AUTOMATED PARKING AND PAYMENT SYSTEM USING LICENSE PLATE AND VEHICLE ATTRIBUTE RECOGNITION WITH MULTIMODAL AI MODELS

YONG TING WEI

UNIVERSITI TUNKU ABDUL RAHMAN

AUTOMATED PARKING AND PAYMENT SYSTEM USING LICENSE PLATE AND VEHICLE ATTRIBUTE RECOGNITION WITH MULTIMODAL AI MODELS

YONG TING WEI

A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Software Engineering (Honours)

Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

September 2025

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Name : Yong Ting Wei

ID No. : 2200787

Date : 16/10/2025

COPYRIGHT STATEMENT

This final year project report is submitted in partial fulfilment of the requirements for the degree of Bachelor of Software Engineering (Honours) at Universiti Tunku Abdul Rahman (UTAR). This final year project report represents the work of the author, except where due acknowledgement has been made in the text. No part of this final year project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor, Lee Ming Jie, for his invaluable advice, continuous guidance, and patience throughout the development of this final year project. His encouragement and constructive feedback have been instrumental in ensuring the successful completion of this work.

I would also like to extend my appreciation to my moderator, Kelwin Tan Seen Tiong, for providing constructive comments and support during the evaluation stages.

In addition, I wish to thank the faculty and departmental members from Lee Kong Chian Faculty of Engineering and Science and the Department of Computing for creating a pleasant and supportive environment that enabled me to carry out my research effectively.

Finally, I would like to thank my family and friends for their encouragement, patience, and support during this journey.

ABSTRACT

As of October 2023, Malaysia recorded over 36.3 million registered vehicles, highlighting the need for more efficient and intelligent parking solutions. Traditional parking systems, which rely on physical tickets, RFID tags, and e-wallets, often lead to congestion, delays, and security vulnerabilities. This project proposes an AI-powered parking payment system that integrates License Plate Recognition (LPR) with Vehicle Attribute Recognition using the Gemini 2.5 Flash multimodal large language model (LLM), complemented by a mobile application designed for drivers, operators, and administrators. The recognition module, developed in Python, was tested using a ground truth dataset of 20 real vehicle images from Roboflow in a simulated environment. Each image was passed directly to the Gemini model to extract license plates and vehicle attributes such as make, model, and color. Recognition was incorporated into two system points: (1) a mobile app feature allowing users to auto-fill vehicle details via photo uploads, and (2) simulated parking facility entry and exit points where vehicle identity was verified against a backend database for automated payment processing. The mobile app was written with Laravel, React Native, and PostgreSQL, and it offers role-based features including vehicle registration, payment tracking, and operational oversight. Testing showed an 85% accuracy for full multi-attribute recognition, with individual accuracies of 95% for license plates, 100% for color and make, and 90% for model detection. Average recognition processing time was 2.495 seconds per image upload. While entry and exit recognition were simulated, the system successfully demonstrated automated vehicle verification and payment workflows. The mobile application facilitated seamless user interactions and system management. Limitations include reliance on free-tier AI services, absence of real-time hardware integration, and limited analytics capabilities. This project illustrates the feasibility of leveraging multimodal LLMs and mobile platforms to create ticketless, contactless, and fraud-resistant parking solutions, contributing to Malaysia’s digital transformation and smart city initiatives.

Keywords: artificial intelligence; license plate recognition; vehicle attribute recognition; smart parking; fraud prevention; multimodal LLMs; smart city

Subject Area: QA75.5–76.95 Electronic computers. Computer science

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

AI	Artificial Intelligence
AVR	Automatic Vehicle Recognition
CNN	Convolutional Neural Network
CRNN	Convolutional Recurrent Neural Network
LLM	Large Language Model
LPR	License Plate Recognition
NMS	Non-Maximum Suppression
OCR	Optical Character Recognition
RDBMS	Relational Database Management System
RFID	Radio Frequency Identification
ROI	Region of Interest
RPN	Region Proposal Network
SAM	Segment Anything Model
SDG	Sustainable Development Goals
SSD	Single Shot Multibox Detector
TNG	Touch ‘n Go
ViT	Vision Transformer
WBS	Work Breakdown Structure
YOLO	You Only Look Once

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 General Introduction

As of October 2023, the number of registered vehicles in Malaysia has exceeded the country's population, with over 36.3 million vehicles recorded (Nuradzimmah Daim, 2023). With rapid urbanization and increasing vehicle ownership, the need for efficient smart parking solutions has become increasingly critical. However, many existing systems still rely on outdated methods, such as physical parking tickets, Touch 'n Go (TNG) cards, eWallets, debit and credit cards, Radio Frequency Identification (RFID)-based payments, and even manual kiosks.

A major issue with the current parking systems in Malaysia is congestion at parking entry and exit points. These delays are often caused by the need to stop the vehicle completely to insert a ticket or pay with cash, faulty card readers, and slow RFID scans. These problems not only inconvenience drivers but also disrupt traffic flow within parking facilities. Additionally, many users forget to reload their TNG cards or eWallets, resulting in failed transactions and further delays.

While LPR technology is gradually being adopted in some modern parking systems to offer a more convenient and quicker experience, it introduces a new challenge, which is license plate fraud. Criminals may clone or tamper with license plates to avoid payment or gain unauthorized access to parking areas, which weakens the reliability and security of LPR-based systems.

This project aims to address these challenges by developing an Artificial Intelligence (AI)-powered parking payment system that utilizes fixed cameras at parking entrances and exits to automatically recognize license plates along with key vehicle attributes, such as make, model, and color, and process payments without the need for manual intervention. By combining license plate recognition with vehicle attribute verification, the system enhances security and helps prevent fraud cases, which are potential vulnerabilities in traditional LPR-based systems.

By eliminating the need for physical cards, parking tickets, eWallets, RFID stickers, or manual kiosks, the proposed system supports Malaysia's digitalization and smart mobility initiatives. It offers a fully automated, ticketless, and contactless parking payment experience, significantly improving operational efficiency, strengthening security, and improving overall user convenience in modern parking management.

1.2 Importance of the Study

The development of an automated parking payment system using LPR and Vehicle Attribute Recognition offers practical and social benefits by addressing major challenges in urban parking. By integrating AI technologies like multimodal Large Language Models (LLMs), this system enhances the efficiency, security, and convenience of parking systems. This research can address the issues of long wait times and inefficiencies in traditional parking methods. Moreover, it aims to reduce human error, minimize fraud, and streamline the payment process, contributing to improved user satisfaction.

The system directly contributes to several United Nations Sustainable Development Goals (SDGs), creating a positive impact on the world. For SDG 9 (Industry, Innovation and Infrastructure), it promotes resilient infrastructure through cutting-edge AI applications while fostering innovation in smart city technologies. In terms of SDG 11 (Sustainable Cities and Communities), the system's ability to reduce congestion and improve parking efficiency supports the creation of more inclusive, safe, and sustainable urban spaces. The solution also advances SDG 16 (Peace, Justice and Strong Institutions) by implementing transparent, fraud-resistant systems that enhance accountability in public services and strengthen institutional trust. These benefits position automated parking management as a key component in building smarter, more sustainable cities for the future.

1.3 Problem Statement

The adoption of cashless payment solutions has significantly improved parking management systems in Malaysia. However, various challenges remain, affecting efficiency, security, and user convenience.

1.3.1 Performance Limitation of Current LPR Solutions

Although LPR technology is being implemented in some parking systems, many of the LPR solutions struggle with accuracy and adaptability in real-world conditions. Environmental factors, such as low-light conditions, motion blur, and obstructed or damaged plates, severely affect LPR performance. For example, LPR systems often struggle during nighttime or in poorly lit environments where image quality is compromised. Additionally, when vehicles are moving at high speeds or plates are dirty or partially blocked, motion blur and distortion further reduce the accuracy of license plate detection. Other than environmental factors, the wide variety of license plate designs, differing in size, color, font style, and layout across regions, can complicate the detection and recognition process. LPR systems often struggle when exposed to designs not present in their training datasets.

1.3.2 Security Risks and Unauthorized Transactions Using LPR

Security is a critical concern in automated parking systems, particularly when it comes to vehicle theft, unauthorized use, and fraud. Traditional LPR systems rely solely on plate numbers for identification, without cross-checking additional vehicle attributes such as make, model, and color. This lack of multi-attribute verification allows fraudulent vehicles with cloned plates to bypass detection. For example, a person could clone a legitimate vehicle's license plate by copying the plate number and placing it on a different vehicle. This cloned vehicle can then enter the parking facilities, as the LPR system recognizes the cloned plate as valid, leading to the transaction being processed under the original vehicle's plate number and deducting the payment from the original vehicle owner's account.

1.3.3 Inefficiency in Existing Parking Systems

Malaysia's parking systems often rely on outdated technologies such as physical parking tickets and cards, eWallets, RFID tags, and manual kiosks. These methods require vehicles to stop for validation or payment, leading to congestion at entry and exit points. Technical issues like malfunctioning card readers, slow RFID scans, or insufficient eWallet balance further delay the

process, frustrating users and reducing system efficiency. The need to carry RFID tags, cards, or cash also contributes to user dissatisfaction.

1.4 Aim and Objectives

1.4.1 Project Aim

This project aims to develop an automated parking payment system that integrates multimodal LLMs for LPR and Vehicle Attribute Recognition. By combining LPR with additional vehicle attributes such as make, model, and color through multimodal AI models, the system will enhance transaction security and accuracy. Additionally, a parking management app will be developed to allow users to manage their parking sessions, view transaction history, and make payments seamlessly. This approach eliminates the need for manual payment methods, reduces congestion, and strengthens the overall security of Malaysia's parking infrastructure.

1.4.2 Project Objectives

1. To examine license plate and vehicle attribute approaches and review similar applications.
2. To develop an automated parking payment system that integrates multimodal LLMs for license plate and vehicle attribute recognition.
3. To develop a parking management application.

1.5 Project Solution

The proposed solution aims to address the limitations of traditional parking payment systems by integrating advanced multimodal LLMs for license plate and vehicle attribute recognition. This solution will not only improve the accuracy and efficiency of vehicle identification but also enhance security by verifying multiple vehicle attributes beyond the license plate, such as make, model, and color.

1.5.1 Automated Parking Payment System with LPR and Vehicle Attribute Recognition

The system will use multimodal LLMs trained on both LPR and vehicle attribute recognition. The LLMs will process real-time data from cameras at entry and

exit points of parking facilities to automatically identify vehicles, verify their license plates, and cross-check additional attributes, such as make, model, and color. This will allow for seamless automated transactions without the need for manual intervention, and also reduce the risk of fraud and increase the speed of the payment process.

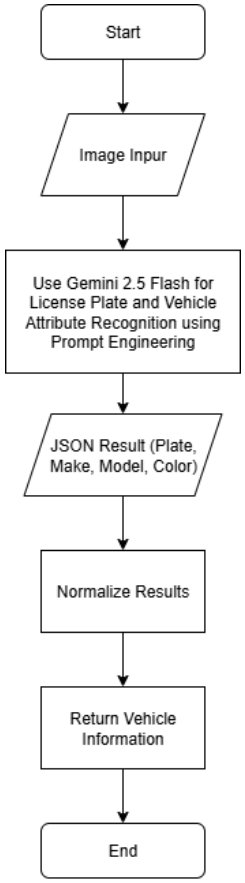


Figure 1.1: High-Level Flow Diagram of LPR and Vehicle Attribute Model

1.5.2 Parking Management Application

A dedicated mobile app will be developed for users to manage their vehicles, register their license plates, and track parking transactions. The app will allow users to link vehicles to their accounts, view parking history, and manage payment preferences. An emergency stop feature will also be included, enabling users to disable automated payments immediately if their vehicle is stolen or used without authorization. In addition, parking operators can also use the application as it provides tools to manage parking lot details, set or adjust

parking rates, and perform analytical reviews of transaction data. This helps both users and the parking operators manage their tasks more easily.

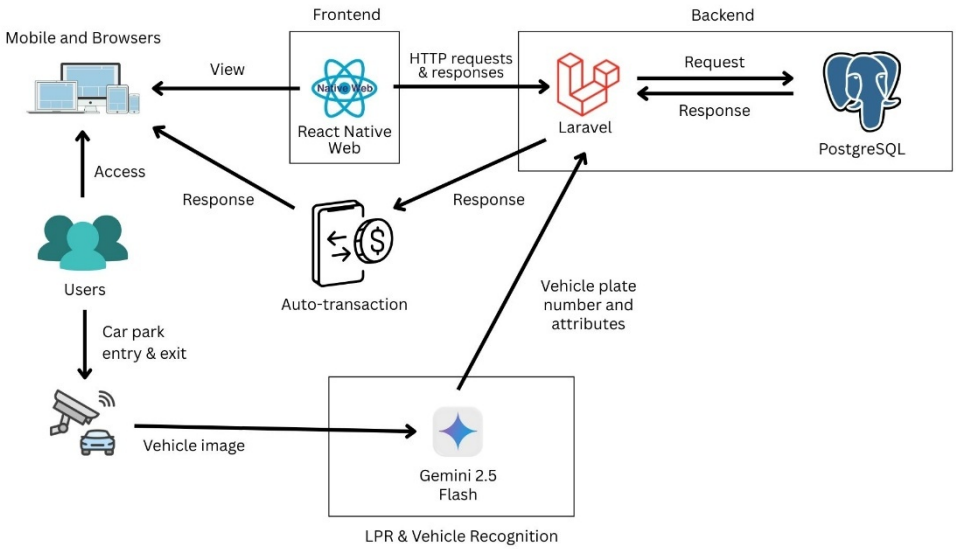


Figure 1.2 Parking Management System Architecture Flowchart

1.6 Project Approach

1.6.1 Research Approach

The research explores various approaches, starting with traditional methods that use simple image processing for plate detection. It then examines machine learning techniques that improve automation and accuracy. The study further investigates deep learning techniques, including deep neural networks for vehicle detection, image segmentation, and Optical Character Recognition (OCR) for reading plates and identifying vehicle attributes like make, model, and color. Finally, the research looks into Multimodal LLMs, which combine image and text processing to enhance recognition accuracy and fraud detection, making them particularly effective in dynamic environments like toll booths and parking areas.

A critical part of the research also involves analyzing existing parking applications to identify strengths and limitations. By reviewing these applications, the study will gain insights into the operational challenges and user experience, helping to shape the design of the proposed system. Furthermore, research into development tools and frameworks will help select the best technologies for the project's implementation.

Additionally, a questionnaire survey will be conducted as part of quantitative research. The survey will gather data from users of current parking systems to understand their preferences, the challenges they face, and their perceptions of automated parking payment solutions. This data will offer valuable insights into the real-world applicability, effectiveness, and user acceptance of the proposed system, ensuring that it meets the needs and expectations of the target users.

1.6.2 Development Approach

The development approach for this project will be based on the Scrum methodology. Although the team consists of a single developer, the principles of Scrum will still be applied to structure the development process effectively. The project will be broken down into smaller, manageable tasks that can be completed in short, time-boxed intervals known as sprints, typically lasting one to two weeks. Each sprint will start with a planning phase, where objectives and tasks for the upcoming period are defined. Daily stand-up meetings, even if brief, will be used to assess progress, address challenges, and ensure that the project stays on track. At the end of each sprint, a review will be conducted to assess the deliverables and make adjustments based on feedback or new insights. The project backlog will be maintained to track tasks and features, which will be prioritized based on the system's requirements and objectives. This approach will allow for continuous improvement of the system, incorporating feedback and ensuring that all aspects of the application, from vehicle recognition to the user interface, are developed efficiently and meet the necessary specifications. By applying Scrum principles, the development process will remain flexible, enabling the project to adapt to any challenges or changes in scope while ensuring steady progress and the delivery of a functional system.

1.7 Scope and Limitations of the Study

The scope and limitations of this study outline the key components and boundaries of the project. This section will cover the specific modules addressed, the target users, and the development tools, languages, and frameworks employed in the project. It will also discuss the project limitations, including any constraints in functionality or technology, as well as aspects that are

intentionally out of scope for this research. This provides a clear understanding of the project's focus while acknowledging areas that fall outside its current scope.

1.7.1 System Modules

The system consists of several key modules, each tailored to specific user roles, ensuring that all users can perform their tasks efficiently.

1.7.1.1 Driver Module

This module is designed for vehicle owners and general users. It includes functionalities such as vehicle registration, license plate management, viewing parking availability, transaction history, and managing payment methods. The module supports integration with the automated recognition system to enable seamless entry and exit based on license plate and vehicle attribute detection.

1.7.1.2 Parking Operator Module

This module allows authorized parking operators to register and manage parking facilities. Operators can input and update parking lot details, edit pricing rates, and monitor the operational performance of their lots through analytical tools. The system ensures that any updates to lot information or pricing are subject to admin approval for validation and standardization.

1.7.1.3 Admin Module

The admin module provides centralized control of the system. It includes features for managing user accounts for both drivers and operators, approving or rejecting parking operator registrations, and verifying changes to parking lot information or rates. This module also includes analytics and reporting tools for overall system monitoring.

1.7.1.4 Multimodal LLM-based License Plate and Vehicle Attribute Recognition Module

This module uses multimodal large language models to perform automated recognition of license plates and vehicle attributes such as make, model, and color. It enhances security and reduces fraud by cross-verifying plate numbers

with vehicle attributes. The system leverages image and text understanding capabilities of LLMs to ensure high accuracy in diverse real-world environments, such as low light or obstructed views.

1.7.2 Target Users

The system is designed to serve three main types of users.

(i) Drivers. These are the primary end-users who use the application to register their vehicles, manage payment preferences, view transaction history, and receive support for any parking-related issues. The system aims to simplify their parking experience through automated license plate recognition and secure payment handling.

(ii) Parking Operators. These users represent parking lot management entities. They will use the system to register and manage their parking lots, set parking rates, and view analytics related to usage and revenue. Operators can also raise support tickets and handle operational queries through the system.

(iii) Administrators. The admin role is responsible for overseeing the platform. This includes managing user accounts, approving or rejecting parking operator registrations and rate changes, monitoring system analytics, and resolving submitted support tickets. Administrators ensure that the platform remains secure, fair, and operationally efficient.

1.7.3 Out-of-Scope

The following aspects are beyond the scope of this project.

(i) Face Recognition or Biometric Authentication. The project will focus solely on LPR and Vehicle Attribute Recognition, without incorporating biometric authentication methods such as facial recognition.

(ii) Hardware Implementation. The project does not involve the development or integration of physical devices such as cameras, sensors, or gate control systems. It will assume that the image inputs are already captured and available for processing.

1.7.4 Project Limitations

While this project aims to enhance parking automation using AI-based LPR and Vehicle Attribute Recognition, it is subject to the following limitations.

(i) Simulation Environment. The system will be developed and tested in a simulated environment for academic purposes. Real-world deployment factors such as network infrastructure, environmental unpredictability, and large-scale user traffic are not fully considered.

(ii) Hardware Constraints. The system will not include physical installation of cameras or sensors. Instead, pre-recorded images or uploaded data will be used for testing the LPR and vehicle attribute modules.

1.7.5 Development Tools, Languages, and Frameworks

This project will utilize a combination of tools and frameworks across both frontend and backend development. Visual Studio Code will serve as the primary code editor. The frontend will be developed using React Native Web, allowing for cross-platform access via both mobile and desktop browsers. The backend will be built using Laravel, a PHP framework, while PostgreSQL will be used for the database system. For AI-based recognition, Gemini 2.5 Flash, a multimodal large language model, is employed to identify license plates and vehicle attributes by processing both visual and textual inputs.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter reviews key technologies and methods related to license plate and vehicle attribute recognition, with a focus on their role in automated parking systems. It covers traditional, machine learning, deep learning, and emerging large language model-based approaches. In addition, it examines existing parking payment applications, relevant software development practices, and development frameworks to provide a foundation for system design and implementation.

2.2 Why Improve Current License Plate And Vehicle Attribute Recognition?

License plate recognition and vehicle attribute recognition are important for systems like toll payments, parking management, traffic monitoring, and law enforcement. However, many current systems still face problems, especially in real-world conditions such as poor lighting, unclear images, occlusion, or different plate formats. Traditional methods often make mistakes and may not recognize important vehicle details like the make, model, or color. With the growing number of vehicles worldwide, which is estimated to surpass 2 billion by 2040, the demand for more reliable and scalable recognition systems is increasing (Manzoor, Morgan and Bais, 2019). Therefore, there is a clear need to improve existing LPR and vehicle recognition technologies to make them more accurate, reliable, and suitable for real-time use. This section explains the problems with current systems, compares older methods with newer AI-based approaches, and shows why better solutions are needed.

2.3 Traditional Approaches to License Plate and Vehicle Attribute Recognition

2.3.1 Manual Observation

One of the earliest ways to identify vehicles was by having officers or staff visually check vehicles and record them down, or match the license plate numbers with existing vehicle registration databases to find details of the vehicle, such as the car's make, model, and owner. Some places still use this method, especially where automated systems are not available. However, manual checks are often unreliable because humans make mistakes, get tired, or struggle with poor lighting, bad weather, or fast-moving traffic. It’s also slow and not practical for busy areas or smart city systems, where quick and accurate identification is needed.

2.3.2 Traditional License Plate Recognition

Traditional LPR systems are based on conventional computer vision techniques. They follow a step-by-step process that includes capturing an image, extracting the feature or region of interest (ROI), preprocessing by grayscale conversion and noise reduction, license plate localization, character segmentation, and OCR using template matching or rule-based logic (Nadira Muda *et al.*, 2007; Abdullah *et al.*, 2021).

A major drawback is that these systems rely heavily on hand-crafted features and are generally optimized for specific types of license plates. For example, template matching works well when characters on the plate are standardized in font and spacing, but tends to fail when dealing with non-standard formats, stylized fonts, or variations in character alignment (Montazzolli and Jung, 2018). In practice, these systems struggle with several real-world challenges, including occlusions, motion blur, low-resolution images, and plate distortions caused by camera angles or vehicle speed (Zherzdev and Gruzdev, 2018).

Furthermore, traditional LPR systems cannot identify vehicle attributes such as make, model, or color, which limits their use in more complex surveillance and intelligent transportation applications. Their dependency on a single modality, which is usually a 2D grayscale image, also makes them less

robust in dynamically changing environments, especially those involving bad lighting or weather conditions (Abdullah *et al.*, 2021).

Due to these limitations, research has gradually shifted toward AI-based approaches, which offer greater flexibility, scalability, and robustness in diverse real-world conditions.

2.4 Machine Learning (ML) Approaches

Machine learning techniques were one of the first approaches to be applied to Automatic Vehicle Recognition (AVR) systems. These methods focus on handcrafted feature extraction, where specific visual patterns are manually engineered and then used for classification tasks. ML models have been widely applied in vehicle make and model recognition (VMMR), particularly in constrained or controlled environments.

The main handcrafted feature descriptors used in ML-based systems are Scale-Invariant Feature Transform (SIFT), Speeded-Up Robust Features (SURF), and Histogram of Oriented Gradients (HOG). SIFT identifies unique keypoints by finding local extremes in scale-space and describes them using gradient histograms. It works well across different scales, rotations, and moderate viewpoint transformations (Lowe, 2004). SURF, a faster alternative to SIFT, locates key points in an image by analyzing distinctive features such as edges, corners, and blobs. It uses integral images and an approximate Hessian matrix to quickly detect keypoints while remaining stable against scale and rotation (Bay, Tuytelaars and Van Gool, 2006). HOG focuses on analyzing the gradients and orientation of pixel intensity changes in an image, making it effective for identifying object shapes such as vehicle outlines (Dalal and Triggs, 2005). These features are typically extracted from segmented vehicle regions and then passed to machine learning algorithms used for classification, such as Support Vector Machines (SVM), K-Nearest Neighbors (KNN), random forest, or multi-layer perceptron (MLP) for vehicle type or model classification.

In the context of VMMR, handcrafted feature-based ML systems have shown reasonable performance in scenarios with good lighting, minimal occlusion, and consistent viewpoints. Their simplicity, low computational

overhead, and interpretability make them suitable for embedded or resource-limited environments (Aly, 2008).

However, ML-based methods have several limitations. These methods require significant effort in feature engineering and may not capture complex features effectively. They also lack the adaptability to complex, real-world conditions such as varying lighting, occlusions, and diverse vehicle models (Sochor, Špaňhel and Herout, 2017). Moreover, separating feature extraction and classification steps prevents the system from being optimized, which often leads to weaker recognition performance in challenging conditions. Due to these limitations, ML approaches are increasingly being replaced or supplemented by deep learning models that automatically learn hierarchical feature representations from raw data, offering superior generalization and accuracy.

2.5 Deep Learning (DL) Approaches

Deep learning techniques have significantly advanced the performance of AVR, particularly in LPR and Vehicle Attribute Recognition (VAR). These methods eliminate the need for manual feature engineering by allowing models to learn directly from image data.

2.5.1 Deep Learning in Vehicle Detection

Object detection models for autonomous vehicles are categorized into one-stage and two-stage detectors, each with distinct advantages and trade-offs in terms of speed and precision. This section reviews well-known models in both categories, focusing on their architectures, strengths, and applications in vehicle detection.

2.5.1.1 Two-Stage Detectors

Two-stage detectors are object detection algorithms that use a two-step process for detecting objects in images, offering high accuracy but requiring more computational resources than one-stage detectors. In the first stage, the algorithm generates potential ROIs in the image that might contain objects. This is done using techniques like Selective Search or Region Proposal Networks (RPN) (Gayen *et al.*, 2024). These regions, called region proposals, narrow

down the areas in the image that are most likely to contain an object, reducing the search space for the next step. The second stage involves classifying each region proposal to determine if it contains an object and then refining the bounding box around the object (Gayen *et al.*, 2024). The model predicts adjustments to the coordinates of the proposed bounding box to improve its accuracy. Two-stage detectors offer higher accuracy, especially for small object detection, as they focus computational resources on promising regions. However, they tend to be slower and more resource-intensive than one-stage detectors.

2.5.1.1.1 Faster R-CNN

Faster R-CNN is a widely adopted and powerful two-stage object detection framework that improves both speed and accuracy over earlier region-based Convolutional Neural Network (CNN) models. It is particularly known for its ability to accurately localize and classify objects in an image by combining a RPN with a Fast R-CNN detection head in a single and unified architecture (Figure 2.1).

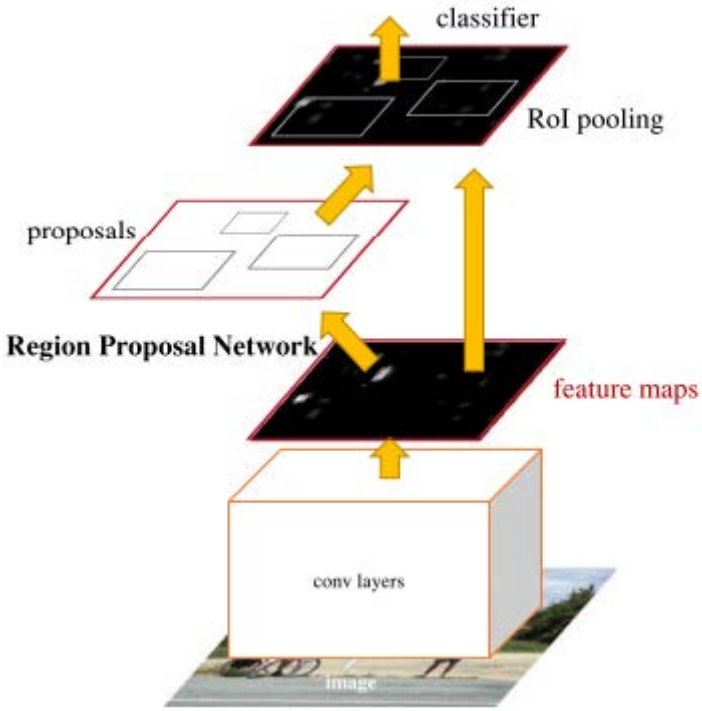


Figure 2.1: Faster R-CNN network structure (Ren *et al.*, 2017)

The architecture of Faster R-CNN is divided into two main stages. First, an input image is passed through a backbone convolutional neural network, such as VGG16 or ResNet, to extract a feature map. This feature map is then fed into the RPN, which slides over the map to generate multiple anchor boxes of varying scales and aspect ratios. For each anchor, the RPN predicts an objectness score, which indicates the presence of an object, and performs bounding box regression to refine the coordinates. To reduce redundancy, Non-Maximum Suppression (NMS) is applied to eliminate overlapping proposals, resulting in a set of high-confidence candidate regions (Ren *et al.*, 2017).

