and address issues efficiently. The codebase should be structured and documented to facilitate easy maintenance and future development.

Fit Criterion: The average time to incorporate a bug fix or implement a minor enhancement should not exceed 2 hours.

Priority: Low

Customer Satisfaction: 3

Customer Dissatisfaction: 2

3.1.3. Hardware and Software Requirements

Software Requirements:

· **Android Platform:** The application will be developed for Android devices.

· **Programming Language:** Java

· **Development Environment:** Android Studio.

· **APIs:** Mapbox API for mapping and navigation services.

· **Database:** SQLite for storing user data and parking spot information.

· **Encryption:** SHA-256 for securing user passwords.

Hardware Requirements:

· **Devices:** Android smartphones.

· **Sensors:** GPS for location tracking.

3.1.4. Platform Requirements

ParkPal is designed to run on Android devices. It will leverage the Mapbox API for mapping and navigation functionalities and will be compatible with various Android versions to ensure a wide user base.

3.1.5. Use Case Diagram for User and Attendant

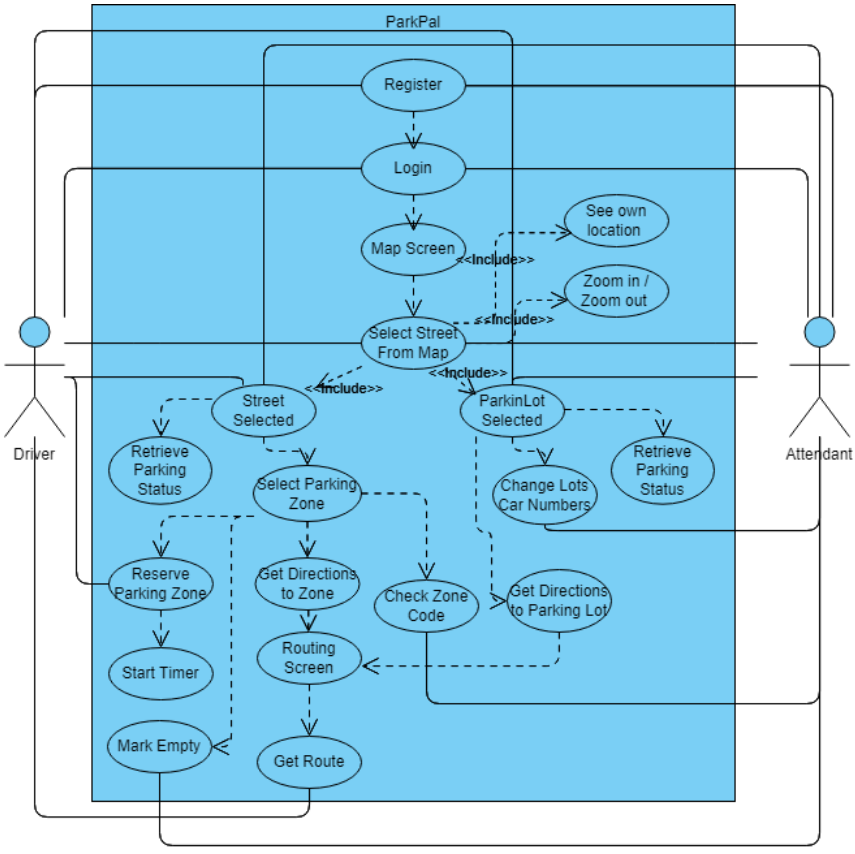

Figure 3.1. Use Case Diagram

The use case diagram provides an overview of the functional requirements of the ParkPal system and illustrates the interactions between the different types of users and the application. The primary users, depicted as actors, are the Driver and the Attendant. Each actor is linked to specific use cases that define the various functionalities they can perform within the ParkPal application.

For Drivers, the primary use cases include "**Sign Up**," "**Log In**," "**Select Street**," "**Choose Parking Spot**," "**Reserve Spot,**" and "**Get Directions**." These use cases represent the key actions that Drivers undertake to locate and secure parking. Once a Driver selects a street and a parking spot, if the spot is available, they can reserve it and receive directions to the location, ensuring a smooth and efficient parking experience.

For Attendants, the primary use cases are "**Sign Up**," "**Log In**," "**Select Street**," "**Choose Parking Spot,**" "**Verify Spot Code**," and "**Mark Spot Empty**." These use cases detail the Attendants' responsibilities in managing parking spot availability. After logging in, an Attendant selects a street and parking spot to verify its status. They can check the spot's unique code and update the status to empty if the spot is available, thereby maintaining accurate real-time data for Drivers.

By visualizing the use cases and their relationships, the diagram provides a comprehensive understanding of the system's functional requirements. It highlights the primary interactions between the users and the application, setting the stage for the subsequent design and implementation phases. This clear depiction of user activities and system functionalities ensures that both Drivers and Attendants can efficiently utilize the ParkPal system to manage and utilize parking resources effectively.

3.2. Design

This section presents a detailed overview of the ParkPal system's design, encompassing various diagrams and algorithms that illustrate its structure and functionality. It includes the use case diagram, class diagram, sequence diagrams and state diagrams, as well as key algorithms. The objective of this section is to provide a thorough understanding of how different components of the ParkPal application interact and collaborate to achieve its objectives. This comprehensive analysis will facilitate a deeper insight into the system's design, paving the way for effective implementation and deployment.

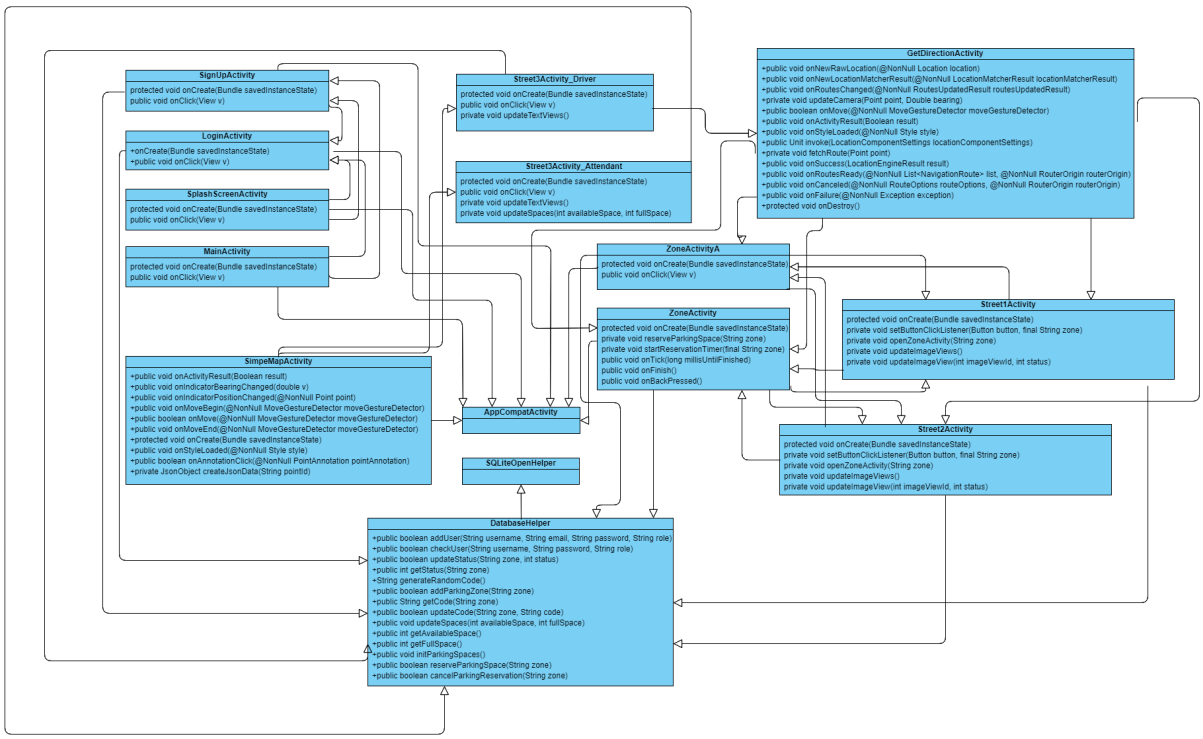

Figure 3.2. Class Diagram

3.2.1. Class Diagram

The provided class diagram offers a detailed representation of the ParkPal system's architecture, illustrating the various classes and their intricate relationships within the application. The diagram identifies key activities such as SignUpActivity, LoginActivity, **InfoActivity, SplashScreenActivity**, **MainActivity**, and multiple **StreetActivity** classes tailored for both drivers and attendants. Each of these classes encapsulates essential methods, including **onCreate**, **onClick**, and other specific functions relevant to their roles. For instance, the **Street3Activity_Driver** and **Street3Activity_Attendant** classes feature methods to update views and manage parking space selections, catering specifically to the distinct needs of drivers and attendants.

Central to the application's mapping functionality is the **SimpleMapActivity** class, which incorporates methods to handle map interactions, position changes, and the creation of JSON data for point annotations. The **GetDirectionActivity** class provides critical methods for route management and location updates, essential for directing drivers to their reserved parking spots efficiently.

The **ZoneActivity** and **ZoneActivityA** classes manage the operations within designated parking zones, including methods such as **reserveParkingSpace()** and **startReservationTimer()**. These are vital for handling reservations and ensuring that parking zones operate smoothly.

The **DatabaseHelper** class is integral for data management and persistence. It contains a wide array of methods for user management (**addUser, getUserStatus**), parking space updates (**updateParkingSpace, getAvailableSpaces**), and handling reservations (**reserveParkingSpace, cancelParkingReservation**). This ensures the system maintains accurate, up-to-date information on parking availability, thereby enhancing reliability and user satisfaction.

Foundational elements like **AppCompatActivity** serve as a base class for other activities, promoting code reuse and maintainability through inheritance. This enables various activities to inherit common functionality, streamlining the development process.

The associations between classes are meticulously depicted, illustrating how different components collaborate to deliver the application's functionality. For example, **SimpleMapActivity** interacts closely with **GetDirectionActivity** to manage navigational tasks, while **Street3Activity_Driver** and **Street3Activity_Attendant** interface with **DatabaseHelper** for retrieving and updating parking space information.

Additional classes such as **Street1Activity** and **Street2Activity** handle specific street-level operations and user interactions. These classes feature methods such as **updateImageView()** and **setButtonClickListener**, ensuring that the user interface remains interactive and responsive.

The **SQLiteOpenHelper** class is a key component for database management, providing foundational support for database creation and version management. Methods like **onCreate** and **onUpgrade** are crucial for setting up the initial database schema and managing schema changes, respectively.

This detailed class diagram serves as a blueprint for understanding the system's structure, guiding the implementation process, and ensuring all components work together seamlessly to achieve the application's objectives. By visualizing the relationships and interactions among classes, the diagram provides a clear and organized framework that supports the development of a robust and user-friendly parking management application. The structured representation underscores the collaborative nature of the system components, ensuring a

cohesive development process that aligns with the overall system goals.

3.2.2. Sequence Diagrams

3.2.2.1. Driver Sequence Diagram. The sequence diagram depicts the step-by-step interaction between a Driver and the system components during their parking journey. It begins with the Driver initiating a login request, prompting the system to validate their credentials. Upon successful validation, the Driver gains access to the system's functionalities. Subsequently, the Driver navigates through available parking streets, triggering the system to retrieve a list of unoccupied spots within the selected street.

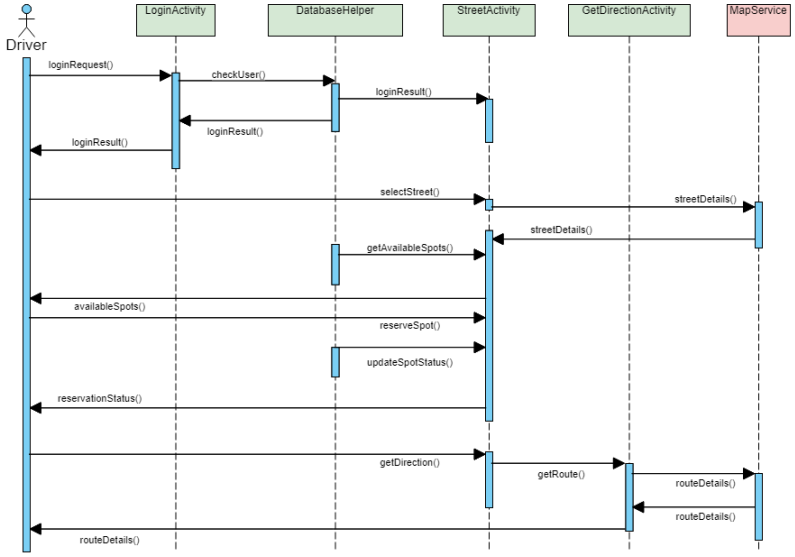

Figure 3.3. Driver Sequence Diagram

Upon receiving the list of available spots, the Driver selects a preferred spot, signaling the system to update the status of the chosen spot to "reserved." This interaction ensures that other users are aware of the spot's unavailability. Once the spot reservation is confirmed, the Driver proceeds to request navigation directions to the reserved parking location.

The system responds to the Driver's navigation request by retrieving detailed route information, including turn-by-turn directions and estimated travel time. This information is then presented to the Driver, guiding them effectively to their reserved parking spot. Throughout this process, the system ensures seamless communication and interaction between the Driver and the various system components, optimizing the parking experience for the Driver.

3.2.2.2. Attendant Sequence Diagram. The sequence diagram illustrates the interaction between an Attendant and the system components during their parking management tasks. It initiates with the Attendant sending a login request, prompting the system to validate their credentials. The system proceeds to verify the provided credentials, ensuring the Attendant's authentication.

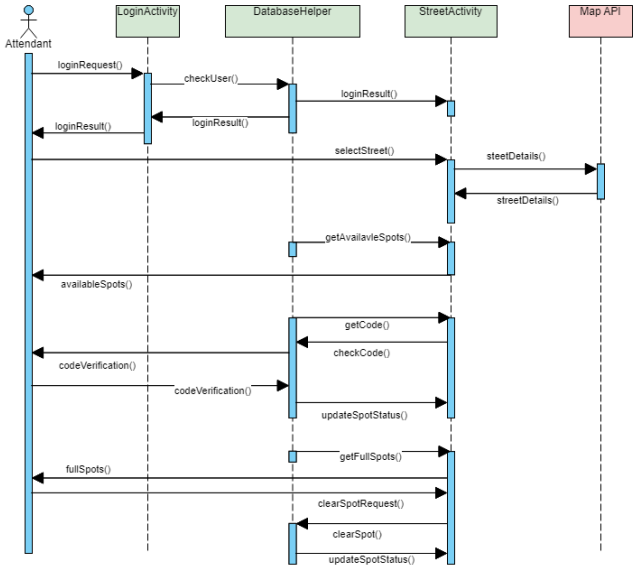

Figure 3.4. Attendant Sequence Diagram

Upon successful validation, the system grants access to the Attendant, allowing them to utilize the system's functionalities. The Attendant then navigates to select a specific parking street, indicating their choice to the system. Subsequently, the system presents the available spots within the selected street to the Attendant. Upon receiving the list of available spots, the Attendant chooses a spot for management. This triggers two actions: firstly, the system updates the status of the selected spot to reflect its "selected" status, and secondly, the Attendant initiates a code verification process for the spot.

The system verifies the code provided by the Attendant to ensure its accuracy and validity. Upon code verification, the system communicates the result back to the Attendant, indicating whether the code verification was successful or not.

Based on the code verification result, the Attendant can proceed with the spot management tasks. If successful, the Attendant continues with the spot management process, while if unsuccessful, appropriate actions are taken to rectify the issue.

Finally, upon completing the spot management tasks, the Attendant requests the system to clear the selected spot. The system updates the status of the spot accordingly and communicates the status update back to the Attendant, confirming the completion of the spot management process. Throughout these interactions, the system ensures seamless communication and coordination between the Attendant and the various system components, facilitating efficient parking management operations.

3.2.3. State Diagram

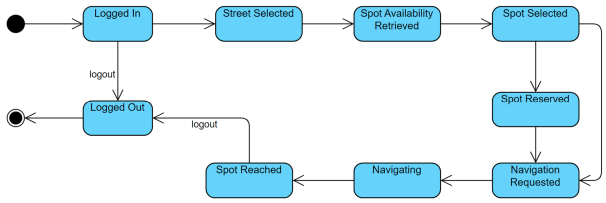

Figure 3.5. Driver's Parking State Diagram

3.2.3.1. Driver's Parking State Diagram. The state diagram for the Driver's parking journey begins with the Driver in the Logged Out state. From the Logged In state, the Driver can select a street, which moves the state to Street Selected. The system then retrieves available spots, transitioning to the Available Spots Retrieved state. The Driver selects a spot, leading to the Spot Selected state, and then reserves the spot, updating the system to the Spot Reserved state. Upon reserving a spot, the Driver can request navigation directions, transitioning to the Navigation Requested state. The system provides the directions, moving to the Directions Retrieved state. The Driver begins navigating, which transitions the state to Navigating, and upon reaching the destination, the state changes to Spot Reached. Finally, the Driver logs out, returning to the Logged Out state. This state diagram effectively captures the sequence of actions and transitions the Driver goes through during the process of parking, from logging in to reaching the reserved spot and logging out.

3.2.3.2. Attendant's Management State Diagram. The Attendant initiates a login request, which transitions the system to the Logged In state upon successful validation of credentials. From the Logged In state, the Attendant can select a street to manage, which moves the system to the Street Selected state. In this state, the system retrieves the availability of parking spots within the selected street, advancing to the Spot Availability Retrieved state. Once in the Spot Availability Retrieved state, the Attendant selects a specific spot, transitioning the system to the Spot Selected state. Here, the Attendant can check a code associated with the

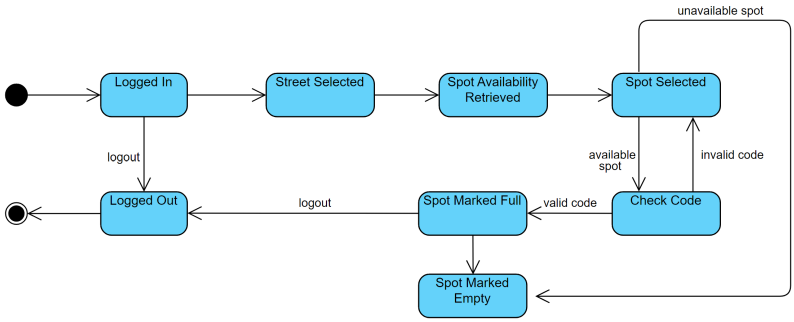

Figure 3.6. Attendant State Diagram

spot. The system then enters the Check Code state to verify the code. If the code is valid, the system transitions to Spot Marked Full, indicating that the spot is now occupied. Conversely, if the code is invalid, the system returns to the Spot Selected state, allowing the Attendant to retry. If the Attendant needs to clear a spot, they can do so from the Spot Marked Full state, which transitions the system to the Spot Marked Empty state. The Attendant can log out from any state, transitioning the system back to Logged Out. Throughout these states, the system ensures that parking spots are accurately managed by the Attendant. The state diagram illustrates the workflow from logging in and selecting streets, to managing spot availability, verifying codes, and marking spots as full or empty. This structured flow ensures that the parking management process is efficient and up-to-date, enabling effective control and organization of parking spaces by the Attendant.