In the second stage, these candidate regions are fed into the Fast R-CNN module. Through a ROI pooling layer, each region is converted into a fixed-size feature vector regardless of its original shape. These vectors are then passed through fully connected layers that perform two tasks, which are classifying the object and further refining its bounding box. Faster R-CNN enables end-to-end training of both the RPN and the Fast R-CNN detector, allowing for joint optimization through backpropagation. The loss function combines classification loss from the RPN's objectness scores and bounding box regression loss for both the RPN and the Fast R-CNN detection network, ensuring simultaneous learning of region proposals and object detection for improved accuracy (Ren *et al.*, 2017).

The final output of Faster R-CNN includes predicted class labels, bounding box coordinates, and confidence scores for each detected object. Due to its efficient design, which reuses features between the RPN and Fast R-CNN, the model significantly outperforms earlier methods that relied on slow proposal algorithms like Selective Search. It typically achieves reasonable speeds of 5 to 17 frames per second while also maintaining high accuracy, depending on the complexity of the network and the hardware used (Ren *et al.*, 2017). Its robust performance has made it a benchmark for object detection tasks, including applications like vehicle detection, face recognition, and surveillance.

2.5.1.2 One-Stage Detectors

One-stage detectors are object detection algorithms that simplify the object detection process by eliminating region proposal generation, which is common in two-stage detectors (Gayen *et al.*, 2024). One-stage detectors directly predict the location of objects and their corresponding class labels in a single step. This is achieved using a single feed-forward, fully convolutional network that outputs both bounding boxes and object classifications for all potential objects in the image (Carranza-García *et al.*, 2021).

One-stage detectors are known for their speed and efficiency. These models process the entire image at once, predicting object classes and bounding box locations for every region of the image simultaneously. This approach makes them much faster than two-stage detectors, as they do not require the additional step of generating region proposals. However, one-stage detectors traditionally faced challenges due to the imbalance between objects of interest and background in images, which could negatively impact detection accuracy (Carranza-García *et al.*, 2021).

2.5.1.2.1 YOLO

You Only Look Once (YOLO) is a deep learning algorithm designed for real-time object detection, known for its unique architecture and efficient processing. Unlike traditional object detection methods that involve separate steps for region proposal and classification, YOLO uses a single CNN that processes the entire image in one pass, simplifying the pipeline.

YOLO's architecture (Figure 2.2) is based on the GoogLeNet image classification model. The network contains 24 convolutional layers and two fully connected layers, making it almost 5 times larger than the ZF-5 used in SPP-Net and Faster R-CNN (Wu, 2018). The process in YOLO starts with image input, where the original image is fed into the system. Following this, a preprocessing step is carried out, which involves resizing the image to a suitable dimension and dividing it into an S × S grid, which prepares the image for analysis by the neural network.

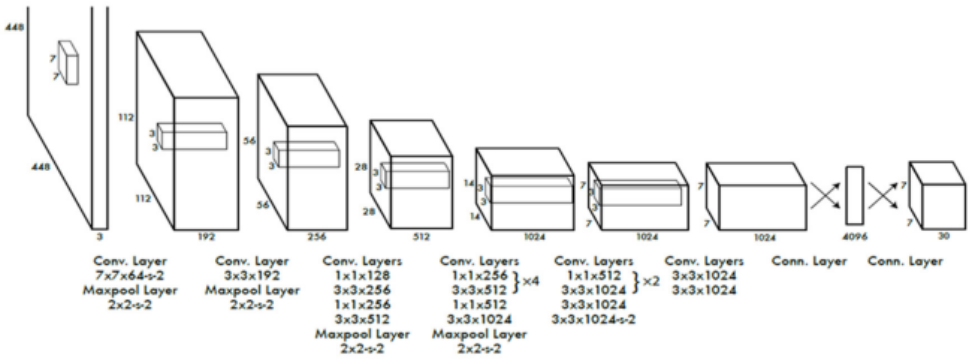


Figure 2.2: YOLO architecture (Wu, 2018)

Next, the image passes through a CNN that includes several layers such as convolutional layers, pooling layers, fully connected layers, and dropout layers. Each grid cell is then responsible for predicting bounding boxes and class probabilities. A bounding box includes five parameters: the x and y coordinates representing the center of the box, the width (w) and height (h) relative to the full image, and a confidence score (Kang *et al.*, 2025).

For every predicted box, a confidence score is calculated to estimate how likely it is that the box contains an object and how accurate the predicted location is. This confidence score is based on how much the predicted box overlaps with the actual object (measured using Intersection over Union, or IoU). In addition, each grid cell also predicts the probability that the object belongs to a certain class. By combining the confidence score and the class probability, the model determines the overall likelihood that a particular object is correctly detected and classified.

After processing by the convolutional neural network, the model outputs a set of predictions, which are then refined in a post-processing step. This includes applying NMS to eliminate overlapping bounding boxes and retain only the most confident detections. The remaining, most confident predictions are then combined and mapped back onto the image to produce the final output, which shows the detected objects along with their corresponding bounding boxes and class labels (Kang *et al.*, 2025).

2.5.1.2.2 Single Shot Multibox Detector (SSD)

SSD, also known as Single Shot Multibox Detector, is a widely used deep learning framework for real-time object detection that strikes a strong balance between speed and accuracy. As a one-stage detector, SSD performs object localization and classification in a single forward pass through the network, making it significantly faster than two-stage detectors like Faster R-CNN.

The SSD architecture (Figure 2.3) consists of three primary components. First, it uses a base network, often adapted from established CNN models such as VGG16, to extract features from input images. Second, it includes a feature extractor that utilizes multiple layers of feature maps at different resolutions, allowing the model to detect objects of varying sizes. Finally, the detection layer comprises convolutional layers that predict both bounding box coordinates and class scores (Cao *et al.*, 2020).

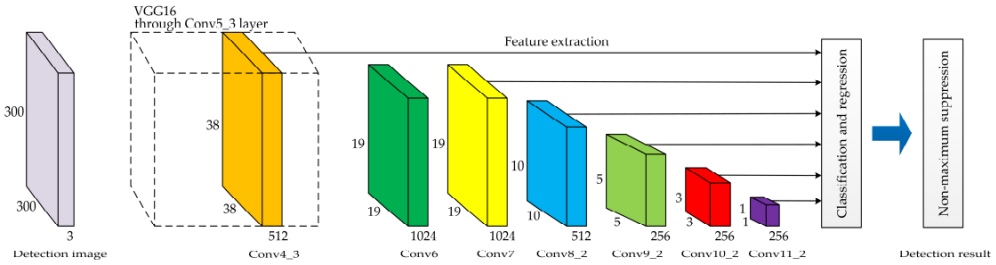


Figure 2.3: SSD architecture (Cao *et al.*, 2020)

The input image is first resized to a standard resolution, such as 300×300 or 512×512 pixels, to maintain consistency. This image is then passed through the base network, which generates feature maps at various levels. These maps represent different levels of abstraction, from low-level textures in earlier layers to high-level object representations in deeper layers. This makes it possible to detect objects of various sizes across different layers (Cao *et al.*, 2020).

SSD relies on these multi-scale feature maps to handle objects of different sizes effectively. Lower layers, such as Conv4_3, detect smaller objects using finer detail, while higher layers, such as Conv7, focus on larger, more abstract features. At each location in the feature maps, SSD defines several default anchor boxes of various shapes and sizes. For each anchor, the model

predicts two things which are the class confidence scores for every object class and bounding box offsets that fine-tune the predicted box location (Cao *et al.*, 2020).

During training, SSD uses a combined loss function, which is a localization loss (Smooth L1) for bounding box accuracy and a classification loss (Softmax) to match predicted classes with the ground truth. During inference, NMS is applied to remove redundant boxes, keeping only the most confident predictions (Cao *et al.*, 2020).

2.5.2 Instance Segmentation in Vehicle and License Plate Recognition

Instance segmentation is a powerful computer vision technique that combines object detection and semantic segmentation to identify and outline objects at the pixel level with unique segmentation masks (Kattenborn *et al.*, 2021). Unlike semantic segmentation, which treats all objects of a class as a single group, instance segmentation can distinguish and segment each object individually, even in cases of overlapping or occluded objects. This makes it particularly useful in complex situations where multiple vehicles or license plates appear close. Instance segmentation algorithms operate in two main stages. First, they detect objects within an image using an object detection model, which typically involves drawing bounding boxes around objects such as vehicles or license plates. Next, a segmentation model produces pixel-wise masks that accurately outline the shape and boundaries of each object. This approach enables precise separation of objects from both the background and nearby instances of the same class.

2.5.2.1 YOLOv8 Seg

YOLOv8 Seg is an advanced instance segmentation model developed by Ultralytics as part of the YOLO family. It extends the capabilities of object detection by incorporating pixel-level segmentation for each detected object. This means that in addition to drawing bounding boxes around objects such as vehicles or license plates, YOLOv8 Seg also generates precise segmentation masks that outline the shape of each object instance (Ultralytics, 2023a). The model operates in real-time and is optimized for applications that require both

speed and accuracy, such as traffic monitoring, license plate recognition, and autonomous driving systems. YOLOv8 Seg supports training and inference on custom datasets, allowing developers to fine-tune it for specific classes relevant to their project.

2.5.2.2 Segment Anything (SAM)

The Segment Anything Model (SAM), developed by Meta AI, is a promptable segmentation model designed to segment any object in an image, even without class labels or prior training on specific objects. SAM can generate segmentation masks in response to user-provided prompts, such as clicks, bounding boxes, or even text descriptions. It uses a powerful image encoder and a flexible prompt encoder to handle a wide range of inputs and produce high-quality masks. One of SAM's most notable features is its ability to perform zero-shot segmentation, meaning it can generalize to new, unseen objects at inference time (Ultralytics, 2023b). This makes it highly versatile for research, labeling tools, and applications that require segmentation without being limited to predefined object classes. However, SAM is not designed for real-time tasks and does not include object detection capabilities.

2.5.2.3 Comparison of Segmentation Model

Although both YOLOv8 Seg and SAM are used for segmentation tasks, they are built for different objectives and perform differently in practical scenarios. YOLOv8 Seg is trained to detect and segment predefined object classes, such as vehicles or license plates, in a single, end-to-end pipeline. This makes it especially suitable for high-speed applications where both accuracy and performance are critical, such as smart parking systems. In contrast, SAM is a prompt-based segmentation model designed for general-purpose segmentation across a wide range of domains as it can segment objects it has never seen during training. However, it is not optimized for real-time processing and lacks built-in object detection, meaning it must be combined with other models to automatically identify and segment specific objects.

In summary, YOLOv8 Seg excels in scenarios where speed, automation, and predefined object classes are essential, while SAM is more

suited for exploratory or manual segmentation tasks where flexibility and generality are required. For this project, where the goal is to automatically identify and segment vehicles and license plates in real-time, YOLOv8 Seg is a more practical and efficient choice.

Table 2.1: Table of comparison of segment models

Feature	YOLOv8 Seg	Segment Anything (SAM)
Task	Instance segmentation and detection	Promptable segmentation only
Real-time performance	Yes	No as it is slower and more complex
Predefined classes	Yes	No
Requires prompts	No	Yes
Use Case	Real-time tasks with known classes	Interactive or general segmentation

2.5.3 Deep Learning for OCR in LPR

Optical Character Recognition plays a crucial role in LPR by converting detected license plate images into readable alphanumeric text. While traditional OCR techniques were widely used in earlier systems, modern LPR applications are increasingly adopting deep learning-based OCR methods due to their superior performance in handling distorted, low-resolution, or variably styled license plates. This section explores three widely used OCR approaches, which are Tesseract OCR, CRNN, and Transformer-based models.

2.5.3.1 Tesseract OCR

Tesseract is a popular open-source OCR engine developed by HP and maintained by Google. It primarily uses a traditional rule-based and pattern-matching approach, making it lightweight and easy to implement. Recent versions, such as those from 4.0 onwards, incorporate a Long Short-Term Memory (LSTM) neural network, enhancing recognition accuracy on distorted or variably spaced text (Tesseract Documentation, n.d.). In LPR systems,

Tesseract has been used for recognizing characters on plates due to its support for multiple languages and high configurability.

However, Tesseract's performance is limited even in newer versions when dealing with real-world license plates that contain noise, blur, occlusions, or stylized fonts (Sporici, Cuşnir and Boiangiu, 2020). It also lacks deep contextual modeling, which can lead to misrecognition of visually similar characters such as '0' and 'O' or '8' and 'B'. Despite its limitations, Tesseract is still widely used in lightweight or edge applications, and it is sometimes combined with modern object detection models in hybrid LPR systems to balance efficiency and cost.

2.5.3.2 CRNN

The Convolutional Recurrent Neural Network (CRNN) is a deep learning architecture specifically designed for sequence-based tasks like OCR. It combines convolutional layers for feature extraction, recurrent layers, typically LSTM, for sequence modeling, and a transcription layer using Connectionist Temporal Classification (CTC) loss to generate text predictions without requiring pre-segmented characters (Shi, Bai and Yao, 2017).

In LPR, CRNN has been widely adopted due to its ability to handle irregular character spacing, stylized fonts, and even cursive or partially occluded text (Shi, Bai and Yao, 2017). Its end-to-end trainable structure allows it to generalize well across different plate styles and layouts. Moreover, its moderate computational footprint makes it suitable for real-time applications in resource-constrained environments. Nevertheless, CRNNs may face limitations when handling complex layouts or sequences that require deeper contextual understanding, where attention-based models like Transformers have shown superior performance.

2.5.3.3 Transformers

Transformer-based OCR models, such as TrOCR and Donut, have recently emerged as cutting-edge solutions for document and scene text recognition. These models use self-attention mechanisms to model global dependencies in

the text sequence, enabling the recognition of complex, distorted, or noisy text (Li et al., 2021).

In the context of LPR, transformers can accurately transcribe license plates even under challenging conditions, such as low lighting, occlusion, or irregular fonts, without requiring character segmentation. Their ability to process entire sequences in parallel, unlike sequential models like RNNs, also enhances speed and scalability, making them highly effective for real-time applications (Tao *et al.*, 2024). Furthermore, their multimodal capabilities also allow better integration with vision-language models, making them suitable for advanced applications in smart transportation and automated surveillance systems.

However, despite their superior accuracy and flexibility, transformer-based models often require substantial computational resources, including high memory usage and processing power (Tabani *et al.*, 2021). This presents practical deployment challenges, especially on edge devices or in resource-constrained environments where low latency and efficiency are crucial. In such scenarios, lighter alternatives like CRNNs may be more suitable due to their reduced model size and hardware demands (Khan et al., 2024).

2.5.4 Deep Learning for Vehicle Attribute Recognition

Vehicle attribute recognition, which involves make, model, and color, relies on deep learning models to handle complex and fine-grained classification tasks. These models work well in extracting and understanding detailed features from images, crucial for recognizing vehicles in diverse real-world conditions.

2.5.4.1 CNN-based Models

Convolutional Neural Networks (CNNs) have been a foundation in image recognition tasks, including vehicle attribute classification. CNNs such as ResNet, VGG, and EfficientNet are widely adopted due to their ability to learn hierarchical spatial features, which are crucial for distinguishing vehicle attributes like make, model, and color. In particular, CNNs are skilled at recognizing fine-grained details, such as the shape of a vehicle’s logo, the style

of headlights, or unique design cues in the grille and bumpers, which are the elements that are often crucial for distinguishing between similar vehicle models.

The structure of CNNs is typically composed of multiple layers. One of the layers is the convolutional layers that extract features from the input image by applying filters. Another layer is the pooling layers that down-sample the image to reduce computational complexity while retaining important features. Finally, the fully connected layers combine features extracted at different layers to make final predictions, such as vehicle make, model, and color (Xia, Feng and Zhang, 2016).

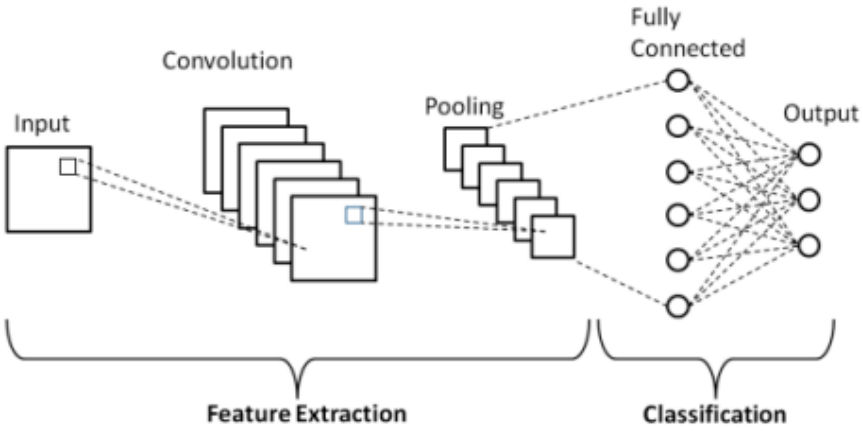


Figure 2.4: CNN architecture diagram (Phung and Rhee, 2018)

The success of CNNs in vehicle recognition can be attributed to their local receptive fields, which allow the network to focus on small, localized patterns, such as the texture of a car logo or the shape of the wheels. Moreover, the deep architectures of models like ResNet, which utilize skip connections, allow them to learn from a large number of layers without losing critical information through vanishing gradients (He *et al.*, 2016). EfficientNet, which balances depth, width, and resolution for optimal performance, has shown significant improvements in terms of accuracy and efficiency (Tan and Le, 2019).

2.5.4.2 ViT-based Models

Vision Transformers (ViTs) represent a more recent development in image recognition, shifting away from traditional convolutional operations to a transformer-based approach, which was initially designed for natural language processing (Han *et al.*, 2023). ViTs have shown great success in image classification tasks due to their ability to capture long-range dependencies across the entire image. This capability makes them effective at understanding the contextual relationships between different parts of an image (Han *et al.*, 2023). This is especially useful in tasks like recognizing subtle differences in vehicle attributes, where distinguishing between visually similar makes or models is essential.

In contrast to CNNs, which apply convolutional filters over the entire image, ViTs divide the image into non-overlapping patches and treat each patch as a "token" in a sequence. These “tokens” are passed through a series of transformer layers that use self-attention mechanisms (Figure 2.5) to assign different levels of importance to each token (Han *et al.*, 2023). This allows the model to focus on crucial regions of the image, such as specific design features, vehicle logos, or color patterns, which are essential for fine-grained vehicle make or model recognition.

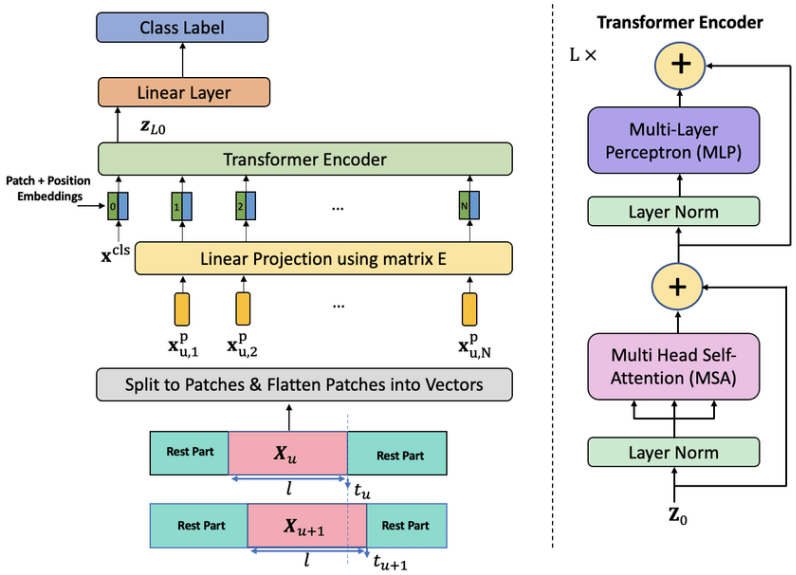


Figure 2.5: ViT Architecture and Transformer Encoder (H. Meybodi *et al.*, 2021)

Furthermore, ViTs excel at handling global contextual information, which allows them to capture intricate details in vehicle design, such as subtle differences in headlight shape, grille patterns, or logo positioning. This ability allows ViTs to outperform traditional CNN models in certain tasks, particularly when dealing with large datasets or images where understanding both local and global features is necessary to distinguish between similar vehicle types.

2.5.4.3 Multitask Learning (MTL)

Multitask Learning (MTL) involves training a single model to predict multiple related outputs simultaneously. The fundamental idea behind MTL is that by sharing information across related tasks, the model can learn more generalizable and robust representations (Figure 2.6), thereby improving overall performance (Ruder, 2017). This approach is particularly effective when the tasks are related and can benefit from shared knowledge, a concept known as inductive transfer, where learning one task helps improve learning in another (Caruana, Pratt and Thrun, 1997).

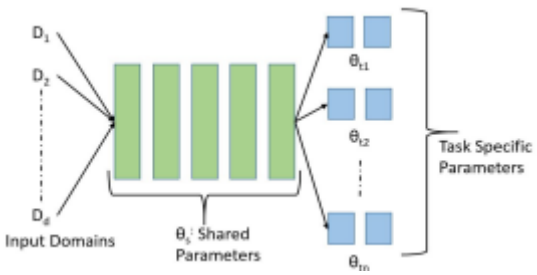


Figure 2.6: Multitask learning framework (Ranjan *et al.*, 2016)

In the context of vehicle attribute recognition, MTL is often used to predict attributes such as make, model, and color within a unified model architecture. Instead of training separate models for each attribute, MTL leverages a shared feature extractor, usually a convolutional backbone, to learn general patterns such as edges, contours, and textures. These shared features are then passed through task-specific branches that refine predictions for each output (Caruana, Pratt and Thrun, 1997). This design allows the model to exploit commonalities across tasks while preserving the flexibility to specialize in each attribute.

Another important aspect of MTL is parameter sharing, which is typically implemented using either hard or soft sharing techniques. In hard sharing, the model uses a common set of parameters, known as a shared backbone, while soft sharing allows different models to have their parameters but introduces constraints to encourage similarity (Caruana, Pratt and Thrun, 1997). Regardless of the approach, this shared learning process increases the model's efficiency and robustness across tasks.

MTL offers several advantages in vehicle recognition. It reduces model complexity by combining multiple tasks into a single network and decreases training and inference time compared to running separate models. Moreover, since many vehicle attributes are naturally correlated, for example, specific models may come in limited colors or shapes, MTL can learn to capture these interdependencies, resulting in more accurate predictions and improved generalization.

However, MTL also has challenges. A major issue is negative transfer, which occurs when learning unrelated or poorly aligned tasks together will negatively affect overall model performance (Ruder, 2017). For example, predicting color, which is a low-level feature, and identifying the vehicle make, which may depend on high-level shape semantics, could conflict if not properly balanced. This challenge can be mitigated through strategies such as weighted loss functions or dynamic task balancing are often used to ensure that one task does not dominate the learning process (Cipolla, Gal and Kendall, 2018; Kongyoung, Macdonald and Ounis, 2020).

2.5.4.4 Fine-Grained Classification

Vehicle attribute recognition is fundamentally a fine-grained classification task, as it requires distinguishing between highly similar vehicle makes, models, and colors that often exhibit only subtle visual differences. Unlike coarse classification, where the goal might simply be to recognize a car versus a truck, fine-grained classification is more precise by noticing small visual details (Yang *et al.*, 2018).

To handle this, fine-grained classification focuses on small, discriminative features, such as the shape of headlights, grille patterns, bumper

contours, logo positioning, or even color hues. These details are usually found in small parts of the image, and detecting them requires high-resolution input and advanced deep learning models, especially CNNs or ViTs.

Deep learning models are suitable for this task because they can extract hierarchical and local features, which are essential for capturing subtle intra-class differences. Moreover, training these models on large, diverse datasets improves their ability to generalize and recognize vehicle variants across different viewpoints, lighting conditions, and occlusions, which are common challenges in real-world environments such as parking lots. By using fine-grained classification, vehicle attribute recognition systems can achieve high accuracy in identifying specific vehicle types, which is crucial for applications such as intelligent parking management.

2.5.5 Summary

Deep learning approaches for LPR and vehicle attribute recognition involve several key stages: detection, segmentation, OCR, and attribute classification. For detection, YOLO and SSD are fast, while Faster R-CNN offers higher accuracy but is slower. Segmentation models like YOLOv8 Seg and Segment Anything help isolate license plates and vehicle features. For OCR, Tesseract OCR and CRNN are common, with transformer models showing better accuracy and reliability. In vehicle attribute recognition, CNNs, ViT, and techniques like multitask learning improve the recognition of vehicle details like make, model, and color.

As outlined in the proposed solution, this project will utilize multimodal LLMs to perform license plate and vehicle attribute recognition. Unlike traditional deep learning models that rely on separate components for each task, multimodal LLMs provide an integrated approach capable of handling detection, segmentation, OCR, and classification within a unified framework. While deep learning models remain efficient and widely adopted, LLMs offer greater flexibility, contextual understanding, and scalability. A hybrid approach may further enhance accuracy and reliability by combining the strengths of both.

2.6 Large Language Models (LLMs)

Large Language Models (LLMs) are a subset of deep learning models designed to understand and generate human-like text based on massive amounts of data. These models are typically built using the Transformer architecture introduced by Vaswani et al. (2017), which uses mechanisms such as self-attention to capture dependencies between tokens across varying distances, enabling context-aware representation of words. It also uses positional encoding to capture contextual relationships between words in a sequence, which is a critical factor since the Transformer lacks inherent sequential processing. LLMs are pre-trained on diverse text corpora using an autoregressive objective, where the model learns to predict the next word in a sequence by adjusting millions to billions of parameters to minimize prediction error. Once pre-trained, these models can be fine-tuned or prompted to perform a wide range of tasks, such as question answering, translation, summarization, and dialogue generation. This adaptability has driven their widespread use in natural language processing, computer vision, and multimodal AI systems.

2.6.1 Multimodal LLMs

Multimodal Large Language Models (LLMs) extend traditional language models by using multiple input types, such as text, images, audio, or video at the same time. Unlike traditional LLMs that only work with text, multimodal models combine information from various sources to perform tasks like describing images, answering questions about pictures, or reading documents with both text and visuals. This is done by connecting different types of encoders, such as visual or audio encoders, with the language by feeding the output from these encoders as prefix tokens into a frozen language model (Tsimpoukelli et al., n.d.).

2.6.1.1 GPT-4o

GPT-4o is a multimodal large language model developed by OpenAI. It is part of the GPT (Generative Pre-trained Transformer) family of models and is designed to handle both text and image inputs, which is a significant advancement in AI. GPT-4o builds upon the architecture of GPT-4 by

incorporating improved reasoning abilities and a larger scale, enabling it to understand and generate more refined responses across a variety of tasks. The model uses a combination of self-attention mechanisms and positional encoding to process and generate text, while also leveraging ViTs for visual input processing (Chiang, 2024). Unlike previous versions of GPT, GPT-4o can perform complex multimodal tasks, such as interpreting images and generating text-based descriptions or answering questions based on visual content. Its ability to handle both text and image inputs is underpinned by a unified model architecture that merges these modalities through shared representations.

2.6.1.2 Gemini 2.0 Flash

Gemini 2.0 Flash is a multimodal large language model developed by Google DeepMind, optimized for speed and efficiency while maintaining robust capabilities in processing both textual and visual inputs. As part of the second generation in the Gemini series, Gemini 2.0 Flash is more advanced than its predecessors by delivering faster inference times and streamlined memory usage, which makes it highly suitable for real-time applications. It utilizes vision-language pre-training and enhanced cross-attention mechanisms to align image and text inputs effectively, enabling high-quality responses across multimodal tasks. Despite its lightweight nature compared to larger Gemini models, Gemini 2.0 Flash excels in tasks such as visual question answering, image captioning, and multimodal reasoning, demonstrating significant improvements in processing speed without compromising accuracy. Its design prioritizes low latency, making it particularly well-suited for mobile and edge deployments where both speed and context-aware responses are critical.