3.2.4. Algorithms & Pseudo Codes

In this section, I will present the fundamental algorithms and pseudo codes that drive my project. Algorithms are precise, step-by-step instructions designed to perform specific tasks or solve particular problems efficiently. Pseudo codes, written in an easily understandable language, outline these algorithms, bridging the gap between human logic and machine execution. I will detail the main algorithms used in my project, along with their corresponding pseudo codes. Each algorithm will be explained in terms of its purpose, inputs, outputs, and the sequence of operations it follows. This will provide a clear understanding of the logic and decision-making processes behind my software. It is important to note that the algorithms and pseudo codes presented here represent only the core components of my project. The full implementation includes numerous XML files and additional algorithms that contribute to the overall functionality and user experience. These supplementary elements are essential for handling various aspects of the application, such as user interface design, data management, and interaction handling.

By examining these fundamental algorithms and pseudo codes, I aim to show the computational techniques and planning involved in my project's development. This section highlights the careful design and systematic approach that underpin the successful implementation of my application, while acknowledging the extensive additional work that supports the core functionality.

Securing the User Passwords:

· In DatabaseHelper class, initialize a hashPassword() method that takes user's password input as parameter.

· hashPassword() method computes the SHA-256 has of the password and store the result in a byte array "hash".

· Convert byte array "hash" to BigInteger "number".

· Convert number to a hexadecimal string "hexString"

· While hexString length is less than 64:

o Insert "0" at the beggining of "hexString"

· Return "hexString"

```
private String hashPassword(String password) {
    try {
        MessageDigest md = MessageDigest.getInstance( algorithm: "SHA-256");
        byte[] hash = md.digest(password.getBytes());
        BigInteger number = new BigInteger( signum: 1, hash);
        StringBuilder hexString = new StringBuilder(number.toString( radix: 16));
        while (hexString.length() < 64) {
            hexString.insert( offset: 0,  c: '0');
        }
        return hexString.toString();
    } catch (NoSuchAlgorithmException e) {
        throw new RuntimeException(e);
    }
}
```

Figure 3.7. Password Hashing Method

Changing the Parking Spot Status:

· In ZoneActivityA class, fetch the zone status by getStatus(zone) method in Database-

Helper class.

· If zone is full(1):

o Using updateStatus() method in DatabaseHelper, change zone status to empty(0).

o Generate new code for the zone using generateRandomCode() in DatabaseHelper class for the next usages of the parking zone.

o Update the code using updateCode() method in DatabaseHelper.

· If zone is empty(0):

o Get the zone code entered by attendant, retrieve the actual zone code from Database-Helper class getCode(zone) method.

o If entered code matched actual zone code: Set status to full(1), update zone status by updateStatus() method in DatabaseHelper class.

```
String enteredCode = codeEditText.getText().toString().trim();
if (TextUtils.isEmpty(enteredCode)) {
    Toast.makeText( context: ZoneActivityA.this,  text: "Please enter the code.", Toast.LENGTH_SHORT).show();
    return;
}
String zoneCode = dbHelper.getCode(zone);
if (enteredCode.equals(zoneCode)) {
    status = 1;
    dbHelper.updateStatus(zone, status);
    Toast.makeText( context: ZoneActivityA.this,  text: "Zone marked full successfully.", Toast.LENGTH_SHORT).show();
} else {
    Toast.makeText( context: ZoneActivityA.this,  text: "Wrong code.", Toast.LENGTH_SHORT).show();
}
```

Figure 3.8. Changing Parking Status Method

Reserve and Cancel a Reservation:

· In ZoneActivity class, when driver attempts to reserve a parking spot, it calls reservePark-ingSpace(zone) method in DatabaseHelper. If reservation is successfull, startReservation-Timer(zone) method is called.

· Retrieve countdown timer associated with the zone, if there is not any existing timer create new with a duration of 180 seconds. Store the newly created countdown timer in the

timerHashMap with the specified zone as the key.

· During each tick of countdown timer, calculate remaining time. If last 30 and 3 seconds, display remaining time.

· When countdown time finishes, cancel the parking reservation using cancelParkingReservation() method in DatabaseHelper class.

o Generate new code for the zone using generateRandomCode() in DatabaseHelper class for the next usages of the parking zone.

o Update the code using updateCode() method in DatabaseHelper.

```
CountDownTimer countDownTimer = new CountDownTimer( millisInFuture: 180000,  countDownInterval: 1000) {
    5 usages  zeynepdukk *
    public void onTick(long millisUntilFinished) {
        long secondsRemaining = millisUntilFinished / 1000;
        String timeRemaining = String.format("%02d:%02d", secondsRemaining / 60, secondsRemaining % 60);

        if (millisUntilFinished <= 30000 && millisUntilFinished > 29000) {
            Toast.makeText( context: ZoneActivity.this,  text: "Last 30 seconds!", Toast.LENGTH_SHORT).show();
        } else if (millisUntilFinished <= 3000) {
            Toast.makeText( context: ZoneActivity.this,  text: "Last 3 seconds", Toast.LENGTH_SHORT).show();
        }
    }
    zeynepdukk
    public void onFinish() {
        dbHelper.cancelParkingReservation(zone);
        Toast.makeText( context: ZoneActivity.this,  text: "Time is up.", Toast.LENGTH_SHORT).show();
        String newCode = dbHelper.generateRandomCode();
        dbHelper.updateCode(zone, newCode);
    }
}.start();
timerHashMap.put(zone, countDownTimer);
```

Figure 3.9. Timer For Reservation Method

Fetching a Route:

· I defined a method named fetchRoute() in GetDirectionActivity class that takes point(lat,lon) as parameter.

· Obtain the location engine instance using LocationEngineProvider and request the last location of user.

· Extract the current location coordinates (latitude, longitude). Create RouteOptions.Builder. to build the desired route options.

· Set origin point to current location, set destination using Point parameter.

· Disable alternative routes (for clarity and efficiency) and set profile to driving mode.

· Request routes from Mapbox Navigation using the constructed route options.

· Handle the route response:

If routes are ready:

§ Set the received navigation routes to the Mapbox Navigation instance.

Trigger a click on the focusLocationBtn to center the map on the current location.

· Handle any failure of cancellation of route request or location engine.

Mapbox Directions API uses a network of roads and paths derived from OpenStreetMap to calculate the fastest path.

```
private void fetchRoute(Point point) {
    LocationEngine locationEngine = LocationEngineProvider.getBestLocationEngine( context: GetDirectionActivity.this);
    zeynepdukk *
    locationEngine.getLastLocation(new LocationEngineCallback<LocationEngineResult>() {
        zeynepdukk *
        @Override
        public void onSuccess(LocationEngineResult result) {
            Location location = result.getLastLocation();
            RouteOptions.Builder builder = RouteOptions.builder();
            Point origin = Point.fromLngLat(Objects.requireNonNull(location).getLongitude(), location.getLatitude());
            builder.coordinatesList(Arrays.asList(origin, point));
            builder.alternatives(false);
            builder.profile(DirectionsCriteria.PROFILE_DRIVING);
            builder.bearingsList(Arrays.asList(Bearing.builder().angle(location.getBearing()).degrees(45.0).build(), null));
            applyDefaultNavigationOptions(builder);
            zeynepdukk
            mapboxNavigation.requestRoutes(builder.build(), new NavigationRouterCallback() {
                1 usage  zeynepdukk
                @Override
                public void onRoutesReady(@NonNull List<NavigationRoute> list, @NonNull RouterOrigin routerOrigin) {
                    mapboxNavigation.setNavigationRoutes(list);
                    focusLocationBtn.performClick();
```

Figure 3.10. Fetch Route Method

Updating the Route when Moving:

· onNewLocationMatcherResult() method in GetDirectionActivity class, updates the route concurrentyly due to drivers changing location.

· Extract the enhanced location information from the received result.

· Update the position of the navigation location provider using this enhanced location,

along with any key points provided.

· If the flag indicating focus on the user's location is enabled:

o Calculate the camera position based on the user's longitude, latitude, and bearing (direction).

o Adjust the map's camera to focus on this updated location, ensuring the user's perspective is maintained.

```
    @Override
    public void onNewLocationMatcherResult(@NonNull LocationMatcherResult locationMatcherResult) {
        Location location = locationMatcherResult.getEnhancedLocation();
        navigationLocationProvider.changePosition(location, locationMatcherResult.getKeyPoints(), latLngTransitionOptions: null, bearingTransitionOptions: null);
        if (focusLocation) {
            updateCamera(Point.fromLngLat(location.getLongitude(), location.getLatitude()), (double) location.getBearing());
        }
    }
};
```

Figure 3.11. Update Route When When Moving Method

3.2.5. Conclusion

In this section, we saw the core algorithms and their corresponding pseudo codes, which are essential for driving the project's functionalities. We started by examining how user passwords are secured, followed by managing parking spot statuses and handling reservations. Then, we explored how navigation routes are fetched and dynamically updated as the user moves. This detailed analysis provides valuable insights into the project's computational techniques and systematic approach, ensuring efficient implementation and user-friendly functionality.

4. IMPLEMENTATION

4.1. System Architecture

The architecture of the ParkPal application, as depicted in the provided screenshot, demonstrates a well-organized structure with a variety of activities dedicated to specific functionalities.

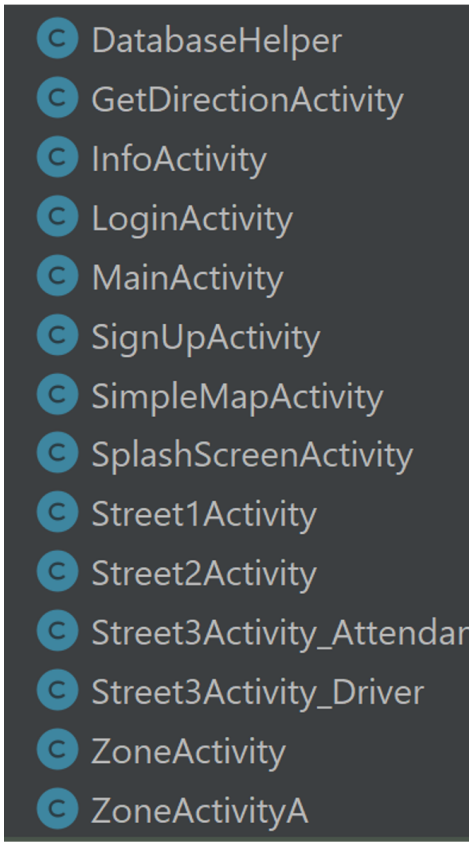

Figure 4.1. Class Architecture

The core components include LoginActivity and SignUpActivity for user authentication, and MainActivity for the primary user interface. The app incorporates several specialized activities such as GetDirectionActivity for navigation, SimpleMapActivity for map interactions, and SplashScreenActivity for the initial loading screen. The DatabaseHelper class suggests a centralized approach for managing database operations. Additional activities like Street1Activity, Street2Activity, and various Street3Activity variants (for attendants and drivers) indicate a modular design tailored for different user roles and tasks within the

parking management system. ZoneActivity and ZoneActivityA likely handle zone-specific functionalities, ensuring a comprehensive and scalable solution for efficient parking management.

· **DatabaseHelper**: The DatabaseHelper class acts as a bridge between the ParkPal application and its underlying SQLite database. It handles essential database operations such as storing, retrieving, and manipulating data related to parking zones, user information, reservations, and configuration settings. By encapsulating database-related logic, it promotes modularity and maintainability while ensuring data integrity and consistency. The class implements best practices for database management, including parameterized queries and error handling, to enhance security and reliability. Overall, the DatabaseHelper class plays a vital role in enabling smooth data management within the ParkPal application, contributing to its functionality and performance.

· **GetDirectionActivity:** The GetDirectionActivity class is a crucial component of the ParkPal application, responsible for enabling navigation functionality using Mapbox services. Upon initialization, this class retrieves the latitude and longitude coordinates of the selected parking zone, either from user input or the application's database. It leverages Mapbox's mapping and navigation capabilities to generate a route from the user's current location to the designated parking zone. Through seamless integration with Mapbox SDK, the class allows users to visualize their navigation route on an interactive map interface. Additionally, it manages user interactions such as zooming to provide a customized and immersive navigation experience. By encapsulating Mapbox navigation functionalities, this class ensures efficient and hassle-free parking navigation for users, enhancing the overall usability and user experience of the ParkPal application.

· **LoginActivity:** This class facilitates the authentication process for registered users. Upon launching the application, users are directed to the LoginActivity where they are prompted to enter their credentials, typically a username/email and password. The LoginActivity then verifies these credentials against the stored user data in the application's database. If the credentials match an existing user account, the user is granted access to the application's main features. However, if the credentials are invalid or the user account does not exist, appropriate error messages are displayed, prompting the user to retry or register for a new account.

· **InfoActivity:** This class serves as a pivotal component within the ParkPal application, facilitating the dissemination of crucial contact information about the developer, as well as providing comprehensive guidelines on how to effectively utilize the application's features.

As the developer, this class offers a platform to showcase pertinent details such as contact email, social media handles, and any other relevant means of communication, fostering transparency.Moreover, the InfoActivity class meticulously outlines step-by-step instructions and best practices for navigating the ParkPal application, ensuring users can maximize its functionality with ease and confidence. By amalgamating developer contact information with user guidance, the InfoActivity class epitomizes user-centric design, prioritizing seamless interaction and support within the ParkPal ecosystem.

· **SignUpActivity:** This class enables new users to create an account within the ParkPal application. It provides a user-friendly interface where individuals can input their personal information, including username, email address, password, and any other required details. Upon submission of the registration form, the SignUpActivity validates the provided information to ensure completeness and adherence to any specified criteria (such as password strength requirements). Upon successful registration, the user's account details are securely stored in the application's database, allowing them to subsequently log in and access the application's functionalities. Additionally, the SignUpActivity may incorporate features such as email verification to enhance account security and prevent fraudulent registrations.

· **SplashScreenActivity:** The SplashScreenActivity class serves as the initial screen displayed to users when they launch the ParkPal application. Its primary purpose is to provide a visually appealing introduction to the app while essential setup tasks are performed in the background. Typically, the SplashScreenActivity displays the application's logo or branding along with animations or brief loading indicators to engage users during the initialization process. Once the necessary setup tasks, such as loading resources or initializing components, are completed, the SplashScreenActivity transitions to the appropriate screen, such as the login or home screen, based on the application's logic. Overall, the SplashScreenActivity enhances the user experience by providing a seamless transition from the application launch to its main functionality.

· **Street1Activity, Street2Activity:** Street1Activity and Street2Activity are classes within the ParkPal application responsible for displaying information about parking zones on different streets. These activities provide a user interface where users can view the availability of parking spaces in various zones and make reservations if needed.

In both activities, the layout typically includes buttons representing individual parking zones, with each button indicating the current availability of parking spaces through visual indicators such as color-coded icons or text labels. Users can interact with these buttons to

view more details about a specific zone or reserve a parking space.

Behind the scenes, these activities interact with the DatabaseHelper class to retrieve information about parking zones and update their availability status based on user actions. Additionally, they utilize SharedPreferences to store user preferences or session data, such as the user's role (driver or attendant), which influences the behavior of certain features within the activities.

Overall, Street1Activity and Street2Activity contribute to the core functionality of the ParkPal application by providing users with real-time information about parking availability and facilitating the reservation process, thereby enhancing the user experience and optimizing parking management.

· **Street3Activity_Attendant, Street3Activity_Driver:** In Street3Activity_Attendant, attendants can oversee the parking space availability and occupancy status. Through a user-friendly interface, attendants can view the number of available spaces, mark spaces as occupied or vacant, and adjust the status in real-time. Utilizing EditText and TextView components, attendants can increment or decrement the available space count and update the database accordingly. Additionally, they can facilitate parking space reservations by interacting with buttons to add or reduce the available space count, ensuring seamless management of parking facilities.

Driver Mode: Conversely, in Street3Activity_Driver, drivers are empowered to access vital information regarding parking availability and directions to designated parking zones. The activity presents drivers with the current availability of parking spaces, enabling them to make informed decisions about where to park. Through TextView components, drivers can view the available and occupied space counts, providing valuable insights into parking availability. Moreover, the activity integrates with the GetDirectionActivity to offer drivers optimized routes to their desired parking locations, enhancing their navigation experience.

In both modes, Street3Activity leverages the DatabaseHelper class to retrieve and update parking space data, ensuring accuracy and reliability. By catering to the distinct needs of attendants and drivers, Street3Activity plays a pivotal role in streamlining parking management and enhancing the overall user experience within the ParkPal application.

· **ZoneActivity**: ZoneActivity is a crucial component of the ParkPal application, facilitating the reservation and management of parking zones for both drivers and attendants. This

activity serves as an interface for users to interact with specific parking zones, enabling them to reserve parking spaces, view zone details, and access navigation instructions.

For drivers, ZoneActivity provides functionalities to reserve parking spaces within designated zones. Upon selecting a parking zone, drivers can initiate the reservation process by clicking the reserve button. The activity then initiates a countdown timer to indicate the remaining reservation time. Additionally, drivers can access navigation directions to their reserved parking zones through integrated map functionalities, enhancing their parking experience.

For attendants, ZoneActivityA offers tools to monitor and manage parking zones effectively. Attendants can view the occupancy status of each zone and update the status based on space availability. Through intuitive UI components, attendants can mark zones as empty or full, facilitating efficient management of parking facilities.

Both versions of ZoneActivity leverage the DatabaseHelper class to retrieve and update parking zone data, ensuring synchronization between user interactions and database records. By providing a user-friendly interface and essential functionalities, ZoneActivity contributes to the seamless operation and optimization of parking facilities within the ParkPal application.

· **ZoneActivityA:** ZoneActivityA is an essential feature of the ParkPal application tailored specifically for attendants responsible for managing parking zones. This activity empowers attendants with tools to monitor and oversee parking zone occupancy, ensuring efficient utilization of parking facilities.

Through ZoneActivityA, attendants can access detailed information about parking zones, including occupancy status and assigned codes. Attendants utilize EditText and Button components to update the occupancy status of parking zones, marking them as either empty or full based on space availability. Additionally, attendants can verify parking zone occupancy by entering zone-specific codes provided by drivers, enabling them to maintain accurate records of parking space utilization.

The activity integrates seamlessly with the ParkPal database through the DatabaseHelper class, enabling attendants to retrieve real-time data and update zone statuses promptly. By providing attendants with intuitive tools to manage parking zones effectively, ZoneActivityA contributes to the optimization of parking operations and enhances overall user experience within the ParkPal application.

4.2. User Interface Design

The user interface design of a mobile application plays a pivotal role in shaping user experiences and interactions. In this section, we delve into the design aspects of the ParkPal application, focusing on key screens such as the login and splash screens. Through a combination of intuitive layouts, thoughtful placement of elements, and adherence to modern design principles, ParkPal aims to provide users with a seamless and engaging experience from the moment they launch the app.