2.6.1.3 Claude 3.5

Claude 3.5 Sonnet is a multimodal large language model developed by Anthropic. It builds upon the Claude 3 model family, introducing enhanced performance across various tasks, including coding, visual reasoning, and complex instruction following. It integrates both text and image processing capabilities, which allows the model to perform complex tasks that require reasoning across multiple types of data, such as visual question answering,

image captioning, and generating detailed descriptions based on visual inputs. By using advanced ViTs and cross-modality attention mechanisms, Claude 3.5 can analyze images and combine them with textual information, enhancing its ability to understand and respond to tasks that involve both texts and images.

2.6.1.4 LLMs on License Plate Recognition

Although Gemini 1.5 has been deprecated for new projects as of April 29, 2025, its performance remains relevant for comparison. For character-level accuracy, GPT-4o demonstrates the highest character-level accuracy at 97.1%, correctly identifying 1,700 out of 1,751 tested characters (AlDahoul *et al.*, 2024). This excellent performance indicates that GPT-4o is highly adept at recognizing text from images. It is particularly well-optimized for complex OCR tasks, including situations with poor image quality or diverse fonts. Meanwhile, both model from Gemini 1.5, including Flash and Pro, achieves a 93.8% character-level accuracy, identifying 1,643 out of 1,751 tested characters (AlDahoul *et al.*, 2024). While it falls behind GPT-4o, it still represents strong performance. The gap suggests that GPT-4o may be more specialized for text recognition tasks, but Gemini 1.5 remains a competent model for general OCR tasks. It's particularly effective in processing and reasoning with multimodal inputs and can still handle various text recognition scenarios efficiently. Lastly, Claude 3.5 achieves a 92.8% character-level accuracy, correctly identifying 1,625 out of 1,751 tested characters (AlDahoul *et al.*, 2024). While slightly lower than both GPT-4o and Gemini 1.5, Claude 3.5's accuracy is still solid. It is adequate for many OCR applications but may not be as precise in extracting text from images as the other two models, particularly in complex or low-quality scenarios.

In LPR, GPT-4o demonstrates the best performance with an 86% plate-level accuracy, recognizing 222 out of 258 plates (AlDahoul *et al.*, 2024). Its strong performance suggests that the model is well-optimized for tasks involving visual input and complex plate recognition scenarios. Next, Gemini 1.5 Pro achieves a 71.7% plate-level accuracy, recognizing 185 out of 258 plates (AlDahoul *et al.*, 2024). While this is lower than GPT-4o, it still demonstrates reasonable performance. Despite this, it remains a viable option for applications where plate recognition is not the sole focus but is still required. The Gemini

1.5 Flash model performs slightly better than the Pro in license plate recognition, with a 77.5% plate-level accuracy, recognizing 200 out of 258 plates (AlDahoul *et al.*, 2024). This result indicates that the Flash variant is optimized for speed without a significant sacrifice in accuracy. The Flash model's focus on real-time applications allows it to strike a balance between recognition accuracy and processing efficiency. Lastly, Claude 3.5 achieves a 72.1% plate-level accuracy, recognizing 186 out of 258 plates (AlDahoul *et al.*, 2024). Although its performance is still lower than GPT-4o and Gemini 1.5 models, it still shows reasonable capability in recognizing plates. Additionally, Claude 3.5 is known for its speed, operating at twice the pace of its predecessor, making it ideal for scenarios where quick processing is more critical than achieving the highest recognition accuracy.

Table 2.2: Table of accuracy results between LLMs (AlDahoul *et al.*, 2024)

Results	GPT-4o	Gemini 1.5 Flash	Gemini 1.5 Pro	Claude 3.5
Character-Level Accuracy	97.1%	93.8%	93.8%	92.8%
Plate-Level Accuracy	86%	77.5%	71.7%	72.1%

2.6.1.5 LLMs on Vehicle Attribute Recognition

At present, there is no existing research that specifically explores the use of LLMs for Vehicle Attribute Recognition, such as identifying a vehicle's make, model, color, or license plate directly from images. While multimodal LLMs like GPT-4o, Gemini, and Claude 3.5 have demonstrated capabilities in understanding both text and images, most studies and practical applications involving vehicle attribute recognition still rely on traditional computer vision models, such as CNNs, YOLO, or Transformers like ViT, rather than LLMs. This indicates a research gap, where the application of multimodal LLMs to vehicle attribute recognition remains largely unexplored. The potential for these

models to handle such tasks, especially when fine-tuned on relevant datasets, presents an emerging opportunity for future research.

2.6.1.6 Comparison of the Multimodal LLMs

GPT-4o, Gemini 2.0 Flash, and Claude 3.5 are all multimodal models capable of handling both text and image inputs. GPT-4o offers strong language generation and recently introduced fine-tuning with images, making it suitable for tasks like vehicle attribute recognition. Gemini 2.0 Flash also supports fine-tuning through Google Cloud's Vertex AI, excelling in real-time image and text processing, but may incur higher costs due to cloud-based services. Claude 3.5, while strong in text generation and ethical AI, is less mature in image processing and lacks robust fine-tuning capabilities for images, making it less ideal for image-heavy tasks like vehicle recognition. Overall, GPT-4o and Gemini 2.0 Flash are more suited for vehicle recognition, offering flexible fine-tuning with images, while Claude 3.5 is better suited for text-focused or ethical AI applications.

Table 2.3: Table of comparison between LLMs

Feature	GPT-4o	Gemini 2.0 Flash	Claude 3.5
Developed by	OpenAI	Google DeepMind	Anthropic
Input Modalities	Text, image, audio	Text, image, code	Text, image
Output Modalities	Text, audio	Text	Text
Fine-Tuning with Images	Available via OpenAI API	Available via Google Cloud’s Vertex AI	Available vie Claude.ai
Strengths	Strong language understanding with fine-tuning support	High performance with cloud scalability	Focus on safety and strong text generation

Limitations	Cloud-based which can be costly	Cloud-based, may incur costs	Limited image processing & fine-tuning options
Pricing	Input price of $2.50 per million tokens, and output price of $10.00 per million tokens. Context length is 128,000 tokens.	Input price of $0.75 per million tokens, and output price of $3.00 per million tokens.	Input price of $1.50 per million tokens, and output price of $6.00 per million tokens.
API Availability	Yes	Yes	Yes

2.7 Review of Similar Parking Payment Application

This section reviews similar parking payment applications, focusing on their features, technologies, and functionalities.

2.7.1 Touch ‘n Go eWallet

Touch 'n Go eWallet (TNG eWallet) is a Malaysian digital wallet and online payment platform established in July 2017 through a joint venture between Touch 'n Go and Ant Financial Services Group (Touch ’n Go, n.d. b). The app supports a wide range of digital transactions, including payments using QR codes, bill payments, mobile top-ups, money transfers, and ticket purchases for various transport services and events. It is also closely integrated with transportation services, enabling payments for tolls, e-hailing, and car-sharing through features like RFID and PayDirect.

In the context of parking, TNG eWallet offers multiple solutions to enhance convenience and automation. Users can access LPR-enabled parking zones where license plates are automatically recognized for entry and exit, use QR code parking at supported facilities, pay for street parking, and purchase insurance coverage that protects against unexpected incidents while their vehicle is parked.

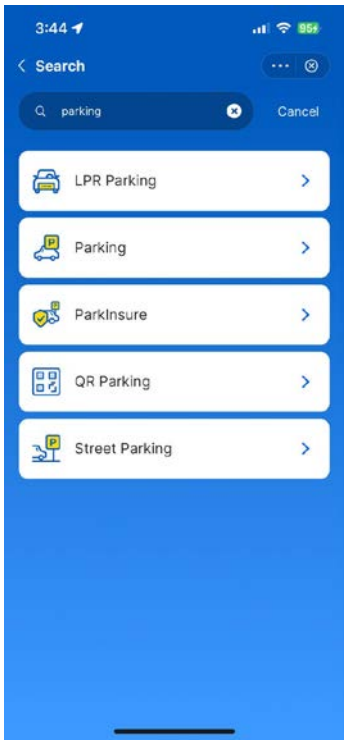


Figure 2.7: TNG eWallet Parking Search Result

2.7.1.1 LPR Parking (License Plate Recognition)

TNG eWallet's LPR Parking feature allows for seamless, ticketless entry and exit at participating parking facilities. Upon arrival at an LPR-enabled parking lot, cameras automatically recognize your vehicle's license plate, lifting the barrier without the need for a physical ticket or card. Upon exit, the system calculates the parking fee and deducts it directly from your eWallet balance (Touch ’n Go, n.d. a).

To activate it, users should tap on "LPR Parking", select “Add vehicle now”, and follow the instructions shown to perform the registration of the vehicle. Users will need to agree to auto-debit terms and confirm with their 6-digit PIN. Lastly, users will get a push notification once the registration for LPR Parking is successful. A user can register up to 10 vehicles under their account and manage their LPR settings individually. While there is no minimum balance required to enter an LPR-enabled parking lot, sufficient funds must be available in the eWallet before exiting to avoid delays. Currently, the LPR Parking feature is available in only 12 locations, and they are working to add in more car parks in Malaysia that support LPR technology (Touch ’n Go, n.d. a).

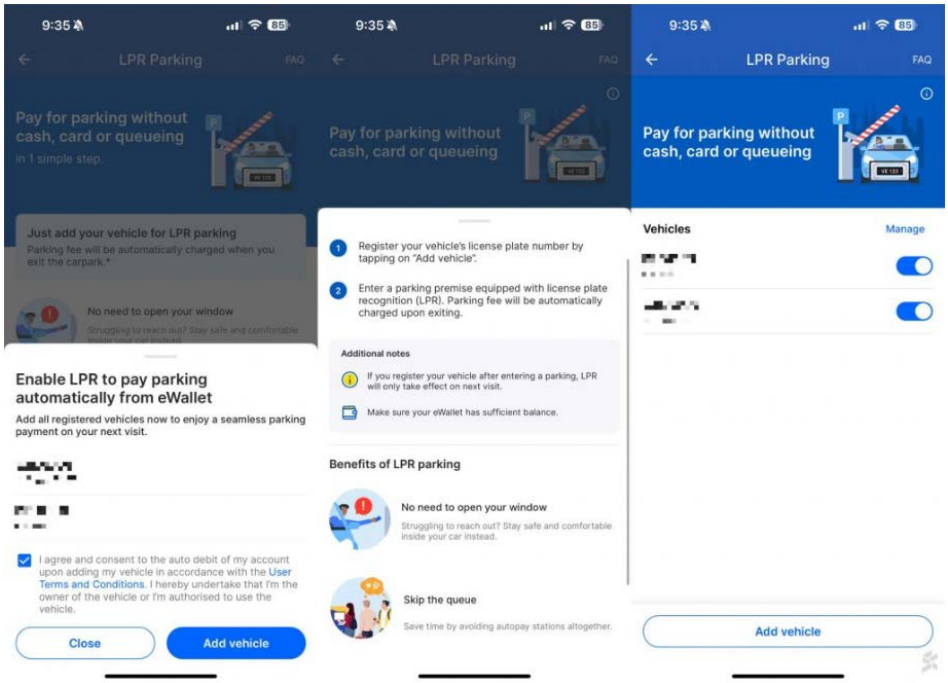


Figure 2.8: TNG eWallet LPR Parking Register Vehicle Steps (Wong, 2025)

2.7.1.2 QR Parking

The QR Parking feature enables a smooth and ticketless parking payment experience through TNG eWallet. Instead of using physical tickets, users scan a QR code at both the entry and exit points of the parking facility. When a user enters a parking facility, the parking reader scans the user's QR code via the eWallet's "Pay" function. This process is repeated during exit, allowing the system to calculate the duration of the parking session. If a payment is required, the fee is automatically deducted from the user's eWallet balance, and a payment notification is sent to the user. Although there is no minimum balance required, it is advisable to maintain sufficient funds in the eWallet to ensure successful payment processing. If the balance is insufficient at the time of exit, the barrier gate will remain closed until payment is completed (Touch 'n Go, n.d. c).

Figure 2.9: TNG eWallet QR Parking Pay Screen (Rozlan, 2023)

2.7.1.3 Street Parking

Touch 'n Go (TNG) eWallet offers a convenient and cashless solution for on-street parking payments across various municipalities in Malaysia. This feature eliminates the need for physical parking meters or paper tickets, allowing users to manage their parking sessions directly through the TNG eWallet app. The app utilizes the device's location services to identify nearby parking zones. This system is currently operational in multiple areas, including Kuala Lumpur, Selangor, Putrajaya, Kota Bharu, and Subang Jaya, among others. Users are advised to check the list of participating local councils within the app to ensure coverage in their desired parking location (Touch ’n Go, n.d. d).

To use the street parking payment feature, users can tap on “Street Parking” within the “Transport” section or search for it on the TNG eWallet app. The app uses GPS to detect nearby supported parking zones. To begin, users must register their vehicle number plate within the app, which will be used to associate and track the parking session with the correct vehicle. Once the vehicle is registered, users can select their parking location and specify the desired duration for their parking session. Depending on the regulations set by the local authority, parking may be booked by the hour or by the day. The app will display the available booking options and the corresponding fees based on the selected zone. These fees are determined by the respective municipal councils and may vary by location and duration. After reviewing the parking details, users can confirm the booking. The parking fee is then automatically deducted from the user’s eWallet balance. A booking is only confirmed once payment is

successfully processed, and a confirmation screen will appear to indicate that the session has started. Users must ensure sufficient balance is available in the eWallet before booking. In addition, the app allows users to monitor their active parking sessions and, if necessary, extend the duration without returning to their vehicle (Touch ’n Go, n.d. d).

2.7.1.4 ParkInsure

TNG ParkInsure is a Personal Accident (PA) insurance or takaful subscription plan offered by Touch 'n Go, designed to provide additional coverage on top of standard motor insurance when using Touch 'n Go cards at TNG-enabled parking sites. It offers protection for incidents occurring within parking compounds, including accidental death or permanent disablement, loss or damage of personal belongings due to accidents, snatch theft, forcible car break-ins, and car accidents. The plan is provided by Allianz General Insurance Company (Malaysia) Berhad for conventional insurance and Zurich General Takaful Malaysia Berhad for takaful coverage (Touch ’n Go, n.d. e). Users can subscribe to ParkInsure through the Insurance section or search for it on the Touch 'n Go eWallet app. The plan costs RM5 per month and is renewed monthly. To be eligible for coverage, users must link their Touch 'n Go card to the ParkInsure plan and use the same card to enter and exit TNG-enabled parking sites.

2.7.2 JomParking

JomParking is a smart parking app developed in Malaysia that provides a cashless and convenient solution for both on-street and off-street parking. By combining digital payments with real-time parking management, the app removes the need for physical tickets or cash, making parking much easier for users.

For on-street parking, the app uses GPS to detect the user's current location. Users select their vehicle and desired parking duration, then confirm the transaction. A countdown timer displays the remaining time, and a notification is sent 15 minutes before expiration. This gives users the chance to extend their parking remotely without needing to go back to their vehicle. On-

street parking is supported in only nine areas managed by local councils (JomParking, n.d. b).

For off-street parking, users select the parking zone within the app. A QR code is generated, which is scanned at the entrance and exit of the parking facility. The parking fee is automatically deducted from the user's token balance upon exit. JomParking also offers monthly parking passes for specific zones, providing a convenient solution for regular commuters (JomParking, n.d. b). Off-street parking is supported at prominent locations such as Terminal Bersepadu Selatan (TBS) and Gurney Mall @ Residensi UTMKL in Kuala Lumpur (Apple, n.d. b).

Other helpful features include compound management, where users can view and pay parking fines directly in the app. The app also supports multiple vehicle registrations under one account, making it suitable for users who own or manage more than one car. All parking transactions are recorded within the app, allowing users to track their parking activities and payments. JomParking provides reminders for vehicle insurance renewals and real-time notifications for session expiry, payment confirmations, and promotional updates.

JomParking operates on a token-based system, where users purchase tokens using various payment methods such as online banking, credit/debit cards, or e-wallets. These tokens are then used to pay for parking sessions, monthly passes, and compounds (JomParking, n.d. a).

2.7.3 ParkEasy

ParkEasy is a Malaysian mobile application designed to help users find and reserve parking spots ahead of time, especially in shopping malls and commercial areas. One of its main features is the parking reservation system, where users can select their location and reserve a parking spot ahead of time. ParkEasy also supports electric vehicle (EV) charging bay reservations, allowing EV owners to book charging spots in advance.

Furthermore, users can easily manage their reservations within the app, including checking and canceling reservations. The app provides clear

navigation instructions to the reserved parking spot and uses a parking lock system to ensure the spot remains available until the user's arrival.

Moreover, the app operates on a credit-based system, where users can purchase credits to make reservations. Payments can be made using credit or debit cards and online banking. New users receive free credits upon registration, and more credits can be earned through referral programs and promotional codes. ParkEasy also provides free insurance coverage against break-ins or theft for users parking at participating malls, offering an added layer of security (ParkEasy, 2016).

Lastly, ParkEasy offers memberships such as Shell Recharge Gold, MyEVOC, and BonusLink (ParkEasy, n.d.). Shell Recharge Gold gives users longer reservation grace periods and cheaper charging rates at Shell Recharge stations. The MyEVOC membership provides special benefits for EV owners, making it easier and more convenient to charge their vehicles. With BonusLink, users can earn points for every transaction made through the ParkEasy app by linking their accounts.

2.7.4 Flexi Parking

Flexi Parking is a Malaysian mobile app developed by Leading Innovative Technologies and Systems Sdn Bhd (LITS). It helps users pay for both on-street and off-street parking easily without the need for physical coupons or tickets. The app supports over 40 municipal councils across 9 states, including popular areas like Selangor, Smart Selangor Parking, and Kuala Lumpur, Wilayah Parking (Apple, n.d. a). With its wide coverage, users can pay for parking in multiple areas using just one app.

Parking in Flexi Parking can be paid in just two steps, which are choosing the vehicle number and selecting the parking duration. Flexi Parking supports multiple languages, including Bahasa Malaysia, English, and Chinese, to suit different users (Apple, n.d. a). It also keeps all receipts in digital form, making it easy to check past transactions or email them when needed.

Flexi Parking works on a prepaid credit system, where users top up credits in advance using online banking, debit or credit cards, and other payment methods. One account can manage up to 10 vehicles, making it convenient for

families or businesses. The app also allows users to pay for parking compounds issued by some municipal councils directly through the app, avoiding the need to visit payment counters (Apple, n.d. a).

In addition to regular parking, the app offers monthly parking passes for up to six months. It also supports gated off-street parking through QR code scanning and LPR. Notifications are sent when parking time is about to expire, helping users avoid getting fined. GPS is used to help users select the correct municipal council based on their location, reducing the chance of mistakes during payment (Apple, n.d. a).

2.7.5 Summary and Comparison of Existing Applications

Each of the four parking payment applications offers unique features but also has its limitations. TNG eWallet offers a comprehensive cashless system with options like LPR, QR-based entry, and pay-on-the-spot methods, while also offering rewards and ParkInsure coverage. It also supports multiple vehicles of up to ten per account. However, its LPR functionality is currently limited to only twelve locations, and users must maintain enough eWallet balance, which may cause delays if funds are insufficient.

JomParking is recognized for its user-friendly interface, token-based payment system, and useful features such as insurance renewal reminders, compound management, and support for up to ten vehicles. It also lets users extend parking sessions remotely and offers monthly passes. However, its token-based system may confuse new users. Additionally, it does not provide guaranteed spot availability and lacks insurance coverage.

ParkEasy distinguishes itself through its advance reservation system, which includes electric vehicle (EV) charging bay bookings, and further increases its value by offering insurance against theft or break-ins. In addition, membership programs such as Shell Recharge Gold, MyEVOC, and BonusLink offer users additional benefits. While these features are appealing, ParkEasy’s coverage is limited, and users may also encounter situations where a reserved parking space is still occupied if the previous user does not leave on time.

Flexi Parking provides the widest coverage nationwide, supporting over 40 municipal councils across nine states. It enables LPR and QR-based

parking, offers monthly passes, compound payments, and multiple vehicle support for up to ten vehicles, and includes Flexi Protect insurance. Its additional features, such as multi-language support and digital receipts, further improve accessibility and usability. However, the application's limitations include the risk of credit expiration, dependence on participating municipal councils, and the potential discontinuation of services in certain areas.

In conclusion, each app has strengths and weaknesses. TNG eWallet and JomParking provide reliable cashless payment systems, but both are limited in terms of coverage and guaranteed parking availability. ParkEasy is strong in reservation and electric vehicle (EV) services, though its use is restricted mainly to certain locations. Flexi Parking has the widest coverage across Malaysia, but it faces challenges such as credit expiry and dependence on participating municipal councils.

Table 2.4: Table of comparison between existing parking applications

Application Name	Touch 'n Go eWallet	Jom Parking	ParkEasy	Flexi Parking
Payment Method	Cashless, with payments deducted directly from the eWallet balance using methods such as LPR, QR code, or pay-on-the-spot.	Cashless, token-based system with options such as QR code, pay-on-the-spot, or monthly passes.	Cashless, credit-based system for both reservations and on-the-spot payments.	Cashless, credit-based system with methods such as LPR, QR code, pay-on-the-spot, or monthly passes.
Parking Types Supported	On-street and off-street	On-street and off-street	On-street and off-street	On-street and off-street
Multiple Vehicle Management	Up to 10 vehicles	Up to 10 vehicles	Multiple vehicles under one	Up to 10 vehicles

			account (not explicitly mentioned)	
Location Coverage	Limited (12 locations for LPR)	Limited, dependent on local council decisions	Limited	Extensive (Multiple Malaysian states)
Booking / Reservation	No	No	Parking and EV charging reservations allowed	No
Rewards & Promotions	Occasional rewards and cashback	Promotional campaigns, including free tokens and special offers, and partnership with brands	Partnerships with brands for exclusive promotions	Promotions such as free credits
Insurance Coverage	Yes, with ParkInsure	No	Yes	Yes, with Flexi Protect
Token / Credit System	No	Yes	Yes	Yes
Compouds payment	Yes	Yes	No	Yes
What stands out	Comprehensive payment system with multiple parking	User friendly interface, provides reminders for vehicle	Parking reservations and EV features	Broad geographic coverage (supports over 40 municipal

	methods and rewards	insurance renewals		councils across 9 states)
Limitations	Currently only 12 locations for LPR system	No guaranteed spot availability & no insurance coverage	Limited locations, reserved spaces may still be occupied by previous users	Limited to certain councils and may be discontinued, credit expiration risk

2.8 Software Development Methodologies

Software development methodologies are structured approaches used to plan, manage, and control the development of software systems. Each methodology defines how a project should be carried out by outlining steps, roles, responsibilities, communication standards, activities, and expected deliverables throughout the software development life cycle. These stages include requirements gathering, planning, analysis, design, implementation, testing, deployment, and maintenance. The wide variety of methodologies available allows teams and organizations to select the one that best suits their specific needs, depending on factors like project size, complexity, timeline, team structure, and the level of flexibility required.

Selecting the right methodology is a crucial decision in any software project, as it can significantly impact cost, timeline, quality, and user satisfaction. However, there is no one-size-fits-all solution, as each project has unique goals and constraints. Therefore, it is important to study and compare different approaches to determine the most effective one for a specific context. Successful outcomes often rely on adopting a management structure that aligns with project needs and objectives.

2.8.1 Waterfall Model

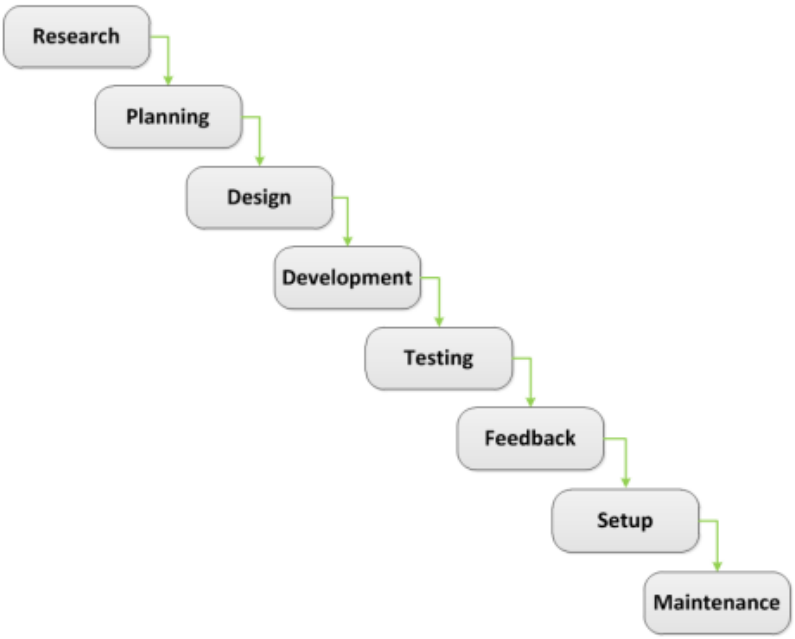


Figure 2.10: Waterfall Methodology (Burtescu *et al.*, 2014)

The Waterfall Model is widely known as the earliest approach to software development. The concept was introduced by Winston W. Royce in 1970 and gained prominence throughout the 1970s and 1980s (A.K.M Zahidul Islam and Ferworn, 2020). It is often referred to as a “top-down” or “linear sequential” development model. The model is heavily inspired by traditional engineering and construction workflows.

One of the key characteristics of the Waterfall Model is its linear sequential flow, where each phase flows into the next like a waterfall with no overlapping or iteration between phases. Each phase must be fully completed with clearly defined deliverables before proceeding to the next. This model relies heavily on documentation, which supports each stage of the process (A.K.M Zahidul Islam and Ferworn, 2020). Since all planning and design are done at the beginning, the process becomes predictable (Burtescu *et al.*, 2014). However, it will be difficult to make changes or feedback once development has started. Feedback from stakeholders usually happens only after the system has been built and tested.

The Waterfall Model has several advantages. It is simple and easy to understand, which makes it suitable for teams with limited experience or

resources. The clearly defined stages make the process easy to manage and implement. Additionally, the structured schedule makes it easier to allocate time and resources for each phase (Vishal Chandra, 2015).

However, the model also has multiple drawbacks. One of the main disadvantages is its inflexibility and rigid structure, which makes it difficult to revisit earlier phases for corrections or improvements. The lack of feedback between phases means that errors or misunderstandings might only be discovered late in the development cycle, leading to costly and time-consuming revisions. Moreover, since each phase must be completed before the next begins, any delays or issues can significantly affect the project schedule (Vishal Chandra, 2015).

2.8.2 Iterative and Incremental Development (IID) Model

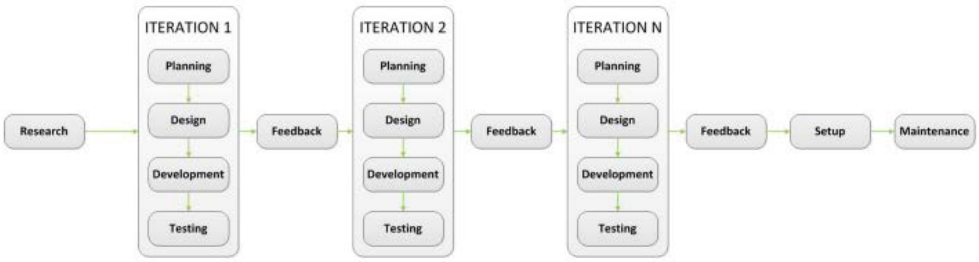


Figure 2.11: IID Methodology (Burtescu *et al.*, 2014)

The Iterative and Incremental Development model is a development approach where the software is built step-by-step through iterative design and incremental delivery. Development starts with a basic version of the system based on initial requirements. After each iteration, feedback is collected from the project owner or stakeholders, and necessary changes or enhancements are made. The model is improved and extended in each cycle without discarding previous work (Burtescu *et al.*, 2014). The model emphasizes design over documentation and ensures that feedback is received after each iteration is completed. This helps maintain alignment with user expectations and project goals throughout development (Burtescu *et al.*, 2014).

In iterative design, requirements and designs are revised throughout the development lifecycle, emphasizing continuous improvement. The team

continuously learns and adapts based on what is built, tested, and observed in each cycle. This feedback loop allows the team to identify areas of improvement, correct misunderstandings, and make informed decisions for the next iteration. In doing so, the system evolves to better match user expectations. In incremental design, the system is delivered in parts. Each part adds new functionality, and these parts are tested independently before being integrated (Suhasini Gadam, 2023). While early increments may have limited functionality, they are fully operational. As more increments are added, the system gradually becomes a complete and functional application (Saeed *et al.*, 2019). This ongoing development allows potential issues to be detected early, reducing the risk of costly revisions later in the process.

The IID model offers key advantages such as early delivery of a working product, allowing users to provide feedback from the beginning. This frequent feedback loop helps align the system with actual user needs. The model is flexible, making it ideal for projects with changing requirements, and it supports early issue detection, which improves risk management. Continuous testing across iterations also enhances overall software quality.

However, IID also comes with drawbacks. Frequent changes can lead to scope creep, causing the project to exceed its original goals. Planning becomes complex when future requirements are unclear, and the repeated cycles may demand more time and resources. Additionally, rushed iterations can result in technical debt if code quality is compromised, and poor planning may cause integration issues between increments.

2.8.3 Agile Methodologies

Agile methods were developed to address challenges in traditional software development, especially in fast-changing environments. These methodologies emphasize flexibility, collaboration, and iterative development, making them ideal for projects where requirements evolve over time. According to Miller (2001), Agile breaks projects into short, manageable steps, eliminating unnecessary work and enabling quick fixes. It prioritizes teamwork and communication over rigid processes.

2.8.3.1 Scrum

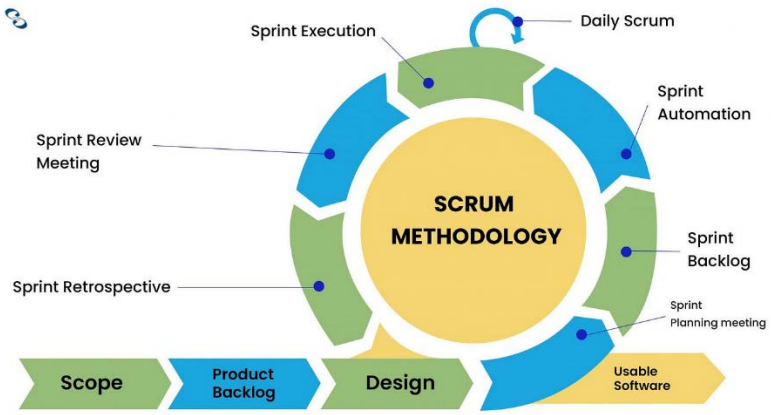


Figure 2.12: Scrum Methodology (Korkut, 2023)

Scrum is an agile, lightweight framework created for managing complex product development, especially in software engineering. The concept was first introduced by Hirotaka Takeuchi and Ikujiro Nonaka in 1986. Unlike the traditional Waterfall model, which relies on sequential stages, Scrum supports iterative and incremental development, allowing teams to adapt quickly to changing requirements (Apoorva Srivastava, Sukriti Bhardwaj and Shipra Saraswat, 2017).

The Scrum methodology is based on three main roles: the Product Owner, Scrum Master, and the Development Team. The Product Owner decides what needs to be built and sets the priorities. The Scrum Master helps the team follow Scrum rules and clears up any problems they face. The Development Team is a group of skilled members who work together to build and deliver a part of the product at the end of each sprint (Sakshi Sachdeva, 2016).

A key element of Scrum is the sprint, a time-boxed iteration typically ranging from one to four weeks, where the team works to complete a part of the product. Each sprint begins with a Sprint Planning meeting where the team defines the Sprint Goal and selects items from the product backlog to be completed. Throughout the sprint, the team participates in daily Scrum meetings to discuss progress, identify obstacles, and coordinate efforts. At the end of each sprint, the team holds a Sprint Review to present the increment to stakeholders for feedback and a Sprint Retrospective to reflect on the process and identify areas for improvement (Sakshi Sachdeva, 2016).

The advantages of Scrum include its adaptability, which makes it suitable for dynamic project environments where requirements frequently change. It also improves communication through regular team meetings and stakeholder involvement. This leads to early and continuous delivery of software, with faster feedback and adjustments. Additionally, Scrum is cost-effective because it identifies and resolves issues early in the process, reducing the need for expensive rework. The methodology also improves product quality through continuous testing and integration, ensuring that each increment meets the desired standards.

Scrum also has some challenges, including requiring a skilled and self-managing team for successful adoption, making it difficult to use in organizations with rigid hierarchies or insufficient training. Moreover, misuse or partial adoption of Scrum practices, often referred to as "ScrumBut", can reduce its effectiveness (Sakshi Sachdeva, 2016). Scrum also doesn’t include detailed engineering practices, so teams often combine it with other methods like Extreme Programming (XP) (Sakshi Sachdeva, 2016). For big or complex projects, using Scrum alone can be difficult, and extra frameworks like SAFe or Nexus may be needed.

2.8.4 Summary and Comparison of Methodologies

The Waterfall model is most suitable for projects with well-defined, stable requirements and a clear end goal. It is ideal in environments where detailed planning can be done upfront and changes during development are unlikely. Industries such as healthcare, aerospace, and defense often favor Waterfall due to its emphasis on formal documentation, regulatory compliance, and structured processes. Waterfall works best when the project scope, timeline, and budget are fixed and when stakeholders can finalize requirements early in the process. However, it is not suited for projects where requirements may evolve, as it offers little flexibility and delays feedback until late in the development cycle. This makes it less appropriate for complex, dynamic, or user-driven projects.

IID is most suitable for projects where requirements are expected to evolve or when the final product is not fully defined at the beginning. It is ideal for uncertain projects, exploratory development, or when early delivery of core

features is critical. By delivering the software in smaller, manageable increments, IID supports continuous feedback and quicker issue resolution, making it beneficial in time-sensitive or innovation-driven contexts. It is also useful when future improvements are anticipated after initial deployment. However, IID may not be the best choice for projects that require a rigid structure, fully defined deliverables from the start, or a lot of planning upfront. Without careful control, the flexibility of IID can lead to integration challenges and uncontrolled changes in scope.

Scrum is most effective in fast-paced, complex projects where requirements are likely to change frequently. It is suitable for projects that focus on adaptability, close collaboration, and frequent delivery of working software. It works best for projects that rely on regular customer feedback and team-based decisions, especially when teams are cross-functional and self-organizing. However, Scrum may not suit projects with fixed scope, rigid deadlines, or limited team freedom. It also requires a certain level of Agile experience and defined roles like Scrum Master and Product Owner, which may not be practical in smaller or more traditional teams. In such cases, alternative methodologies may offer a better fit.

Table 2.5:Table of differences between methodologies compared

	Waterfall	IID	Scrum
Development Style	Linear and sequential	Build in increments and improve in iterations	Agile, iterative, and time-boxed
Flexibility	Very low	Medium	Very high
User Involvement	Only at the beginning and end	Moderate (feedback after increments)	Very high (constant feedback every sprint)
Risk Management	High risk (late discovery of issues)	Moderate (issues found after each increment)	Low (issues found early and

			corrected quickly)
Delivery	Single delivery at the end	Partial delivery after each increment	Working product delivered at the end of each sprint
Time Estimation	Predictable (if requirements are stable)	Moderate	Unpredictable
Project Type Suitability	Best for clear, fixed requirements, small projects	Good for evolving requirements, medium projects	Best for complex, changing projects

2.9 Development Framework

Choosing the right frameworks for the frontend, backend, and database is essential for building an efficient, scalable, and maintainable application. The frontend determines how users interact with the app, the backend handles the business logic and data processing, and the database stores and retrieves the data. In this section, we will compare different frameworks and tools to evaluate their suitability for developing a web and mobile app for a parking payment system.

2.9.1 Frontend Framework

2.9.1.1 React Native (with React Native for Web)

React Native is an open-source framework developed by Meta that enables developers to build cross-platform mobile applications using JavaScript and React. Unlike traditional web-based frameworks, React Native uses native components instead of HTML elements, allowing applications to achieve near-native performance on both Android and iOS platforms while maintaining a single shared codebase. This approach simplifies development and reduces maintenance effort compared to creating separate native apps for each platform.

React Native operates by bridging JavaScript code to native mobile components, ensuring that applications deliver smooth performance and a consistent user experience. The framework supports features such as hot

reloading, flexible styling, and integration with native modules, which enhance development efficiency and scalability. Because of its architecture, developers can reuse a significant portion of code across multiple platforms while still having the flexibility to implement platform-specific features when needed.

In addition to mobile platforms, React Native can be extended to support web development through React Native for Web. This library bridges React Native components to the React DOM, enabling the same codebase to run seamlessly in web browsers. It allows developers to build applications that function consistently across web and mobile without rewriting core logic. React Native for Web leverages modern React features such as function components and hooks, and converts native components into standard web elements while maintaining their layout and design behavior. This makes it a practical choice for projects that require both mobile and web interfaces with a unified user experience.

Overall, React Native (with React Native for Web) provides a powerful and flexible solution for cross-platform application development. It offers the performance and feel of native mobile apps while maintaining the adaptability of web technologies. Major companies such as Meta, Microsoft, and Shopify use React Native for their production applications, demonstrating its reliability and scalability for real-world use.

2.9.1.2 Flutter

Flutter is an open-source UI framework developed by Google for building cross-platform applications from a single codebase. It allows developers to create applications for Android, iOS, web, and desktop platforms using the Dart programming language. Flutter's key advantage lies in its high-performance rendering engine, known as Skia, which allows it to draw UI elements directly on the screen rather than relying on native platform components. This approach ensures consistent appearance and performance across platforms.

Flutter provides a rich set of customizable widgets, enabling developers to create visually appealing and highly responsive interfaces. Its hot reload feature allows developers to instantly view code changes without restarting the application, which significantly speeds up the development and

debugging process. Flutter also includes a powerful layout system that supports adaptive design, ensuring that applications function well on devices with different screen sizes and resolutions.

For web development, Flutter compiles Dart code to JavaScript, allowing applications to run efficiently in modern browsers. Although Flutter's web support continues to evolve, it already enables the deployment of lightweight and visually consistent web interfaces that share the same business logic and UI design as mobile apps. This makes it an attractive option for projects aiming to achieve a unified design system across multiple platforms.

Overall, Flutter offers an efficient and versatile framework for building cross-platform applications with a single codebase. Its strong performance, expressive UI components, and growing community make it a leading choice for both mobile and web application development.

2.9.1.3 Ionic React

Ionic React is a hybrid application development framework that integrates the Ionic UI library with the React ecosystem to build cross-platform web and mobile applications. It uses standard web technologies, such as HTML, CSS, and JavaScript, to create applications that run in a WebView for mobile or directly in a web browser. Ionic is designed with a mobile-first approach and provides a native-like look and feel through a rich collection of pre-built, mobile-optimized UI components, as well as tools for navigation, routing, gestures, and animations. It works well with popular JavaScript frameworks like React, Angular, and Vue, making it ideal for developers already familiar with these technologies (Haire, n.d.). This approach enables developers to maintain a single codebase for multiple platforms, reducing development time and effort.

Ionic React emphasizes UI consistency through its pre-built and customizable components that follow modern mobile design principles such as Material Design and iOS Human Interface Guidelines. The framework integrates with Capacitor, a native runtime that allows access to device hardware features like the camera, GPS, and file system. This enables web-based applications to function more like native mobile apps while maintaining the flexibility of web technologies.

For web deployment, Ionic React applications can run directly in browsers with minimal modification, making it suitable for projects that prioritize rapid development and easy maintenance. However, since it relies on a WebView for mobile platforms, its performance may not match that of fully native or compiled frameworks like React Native or Flutter. Despite this, Ionic React remains a popular choice for developing applications that need to reach both web and mobile users efficiently.

In summary, Ionic React provides a balanced solution between web flexibility and mobile accessibility, making it ideal for applications that require cross-platform compatibility without complex native development.

2.9.1.4 Summary and Comparison of Frontend Framework

When comparing React Native, Flutter, and Ionic React, several key factors influence their suitability for cross-platform development. React Native excels in scenarios where developers aim to build mobile and web applications with a shared codebase using the React ecosystem. It provides near-native performance by bridging JavaScript components to native platform APIs. Its ability to extend to the web through React Native for Web makes it suitable for teams seeking a unified development experience across platforms.

Flutter is best suited for applications requiring high performance, custom UIs, and complex animations. Since it compiles directly to native code, Flutter delivers smooth performance and a consistent UI across devices. However, its larger app size and the need to learn the Dart language can pose challenges for developers who are new to the framework.

Ionic React, on the other hand, provides an accessible entry point for developers familiar with web technologies such as HTML, CSS, and JavaScript. It enables rapid development for both mobile and web applications through a single codebase and integrates with Capacitor for accessing native device features. However, as it primarily relies on WebView rendering, it may not match the performance of frameworks that compile to native code, especially in graphically intensive applications.

Overall, React Native offers the best balance between performance and code reusability, Flutter delivers superior native-like performance with

flexibility in design, and Ionic React provides simplicity and fast development cycles for web-oriented teams.

Table 2.6: Table of differences between frontend frameworks compared

Feature	React Native	Flutter	Ionic React
Programming Language	JavaScript with React syntax	Dart	JavaScript with HTML, CSS, and React
Performance	Good with native-feel	Excellent with native-like speed, especially on mobile	Moderate as WebView-based and may be slightly slower for complex apps
UI Look	Native look on web and mobile	Consistent UI across platforms with custom widgets	Web-style UI
Learning Curve	Easy if familiar with React	Steeper as need to learn Dart and Flutter structure	Easiest if familiar with web development and React
App Size	Moderate	Larger as Flutter bundles its rendering engine	Light as it relies on the browser's engine
Web Support	Supported via React Native for Web	Available but still maturing	Excellent, designed for web-first applications
Mobile Support	Strong native performance for Android and iOS	Strong native performance for Android and iOS	Supported via WebView with Capacitor bridge
Use Case	Best for apps with shared mobile &	Priority on beautiful UI, animation-heavy	Priority on rapid development, web-first experience

	web code using React	apps, and term scalability	

2.9.2 Backend Framework

2.9.2.1 NestJS

NestJS is a framework for building server-side applications with Node.js. It is built with TypeScript and also supports regular JavaScript. NestJS stands out because it provides a clear structure for organizing applications, making it easier to manage and scale large projects. The framework follows a modular approach, where the code is divided into modules, controllers, and services. This helps separate responsibilities within the application, making the code cleaner and more maintainable. NestJS is inspired by Angular's architecture, so developers familiar with Angular may find it easier to learn. NestJS uses Express as its default web server, but developers can also choose Fastify for better performance. While Nest adds a layer of abstraction, it still allows full access to the Express or Fastify features. Additionally, it supports modern programming styles like object-oriented, functional, and functional reactive programming (NestJS, n.d.). NestJS also offers a strong foundation for building reliable and scalable applications and is suitable for teams that value clean architecture, maintainability, and good development practices.

Despite that, NestJS has some limitations. It has a steep learning curve, particularly for developers unfamiliar with TypeScript or advanced programming concepts. It also requires a lot of repetitive code, which can feel unnecessary for small projects. Additionally, NestJS may introduce some performance overhead due to its use of decorators and abstraction layers. Lastly, its community is still growing, which can make finding solutions for specific issues more challenging compared to more established frameworks.

2.9.2.2 Laravel

Laravel is a modern PHP framework known for its elegant syntax and powerful features that simplify web application development. It follows the Model-View-Controller (MVC) pattern, helping developers organize their code and maintain separation of concerns. Laravel offers built-in tools for routing, session

management, caching, authentication, and more, reducing the need for third-party packages. It also provides robust security features, such as protection against SQL injection, CSRF, and XSS, ensuring secure applications out of the box (Sehouli, 2025). One of Laravel's key strengths is its scalability. The framework is designed to handle large workloads and can be easily scaled horizontally using distributed caching systems like Redis (Laravel, n.d.). Laravel also supports task scheduling and job queues, making it easier to manage background processes such as email sending or file uploads. With a large and active community, Laravel also benefits from a rich ecosystem of packages and resources. The framework also supports unit testing and provides tools for debugging and error handling, enhancing the overall developer experience.

However, Laravel's main disadvantage is its performance, as it tends to be heavier than other PHP frameworks, which may cause slower response times in high-traffic applications. The learning curve can also be steep for beginners, especially with the large number of the framework’s features and tools. Laravel’s heavy reliance on third-party packages can also lead to extra dependencies, making it harder to manage as the project grows. Additionally, deploying Laravel can be more complicated than other frameworks as it requires specific server environments and tools.

2.9.2.3 Summary and Comparison of Backend Framework

NestJS and Laravel are both powerful frameworks, but they fit different ecosystems and use cases. NestJS is a TypeScript-based framework built on Node.js, ideal for scalable, high-performance applications, especially those requiring microservices, APIs, and real-time features. It's well-suited for developers familiar with JavaScript/TypeScript and offers flexibility and modularity. However, it has a steeper learning curve and might be more complex for beginners. Meanwhile, Laravel is a PHP-based framework that focuses on rapid development with an elegant syntax and a robust set of built-in tools like Eloquent ORM, Blade templating, and Artisan CLI. It is best for web applications and traditional server-rendered projects. Laravel’s ease of use,

excellent documentation, and strong ecosystem make it great for developers who prefer PHP.

Table 2.7: Table of differences between backend frameworks compared

Feature	NestJS	Laravel
Approach	Structured, modular, and built for scalability	MVC framework
Performance	High performance	Good
Development Speed	Slower for small projects but faster for large-scale applications	Fast development with lots of built-in tools
Scalability	Excellent scalability with built-in support	Scalable but requires manual configuration for advanced features
Built-in Features	Many	Many
Learning Curve	Steeper	Moderate
Database Integration	Supports TypeORM or Sequelize for ORM	Eloquent ORM that is built-in for smooth database integration
Ecosystem	Built on Node.js, uses npm but with additional tooling and structure	Huge PHP ecosystem with tools like Composer, Laravel Mix, Eloquent ORM
Testing	Built-in testing tools like Jest	Built-in testing tools like PHPUnit
Use Case	Best for large-scale, enterprise-level, or complex applications	Best for full-stack web applications, especially with a database and admin panel

2.9.3 Database

2.9.3.1 MySQL

MySQL is a popular open-source relational database management system (RDBMS) widely used for storing and managing data. It is reliable, fast, scalable, and easy to use, making it essential for high-traffic apps like Facebook, Netflix, Uber, Airbnb, Shopify, and Booking.com (Erickson, 2024). MySQL stores data in tables organized by schemas and uses SQL (Structured Query Language) for querying and managing data. It supports ACID transactions, ensuring data consistency even during system failures, and can handle a large number of concurrent connections. As MySQL is open source, it allows developers to freely access and modify its code, making it a cost-effective solution for both small projects and enterprise-level systems. It supports a variety of data types, including traditional SQL data and JSON, enabling flexibility in how data is stored and accessed. MySQL also scales efficiently, with native replication and failover features that ensure high availability and performance under demanding conditions. Over its nearly 30-year history, MySQL has developed a robust community that continually improves the system. It is compatible with various programming languages like Java, Python, and PHP, and is known for its easy setup and management. MySQL’s flexibility, low cost, and support for both SQL and NoSQL applications make it a reliable choice for developers across various industries (Erickson, 2024).

However, MySQL has some limitations, including a lack of advanced features for complex queries, recursive operations, and certain joins, which can be restrictive for complex applications. It also doesn't fully comply with SQL standards, leading to potential compatibility issues when migrating to other databases. MySQL is also less extensible than systems like PostgreSQL, limiting customization options for developers. While it offers some JSON support, MySQL's NoSQL capabilities are weaker compared to databases like MongoDB or PostgreSQL, making it less suitable for handling unstructured or semi-structured data.

2.9.3.2 PostgreSQL

PostgreSQL is a powerful, open-source object-relational database system that extends SQL and is known for its reliability, data integrity, and extensibility. Major companies such as Apple, IMDB, Instagram, Reddit, Skype, Spotify, and Twitch use PostgreSQL due to its powerful features, flexibility, scalability, and cost-effectiveness (Romanowski, 2020). It is ACID-compliant, compatible with all major operating systems, and offers robust features such as the PostGIS geospatial extender for better spatial data handling. PostgreSQL is highly extensible and allows developers to define custom data types, functions, and even use different programming languages without recompiling the database. It adheres closely to SQL standards and supports advanced features like indexing, multi-version concurrency control, and parallel query processing. PostgreSQL also offers strong disaster recovery capabilities, including write-ahead logging, replication, and point-in-time recovery. Security features include robust authentication methods and access controls, while its extensibility allows for a wide range of customizations, including stored functions and foreign data wrappers (The PostgreSQL Global Development Group, n.d.). With its scalability and ability to manage large data sets, PostgreSQL is widely used in both small and enterprise-level applications.

However, However, PostgreSQL's advanced features can lead to performance overhead, making it less optimal for read-heavy applications. It can also be complex to configure and manage, requiring more effort and expertise to optimize the system, which could lead to longer setup times and higher maintenance costs. Additionally, PostgreSQL has a steeper learning curve due to its extensive features, making it more challenging for beginners.

2.9.3.3 Summary and Comparison of Database

When comparing MySQL and PostgreSQL, the main differences are their features, performance, and extensibility. MySQL is better for applications that need fast read operations, as it offers faster performance and easier management, making it ideal for applications like e-commerce platforms and websites with high traffic. Its simplicity and speed make it a good choice for small to medium-sized projects. Meanwhile, PostgreSQL, which is known for its advanced

features and scalability, performs well in complex applications requiring high data integrity, custom data types, and extensibility. It is fully SQL-compliant, supports complex queries, and provides great disaster recovery features, making it ideal for large-scale applications, analytical processing, or geospatial data.

Table 2.8: Table of differences between the databases compared

Feature	MySQL	PostgreSQL
Type	Relational DBMS	Object-relational DBMS
Performance	Fast for read-heavy operations	Better for complex queries & write-heavy
Data Integrity	Decent (less strict)	Strong (strict ACID compliance)
Complex Queries	Basic joins & subqueries supported	Advanced joins, CTEs, window functions
Geolocation (GIS)	Basic support via MySQL Spatial Extensions	Advanced with PostGIS
Ease of Setup	Simple	Slightly more complex

CHAPTER 3
METHODOLOGY AND WORK PLAN

3.1 Introduction

This chapter outlines the methodology and work plan for the parking system project. It covers the chosen methodology, which will guide the project's development process, as well as the Work Breakdown Structure (WBS), which breaks the project into smaller tasks. A Gantt chart is provided to visualize the project timeline and track progress. Additionally, the chapter discusses the development tools and techniques that will be utilized throughout the project to ensure efficient and effective implementation.

3.2 Software Development Methodology: Scrum Methodology

The software development of the Parking Payment System for License Plate and Vehicle Attribute Recognition will adopt the Scrum methodology. Scrum was chosen due to its flexibility, iterative nature, and ability to adapt to evolving requirements, which is important in this project with the integration of advanced AI techniques. The development is divided into Sprints, with each producing usable increments or improvements of the system.

3.2.1 Phase 1: Initiation

The first phase involves initiating the project by clearly defining the overall vision. The objective of this project is to build a system that automates parking transactions through license plate and vehicle attribute recognition powered by multimodal AI models. This system aims to streamline the parking experience for drivers and parking operators by leveraging advanced AI technologies to enable seamless, secure, and efficient payment processing.

The key stakeholders for this project include drivers, parking operators, and system administrators. Drivers will use the system to view parking rates, activate auto-payments, and check their parking history. Parking operators will use the system to monitor parking activity through dashboards and reports, as well as manage parking rates and related details. System administrators will use

the system to manage account approvals, review support tickets, and oversee changes requested by parking operators.

Although Scrum methodology is traditionally used for team-based development, its practices are adapted in this project for a single-developer environment. The Scrum roles are assigned where the developer takes on the responsibilities of both the Scrum Master and Developer roles, while the supervising lecturer or the developer assumes the role of Product Owner. This approach ensures that the project remains organized, iterative, and focused on delivering incremental improvements.

An initial Product Backlog is created based on the outlined project scope. Key features in the Product Backlog include user account registration, vehicle registration, activation of auto-payment, viewing of parking rates, fraud prevention mechanisms, support ticket submission and management, management of parking zones by operators, user account management by administrators, and analytical dashboards for both operators and administrators. This initial backlog forms the foundation for planning and prioritizing tasks during the subsequent phases of the project.

3.2.2 Phase 2: Planning and Estimation

In the Planning and Estimation phase, the development process begins by conducting Sprint Planning sessions to prioritize the items listed in the Product Backlog. Each feature or requirement is evaluated based on its importance and contribution to the overall project vision, ensuring that critical functionalities are addressed early in the development cycle.

Major features are broken down into smaller, manageable User Stories to make the development tasks clearer and more achievable. For example, a User Story might be framed as "As a driver, I want to register my vehicle so I can activate auto-payment." This method allows for a user-centered approach, ensuring that all system functionalities align closely with real-world needs and user expectations.

Effort estimation is then carried out for each User Story using story points or simple relative sizing. This helps in understanding the complexity and workload associated with each task, helping in better Sprint planning and time

management. Each Sprint is set to a duration of two weeks to ensure consistent development and allow for regular reviews and adjustments.

Sprint Goals are established at the beginning of each Sprint to provide clear direction and focus. These goals might include completing specific modules such as the vehicle registration flow, implementing basic login and authentication functionality, or integrating early versions of the multimodal AI recognition system. Setting clear Sprint Goals helps ensure that each Sprint moves the project closer to completion.

Additionally, the phase involves creating a WBS to break down the project into manageable tasks and a Gantt chart to schedule and track these tasks. These help ensure effective project management and provide a clear visual timeline to monitor progress, leading to better organization, resource allocation, and timely project completion. The development tools for the project should also be decided, including programming languages, frameworks, and AI model libraries. It also involves system design, where it defines functional and non-functional requirements, creates use case diagrams and use case descriptions to outline system interactions, designs interface flow diagrams to map user interactions, and develops user interface prototype designs to visualize the final product.

3.2.3 Phase 3: Sprint Execution

The Sprint Execution phase focuses on implementing the features defined during Sprint Planning. Development activities are initiated by working on both backend and frontend components. Backend development includes creating modules for account management, payment handling, support ticketing, and data analytics. These backend services are the foundation that supports secure transactions, user management, and operational reporting for both drivers and parking operators. Meanwhile, frontend development is carried out to build intuitive and responsive mobile application interfaces. Separate but complementary interfaces are designed for drivers and parking operators, ensuring that each user group can access the functionalities they require. Besides that, user experience and usability are prioritized to enhance system performance and promote widespread adoption.

Another part of this phase includes the integration of multimodal AI models for license plate and vehicle attribute recognition. The system incorporates AI models to automatically identify vehicles and activate auto-payment. Significant efforts are made to ensure that the AI components are robust and capable of operating under real-world conditions. Prompt engineering techniques are applied to optimize model performance, ensuring reliable operation under real-world conditions. Data preprocessing, model evaluation, and backend integration further strengthen the AI component, supporting accurate recognition and smooth automated parking transactions. Moreover, Daily Scrum meetings are conducted in this phase to monitor development progress, identify and resolve any obstacles, and adjust task priorities as necessary. Even as a solo developer, brief daily reflections are held to assess completed work, plan upcoming tasks, and ensure that the Sprint remains on track. Throughout the Sprint, the Sprint Backlog is continuously updated to reflect the current status of tasks, allowing for better tracking, transparency, and management of the development workflow.

3.2.4 Phase 4: Review and Retrospective

At the end of each Sprint, a Sprint Review is conducted to evaluate the progress made and demonstrate the completed features, where the key functionalities are presented to gather feedback. Feedback may be collected from the supervising lecturer acting as the Product Owner, project moderator during the presentation, or from mock users simulating the roles of drivers and parking operators. This feedback is crucial for validating the system’s functionality and ensuring that the development remains aligned with user needs and project objectives.