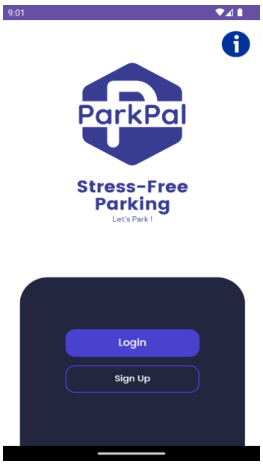

Figure 4.2. Splash Screen

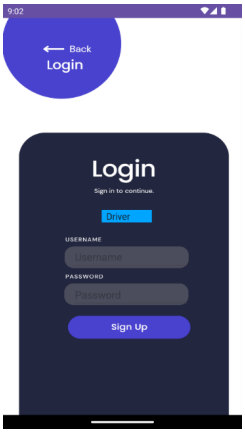

Figure 4.3. Login Screen

The login and splash screens are integral components of the ParkPal application's user interface. The login screen offers users a streamlined authentication process, featuring input fields for username and password, along with the option to choose between driver and attendant roles. This facilitates secure and personalized access to the app's features. On the

other hand, the splash screen serves as a visual introduction to the ParkPal brand, displaying captivating visuals and the app logo during the loading process. Together, these screens create a seamless and engaging user experience, setting the stage for users to explore the functionalities of ParkPal with ease.

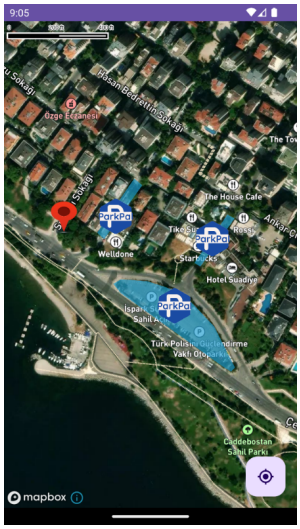

Figure 4.4. Map Screen for Selecting Streets

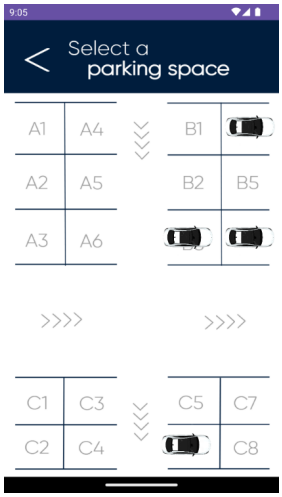

Figure 4.5. Selecting Park Space Screen

ParkPal utilizes the Mapbox API to enhance its user interface, providing an intuitive mapping system that showcases streets with dynamic overlays. These overlays, depicted as blue polygons, allow users to easily differentiate between active streets and select their desired location by simply tapping on the map. Once a street is selected, the interface reveals a comprehensive view of both vacant and occupied parking spots. Occupied spots are represented by car icons, enabling users to effortlessly discern between available and occupied spaces.

This visual approach not only enhances the aesthetic appeal of the interface but also improves usability, empowering users with clear, actionable information for efficient parking spot selection.

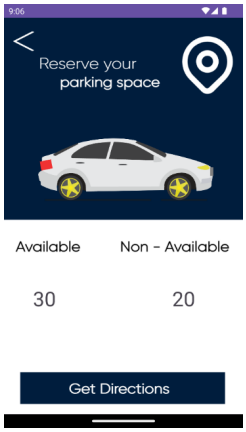

Figure 4.6. Parking Area Availability Screen For Drivers

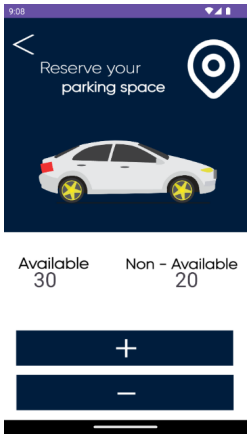

Figure 4.7. Parking Area Management Screen For Attendants

In ParkPal, a unique system is implemented for parking lots. This system is divided into two modes: driver and attendant. Attendants have the capability to manually update the status of parking spaces, marking them as vacant or occupied as vehicles enter or leave so in their interface there are plus an. On the other hand, drivers can view the number of available and occupied parking spaces and receive directions to their desired parking lot. As a result, ParkPal users can effectively manage parking lots in both attendant and driver modes, enhancing both the visual appeal and usability of the system.

ParkPal integrates LottieFiles, a library for utilizing Lottie animations, to introduce dynamic visual elements into the user interface. By leveraging Lottie animations, ParkPal en-

riches the user experience with engaging visual feedback and interactive elements, enhancing both the aesthetic appeal and usability of the application.

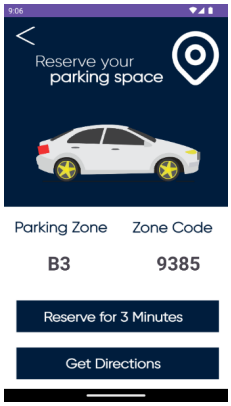

Figure 4.8. Parking Zone Screen For Drivers

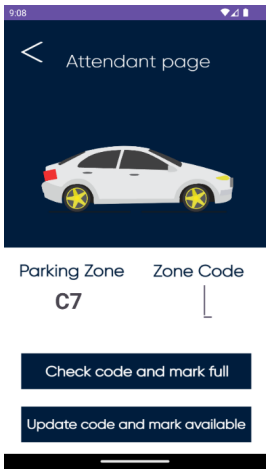

Figure 4.9. Parking Zone Screen For Attendants

After selecting a parking spot on the street, users are presented with a subsequent screen tailored to their mode selection: either ”Driver” or ”Attendant.” If the user opts for the ”Driver” mode, they receive a personalized zone code assigned to the chosen parking space. This code serves as a unique identifier, enabling the user to communicate effectively with parking attendants or other relevant personnel. With this zone code in hand, drivers can proceed to park in their designated spot, or alternatively, they have the option to reserve it for a brief period, typically 3 minutes, to facilitate a smooth parking process. Moreover, drivers can utilize the application to request directions to their selected parking area, streamlining navigation within urban environments.

Conversely, when operating in ”Attendant” mode, users are tasked with managing park-

ing spaces and verifying the status of allocated zones. Attendants are responsible for checking the validity of zone codes provided by drivers seeking to park in specific spots. Upon receiving a code from a driver, attendants can cross-reference it with the corresponding vacancy to ascertain its authenticity. If the code matches, indicating a successful reservation, the attendant marks the spot as occupied, ensuring accurate monitoring of available parking spaces. Furthermore, attendants have the authority to clear occupied spots if necessary. To accomplish this, attendants update the code associated with the parking space and utilize the application to designate the spot as vacant, allowing for efficient turnover and utilization of parking resources.

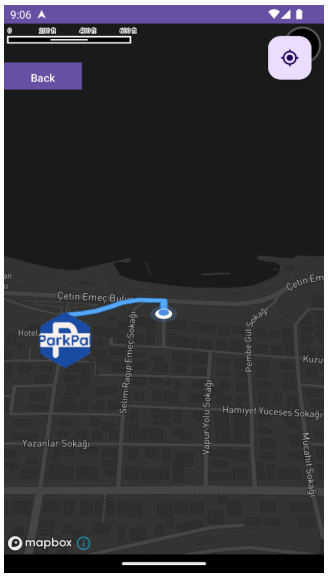

Figure 4.10. Routing Screen

Upon pressing the ”Get Direction” button on the respective pages, our routing interface is activated, guiding users from their current location to the designated parking spot based on its latitude and longitude coordinates. Embracing a dark-themed aesthetic, we have curated an immersive experience that enhances visibility and usability. By tilting the camera at a 45-degree angle, we have optimized the viewing perspective, ensuring a pleasant and intuitive navigation experience for users.

4.3. Mapbox API : Marking the Streets

In the ”Marking The Streets with Mapbox API” section, the process begins by selecting the appropriate Mapbox style, specifically opting for ”satellite streets” to ensure clear visibility of the streets within their surroundings. Following this, a dataset is created to define the streets where ParkPal operates, facilitating easy identification by users. Utilizing the drawing

tool provided by Mapbox, polygons are drawn to accurately outline the streets, with a focus on three specific streets within the dataset. [7]

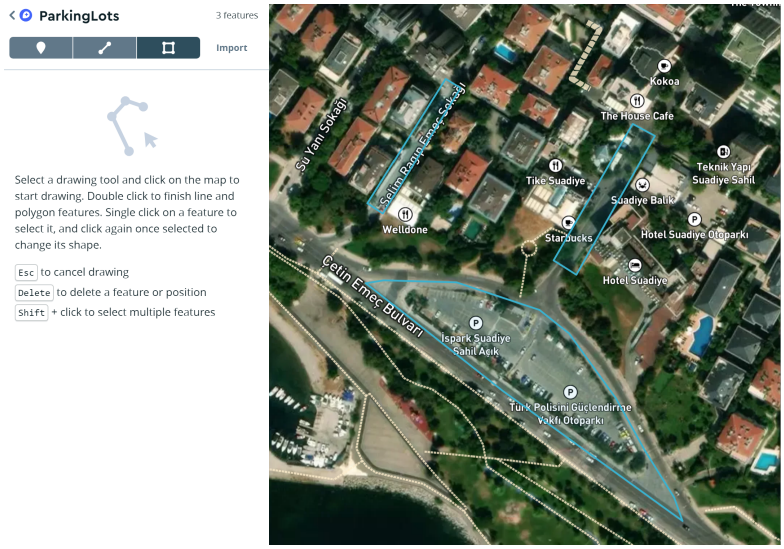

Figure 4.11. Mapbox Polygon Drawing For Parking Lot Streets

Once the polygons are drawn and the streets are clearly defined, the dataset is named and exported. This exported dataset is then integrated into the MAP style used in the main application, ensuring seamless incorporation of the streets' delineations into the overall design and functionality.