Following the Sprint Review, a Sprint Retrospective is held to reflect on the overall development process. During the retrospective, the project examines what aspects of the Sprint went well, what challenges were encountered, and what areas require improvement. This continuous reflection helps identify workflow obstacles and shape better practices for future Sprints. Action items are documented at the end of each retrospective session. These may include steps such as improving testing procedures, enhancing user interface designs, optimizing AI model performance, or refining project

management approaches. The lessons learned and adjustments made during this phase contribute significantly to the continuous improvement of the system throughout the project lifecycle.

3.2.5 Phase 5: Finalization and Release

In the Finalization and Release phase, the primary focus is on completing all necessary testing and preparing the system for deployment. Comprehensive system testing is conducted, covering functional tests to verify the correct operation of features, usability tests to ensure a smooth and intuitive user experience, and performance tests to evaluate the accuracy and robustness of the multimodal AI models for license plate and vehicle attribute recognition. Any issues identified during testing are addressed immediately to ensure a refined and reliable system.

Once testing is completed, the mobile application prototype is finalized and deployed, ready for demonstration and evaluation purposes. Alongside the deployment, essential documentation such as user guides, technical documentation, and the final project report is prepared to support users and stakeholders in understanding and operating the system. These documents serve to enhance the project's professionalism and completeness.

A final presentation and system demonstration are conducted to showcase the project’s outcomes. This provides an opportunity to highlight key features, demonstrate the effectiveness of the AI models, and explain how the system meets the original project objectives. Other than software deliverables, research findings from developing and evaluating the License Plate and Vehicle Attribute Recognition models are included in the final report. This ensures that both the practical implementation and research findings are documented and presented as part of the project's completion.

3.2.6 Justification for Scrum Methodology

Scrum is selected as the methodology for this project because it provides a flexible and structured approach that suits the project's needs. Scrum’s iterative approach enables the quick development of key modules while managing the growing complexity of integrating multimodal AI models. Since fine-tuning and

testing the AI components often bring new challenges, using Scrum helps the team adapt quickly during development.

Furthermore, Scrum's focus on incremental delivery ensures that a working prototype is always ready for demonstration and feedback. This continuous delivery of functional features helps detect issues early and align the system more closely with stakeholder expectations. Each Sprint produces tangible outcomes that can be evaluated and improved, enhancing the quality and robustness of the final system.

Another key advantage of Scrum is its flexibility in accommodating changes. Since the performance of AI models might require frequent adjustments based on real-world test results, and the need to refine app functionalities to meet user feedback, Scrum's flexibility allows these changes to be integrated smoothly without disrupting overall project progress. This makes Scrum a highly effective and practical choice for managing the technical and research needs of the project.

3.3 Project Planning and Scheduling

In order to plan and organize the project effectively, project planning tools such as the WBS and Gantt chart will be used. The WBS will help break down the overall project into smaller, manageable tasks to ensure that all necessary work is clearly defined. The Gantt chart will be used to schedule these tasks over time, providing a visual timeline that helps track progress and ensure timely completion. Together, these tools will support better organization, resource management, and project control.

3.3.1 Work Breakdown Structure (WBS)

0.0 Development for License Plate and Vehicle Attribute Recognition Using Multimodal AI Models

1.0 Project Initialization

- 1.1 Identify the Importance of the Project
- 1.2 Identify Problems
  - 1.2.1 Analyze Problem Background
  - 1.2.2 List Problem Statements

1.3 Define Project Aim and Objectives

1.4 Propose Project Solution and Approach

1.5 Define Project Scope and Limitations

1.6 Conduct Literature Review

1.6.1 Research on License Plate and Vehicle Attribute Recognition Systems

1.6.2 Research on Existing Parking Systems and Applications

1.7 Identify Stakeholders

1.8 Requirement Gathering and Elicitation

1.8.1 Questionnaire

1.8.1.1 Plan Questionnaire

1.8.1.2 Request and Obtain Ethical Clearance

1.8.1.3 Distribute Questionnaire

1.8.1.4 Analyze and Interpret Data Collected

2.0 Planning and Design

2.1 Select Software Development Methodology

2.2 Schedule Project Activities

2.2.1 Create Work Breakdown Structure (WBS)

2.2.2 Create Gantt Chart

2.3 Define Functional and Non-Functional Requirements

2.4 Create Use Case Diagrams and Descriptions

2.5 Design Flow Diagrams

3.0 Development and Sprints

3.1 Sprint 1

3.1.1 Sprint Planning

3.1.2 Sprint Execution

3.1.2.1 AI Recognition and Classification Research

3.1.2.1.1 License Plate and Vehicle Attribute Recognition

3.1.2.1.1.1 Recognition Segmentation

3.1.3 Sprint Testing

3.1.3.1 Perform Bug Fixing and Regression Testing

3.1.4 Sprint Review

3.3.2 Gantt Chart

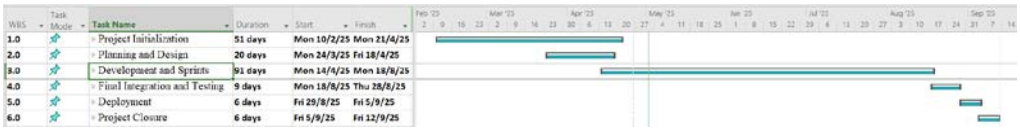


Figure 3.1: Overview of project timeline

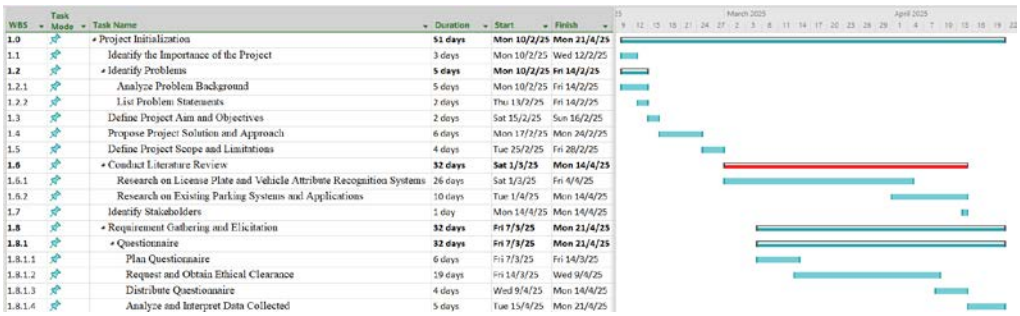
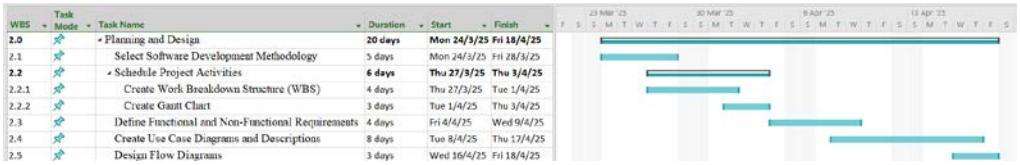
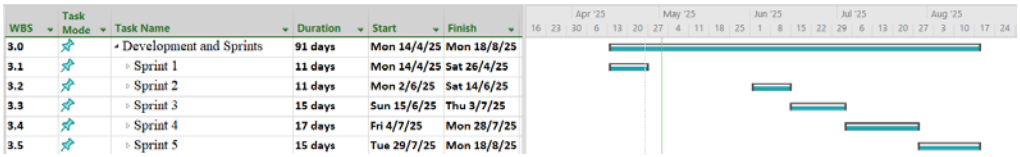


Figure 3.2: Project initiation timeline



WBS	Task Mode	Task Name	Duration	Start	Finish
2.0		Planning and Design	20 days	Mon 24/3/25	Fri 18/4/25
2.1		Select Software Development Methodology	5 days	Mon 24/3/25	Fri 28/3/25
2.2		Schedule Project Activities	6 days	Thu 27/3/25	Thu 3/4/25
2.2.1		Create Work Breakdown Structure (WBS)	4 days	Thu 27/3/25	Tue 1/4/25
2.2.2		Create Gantt Chart	3 days	Tue 1/4/25	Thu 3/4/25
2.3		Define Functional and Non-Functional Requirements	4 days	Fri 4/4/25	Wed 9/4/25
2.4		Create Use Case Diagrams and Descriptions	8 days	Tue 8/4/25	Thu 17/4/25
2.5		Design Flow Diagrams	3 days	Wed 16/4/25	Fri 18/4/25

Figure 3.3: Planning and design timeline

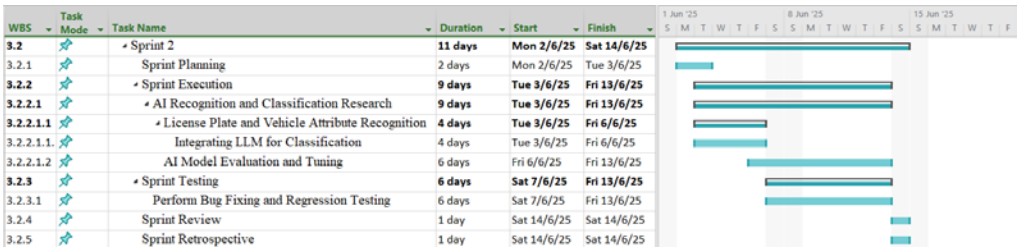


WBS	Task Mode	Task Name	Duration	Start	Finish
3.0		Development and Sprints	91 days	Mon 14/4/25	Mon 18/8/25
3.1		Sprint 1	11 days	Mon 14/4/25	Sat 26/4/25
3.2		Sprint 2	11 days	Mon 2/6/25	Sat 14/6/25
3.3		Sprint 3	15 days	Sun 15/6/25	Thu 3/7/25
3.4		Sprint 4	17 days	Fri 4/7/25	Mon 28/7/25
3.5		Sprint 5	15 days	Tue 29/7/25	Mon 18/8/25

Figure 3.4: Development and Sprints Timeline Overview

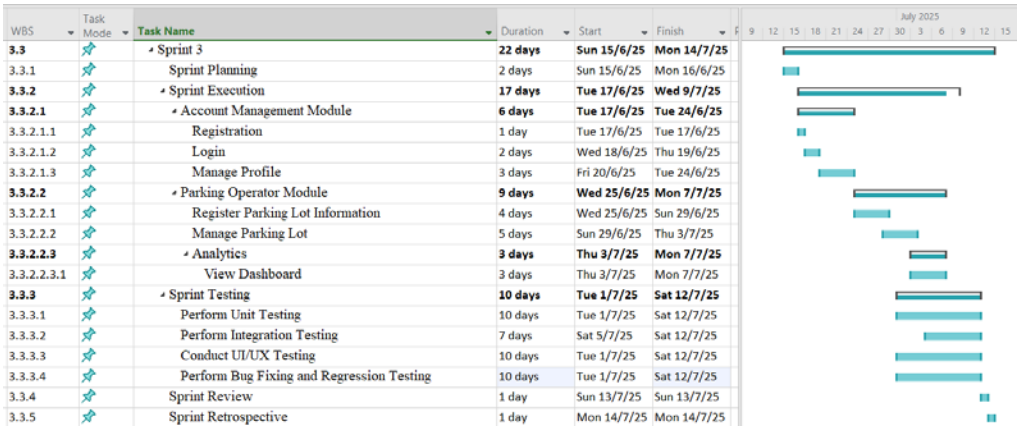
WBS	Task Mode	Task Name	Duration	Start	Finish
3.1		Sprint 1	11 days	Mon 14/4/25	Sat 26/4/25
3.1.1		Sprint Planning	2 days	Mon 14/4/25	Tue 15/4/25
3.1.2		Sprint Execution	9 days	Tue 15/4/25	Fri 25/4/25
3.1.2.1		AI Recognition and Classification Research	9 days	Tue 15/4/25	Fri 25/4/25
3.1.2.1.1		License Plate and Vehicle Attribute Recognition	9 days	Tue 15/4/25	Fri 25/4/25
3.1.2.1.1.		Recognition Segmentation	9 days	Tue 15/4/25	Fri 25/4/25
3.1.3		Sprint Testing	9 days	Tue 15/4/25	Fri 25/4/25
3.1.3.1		Perform Bug Fixing and Regression Testing	6 days	Fri 18/4/25	Fri 25/4/25
3.1.4		Sprint Review	1 day	Sat 26/4/25	Sat 26/4/25
3.1.5		Sprint Retrospective	1 day	Sat 26/4/25	Sat 26/4/25

Figure 3.5: Sprint 1 timeline



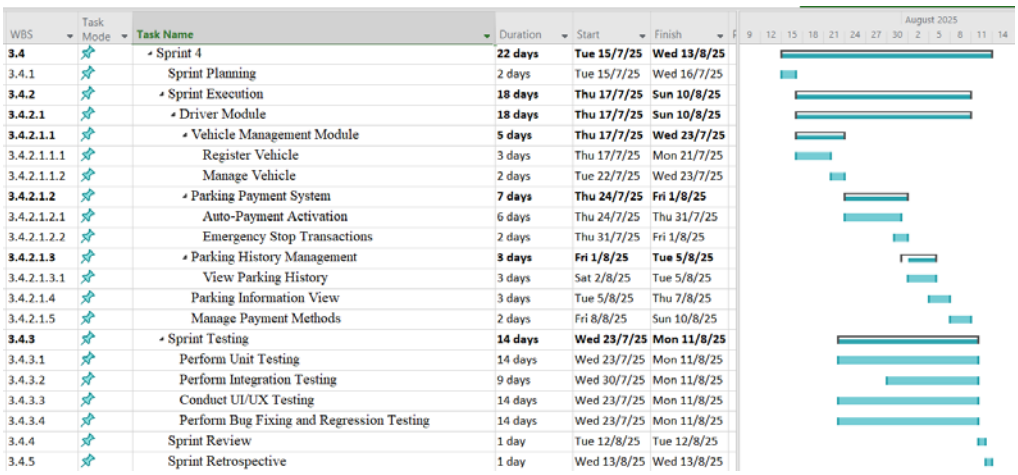
WBS	Task Mode	Task Name	Duration	Start	Finish
3.2		Sprint 2	11 days	Mon 2/6/25	Sat 14/6/25
3.2.1		Sprint Planning	2 days	Mon 2/6/25	Tue 3/6/25
3.2.2		Sprint Execution	9 days	Tue 3/6/25	Fri 13/6/25
3.2.2.1		AI Recognition and Classification Research	9 days	Tue 3/6/25	Fri 13/6/25
3.2.2.1.1		License Plate and Vehicle Attribute Recognition	4 days	Tue 3/6/25	Fri 6/6/25
3.2.2.1.1.		Integrating LLM for Classification	4 days	Tue 3/6/25	Fri 6/6/25
3.2.2.1.2		AI Model Evaluation and Tuning	6 days	Fri 6/6/25	Fri 13/6/25
3.2.3		Sprint Testing	6 days	Sat 7/6/25	Fri 13/6/25
3.2.3.1		Perform Bug Fixing and Regression Testing	6 days	Sat 7/6/25	Fri 13/6/25
3.2.4		Sprint Review	1 day	Sat 14/6/25	Sat 14/6/25
3.2.5		Sprint Retrospective	1 day	Sat 14/6/25	Sat 14/6/25

Figure 3.6: Sprint 2 timeline



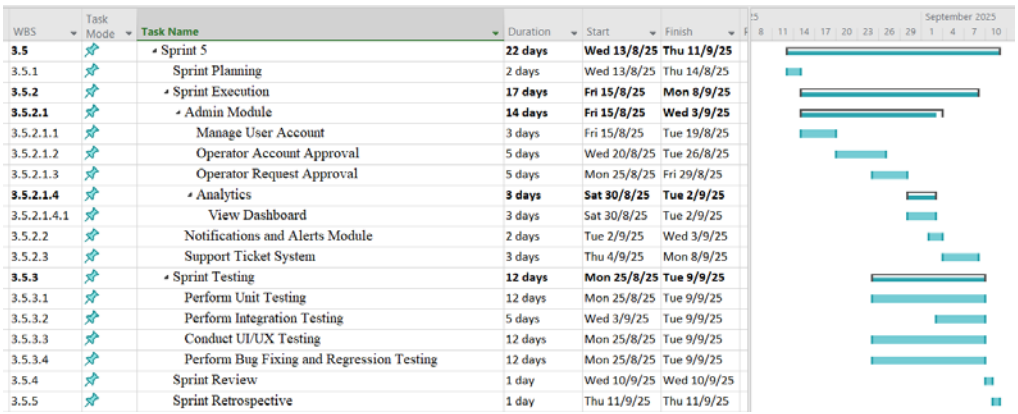
WBS	Task Mode	Task Name	Duration	Start	Finish
3.3		Sprint 3	22 days	Sun 15/6/25	Mon 14/7/25
3.3.1		Sprint Planning	2 days	Sun 15/6/25	Mon 16/6/25
3.3.2		Sprint Execution	17 days	Tue 17/6/25	Wed 9/7/25
3.3.2.1		Account Management Module	6 days	Tue 17/6/25	Tue 24/6/25
3.3.2.1.1		Registration	1 day	Tue 17/6/25	Tue 17/6/25
3.3.2.1.2		Login	2 days	Wed 18/6/25	Thu 19/6/25
3.3.2.1.3		Manage Profile	3 days	Fri 20/6/25	Tue 24/6/25
3.3.2.2		Parking Operator Module	9 days	Wed 25/6/25	Mon 7/7/25
3.3.2.2.1		Register Parking Lot Information	4 days	Wed 25/6/25	Sun 29/6/25
3.3.2.2.2		Manage Parking Lot	5 days	Sun 29/6/25	Thu 3/7/25
3.3.2.2.3		Analytics	3 days	Thu 3/7/25	Mon 7/7/25
3.3.2.2.3.1		View Dashboard	3 days	Thu 3/7/25	Mon 7/7/25
3.3.3		Sprint Testing	10 days	Tue 1/7/25	Sat 12/7/25
3.3.3.1		Perform Unit Testing	10 days	Tue 1/7/25	Sat 12/7/25
3.3.3.2		Perform Integration Testing	7 days	Sat 5/7/25	Sat 12/7/25
3.3.3.3		Conduct UI/UX Testing	10 days	Tue 1/7/25	Sat 12/7/25
3.3.3.4		Perform Bug Fixing and Regression Testing	10 days	Tue 1/7/25	Sat 12/7/25
3.3.4		Sprint Review	1 day	Sun 13/7/25	Sun 13/7/25
3.3.5		Sprint Retrospective	1 day	Mon 14/7/25	Mon 14/7/25

Figure 3.7: Sprint 3 timeline



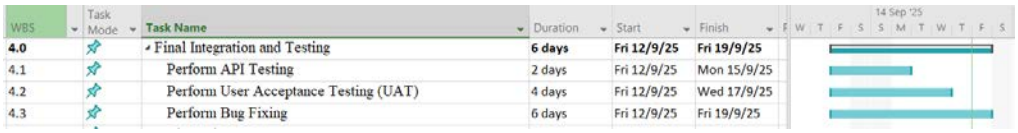
WBS	Task Mode	Task Name	Duration	Start	Finish
3.4		**Sprint 4**	**22 days**	**Tue 15/7/25**	**Wed 13/8/25**
3.4.1		Sprint Planning	2 days	Tue 15/7/25	Wed 16/7/25
3.4.2		**Sprint Execution**	**18 days**	**Thu 17/7/25**	**Sun 10/8/25**
3.4.2.1		**Driver Module**	**18 days**	**Thu 17/7/25**	**Sun 10/8/25**
3.4.2.1.1		**Vehicle Management Module**	**5 days**	**Thu 17/7/25**	**Wed 23/7/25**
3.4.2.1.1.1		Register Vehicle	3 days	Thu 17/7/25	Mon 21/7/25
3.4.2.1.1.2		Manage Vehicle	2 days	Tue 22/7/25	Wed 23/7/25
3.4.2.1.2		**Parking Payment System**	**7 days**	**Thu 24/7/25**	**Fri 1/8/25**
3.4.2.1.2.1		Auto-Payment Activation	6 days	Thu 24/7/25	Thu 31/7/25
3.4.2.1.2.2		Emergency Stop Transactions	2 days	Thu 31/7/25	Fri 1/8/25
3.4.2.1.3		**Parking History Management**	**3 days**	**Fri 1/8/25**	**Tue 5/8/25**
3.4.2.1.3.1		View Parking History	3 days	Sat 2/8/25	Tue 5/8/25
3.4.2.1.4		Parking Information View	3 days	Tue 5/8/25	Thu 7/8/25
3.4.2.1.5		Manage Payment Methods	2 days	Fri 8/8/25	Sun 10/8/25
3.4.3		**Sprint Testing**	**14 days**	**Wed 23/7/25**	**Mon 11/8/25**
3.4.3.1		Perform Unit Testing	14 days	Wed 23/7/25	Mon 11/8/25
3.4.3.2		Perform Integration Testing	9 days	Wed 30/7/25	Mon 11/8/25
3.4.3.3		Conduct UI/UX Testing	14 days	Wed 23/7/25	Mon 11/8/25
3.4.3.4		Perform Bug Fixing and Regression Testing	14 days	Wed 23/7/25	Mon 11/8/25
3.4.4		Sprint Review	1 day	Tue 12/8/25	Tue 12/8/25
3.4.5		Sprint Retrospective	1 day	Wed 13/8/25	Wed 13/8/25

Figure 3.8: Sprint 4 timeline



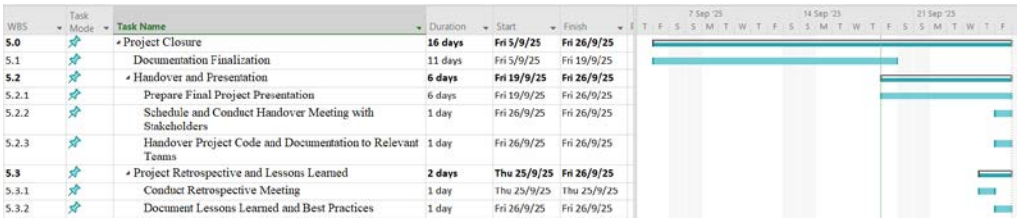
WBS	Task Mode	Task Name	Duration	Start	Finish
3.5		**Sprint 5**	**22 days**	**Wed 13/8/25**	**Thu 11/9/25**
3.5.1		Sprint Planning	2 days	Wed 13/8/25	Thu 14/8/25
3.5.2		**Sprint Execution**	**17 days**	**Fri 15/8/25**	**Mon 8/9/25**
3.5.2.1		**Admin Module**	**14 days**	**Fri 15/8/25**	**Wed 3/9/25**
3.5.2.1.1		Manage User Account	3 days	Fri 15/8/25	Tue 19/8/25
3.5.2.1.2		Operator Account Approval	5 days	Wed 20/8/25	Tue 26/8/25
3.5.2.1.3		Operator Request Approval	5 days	Mon 25/8/25	Fri 29/8/25
3.5.2.1.4		**Analytics**	**3 days**	**Sat 30/8/25**	**Tue 2/9/25**
3.5.2.1.4.1		View Dashboard	3 days	Sat 30/8/25	Tue 2/9/25
3.5.2.2		Notifications and Alerts Module	2 days	Tue 2/9/25	Wed 3/9/25
3.5.2.3		Support Ticket System	3 days	Thu 4/9/25	Mon 8/9/25
3.5.3		**Sprint Testing**	**12 days**	**Mon 25/8/25**	**Tue 9/9/25**
3.5.3.1		Perform Unit Testing	12 days	Mon 25/8/25	Tue 9/9/25
3.5.3.2		Perform Integration Testing	5 days	Wed 3/9/25	Tue 9/9/25
3.5.3.3		Conduct UI/UX Testing	12 days	Mon 25/8/25	Tue 9/9/25
3.5.3.4		Perform Bug Fixing and Regression Testing	12 days	Mon 25/8/25	Tue 9/9/25
3.5.4		Sprint Review	1 day	Wed 10/9/25	Wed 10/9/25
3.5.5		Sprint Retrospective	1 day	Thu 11/9/25	Thu 11/9/25

Figure 3.9: Sprint 5 timeline



WBS	Task Mode	Task Name	Duration	Start	Finish
4.0		**Final Integration and Testing**	**6 days**	**Fri 12/9/25**	**Fri 19/9/25**
4.1		Perform API Testing	2 days	Fri 12/9/25	Mon 15/9/25
4.2		Perform User Acceptance Testing (UAT)	4 days	Fri 12/9/25	Wed 17/9/25
4.3		Perform Bug Fixing	6 days	Fri 12/9/25	Fri 19/9/25

Figure 3.10: Final Integration and Testing Timeline



WBS	Task Mode	Task Name	Duration	Start	Finish
5.0		**Project Closure**	**16 days**	**Fri 5/9/25**	**Fri 26/9/25**
5.1		Documentation Finalization	11 days	Fri 5/9/25	Fri 19/9/25
5.2		**Handover and Presentation**	**6 days**	**Fri 19/9/25**	**Fri 26/9/25**
5.2.1		Prepare Final Project Presentation	6 days	Fri 19/9/25	Fri 26/9/25
5.2.2		Schedule and Conduct Handover Meeting with Stakeholders	1 day	Fri 26/9/25	Fri 26/9/25
5.2.3		Handover Project Code and Documentation to Relevant Teams	1 day	Fri 26/9/25	Fri 26/9/25
5.3		**Project Retrospective and Lessons Learned**	**2 days**	**Thu 25/9/25**	**Fri 26/9/25**
5.3.1		Conduct Retrospective Meeting	1 day	Thu 25/9/25	Thu 25/9/25
5.3.2		Document Lessons Learned and Best Practices	1 day	Fri 26/9/25	Fri 26/9/25

Figure 3.11: Project Closure Timeline

3.4 Development Tools and Techniques

3.4.1 Tools and IDEs

3.4.1.1 Enterprise Architecture

Enterprise Architecture (EA) is a conceptual blueprint that shapes the structure and operation of an organization. It helps to determine how an organization can effectively achieve its current and future goals (Alexander Gillis, n.d.). In this project, Enterprise Architecture will be used to design and organize the system in a clear and structured way. It will guide how different parts of the system connect and work together. To support this, charts such as use case diagrams and flowcharts will be created. Use case diagrams will show how users interact with the system, while flowcharts will map out the steps and processes within the system. This approach will make it easier to plan, develop, and manage the system efficiently.

3.4.1.2 Visual Studio Code

Visual Studio Code (VS Code) is a lightweight but powerful source code editor developed by Microsoft. It supports a wide range of programming languages and comes with built-in features such as debugging, version control integration, and intelligent code completion (Visual Studio Code, n.d.). In this project, Visual Studio Code will be used as the main integrated development environment (IDE) for writing, editing, and managing code for both the frontend and backend components. Additionally, VS Code offers many extensions that assist in coding by providing syntax highlighting, code formatting, snippets, and AI-powered suggestions, which help to improve development efficiency and code quality.

3.4.1.3 Git and GitHub

Git is a version control system that allows developers to track changes in their codebase over time. It helps manage different versions of the project, making it easy to identify and fix issues. On the other hand, GitHub is a cloud-based platform that hosts Git repositories and provides a space for developers to collaborate on code by sharing and managing their repositories. In this project, I will use Git and GitHub to track changes, manage versions, and maintain a

history of the project to easily reference older versions of the code when needed. I will also use branches to test new functionalities without affecting the main codebase. Additionally, GitHub will act as a backup, ensuring the project is securely stored and accessible from anywhere. These tools will help streamline development, improve organization, and provide a safe, collaborative environment for the project in the future.

3.4.2 Languages

3.4.2.1 Python

Python is a flexible and powerful language used widely in machine learning and AIIn this project, it is used mainly for prompt engineering to interact with multimodal models, enabling recognition of license plates and vehicle attributes. It also supports essential tasks such as data preprocessing, model evaluation, and API integrations to ensure smooth system functionality.

3.4.2.2 HTML, CSS, JavaScript

HTML, CSS, and JavaScript are the basic tools for web development. These technologies will be used for developing the frontend of the parking payment system, ensuring that the user interface (UI) is visually appealing, functional, and interactive across both mobile and desktop platforms. HTML will provide the basic structure of the web pages, defining elements such as headers, forms, and buttons. CSS will be used for styling, ensuring that the application has a responsive design that adapts to different screen sizes and devices. JavaScript will bring interactivity to the frontend, handling tasks like form validation, dynamic content updates, and user interactions.