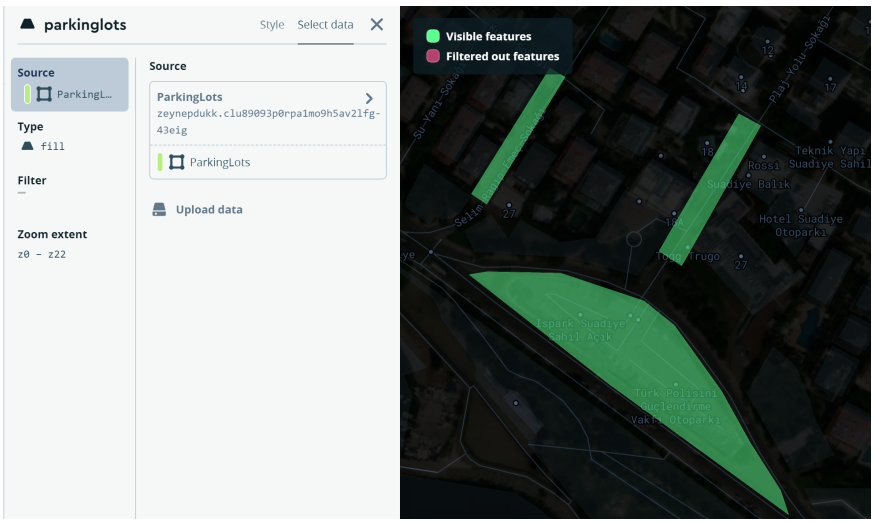

Figure 4.12. Dataset for Streets

Next, the "parkingLots" dataset is overlaid onto the selected style to effectively integrate it into the application's visual representation. User interaction is enhanced by making the

streets clickable. This is achieved by adding point annotations to the streets, enabling users to interact with them directly within the application interface.

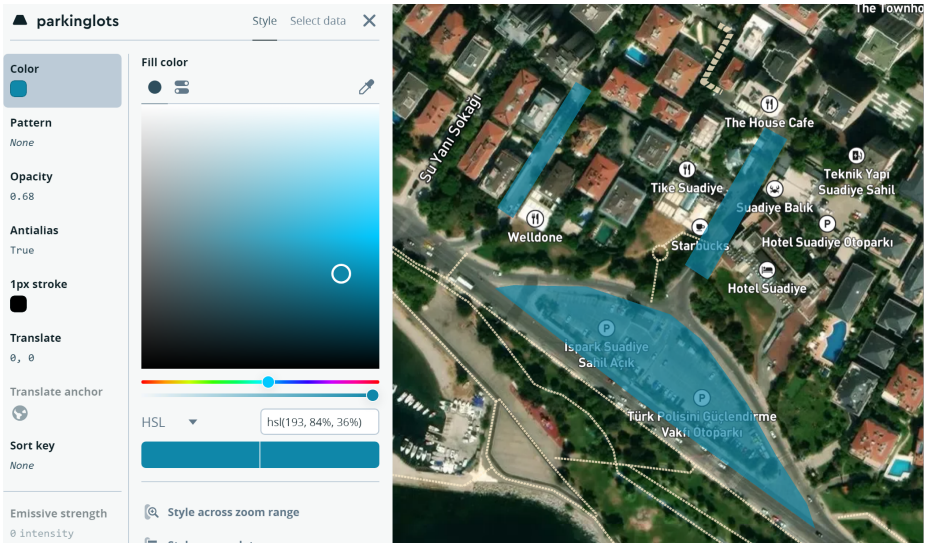

Figure 4.13. Mapbox Dataset Integration To Map Style

The appearance of the delineated streets within the application is determined, with a blue fill color chosen to maintain consistency with the application's theme.

Furthermore, the versatility and adaptability of the Mapbox API are highlighted, as it offers easy customization and flexibility for future enhancements and modifications. By integrating this API into the application and making necessary adjustments using a token, seamless compatibility and functionality are ensured.

The process of marking the streets with the Mapbox API involves selecting the style, creating and exporting the dataset, integrating it into the application's MAP style, overlaying the dataset, determining visual appearance, enhancing user interaction, and ensuring adaptability for future developments. (MAPBOX)

4.4. Implementation Details

The implementation process of ParkPal App, involves important coding approaches and well defined structure for maintainable code. In this section provides the approach taken, the overall structure of the implementation, and the key libraries utilized throughout the development process. By exploring these details, we'll gain a comprehensive understanding of the methodologies, technologies, and tools employed to create ParkPal.

4.4.1. Approach and Structure

The implementation of the ParkPal application followed a structured approach aimed at achieving modularity, scalability, and maintainability. The project utilized the Model-View-ViewModel (MVVM) architecture pattern to separate concerns and promote code organization. This architecture facilitated the decoupling of UI logic from business logic, enhancing testability and facilitating future updates.

The project structure was organized into distinct modules representing different layers of the application. The data layer comprised classes responsible for database operations, such as storing parking zone information, user data, and reservations. The presentation layer included activities and fragments that interacted with the user, while the domain layer encapsulated business logic and data processing operations.

Furthermore, the implementation embraced design principles such as single responsibility and abstraction to ensure clean and maintainable code. By adhering to these principles and leveraging architectural patterns, the implementation facilitated seamless collaboration among team members and facilitated future enhancements.

4.4.2. Libraries

The implementation of ParkPal relied on several third-party libraries to streamline development and enhance functionality. Key libraries utilized in the project included:

· **AndroidX Libraries:** ParkPal uses several AndroidX libraries to enhance its functionality. The androidx.appcompat:appcompat:1.6.1 library ensures compatibility across different Android versions, while com.google.android.material:material:1.11.0 provides modern UI components based on Material Design. The androidx.constraintlayout:constraintlayout:2.1.4 library is employed to create responsive layouts, optimizing the app for various screen sizes and improving performance.

o androidx.appcompat:appcompat:1.6.1

o com.google.android.material:material:1.11.0

o androidx.constraintlayout:constraintlayout:2.1.4

· **Testing Libraries:** To ensure reliability, ParkPal incorporates essential testing libraries. junit:junit:4.13.2 is used for unit testing individual components. androidx.test.ext:junit:1.1.5 facilitates Android-specific instrumentation tests, and androidx.test.espresso:espresso-core:3.5.1 is used for UI testing, simulating user interactions to verify UI behavior.

o junit:junit:4.13.2 (for unit testing)

o androidx.test.ext:junit:1.1.5 (for Android instrumentation testing)

o androidx.test.espresso:espresso-core:3.5.1 (for UI testing)

· **Mapbox SDK:** The Mapbox SDK provides advanced mapping and navigation features for ParkPal. The com.mapbox.maps:android:10.16.3 library integrates customizable Mapbox maps, while com.mapbox.extension:maps-compose:11.2.1 supports map integration within Jetpack Compose. The com.mapbox.navigation:android:2.17.1 library offers robust navigation functionalities, including turn-by-turn directions and real-time traffic updates.

o com.mapbox.maps:android:10.16.3 (for integrating Mapbox maps)

o com.mapbox.extension:maps-compose:11.2.1 (for composing Mapbox maps with Jetpack Compose)

o com.mapbox.navigation:android:2.17.1 (for navigation functionalities)

· **Animation Library:** ParkPal enhances its visual appeal with the Lottie animation library. The com.airbnb.android:lottie:6.0.0 library enables the integration of high-quality animations created in Adobe After Effects. This allows ParkPal to feature engaging animations, improving the user experience with dynamic and interactive visual elements.

o com.airbnb.android:lottie:6.0.0 (for integrating Lottie animations)

4.5. Implementation Challanges and Solutions

During the implementation process of ParkPal App, I faced many challanges. In following section, we will discuss these challenges and the solutions we found to tackle them effectively.

Mapbox API Integration with Java Language: Integrating Mapbox API features with Java posed challenges as the latest documentation and codes are primarily compatible with Kotlin. Navigating through the newer Kotlin-centric resources while working with Java required extensive research. Communication with Mapbox representatives played a vital role in identifying workarounds and understanding the compatibility issues. For example, newer versions often favored Kotlin for drawing polygons programmatically, leading to the utilization of point annotations for creating clickable elements.

Integration with the Mapbox API: Integrating Mapbox API to handle map functionalities such as marking streets, defining datasets, and styling maps is complex, especially when dealing with various components like annotations and route handling.

Layout Design for Streets: Designing layout files for streets with multiple parking spots posed challenges, especially in representing parking spots dynamically with car icons as they filled up. Experimenting with different layout styles and manually adjusting each space was time-consuming but essential for achieving the desired visual appearance.

Managing Class Hierarchies and Connections: As the application evolved, tracking hierarchies and connections between classes became challenging. However, adopting logical naming conventions and defining a clear architecture helped minimize confusion and streamline development.

Handling Dependencies and Libraries: Dealing with dependencies, particularly older libraries required by some functionalities, posed challenges in the dependency section. However, overcoming this hurdle was possible by sourcing specific resources online and implementing them effectively within the project structure.

Handling Location and Permissions: Managing user location and obtaining necessary location permissions was challenging, especially when ensuring smooth transitions between different map views and functionalities.