3.4.2.3 SQL

SQL (Structured Query Language) is a standard programming language used to manage and manipulate relational databases. It allows users to perform tasks such as querying data, inserting, updating, and deleting records in a database. In this project, SQL will be used to manage the data associated with the parking system. It will store user information and manage parking records. SQL will also have efficient data retrieval and querying, like fetching a user’s transaction

history or checking parking space availability. SQL will ensure data integrity, providing a secure and consistent way to store and manipulate the system's information.

3.4.3 Software Frameworks

3.4.3.1 React Native

In this project, React Native is used as the primary frontend framework for developing both mobile and web applications using a shared codebase. React Native enables the development of mobile applications for Android and iOS using JavaScript and React, while maintaining native-like performance and interface consistency. To extend the application's reach to web browsers, the project also integrates React Native for Web, which allows React Native components to be rendered on the web through React DOM. This approach ensures that the application maintains a consistent user interface and functionality across both mobile and web platforms without the need to build separate applications. By adopting this framework, the project benefits from reduced development time, easier maintenance, and a cohesive user experience across multiple platforms. React Native's reusable components and modular structure also enhance scalability and performance, making it an ideal choice for cross-platform development.

3.4.3.2 Laravel

Laravel is an open-source PHP web framework that follows the MVC architecture and is known for its elegant syntax, scalability, and developer-friendly features. It includes built-in tools for database management, authentication, routing, and security, making it ideal for backend development. I will use Laravel for the backend of the parking system because of its ability to efficiently handle user accounts, vehicle registration, payment transactions, and data processing. Its also integrates with PostgreSQL to ensure reliable data management.

3.4.3.3 Expo

Expo is an open-source framework and platform built around React Native that makes it easier to develop cross-platform applications. It includes a range of tools and services to help developers build, deploy, and quickly test apps for iOS, Android, and the web using a single JavaScript or TypeScript codebase. Expo includes features like a built-in development server, over-the-air (OTA) updates, and pre-configured native modules, such as camera, notifications, and biometrics, which reduces complex native code setup (Kwiatkowski, 2024). In this project, Expo is used alongside React Native and React Native for Web to streamline development across both mobile and web platforms. It simplifies testing and debugging across devices, supports fast iteration through its development tools, and provides easy integration with native features without requiring manual configuration of native code. Overall, Expo enhances development efficiency and ensures a smooth deployment process across all supported platforms.

3.4.4 Database

3.4.4.1 PostgreSQL

PostgreSQL is an open-source, relational database management system known for its scalability, reliability, and advanced data handling capabilities. For the LPR and vehicle attribute recognition in a parking system, PostgreSQL is an ideal choice due to its ability to efficiently manage large volumes of structured data, handle complex queries, and store interrelated data like vehicle registration details and transaction histories. Its support for advanced data types like JSON is useful for storing vehicle attributes, while its strong security features ensure data protection. Additionally, PostgreSQL integrates well with Laravel, making it suitable for building and managing the backend of the parking system and its AI components.

3.5 Summary

This chapter outlines the methodology, WBS, Gantt chart, and tools and technologies used in the development of the parking system project. The Scrum methodology is adopted for iterative development and continuous feedback through short sprints, ensuring flexibility and efficient task management. A WBS was created to break the project into manageable tasks, while a Gantt chart visually tracks the project timeline, ensuring timely completion. For the development process, Enterprise Architecture (EA) guides the overall system design, Visual Studio Code (VS Code) serves as the primary integrated development environment (IDE), and Git with GitHub is used for version control and team collaboration. The frontend will be developed using React Native, which supports both mobile and web platforms through React Native for Web. Expo is used to facilitate cross-platform development, testing, and access to native device features. The backend will be powered by Laravel, integrated with PostgreSQL for secure and efficient data management, while Python will handle license plate recognition and vehicle attribute identification.

CHAPTER 4
PROJECT SPECIFICATION

4.1 Introduction

This chapter presents the project specification, which was developed based on the analysis of data collected through both primary and secondary research. Using the insights gathered, the system requirements are defined, followed by the construction of requirement specifications, use case diagrams, interface flow diagrams, and initial user interface designs. A preliminary experiment on vehicle detection and segmentation is also included to evaluate the feasibility of the proposed approach. Altogether, this chapter provides a comprehensive overview of the system's expected functionalities, design structure, and technical foundation.

4.2 Fact Finding

This project uses both primary and secondary data collection methods to support the system analysis and development phases. For primary data collection, an online questionnaire was created and distributed through Google Forms to gather insights from Malaysian vehicle owners and drivers, who are the main target users of the proposed system. The questionnaire aims to gather insights on vehicle usage, commuting habits, challenges faced by road users regarding current parking payment systems, and user perceptions of automated parking payment systems using vehicle recognition technology. It is also designed to understand user expectations for a multimodal AI-based vehicle recognition system.

For secondary data collection, a literature review explored existing LPR technologies, vehicle attribute recognition methods, and their applications in automated parking systems. Sources included journal articles, research papers, case studies, and technical documentation on both traditional and AI-driven approaches, such as machine learning, deep learning, and LLMs. Additionally, commercial parking solutions were reviewed to benchmark current systems. Software development methodologies and frameworks were also examined to

guide the system’s implementation. This review provided insights into the proposed system's current capabilities, limitations, and opportunities.

Both of these data collection methods were important in defining the scope and direction of the project. The primary data highlighted user preferences, experiences, and expectations regarding parking payment systems, while the secondary data provided insights into existing technologies and AI-based vehicle recognition methods. These findings will guide the development to be more practical and user-focused in addressing common issues of current parking systems.

4.2.1 Responses on Google Form Questionnaire Survey

The questionnaire was distributed over social media platforms and messaging apps to ensure a wide reach and cost-effective data collection. A total of 31 responses were collected. The questionnaire was divided into four sections, consisting of 21 questions in total.

4.2.1.1 Section A: Demographics of Respondents

In this section, demographic information is collected to understand the background of the respondents, including factors such as age, gender, and vehicle ownership. This helps provide context for analyzing the responses and identifying any trends or patterns based on different user groups.

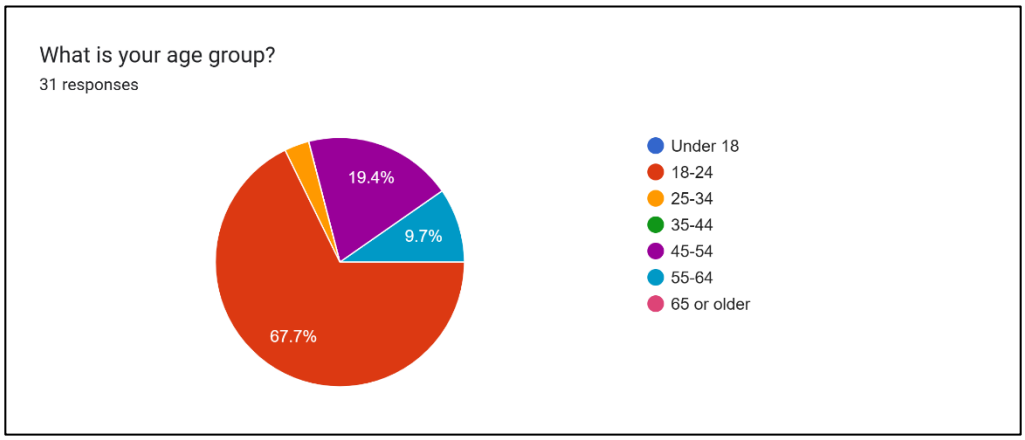


Figure 4.1: Pie Chart of Respondents' Age Group

The first question of the questionnaire collects the age of the respondents. As shown in Figure 4.1, the majority of respondents, which is 67.7%, fall within the 18 to 24 age group. The second largest group is aged between 45 and 54, making up 19.4% of the total responses. Meanwhile, the 55 to 64 and 25 to 34 age groups have the lowest representation, with only three (9.7%) and one (3.2%) respondents respectively. No responses were recorded from the under-18 or 35 to 44 age categories. Overall, the data shows that all respondents are aged 18 and above.

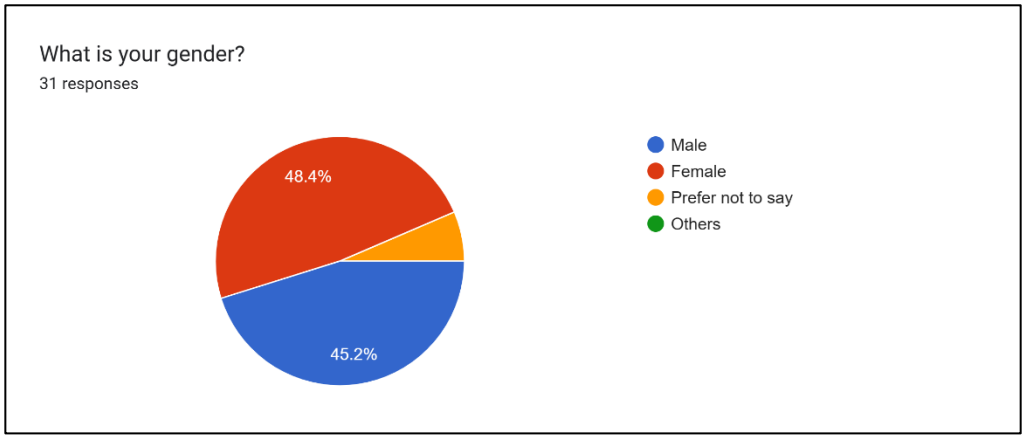


Figure 4.2: Pie Chart of Respondents' Gender

The majority of respondents are female, making up 48.4% of the total. Meanwhile, 45.2% of respondents are male, and 6.5% preferred not to disclose their gender. No respondents selected the "Others" option. Overall, the results show a nearly equal gender distribution among respondents.

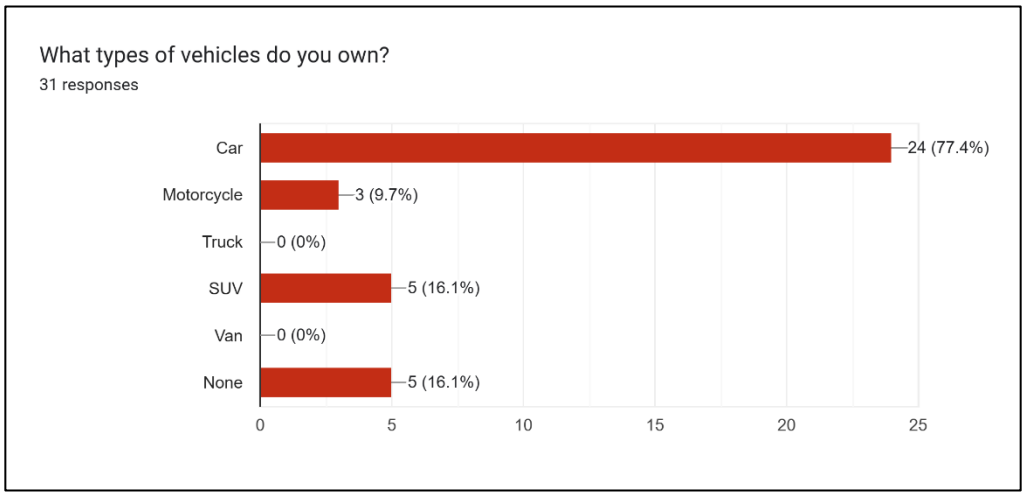


Figure 4.3: Bar Chart of Types of Vehicles Owned by Respondents

The majority of respondents own a car, making up 77.4% of the total. SUVs are the next most commonly owned vehicle at 16.1%. Meanwhile, 16.1% of respondents also reported that they do not own any vehicle. However, they are still relevant to the survey, as they may use a family member's vehicle. Motorcycles are the least owned type among the options provided, with only 9.7% of respondents. Additionally, there were no respondents who own a truck or van, as both categories received 0%. This suggests that private cars are the most preferred mode of transportation among the respondents.

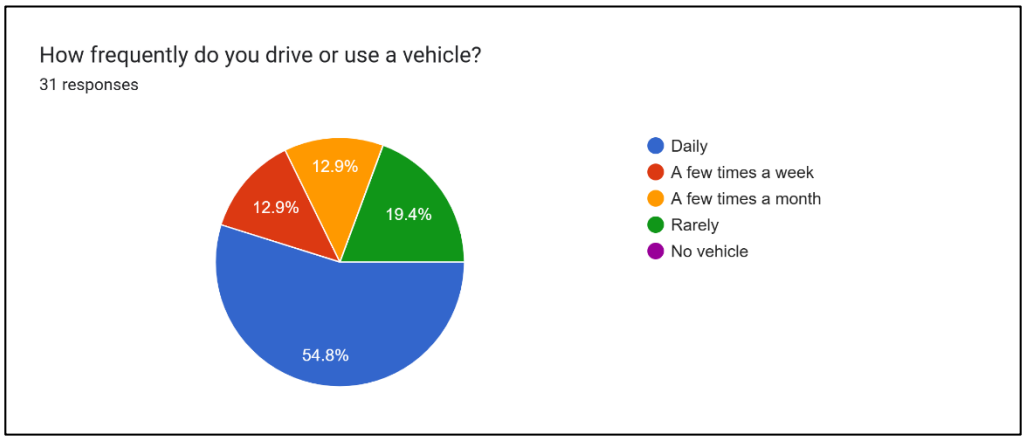


Figure 4.4: Pie Chart of Respondents' Frequency of Driving

The majority of respondents, which is 54.8%, reported that they drive daily, showing a high level of reliance on their vehicles for transportation. 19.4% of respondents drive rarely, suggesting occasional use of their vehicle. Meanwhile, 12.9% of respondents drive a few times a month, while another 12.9% drive a few times a week, showing moderate but less frequent vehicle use. These lower frequencies may be due to the higher number of younger respondents in the survey, such as university students who live in hostels without a car but drive when they return to their hometowns. Lastly, no respondents indicated that they do not own a vehicle, showing that all participants in this survey drive or have access to a vehicle. This chart highlights varying levels of vehicle use among the respondents, with daily driving being the most common.

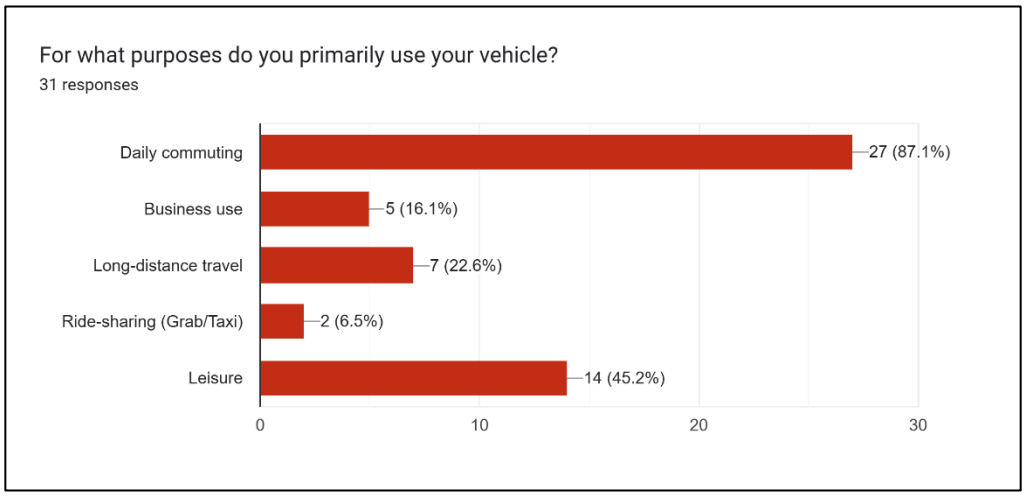


Figure 4.5: Bar Chart of Respondents' Purpose for Using their Vehicle

The most common purpose for using a vehicle is daily commuting, which is 87.1% of respondents. It indicated that the majority of respondents rely on their vehicle for regular travel to work, school, or other daily activities. 45.2% of respondents use their vehicle for leisure, suggesting that a significant portion of respondents also use their vehicle for recreational purposes. 22.6% of respondents use their vehicle for long-distance travel, while 16.1% use it for business purposes. Finally, 6.5% of respondents reported using their vehicle for ride-sharing services such as Grab or Taxi, showing a smaller but relevant portion of respondents using their vehicle in the ride-hailing market. This suggests that most respondents use their vehicles primarily for commuting, with leisure and travel also being important reasons.

4.2.1.2 Section B: Difficulties in the Existing Parking Payment System

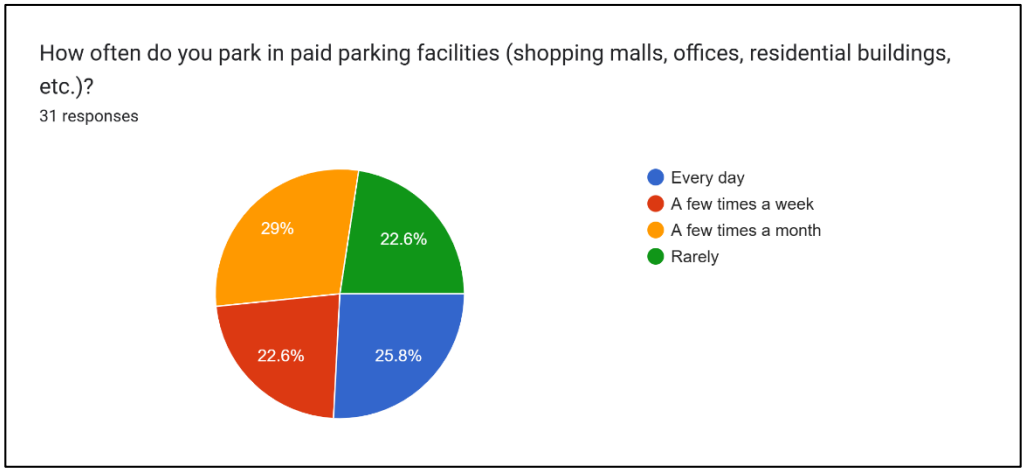


Figure 4.6: Pie Chart of Respondents' Frequency of Using Paid Parking Facilities

The largest group, with 29% of total responses, reported using paid parking a few times a month, indicating occasional use of such facilities. 25.8% of respondents use paid parking every day, indicating that a significant portion of respondents likely rely on paid parking for daily commuting or work-related travel. 22.6% use paid parking a few times a week, suggesting moderate usage of such facilities. The remaining 22.6% of respondents reported using paid parking rarely, showing that for some, paid parking is not a regular necessity. This distribution highlights that paid parking is commonly used, especially by those who commute daily or have frequent parking needs. The lower usage of paid parking facilities among some respondents may be due to the younger age group, who may not use paid parking as frequently as working adults.

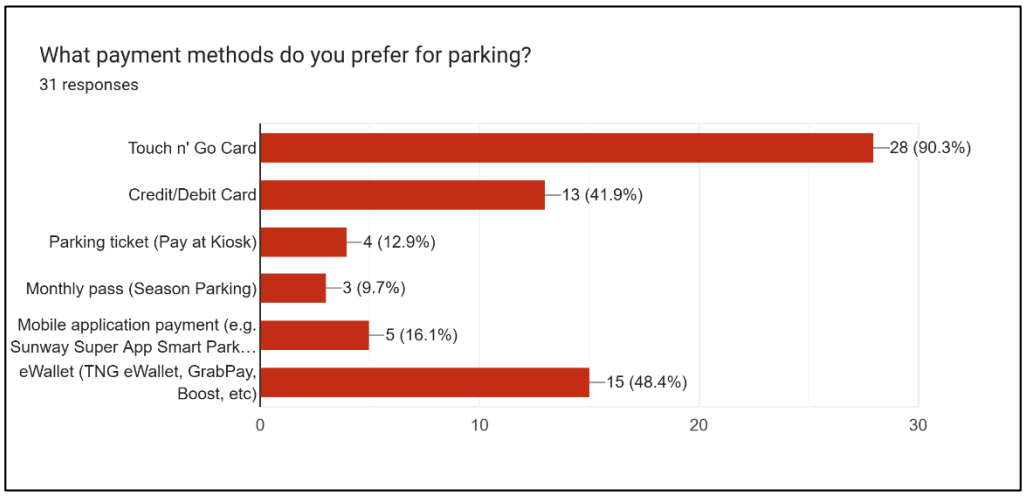


Figure 4.7: Bar Chart of Respondents' Preferred Parking Payment Methods

The majority of respondents, which is 90.3%, prefer using the Touch 'n Go card, making it the most popular method. It is then followed by eWallets like TNG eWallet, GrabPay, and Boost, which are used by 48.4% of respondents. Credit or debit card payments are also fairly common at 41.9%. Other methods include mobile app payments (e.g., Sunway Super App Smart Park) at 16.1%, paying at kiosks using a parking ticket at 12.9%, and monthly passes or season parking at 9.7%. This suggests that digital and contactless payments are more favored over traditional methods like kiosk payments or monthly passes.

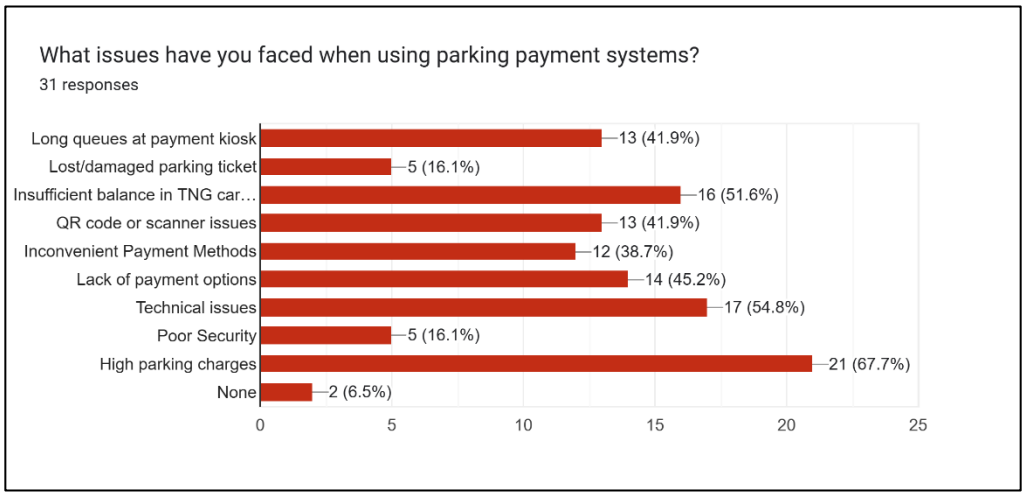


Figure 4.8: Bar Chart of Issues Faced by Respondents with Current Parking Payment Systems

The most frequently reported issue with parking payment systems was high parking charges, which were 67.7% of total responses. This suggests that parking price remains a major concern for users. Technical issues were the second most common problem, affecting 54.8% of respondents and indicating that system reliability is a significant area for improvement. Additionally, 51.6% of participants reported problems due to insufficient balance in their TNG Card or eWallet, highlighting the need for easier ways to check and manage balance. Other commonly faced issues include a lack of payment options at 45.2%, long queues at payment kiosks at 41.9%, QR code or scanner issues at 41.9%, and inconvenient payment methods at 38.7%, all of which suggest both physical and digital aspects of the payment process need refinement. Less frequent issues, such as lost or damaged parking tickets and poor security, were each reported by 16.1% of respondents. Only 6.5% of participants indicated they faced no issues at all, implying that the majority encounter some form of difficulty when using parking payment systems.

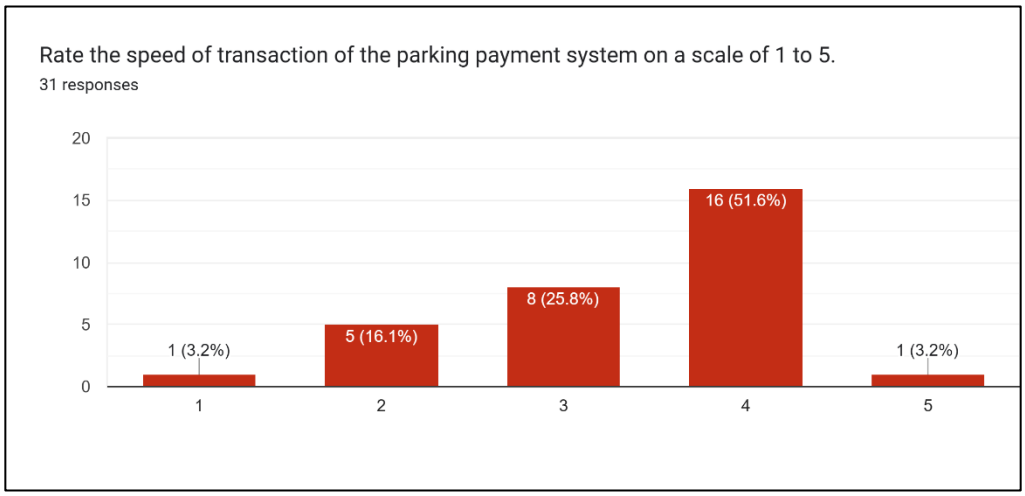


Figure 4.9: Column Chart of Respondents' Rating on Parking Payment Transaction Speed

The majority of respondents, which is 51.6%, rated the transaction speed as 4, indicating that they find the process to be fairly fast and acceptable for daily use. Another 25.8% chose a neutral score of 3, suggesting that while the speed is not problematic, it may not be particularly efficient either. Meanwhile, 16.1% rated the speed as 2, pointing to noticeable delays or inefficiencies in some payment systems. A small portion of respondents rated the speed as 1 (very slow) and 5 (very fast), both at 3.2%. These findings reflect that although a large portion of drivers are relatively satisfied with the transaction speed, there remains a need to optimize the system further. Improving processing speed through automation or better system design could improve the overall parking payment experience for drivers.

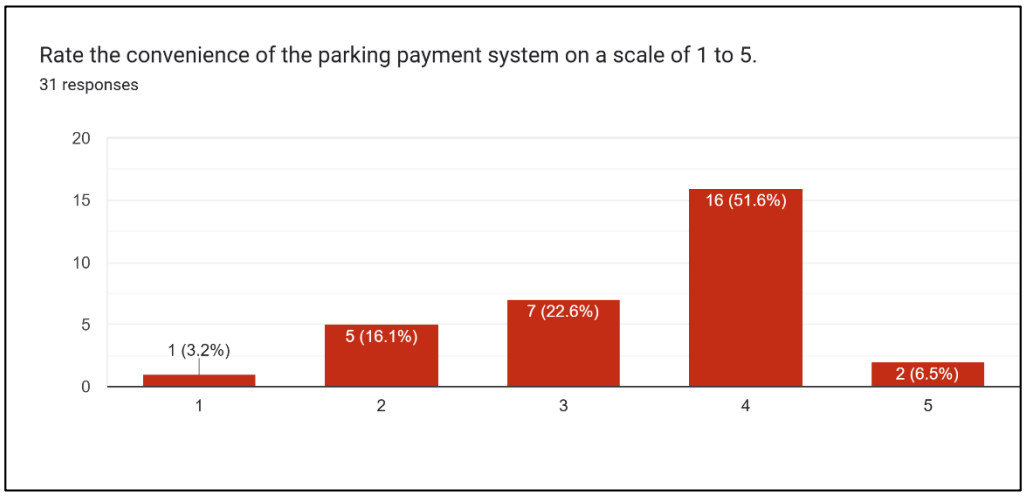


Figure 4.10: Column Chart of Respondents' Rating on Parking Payment Transaction Convenience

The majority of respondents, which is 51.6%, rated the transaction convenience as 4, indicating that they find the payment process somewhat convenient. Following that, 22.6% of respondents chose a neutral score of 3, suggesting that while the transaction is neither particularly convenient nor inconvenient, improvement is still needed. Meanwhile, a smaller portion of 16.1% rated the convenience as 2, which points to some dissatisfaction or challenges faced by respondents with the current system. Only 6.5% of respondents rated the system as a 5, which represents very convenient, indicating that while some drivers find the system convenient, it is not the case for the majority of drivers. Additionally, 3.2% of respondents rated the system as a 1, which means very inconvenient, highlighting that there are still drivers who experience significant difficulties when using the current parking payment system. This suggests that while the majority of drivers find the system overall convenient, there is potential for improvement. Enhancing the system's convenience could help in addressing the concerns of those who rated it as less convenient. Improvements could focus on simplifying the transaction process, reducing wait times, or introducing more user-friendly interfaces.