Database Management: Implementing database operations for tasks like updating parking spot statuses, reserving spots, and retrieving routes efficiently while maintaining data integrity was demanding. Ensuring smooth functionality across various user actions, such as updating parking spot statuses from occupied to vacant or vice versa, required meticulous handling of database transactions. Furthermore, implementing reservation systems involved intricate logic to handle concurrent requests while preventing data inconsistencies. Main-

taining data integrity throughout these operations was crucial to ensure accurate and reliable information for users interacting with the application.

4.6. Conclusion

In the implementation phase of the ParkPal application, we meticulously designed and developed various components to deliver a seamless and intuitive parking management solution. The chapter on System Architecture provided insights into critical components such as the DatabaseHelper and activity classes, emphasizing their roles in enabling essential functionalities like database management, user authentication, and navigation.

Our exploration of User Interface Design showcased the importance of visually engaging interfaces in enhancing user experiences. By integrating Mapbox API and Lottie animations, we elevated the application's aesthetic appeal while ensuring usability and functionality.

The section on Mapbox API: Marking the Streets demonstrated our efforts to integrate mapping functionalities seamlessly into ParkPal. From selecting map styles to defining datasets and enhancing user interaction, we leveraged Mapbox's capabilities to provide users with a dynamic and immersive experience. Despite encountering challenges such as integrating Mapbox API features with Java and managing class hierarchies, dependencies, and database operations, we remained resilient. Through strategic problem-solving and collaboration, we overcame these obstacles, ensuring the successful implementation of ParkPal.

5. TEST AND RESULTS

5.1. User Survey

To assess the user experience of the ParkPal application, a comprehensive survey was conducted. The survey aimed to gather valuable feedback on various aspects of the application, including ease of use, functionality, visual design, performance, and overall satisfaction. Participants were asked to rate their experience based on a series of structured questions, providing insights that are crucial for identifying strengths and areas for improvement. The feedback collected from this survey will play a pivotal role in guiding future enhancements and ensuring that ParkPal meets the needs and expectations of its users effectively. Total 15 responses were collected for each question, below you can see the pie charts for results.

5.1.1. Question 1

First question of User Survey: How easy was it to navigate through the application?

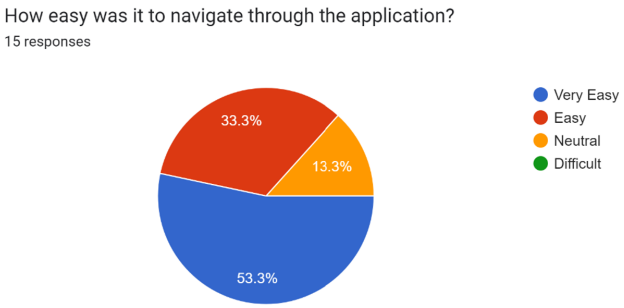

Figure 5.1. First Question of Survey

The first pie chart represents the responses to the question, ”How easy was it to navigate through the application?” Out of 15 respondents, a majority (53.3%) found it ”Very Easy” to navigate, while 33.3% rated it as ”Easy.” A smaller portion (13.3%) felt ”Neutral” about the navigation ease, and no respondents rated it as ”Difficult.” The majority of users find the application easy to navigate, indicating a user-friendly interface.

5.1.2. Question 2

Second question of the User Survey: How satisfied are you with the functionality and features of the application?

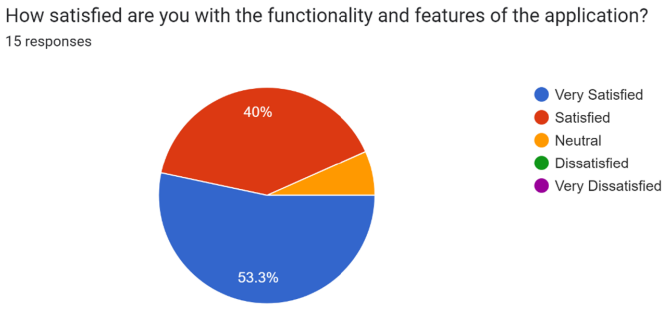

Figure 5.2. Second Question of Survey

The second pie chart shows the results for ”How satisfied are you with the functionality and features of the application?” Here, 53.3% of users reported being ”Very Satisfied” with the functionality. Meanwhile, 40% indicated they were ”Satisfied,” and a small fraction (6.7%) felt ”Neutral” about the application’s features. There were no responses indicating dissatisfaction. Most users are highly satisfied with the application’s functionality, suggesting it meets their needs effectively.

5.1.3. Question 3

Third question of the User Survey: How do you rate the visual design and layout of the application?

The third pie chart addresses the question, ”How do you rate the visual design and layout of the application?” Almost half of the respondents (46.7%) rated the visual design as ”Excellent,” and 40% rated it as ”Good.” A smaller segment (13.3%) felt ”Neutral,” with no ratings for ”Poor” or ”Very Poor.” The visual design and layout of the application are well-regarded, with a significant majority rating it positively.

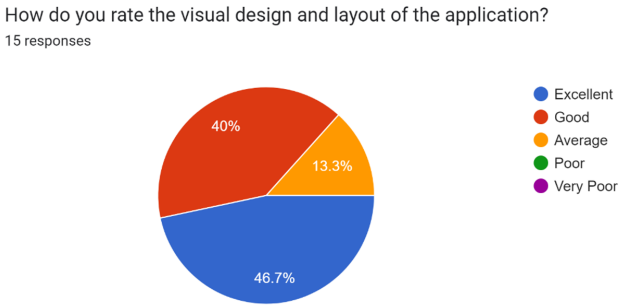

Figure 5.3. Third Question of Survey

5.1.4. Question 4

Fourth question of the User Survey: How would you rate the performance and speed of the application?

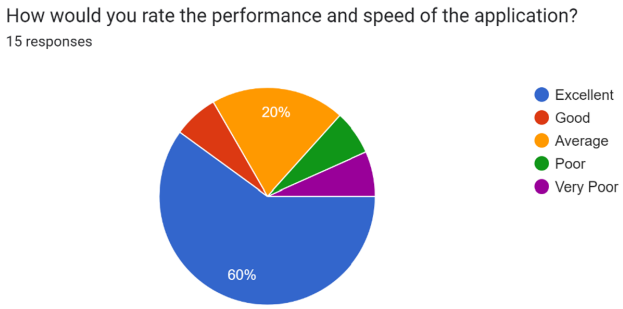

Figure 5.1.4 Fourth Question of Survey

Figure 5.4. Fourth Question of Survey

The fourth pie chart illustrates the responses to ”How would you rate the performance and speed of the application?” A significant majority (60%) rated the performance as ”Excellent.” Additionally, 20% found it ”Good,” and another 20% rated it as ”Average.” No users rated the performance as ”Poor” or ”Very Poor.” Users are very pleased with the application’s performance and speed, indicating a robust and efficient system.

5.1.5. Question 5

Fifth question of the User Survey: Overall, how satisfied are you with your experience using the application?

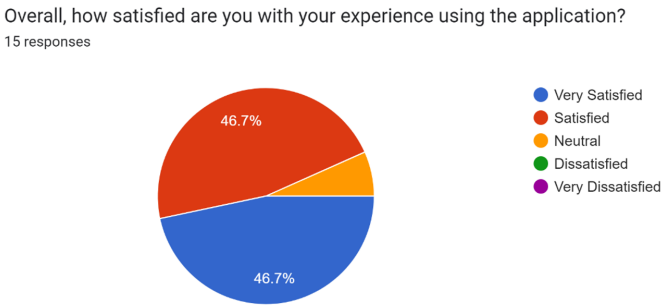

Figure 5.1.5. Fifth Question of Survey

Figure 5.5. Fifth Question of Survey

The final pie chart summarizes the overall satisfaction with the application, with the question ”Overall, how satisfied are you with your experience using the application?” Both ”Very Satisfied” and ”Satisfied” received equal ratings of 46.7%. Only a small portion (6.7%) felt ”Neutral,” with no users indicating dissatisfaction. Overall user satisfaction is high, reflecting a generally positive user experience with the application. The user survey results highlight a positive reception of the application, with high marks across ease of navigation, functionality, visual design, performance, and overall satisfaction. The consistent positive feedback underscores the application’s success in providing a user-friendly and effective solution.

5.2. Testing

This chapter presents the tests and results conducted to evaluate the ParkPal Application.

5.2.1. Test1 : Login Functionality

The table titled ”Tests for Login Functionality” presents a summary of test cases for evaluating the login process of an application. It lists three distinct events: authenticating a user with correct input data, incorrect input data, and null input values. Each event’s expected outcome, frequency of tests, and the results in terms of success and failure are detailed. For correct input data, out of 10 attempts, 9 resulted in successful logins, indicating one failure.

Both incorrect input data and null input values consistently resulted in the expected outcomes of failed login and showing that input is required, respectively, with all 10 attempts in each case being successful. This data helps in assessing the reliability and robustness of the login functionality.

Table 5.1. Tests for Login Functionality

Event	Expected Outcome	Frequency	Success	Failure
Authenticate user with correct input data.	Succesfull login.	10	9	1
Authenticate user with incorrect input data.	Failed login.	10	10	-
Authenticate user with null input value.	Show input is required.	10	10	-

5.2.2. Test2 : Sign Up Functionality

The table titled ”Tests for Sign Up Functionality” outlines the results of testing the sign-up process within an application. It evaluates two key events: authenticating a user with correct input data and with null input values. For each event, the table records the expected outcomes, the number of times the test was performed (frequency), and the results in terms of success and failure. When users provided correct input data, 9 out of 10 attempts resulted in a successful sign-up, indicating a single failure. When users provided null input values, all 10 attempts correctly prompted that input is required, with no failures recorded. This data is crucial in understanding the effectiveness and reliability of the sign-up functionality, ensuring it operates as intended under different scenarios.

Table 5.2. Tests for Sign Up Functionality

Event	Expected Outcome	Frequency	Success	Failure
Authenticate user with correct input data.	Successfull login.	10	9	1
Authenticate user with null input value.	Show input is required.	10	10	-

5.2.3. Test3 : Overall Application Functionality

The table titled ”Tests for Overall Functionality” provides a comprehensive overview of the application’s performance across various functionalities, detailing the results of several key events. The map functionality after the login process succeeded in 6 out of 10 tests, indicating room for improvement with 4 failures. The street functionality and selecting parking spots both performed flawlessly, achieving 100

After a reservation timer ends, the parking lot correctly appeared as empty in 9 out of 10 tests, showing one failure. The routing functionality provided directions accurately 8 times out of 10, but failed in 2 instances. User location retrieval on the map was consistently accurate across all tests. Finally, the ”Back” button on the street page successfully reloaded the map in 7 out of 10 attempts, with 3 failures noted. This data highlights both the strengths

and areas needing improvement in the overall application functionality, ensuring it remains user-friendly and reliable.

Table 5.3. Tests for Overall Functionality

Event	Expected Outcome	Frequency	Success	Failure
Map functionality after login process.	Successfully loaded map required.	10	6	4
Street functionality after selecting from map.	Successfully retrieved parking status required.	10	10	-
Selecting parking spot in a street.	Click successfully on parking lots buttons required.	10	10	-
Reserving parking spot as driver.	After reservation button , parking spot should reserved for 3 minutes.	10	10	-
Check for correct code functionality as attendant.	Parking lot successfully marked full.	10	10	-
Check for wrong code functionality as attendant.	Show error toast due to wrong code.	10	10	-
Cancelling reservation after timer ends for driver.	Parking lot should appear as empty after 3 minutes of reservation.	10	9	1
Routing functionality after clicking “set route” button.	Provides directions from the current location to the selected parking spot.	10	8	2
User’s location appear correctly on map.	User’s lat lon should be retrieved successfully.	10	10	-
“Back” button functionality in Street’s Page	Successfully load the map when clicking “back” button.	10	7	3

5.2.4. Test Results

The bar chart illustrates the results of functionality tests conducted on the application, focusing on three key areas: Login Functionality, Sign Up Functionality, and Overall Application Tests.

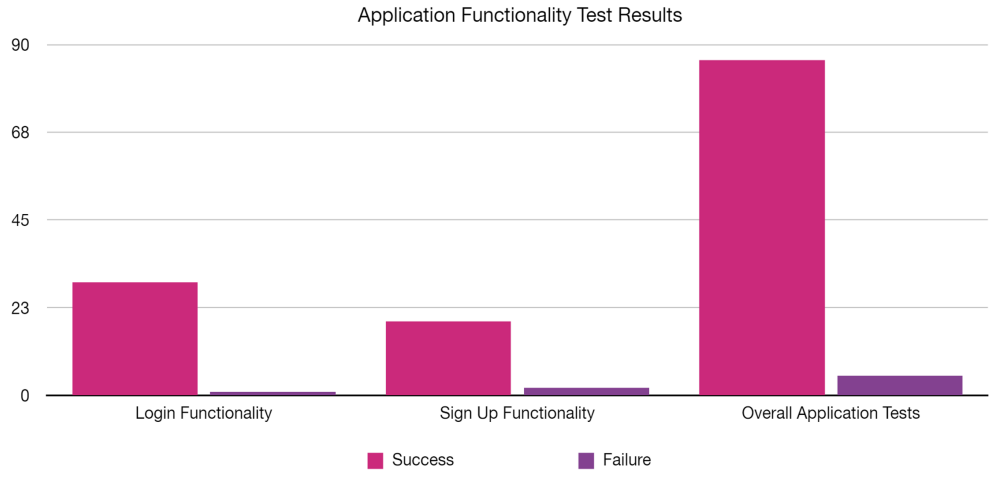

Figure 5.6. Column Chart for Application Functionality Test Results

For the Login Functionality, the chart shows a high success rate with 23 successful attempts and minimal failures. The Sign Up Functionality also demonstrates a good success rate, though slightly lower than the login functionality, with fewer successful attempts but still maintaining a low failure rate. The Overall Application Tests category indicates a robust performance, with a significant number of successful tests (nearly 90) and very few failures.

The application exhibits strong and reliable functionality across its critical areas, with high success rates in login and overall application performance, and slightly lower but still satisfactory success rates in sign-up functionality. This indicates a well-implemented and dependable system.

5.2.5. Tests For Application Performance for Average Response Times

To assess the performance of the ParkPal mobile application, I conducted a series of performance tests aimed at measuring response times and system efficiency for various user interactions.

Table 5.4. Tests for Overall Application Performance for Response Times

Event	Expected Result	Average Time
Authenticate user with incorrect input data.	Map displayed within a reasonable time.	2.2 sec
Map interface loads with markers for available parking spots.	The map loads in a reasonable time	4.5 sec
A user selects an available parking spot, and the app generates a route.	The route is generated.	2.9 sec
Marking a parking spot as occupied or vacant.	Marked in a reasonable time.	1.5 sec

The tests included scenarios such as user login, map loading with parking spot detection, navigation route generation and status updates by attendants. On average, user authentication took 2.2 seconds, map loading and parking spot detection took 4.5 seconds, generating navigation routes took 2.9 seconds and updating parking spot status took 1.5 seconds. These results indicate that the app performs efficiently, providing a smooth and responsive user experience across key functionalities.

6. CONCLUSION & FUTURE WORK

6.1. Project Achievements

The development and deployment of the park management system leveraging the Mapbox API have been highly successful. The project has met all its primary objectives, including real-time park occupancy tracking, interactive map features for navigation, and user-friendly interfaces for both park administrators and visitors. In a big city, this system significantly reduces the time people spend searching for parking lots, enhancing convenience and reducing congestion. The integration of the Mapbox API has enabled high precision and visually appealing maps, significantly improving the user experience. Additionally, the system's robust backend supports efficient data handling and ensures scalability for future expansions. Overall, the project has demonstrated the feasibility and benefits of a technology-driven approach to park management.

6.2. Project Review

The ParkPal project review provides a comprehensive evaluation of the system's limitations, strengths, and areas for improvement.

6.2.1. Project Strengths

One of the key strengths of the project is its intuitive user interface, which simplifies navigation and access to park information. The use of the Mapbox API has added significant value by providing detailed and interactive maps, which are crucial for effective park management and enhancing visitor experience. Furthermore, the real-time data capabilities of the system allow for timely updates on park occupancy and availability of amenities, which improves operational efficiency. The project's modular architecture also ensures that it is easily maintainable and can be expanded with additional features as needed.

6.2.2. Project Limitations

Despite its many strengths, the project has a few limitations. One of the primary constraints is the dependency on internet connectivity for real-time updates, which can be a challenge in areas with poor network coverage. Additionally, while the system is scalable, initial deployment costs can be high due to the need for advanced hardware and software

resources. There is also a learning curve associated with the system for park staff, which requires adequate training and support. Lastly, integrating additional third-party services, while feasible, can be complex and time-consuming.

6.3. Future Work

Looking ahead, several enhancements and expansions can be pursued to further improve the park management system. Future work could include the integration of AI-driven analytics to predict visitor trends and optimize resource allocation. Expanding the system to include more comprehensive environmental monitoring features, such as air quality and wildlife tracking, could also provide valuable data for park management. Additionally, developing a mobile app version of the system would increase accessibility for both park staff and visitors. The implementation of a payment system and QR code integration could streamline the parking process, allowing users to easily pay for parking and access services via their smartphones. Finally, incorporating more offline capabilities could address the limitations posed by unreliable internet connectivity. (Veeraiah, Smart Parking System with QR Code, 2019)

6.4. Conclusions

The project has successfully developed a robust and user-friendly park management system that leverages the advanced mapping capabilities of the Mapbox API. The system's strengths in providing real-time data, ease of navigation, and scalability have been well demonstrated. While there are some limitations, such as the reliance on internet connectivity and initial deployment costs, these do not overshadow the significant benefits and potential of the system. Future enhancements, particularly in AI integration, offline capabilities, and the addition of payment and QR code features, are poised to further elevate the system's effectiveness and user satisfaction. Overall, the project represents a significant step forward in modernizing park management through technology.

Bibliography

[1] N. Idris Leng Tamil, *Car Park System: A Review of Smart Parking System and its Technology*. Information Technology Journal, 2009.

[2] . C. Aichner T., "Evaluation of mobile map apis.," in *Design Science Research in Information Systems and Technology (DESRIST) IEEE Xplore*, IEEE Xplore, 2017.

[3] M. Bajçinovci, "Anxiety and urban stress for parking spots," *Journal of Science Humanities and Arts - JOSHA*, 2019.

[4] N. K. Weng, "The development of mobile application for parking lot," *Universiti Tunku Abdul Rahman*,

[5] A. R. S. S. A. Arshad, "Design and development of smart parking," 2017.

[6] . A. Veeraiah Padmaja, "Smart parking system with qr code," *International Journal of Engineering and Advanced Technology (IJEAT)*, 2019.

[7] MAPBOX. "Annotations maps sdk for android." (1998), [Online]. Available: `https://docs.mapbox.com/android/maps/guides/annotations/annotations/`.

APPENDIX A: SOURCE CODE

The ParkPal application is a smart parking solution designed to optimize parking operations and enhance user experience. The source code and detailed information are available on GitHub at https://github.com/zeynepdukk/ParkPal1. The GitHub repository includes the complete codebase, instructions, and comprehensive documentation, making it a valuable resource for developers and researchers interested in smart parking solutions.