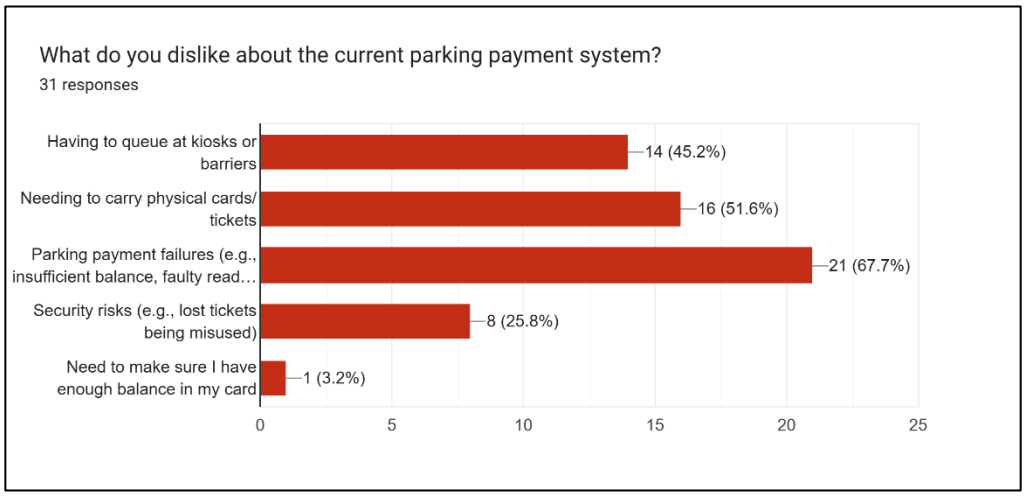


Figure 4.11: Bar Chart of Respondents' Dissatisfaction with Current Parking Payment System

The most common issue reported by 21 respondents (67.7%) is parking payment failures, such as insufficient balance or faulty card readers, which often lead to delays and inconvenience. The second most frequent dissatisfaction, selected by 16 respondents (51.6%), is the need to carry physical cards or tickets. This can be inconvenient for drivers who may forget their cards or have to manage multiple cards for different parking locations. Having to queue at kiosks or barriers was also a major dissatisfaction reported by 14 respondents (45.2%). This suggests that delays during peak hours or in high-traffic areas give a negative experience to drivers. Security risks, such as the misuse of lost tickets, were selected by 8 respondents (25.8%). This shows that drivers are not only concerned about convenience but also the safety and reliability of the current systems. Additionally, 1 respondent (3.2%) selected the "Others" option and mentioned the hassle of ensuring sufficient balance on a card before entering a parking facility. This further emphasizes the limitations of prepaid card systems and the potential benefit of real-time balance tracking or auto top-up features.

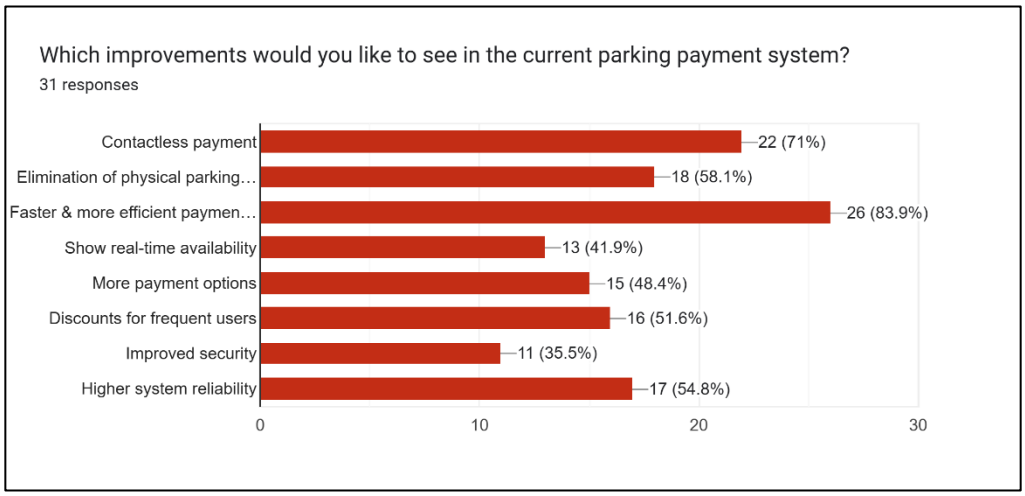


Figure 4.12: Bar Chart of Respondents' Suggested Improvements for Parking Payment System

The most highly requested improvement is faster and more efficient payment processing, with 83.9% of respondents selecting this option. This highlights the importance of minimizing delays during entry and exit, possibly through better system integration or automation. Contactless payment methods were also a popular choice, which was chosen by 71% of respondents. It indicates a strong demand for more hygienic and convenient payment experiences. Following closely, eliminating the need for physical parking tickets was chosen by 58.1%, reflecting a preference for digital or app-based alternatives that reduce the hassle of handling or losing tickets. Other suggestions include increasing system reliability (54.8%) and offering discounts for frequent users (51.6%), both showing a need for consistent system performance and customer loyalty benefits. More payment options, such as eWallets, mobile apps, or credit cards, were selected by 48.4% of respondents, suggesting that flexibility in payment is also important to users. Additionally, 41.9% of respondents would like real-time availability displays, which could help drivers better plan and reduce time spent searching for parking. Finally, improved security was selected by 35.5%, indicating that users also value safer, more trustworthy payment systems.

4.2.1.3 Section C: Exposure to Technologies for Automated Parking Payment

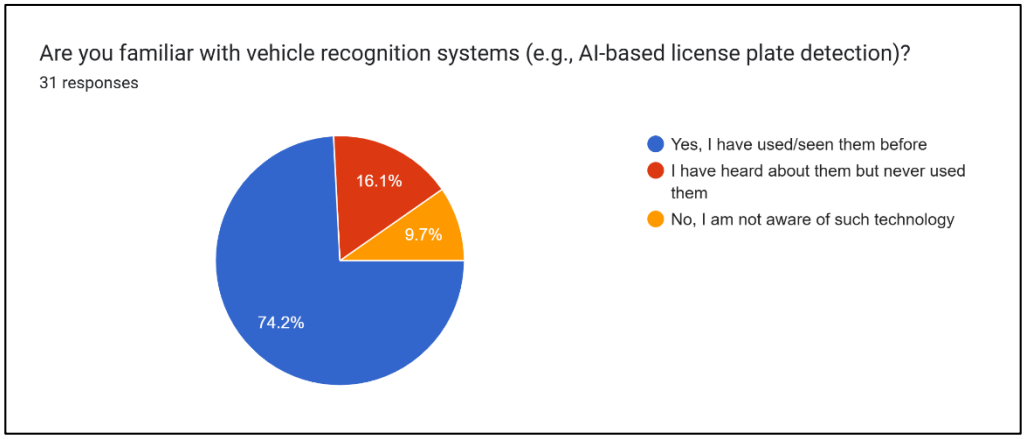


Figure 4.13: Pie Chart of Respondents' Familiarity with Vehicle Recognition System

A majority of respondents (74.2%) expressed that they have used or seen vehicle recognition systems before, suggesting a widespread exposure to the technology. A smaller portion (16.1%) has heard of these systems but has not used them, suggesting some awareness but limited direct experience. The remaining 9.7% are unaware of vehicle recognition technology, highlighting that there are still individuals unfamiliar with this emerging technology. Overall, this indicates a high level of awareness and experience among respondents, but also shows that there is room for greater adoption and familiarity.

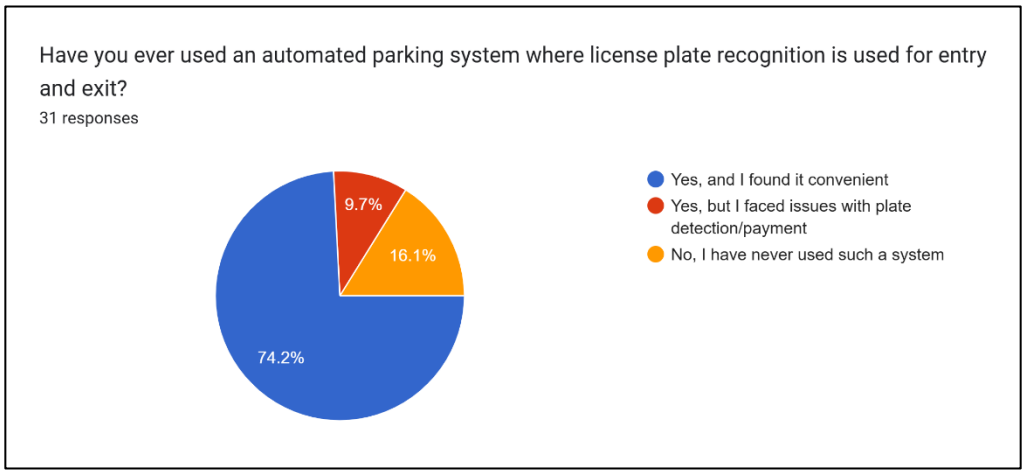


Figure 4.14: Pie Chart of Respondents' Experience Using License Plate Recognition Parking Systems

The chart shows that 74.2% of respondents have used a license plate recognitio parking system and found it convenient, indicating a generally positive experience with the technology. A smaller portion, 9.7%, have used LPR systems but encountered issues with plate detection or payment, suggesting that while the technology has been adopted, there are still challenges to be addressed. The remaining 16.1% have never used such a system, highlighting that there is still a segment of users unfamiliar with LPR parking systems. This suggests that while LPR technology is gaining popularity, there is still room for improvement and greater adoption.

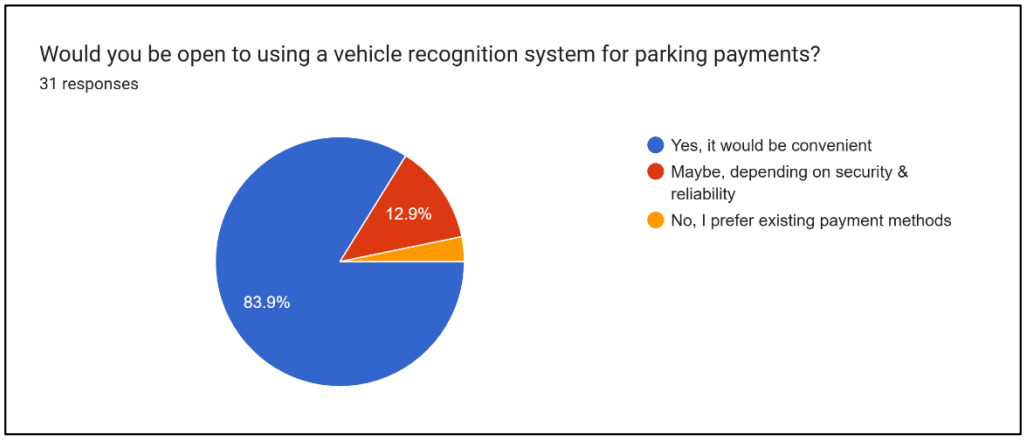


Figure 4.15: Pie Chart of Respondents' Willingness to Use Vehicle Recognition System for Parking Payments

The chart indicates that a majority of respondents (83.9%) would be willing to use a vehicle recognition system for parking payments, expressing that convenience is important to them. A smaller portion (12.9%) expressed limited interest and is dependent on the security and reliability of the system. Only 3.2% of respondents preferred to stick with existing payment methods and were unwilling to use a vehicle recognition system for parking payments. This suggests that most respondents are open to adopting vehicle recognition systems if they offer practical benefits like convenience and ease of use. While there is a positive response to the system, concerns around security and reliability must still be addressed to ensure more acceptance.

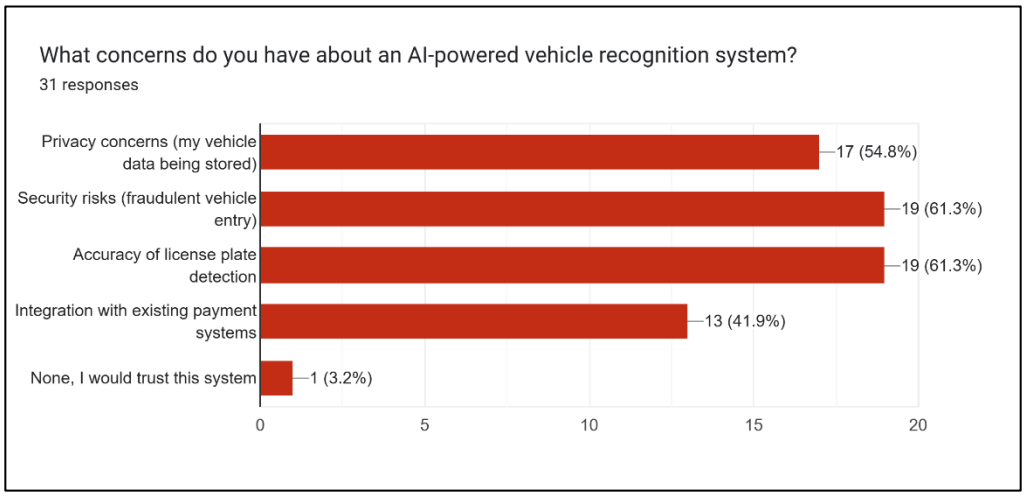


Figure 4.16: Bar Chart of Respondents' Concerns Regarding AI-Powered Vehicle Recognition System

The primary concern among respondents regarding AI-powered vehicle recognition systems is security, with 61.3% expressing worries about fraudulent vehicle entry. With an equal percentage of 61.3%, the accuracy of license plate detection is also another significant concern. Inaccuracies could lead to errors in billing, processing delays, and misidentification, affecting user trust and system reliability. Privacy concerns, specifically about vehicle data being stored, were also raised by 54.8% of respondents. Additionally, 41.9% are concerned about how well the system integrates with existing payment methods. Only 3.2% of respondents indicated they would trust the system without any concerns. This highlights that while there is interest in AI-powered systems, addressing privacy, security, and accuracy concerns will be crucial to gaining wider acceptance.

4.2.1.4 Section D: User Expectations for AI-powered Parking System

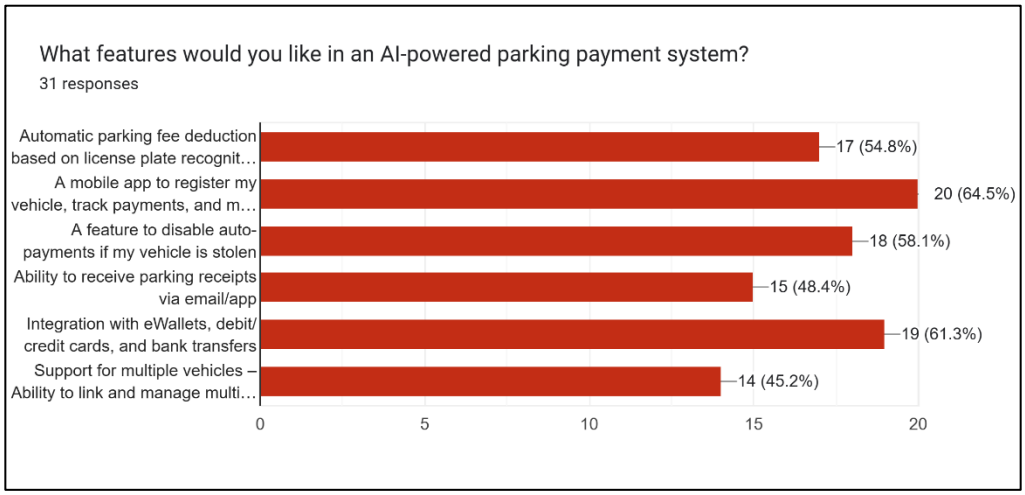


Figure 4.17: Bar Chart of Features Respondents Want in an AI-Powered Parking Payment System

The most popular feature is a mobile app to register vehicles, track payments, and manage parking history (64.5%). This highlights the demand for a simpler and accessible way to handle parking tasks. The next popular feature is the integration of various payment methods, such as eWallets, debit/credit cards, and bank transfers (61.3%). This reflects the importance of flexible and smooth payment options. Other highly requested features include automatic parking fee deduction based on license plate recognition (54.8%) and the ability to disable auto-payments if a vehicle is stolen (58.1%). This indicates concerns around security and the need for convenience in managing payments. Additionally, 48.4% of respondents would like to receive parking receipts via email or app, showing a preference for digital records. Lastly, 45.2% of respondents would like to manage multiple vehicles under one account, further emphasizing the desire for a more user-friendly, multi-functional system. These preferences demonstrate the need for a comprehensive, secure, and flexible solution that meets the diverse needs of users.

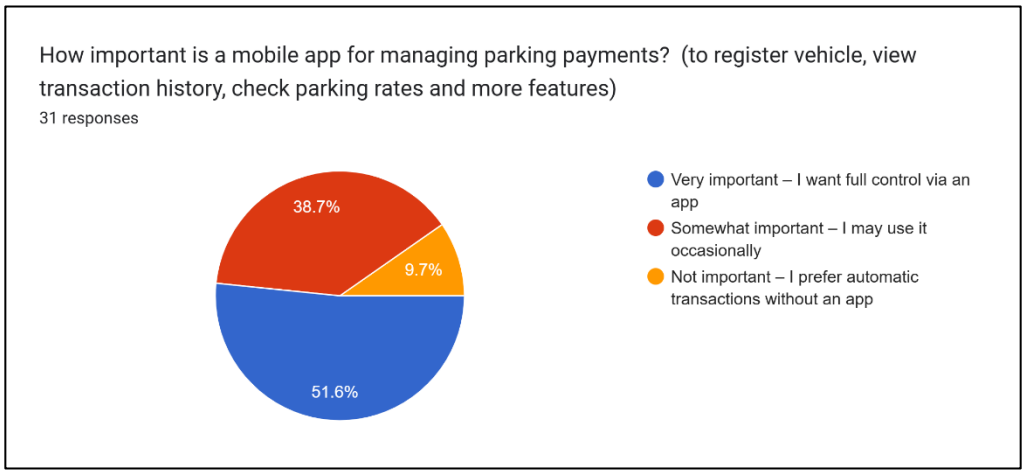


Figure 4.18: Pie Chart of the Importance of a Mobile App for Managing Parking Payments

The majority of respondents (51.6%) consider a mobile app "very important" for managing parking payments, as they desire full control over their parking payments through the app. This reflects a preference for greater convenience and autonomy in managing parking transactions. A smaller portion of respondents (38.7%) find the app "somewhat important," indicating that they might use the app occasionally but are not as dependent on it. Lastly, a minority of respondents (9.7%) view the app as "not important" and prefer automatic transactions without an app, suggesting that they favor a more straightforward payment experience. This chart emphasizes the importance of offering a mobile app for managing parking payments, while also highlighting that some users still prefer fully automated methods.

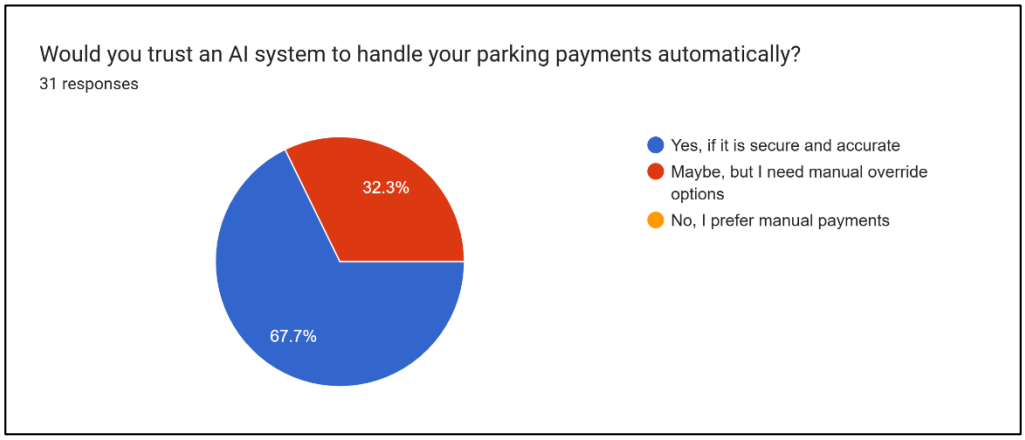


Figure 4.19: Pie Chart of Respondents' Trust in AI System for Handling Parking Payments

The majority of respondents (67.7%) are open to trusting an AI system for handling parking payments, provided it is secure and accurate. This shows that most users are willing to adopt AI technology if it offers reliability and safety. Meanwhile, 32.3% of respondents are more cautious as they would trust the system only if manual override options are available. This suggests a need for flexibility and user control within the system. Notably, none of the respondents rejected the idea entirely, showing a general openness toward AI-driven solutions for parking payments.

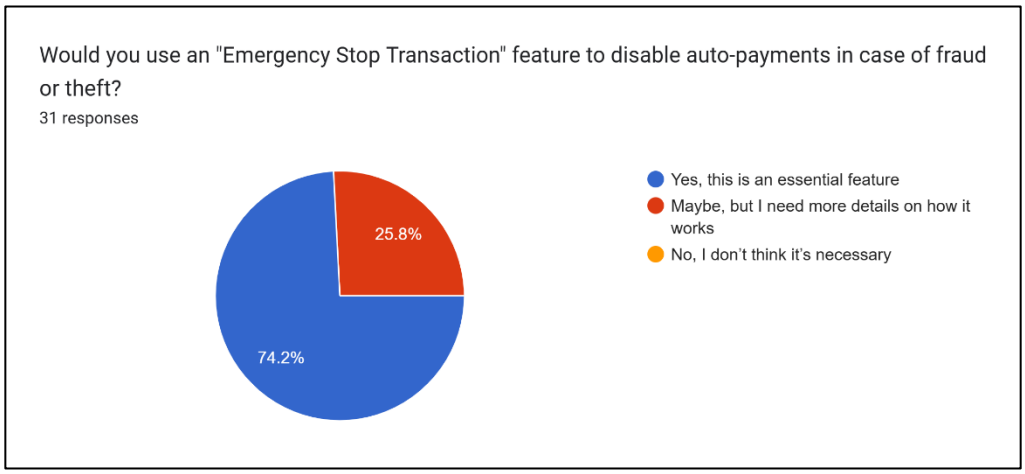


Figure 4.20: Pie Chart of Respondents' Need for an "Emergency Stop Transaction" Feature

The majority of respondents (74.2%) believe that an "Emergency Stop Transaction" feature is essential. This shows that there is strong support for having the ability to immediately stop parking payments in urgent situations, such as vehicle theft or unauthorized use. Meanwhile, 25.8% of respondents expressed interest but would like more information about how the feature works, suggesting clearer policies and transparency could help increase confidence. Notably, no respondents dismissed the need for this feature, indicating overall support for having added safety and control measures within the parking payment system.

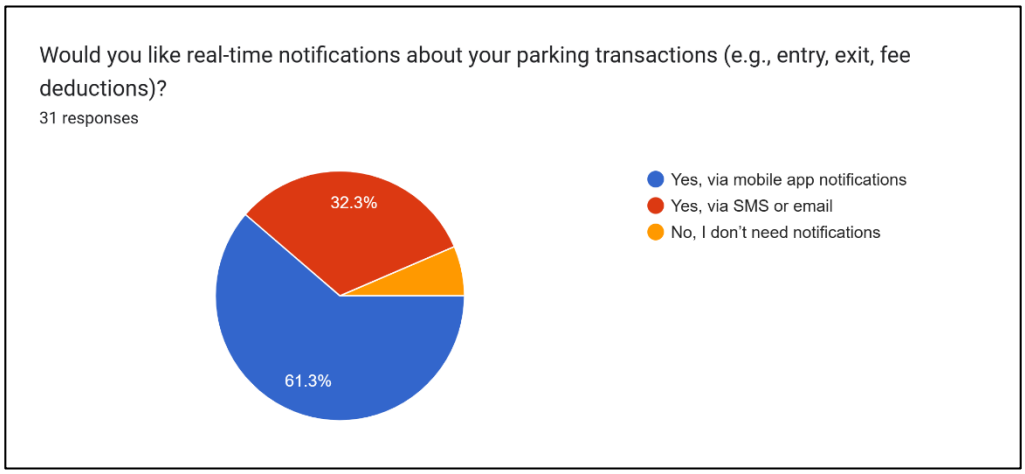


Figure 4.21: Pie Chart of Respondents' Preference for Real-Time Parking Transaction Notifications

Most respondents (61.3%) prefer to receive real-time parking transaction notifications through mobile app alerts, showing a strong demand for instant updates. Another 32.3% prefer notifications via SMS or email, suggesting that while they value updates, they may not actively use mobile apps or prefer more traditional channels. Only 6.5% indicated that they do not require any notifications, as they may prioritize simplicity or find frequent notifications unnecessary or intrusive. Overall, the majority want to stay informed about their parking activities through some sort of notification. The chart also highlights the importance of offering flexible notification options to suit different user preferences and enhance overall user engagement.

4.3 Requirement Specification

4.3.1 Functional Requirements

Table 4.1: Functional Requirements.

ID	Functional Requirement
FR1	The system shall allow drivers and parking operators to register user accounts.
FR2	The system shall allow drivers, parking operators, and admins to log in securely.
FR3	The system shall allow drivers, parking operators, and admins to manage their own profile information.
FR4	The system shall allow drivers to manage one or more vehicles.
FR5	The system shall allow drivers to manage the auto-transaction settings for parking through license plate and vehicle attribute recognition using multimodal AI.
FR6	The system shall allow drivers, parking operators, and admins to view dashboards personalized to their roles.
FR7	The system shall allow drivers to view nearby parking information.
FR8	The system shall allow drivers to view their parking history.
FR9	The system shall allow drivers to view nearby EV charger information.
FR10	The system shall allow drivers to make EV reservations.
FR11	The system shall allow drivers to view their EV reservations.
FR12	The system shall allow drivers to manage their payment methods.
FR13	The system shall process automatic parking fee transactions when drivers exit a parking lot.
FR14	The system shall provide drivers with notifications and alerts related to transactions.
FR15	The system shall require parking operators to set up parking rates, parking lot details, and all relevant information during the registration process.
FR16	The system shall allow parking operators to update and manage parking lot details, which will be reviewed and approved by the admin.

FR17	The system shall allow drivers and parking operators to submit support tickets for system-related issues.
FR18	The system shall allow admins to manage accounts for drivers and parking operators.
FR19	The system shall require admin approval for new parking operator accounts before they can start managing parking lots, rates, and other features.
FR20	The system shall require admin approval for any changes made to parking rates, parking zones, or other features requested by parking operators.
FR21	The system shall allow admins to manage support tickets submitted by drivers or parking operators.

4.3.2 Non-functional Requirements

Table 4.2: Non-functional Requirements.

ID	Non-functional Requirements	Category of NFR
NFR1	The system shall achieve a minimum of 90% accuracy in license plate recognition under optimal conditions.	Accuracy
NFR2	The system shall correctly match vehicle attributes (e.g., color, make, model) with a minimum accuracy rate of 90% under optimal conditions.	
NFR3	The system shall ensure that auto-payments are only triggered when both the license plate and registered vehicle attributes are correctly recognized.	
NFR4	The system shall respond to user actions within 2 seconds under normal network conditions.	Performance
NFR5	The system shall process license plate recognition and return a decision within 5 seconds at the point of entry/exit.	

NFR6	The system shall be compatible with modern web browsers, including Chrome, Firefox, Edge, and Safari, for the website, and compatible with Android and iOS platforms for the mobile application.	Compatability
NFR7	The system shall integrate with third-party payment gateways and support APIs for external system access (e.g., payment services).	
NFR8	The system shall enforce authentication and role-based access control for drivers and parking operators.	Security
NFR9	The system shall maintain accurate and tamper-proof records of transactions	
NFR10	All sensitive data, like user credentials, personal data, and payment information, shall be encrypted using AES-256 encryption or higher.	
NFR11	The application shall adhere to relevant data protection regulations to safeguard user privacy and ensure the security of personal data.	
NFR12	The system shall provide a user-friendly, simple, and intuitive interface with clear icons, buttons, and navigation.	Usability
NFR13	When an error occurs, the system shall display clear and understandable error messages.	
NFR14	The system shall be able to store and retrieve user data accurately without loss or corruption.	Reliability

NFR15	The system shall follow a modular architecture to facilitate future enhancements and updates, with a structured and maintainable codebase that adheres to established coding standards and industry best practices.	Maintainability
NFR16	The system shall be designed to easily accommodate additional parking lots or parking zones as the system expands, without requiring significant changes to the infrastructure.	Scalability
NFR17	The system shall be designed to accommodate future growth in user base, traffic, and transaction volume.	

4.4 Use Case Modelling

4.4.1 Use Case Diagram

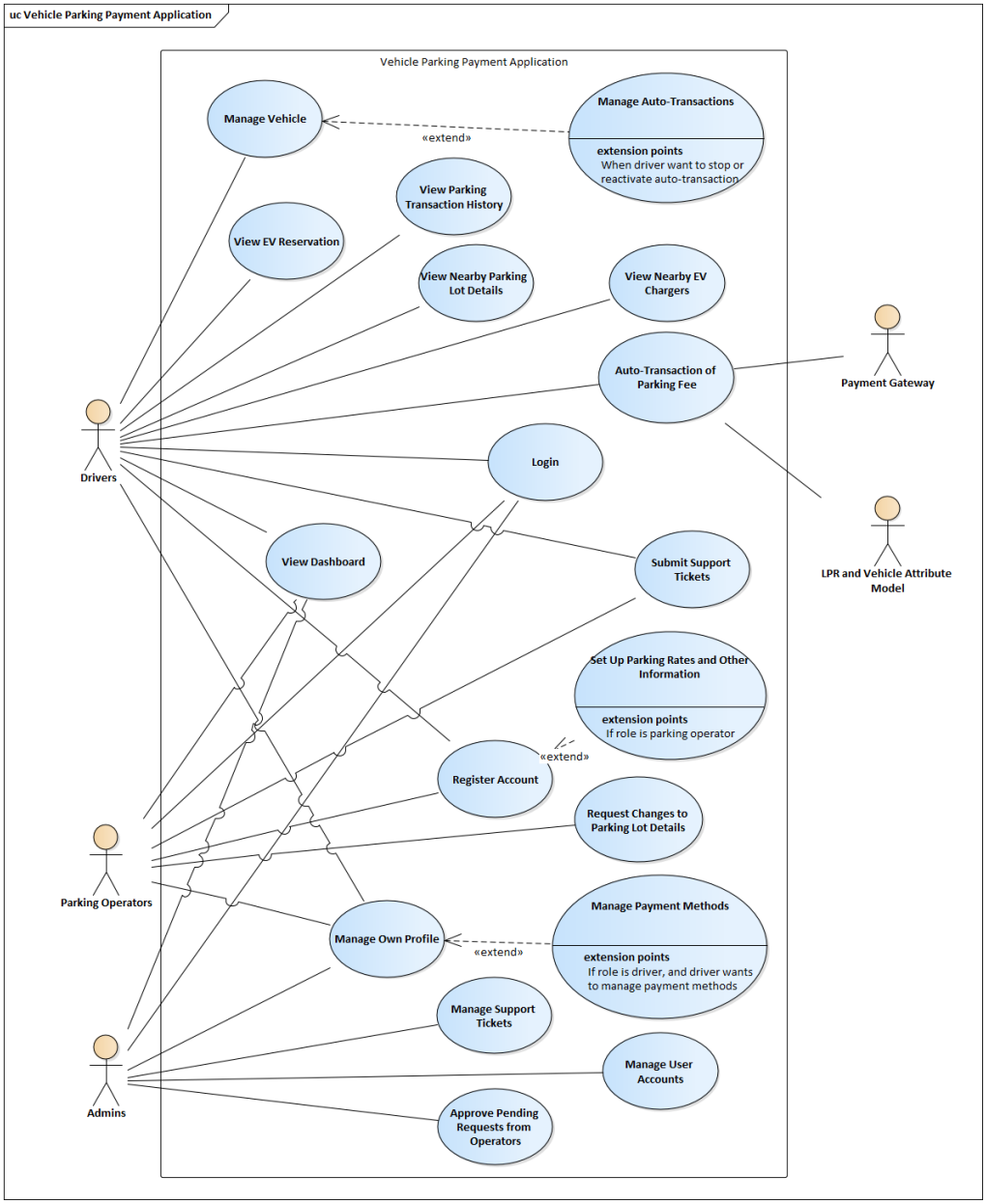


Figure 4.22: Use case diagram of Vehicle Parking Payment Application.

4.4.2 Use Case Description

Table 4.3: Use case description of login.

Use Case Name: Login	ID: UC-1	Importance Level: High
Primary Actor: Drivers, Parking Operators, Admin	Use Case Type: Detail, Real	
Stakeholders and Interests: N/A		
Brief Description: Allows Admins, Parking Operators, and Drivers to securely access the system based on their assigned roles.		
Trigger: The actor initiates the login process by selecting login on the home page of the platform.		
Relationships: Association : Driver, Parking Operator, Admin Include : - Extend : - Generalization: -		
Normal Flow of Events: 1. The actor navigates to the login page. 2. The system prompts the user to enter their login credentials. 3. The actor enters their login credentials. 4. The system verifies the credentials. Continue S-1 5. The actor is redirected to their dashboard. 6. The login session is initiated.		
Sub-flows: S-1: Perform 4.1 or 4.2 or 4.3 or 4.4 4.1 If invalid credentials, the system displays an error message and prompts the user to re-enter their credentials. Continue to flow 3. 4.2 If valid credentials, the actor successfully logs in to their account. Continue to flow 5.		

4.3 If unapproved parking operators, the system displays a pending approval message. Continue to flow 3. 4.4 If the actor does not have an account, UC-2 will be performed.
Alternate/Exceptional Flows: Assumptions:

Table 4.4: Use case description of Register.

Use Case Name**:** Register	ID**:** UC-2	Importance Level**:** High
Primary Actor**:** Drivers, Parking Operators	Use Case Type**:** Detail, Real	
Stakeholders and Interests**:** N/A		
Brief Description**:** Allows Drivers and Parking Operators to create a new account on the platform.		
Trigger**:** The actor initiates the register process by selecting register on the home page of the platform.		
Relationships: Association **:** Driver, Parking Operator Include **: -** Extend **:** Set Up Parking Rates and Other Related Information Generalization**: -**		
Normal Flow of Events**:** 1. The actor navigates to the register page. 2. The system prompts the user to enter the required information, including email and password. 3. The actor enters their required information and submits the registration form. 4. The system validates the credentials. Continue S-1 5. The system creates a new user account. 6. The actor is redirected to their dashboard.		

7. The login session is initiated.
Sub-flows: S-1: Perform 4.1 or 4.2 or 4.3 4.1 If invalid input, the system displays an error message and prompts the user to re-enter the required information. Continue to flow 3. 4.2 If valid input, the actor successfully registered their account. Continue to flow 5. 4.3 If the actor is a parking operator, proceed to E-1.
Alternate/Exceptional Flows: E-1: 4.3.1 The system prompts the parking operator to enter required information for parking lots information. 4.3.2 The parking operator enters the required parking lots information and submits the registration form. 4.3.3 The system prompts a message stating to wait for admin approval. Continue to flow 1. Assumptions:

Table 4.5: Use case description of Manage Vehicles

Use Case Name: Manage Vehicles	ID: UC-3	Importance Level: High
Primary Actor: Driver	Use Case Type: Detail, Real	
Stakeholders and Interests: N/A		
Brief Description: Allows driver to register their vehicles for auto-transaction for parking payments.		
Trigger: The driver wants to register their vehicles into the application.		
Relationships: Association : Drivers Include : - Extend : Manage Auto-Transaction		

Generalization**: -**

Normal Flow of Events**:**

1. The driver navigates to the “manage vehicle” screen.
2. The system displays the currently registered vehicles if they exist, and prompts the user to add, edit, delete or manage auto-transaction of vehicles. Continue S-1.
3. The system updates the records of the vehicle.
4. The system will display a message showing that the action performed is successful.

Sub-flows**:**

S-1: Perform 2.1 or 2.2 or 2.3 or 2.4

2.1 If driver selects to add vehicle, the system request vehicle information.

2.1.1 The user enters the required vehicle details, including vehicle type, license plate, make, model, and color, and submit the registration form.

2.1.1.1 If invalid input, the system displays an error message and prompts the driver to re-enter the required information. Continue to flow 2.

2.1.1.2 If valid input, the vehicle is successfully registered.

2.2 If driver selects to edit vehicle, the system request vehicle information for modification.

2.2.1 The user enters the required vehicle information.

2.2.1.1 If invalid input, the system displays an error message and prompts the driver to re-enter the required information. Continue to flow 2.

2.2.1.2 If valid input, the vehicle detail is modified. Continue to flow 3

2.2.2 If the enter selects to manage payment methods of the vehicle, perform UC-5.

2.3 If driver selects to delete vehicle method, the system prompts a confirmation message.

2.3.1 If the driver selects confirm, the vehicle will be removed. Continue to flow 3.

2.3.2 If the driver selects cancel, the process will stop. Continue to flow 2.

2.4 If the driver selects to manage auto-transaction settings, proceed to E-1

Alternate/Exceptional Flows:

E-1: Perform 2.4.1 or 2.4.2

2.4.1 If the auto-transactionis active and the driver wants to stop, the system will prompt a confirmation message.

2.4.1.1 If the driver selects confirm, the auto-deduction will be deactivated.

2.4.1.1.1 The driver receives a notification regarding the stop. Continue to flow 3.

2.4.1.2 If the driver selects cancel, the process will stop. Continue to flow 2.

2.4.2 If the auto-transaction is deactivated and the driver wants to reactivate, the system will prompt a verification step.

2.4.2.1 If verification successful, the auto-transaction will be reactivated.

2.4.2.1.1 The driver receives a notification regarding the reactivation. Continue to flow 3.

2.4.2.2 If the driver selects cancel, the process will stop. Continue to flow 2.

Assumptions:

A. Actor is an authenticated user.

B. Driver can only edit or delete the specific vehicle when it is registered

Table 4.6: Use case description of Manage Payment Methods

Use Case Name: Manage Payment Methods	ID: UC-4	Importance Level: High
Primary Actor: Driver	Use Case Type: Detail, Real	
Stakeholders and Interests: N/A		
Brief Description: Allows the driver to add, modify, or delete their payment methods. This use case extends Manage Own Profile (UC-16) when the actor is a Driver.		
Trigger: The driver wants to manage their payment methods.		
Relationships: Association : Drivers Include : - Extend : - Generalization: -		
Normal Flow of Events: 1. The driver selects *"Manage Payment Methods"* from their profile page. 2. The system displays the driver's current payment methods if they exist, and prompts the user to add, edit, or delete payment methods. Continue S-1. 3. The system updates the records of payment methods. 4. The system will display a message showing that the action performed is successful.		
Sub-flows: S-1: Perform 2.1 or 2.2 or 2.3 2.1 If the driver selects to add payment methods, the system requests payment method information. 2.1.1 The user enters the required payment method information.		

2.1.1.1 If invalid input, the system displays an error message and prompts the driver to re-enter the required information. Continue to flow 2.1. 2.1.1.2 If valid input, the payment method is added. Continue to flow 3. 2.2 If the driver selects to edit payment methods, the system displays the selected payment method information available for modification. 2.2.1 The user enters the required payment method information for modification. 2.2.1.1 If invalid input, the system displays an error message and prompts the driver to re-enter the required information. Continue to flow 2.2. 2.2.1.2 If valid input, the payment method is modified. Continue to flow 3. 2.3 If the driver selects to delete a payment method, the system prompts a confirmation message. 2.3.1 If the driver selects confirm, the payment method will be removed. Continue to flow 3. 2.3.2 If the driver selects cancel, the process will stop. Continue to flow 2.
Alternate/Exceptional Flows: Assumptions: A. Actor is an authenticated user.

Table 4.7: Use case description of View Dashboard

Use Case Name**:** View Dashboard	ID**:** UC-5	Importance Level**:** High
Primary Actor**:** Driver, Parking Operator, Admin	Use Case Type**:** Detail, Real	
Stakeholders and Interests**:** Drivers want to see their current sessions, reservations, and statistics for better tracking and planning. Parking Operators want to monitor operational metrics, financial performance, and live parking		

status. Admin wants to oversee the entire system, user activities, resources, and growth trends.

Brief Description: Allows the actor to view their personalized dashboard showing role-specific information, statistics, and quick actions.

Trigger: The actor logs into the system and navigates to the Dashboard screen.

Relationships:

Association : Drivers

Include : -

Extend : -

Generalization: -

Normal Flow of Events:

1. The actor logs in and navigates to the Dashboard screen.
2. The system checks the actor's role. Continue to S-1.
3. The actor browses the dashboard and interacts with available quick actions or analytics.

Sub-flows:

S-1: Perform 2.1 or 2.2 or 2.3

2.1 If the role is Driver, the system displays the appropriate dashboard content for drivers, which includes current parking sessions, EV charging reservations, stats such as total sessions, total spent, sessions this month, and usage trends, and quick info and actions, continue to flow 3.

2.2 If the role is Parking Operator, the system displays the appropriate dashboard content for parking operators, which includes parking lots overview, parked today count, today's revenue, occupancy rate and average duration, customer analysis and revenue trends, peak hour analysis, live parking status, and current parking rates, continue to flow 3.

2.3 If the role is Admin, the system displays the appropriate dashboard content for admins, which includes total users,

parking lots, EV chargers in the system, quick actions, alerts, growth trends, parking usage statistics, and user breakdowns, continue to flow 3.
Alternate/Exceptional Flows: A. If system data sync fails, the system shows a “Data temporarily unavailable” message with a retry option. B. If the actor does not have proper role permissions, the system restricts access and displays “Unauthorized access”. Assumptions: A. Actor is an authenticated user and has a valid role assigned. B. Dashboard data is updated in real-time or near real-time to ensure accuracy.

Table 4.8: Use case description of View Nearby Parking Lot Details

Use Case Name**:** View Nearby Parking Lot Details	ID**:** UC-6	Importance Level**:** High
Primary Actor**:** Driver	Use Case Type**:** Detail, Real	
Stakeholders and Interests**:** N/A		
Brief Description**:** Allows the driver to view details of nearb parking lots to facilitate parking decisions.		
Trigger**:** Driver wants to view parking lot details		
Relationships: Association **:** Drivers Include **: -** Extend **: -** Generalization**: -**		
Normal Flow of Events**:** 1. The driver navigates to the “parking lots” screen. 2. The system requests permission to access the driver’s current location. Continue S-1 3. If permission is granted, the system retrieves the driver’s location.		

4. The system checks if there is any parking lots available. Continue S-2 5. The system displays the list of nearby parking lots on a map or list view. 6. The driver browses the list or map to view available parking lots. 7. The driver can select to view the details of the specific parking lot. 8. The system will display the details of the specific parking lot, including the location, rate, zones, and more.
Sub-flows: S-1: Perform 2.1 or 2.2 2.1 If the driver denies location permission, the system cannot display nearby parking lot. 2.2 If the driver accepts location permission, the system retrieves the driver's location, continue to flow 3. S-2: Perform 3.1 or 3.2 3.1 If there is no parking lots available, the system displays a message showing no parking lots and redirect to previous screen. 3.2 If there are parking lots, continue to flow 4.
Alternate/Exceptional Flows: Assumptions: A. Actor is an authenticated user. B. Parking lot data is synced with the system in real-time for accurate availability.

Table 4.9: Use case description of View Nearby EV Chargers

Use Case Name: View Nearby EV Chargers	ID: UC-7	Importance Level: High
Primary Actor: Driver	Use Case Type: Detail, Real	
Stakeholders and Interests: N/A		

Brief Description: Allows the driver to view details of nearby EV chargers, including availability, type, and location, to facilitate charging decisions.
Trigger: The driver wants to check EV charger availability.
Relationships: Association : Drivers Include : - Extend : - Generalization: -
Normal Flow of Events: 1. The driver navigates to the "EV Reservation" screen. 2. The system requests permission to access the driver's current location. Continue to S-1. 3. The system checks if there are any EV chargers available. Continue to S-2. 4. The system displays the list of nearby EV chargers on a map or list view. 5. The driver browses the list or map to view available chargers. 6. The driver can select a specific EV charger to view detailed information. 7. The system displays the details of the charger, including Charger type, location, and availability status.
Sub-flows: S-1: Perform 2.1 or 2.2 2.1 If the driver denies location permission, the system cannot display nearby EV chargers. 2.2 If the driver accepts location permission, the system retrieves the driver's location, continue to flow 3. S-2: Perform 3.1 or 3.2 3.1 If there are no chargers available, the system displays a message indicating "No EV chargers available". 3.2 If there are chargers available, continue to flow 4.

Alternate/Exceptional Flows: A. If a charger's status changes while viewing (e.g., from available to reserved), the system updates the display in real-time. Assumptions: A. Actor is an authenticated user. B. EV charger data is synced with the system in real-time for accurate availability.

Table 4.10: Use case description of View Parking Transaction History

Use Case Name**:** View Parking Transaction History	ID**:** UC-8	Importance Level**:** High
Primary Actor**:** Driver	Use Case Type**:** Detail, Real	
Stakeholders and Interests**:** N/A		
Brief Description**:** Allows driver to view parking their parking transaction history.		
Trigger**:** Driver wants to view their parking transaction history.		
Relationships: Association **:** Drivers Include **: -** Extend **: -** Generalization**: -**		
Normal Flow of Events**:** 1. The driver navigates to the "parking history" screen 2. The system checks if there is any parking history available. Continue S-1 3. The system displays the list of parking histories to the user. 4. The driver views the list of parking histories. 5. The driver can select to view the details of the specific parking history. 6. The system will display the details of the specific parking history, including the transaction fee, timestamp, parking location, zones, reference number, status, and payment method used.		

Sub-flows: S-1: Perform 2.1 or 2.2 2.1 If there is no parking history available, the system displays a message showing no parking history and redirect to previous screen. 2.2 If there are parking histories, continue to flow 3.
Alternate/Exceptional Flows: Assumptions: A. Actor is an authenticated user.

Table 4.11: Use case description of View EV Reservation

Use Case Name: View EV Reservation	ID: UC-9	Importance Level: High
Primary Actor: Driver	Use Case Type: Detail, Real	
Stakeholders and Interests: N/A		
Brief Description: Allows the driver to view their EV charging reservation history, separated into active reservations (reserved and ongoing) and past reservations.		
Trigger: Driver wants to view their EV charging reservations.		
Relationships: Association : Drivers Include : - Extend : - Generalization: -		
Normal Flow of Events: 1. The driver navigates to the “EV Reservation” screen 2. The system checks if there is any EV reservations associated with the driver. Continue S-1 3. The system displays two sections, which are Active Reservations that show reservations with status reserved or ongoing, and Past Reservations that show completed, expired, or cancelled reservations.		

4. The driver browses through active or past reservations. 5. The driver can select a specific reservation to view detailed information. 6. The system displays details of the reservation, including reservation time, charger type, location, zone, EV details, and current status.
Sub-flows: S-1: Perform 2.1 or 2.2 2.1 If there are no EV reservations, the system displays a message "No reservations found" and redirects to the previous screen. 2.2 If there are EV reservations, continue to flow 3.
Alternate/Exceptional Flows: Assumptions: A. Actor is an authenticated user. B. Reservation data is synced with the system in real-time for accurate availability and status.

Table 4.12: Use case description of Auto-Transaction of Parking Fee

Use Case Name: Auto-Transaction of Parking Fee	ID: UC-10	Importance Level: High
Primary Actor: Driver	Use Case Type: Detail, Real	
Stakeholders and Interests: Payment Gateway		
Brief Description: Allows driver to pay for their parking automatically.		
Trigger: When driver exits the parking lot through the gate sensor and the vehicle recognition is successfully performed.		
Relationships: Association : Drivers Include : - Extend : - Generalization: -		

Normal Flow of Events**:** 1. The driver's vehicle is scanned by the exit gate sensors. 2. Vehicle recognition is successful. 3. The system calculates the total parking fee based on entry and exit timestamps, and the parking location and rates. 4. The system initiates auto-transaction using linked payment methods. Continue to S-1. 5. The system will send a notification regarding the success of transaction to the drivers . 6. Transaction detail is saved to history.
Sub-flows**:** S-1: Perform 3.1 or 3.2 3.1 If auto-transaction is unsuccessful, the system will send a notification regarding the failure to the user. 3.2 If auto-transaction is successful, the payment is confirmed. Continue to flow 4.
Alternate/Exceptional Flows: Assumptions: A. Actor is an authenticated user. B. Driver anable the auto-transaction in the application C. Linked a valid payment method D. Vehicle recognition is successfully performed.

Table 4.13: Use case description of Submit Support Tickets

Use Case Name**:** Submit Support Tickets	ID**:** UC-11	Importance Level**:** High
Primary Actor**:** Drivers, Parking Operators	Use Case Type**:** Detail, Real	
Stakeholders and Interests**:** N/A		

Brief Description: Drivers and Parking Operators are able to submit support tickets when they faced any issues or bugs from the application.
Trigger: When there is an issue or bug that drivers and parking operators would like to submit to admin to resolve.
Relationships: Association : Driver, Parking Operator Include : - Extend : - Generalization: -
Normal Flow of Events: 1. The actor navigates to the “Support” page. 2. Actor selects "Submit New Ticket". 3. The system displays a form, prompting the user to enter the details of the issue faced, including subject, category and description. 4. The actor enters their required information and submits the support ticket form. 5. The system validates the input. Continue to S-1. 6. The system creates a support ticket. 7. Confirmation message is displayed
Sub-flows: S-1: Perform 5.1 or 5.2 5.1 If invalid input, the system displays an error message and prompts the user to re-enter the required information. Continue to flow 4. 5.2 If valid input, continue to flow 6.
Alternate/Exceptional Flows: Assumptions: A. Actor is an authenticated user.

Table 4.14: Use case description of Request Change to Parking Lot Details

Use Case Name: Request Change to Parking Lot Details	ID: UC-12	Importance Level: High
Primary Actor: Parking Operators	Use Case Type: Detail, Real	
Stakeholders and Interests: N/A		
Brief Description: Allows parking operator to request change to parking lot details. These changes are subject to admin review and approval. If a parking lot is marked as inactive, no further parking sessions or auto-transactions will be allowed for that location.		
Trigger: The parking operator have to make some changes to the information of parking lots.		
Relationships: Association : Parking Operators Include : - Extend : - Generalization: -		
Normal Flow of Events: 1. The parking operator navigates to the "My Parking Lots" page. 2. The system displays the available parking lots of the parking operator, 3. The parking operator selects a parking lot and clicks "Request Change". 4. The system displays a form with editable fields such as parking name, address, operating hours, pates, contact info, and status. 5. Operator modifies the desired fields and submit the request. Continue to S-1 6. System prompts a message stating to wait for admin's approval and records the request as "Pending Approval".		
Sub-flows: S-1: Perform 5.1 or 5.2 5.1 If invalid input, the system displays an error message and prompts the driver to re-enter the required information. Continue to flow 5.		

5.2 If valid input, the payment method is added. Continue to flow 6.
Alternate/Exceptional Flows: Assumptions: A. Actor is an authenticated user. B. Parking operator have existing parking lot as they set up during registration and approved by admin.

Table 4.15: Use case description of Manage Support Tickets

Use Case Name**:** Manage Support Tickets	ID**:** UC-13	Importance Level**:** High
Primary Actor**:** Admin	Use Case Type**:** Detail, Real	
Stakeholders and Interests**:** N/A		
Brief Description**:** Enables the admin to view, respond to, and manage all support tickets submitted by drivers and parking operators. Admins can update ticket status, provide responses, and close resolved tickets.		
Trigger**:** The admin want to manage the support tickets submitted by drivers and parking operators.		
Relationships: Association **:** Admin Include **: -** Extend **: -** Generalization**: -**		
Normal Flow of Events**:** 1. The admin navigates to the “Support Tickets” page. 2. The system checks if there are support tickets. Continue to S-1.		

3. The system displays a list of submitted tickets, including status that are open, in progress, resolved, and closed, while also offering filter by status, user type, and submission date. 4. The admin selects a ticket to view full details, including ticket id, submitter name and role, issue category, subject, and message. 5. The admin writes a response. 6. The admin updates the ticket status. 7. The system will send notification of the response to the users.
Sub-flows: S-1: Perform 2.1 or 2.2 2.1 If there is no support ticket available, the system displays a message showing no support ticket and redirect to previous screen. 2.2 If there are support tickets, continue to flow 3.
Alternate/Exceptional Flows: Assumptions: A. Actor is an authenticated user. B. Admin cannot respond to ticket that is closed.

Table 4.16: Use case description of Approve Pending Requests from Operators

Use Case Name: Approve Pending Requests from Operators	ID: UC-14	Importance Level: High
Primary Actor: Admin	Use Case Type: Detail, Real	
Stakeholders and Interests: Admin wants to ensure only legitimate operators and valid changes are approved. Parking Operators want their accounts and submitted changes to be processed efficiently.		
Brief Description: Allows the Admin to review, approve, or reject pending requests submitted by parking operators. Pending requests can include new operator account applications or change requests (e.g., parking rate adjustments, lot capacity		

updates, enabling/disabling EV chargers, or marking a lot inactive).
Trigger**:** When there are pending requests (account approvals or parking change requests) that require admin action.
Relationships: Association **:** Admin Include **: -** Extend **:** Generalization**: -**
Normal Flow of Events**:** 1. The admin navigates to the Pending Requests page. 2. The system checks if there are any pending requests. Continue to S-1. 3. The system displays a list of pending requests (e.g., operator accounts, parking changes) with filter options (e.g., request type, status, submission date). 4. The admin selects a request to view details. 5. The system displays full request details (e.g., operator info, requested change, justification, timestamp). 6. The admin decides to approve or reject the request. Continue to S-2. 7. The system displays a message showing the action is successful. 8. The system updates the status of the request accordingly. 9. The system sends a notification to the operator regarding the decision.
Sub-flows**:** S-1: Perform 2.1 or 2.2 2.1 If there is no pending parking change request, the system displays a message showing no pending request and redirect to previous screen. 2.2 If pending requests exist, continue to flow 3. S-2: Perform 6.1 or 6.2 6.1 If the admin selects **Approve**, the system finalizes the approval (account is activated or parking change applied). Continue to flow 7.

6.2 If the admin selects reject, the system will prompt a confirmation message. 4.2.1 If the admin confirms, the system rejects the request. Continue to flow 7. 4.2.2 If the admin selects cancel, the process stops. Continue to flow 5.
Alternate/Exceptional Flows: Assumptions: A. Actor is an authenticated user. B. Operators must have submitted requests through the system.

Table 4.17: Use case description of Manage User Accounts

Use Case Name**:** Manage User Accounts	ID**:** UC-15	Importance Level**:** High
Primary Actor**:** Admin	Use Case Type**:** Detail, Real	
Stakeholders and Interests**:** N/A		
Brief Description**:** Allows the Admin to view, search, add, edit, and manage all driver and parking operator accounts in the system. Admins can create new user accounts, update existing profiles, deactivate/reactivate accounts, monitor suspicious or inactive users, and manage other admins when necessary.		
Trigger**:** The admin want to manage user accounts.		
Relationships: Association **:** Admin Include **: -** Extend **: -** Generalization**: -**		
Normal Flow of Events**:** 1. The admin navigates to the “User Accounts” page. 2. The system checks if there are any users. Continue to S-1.		

3. System displays a list of all registered accounts with filters by user role, account status, registration date, and search by email, name, or id.
4. The admin can select a specific account to view full profile
5. The system displays the full profile, including name, email, contact info, role, login history, and status.
6. Admin can perform actions. Continue to S-2

Sub-flows:

S-1: Perform 2.1 or 2.2

- 2.1 If there is no users, the system displays a message showing no users and redirect to previous screen.
- 2.2 If there are users, continue to flow 3.

S-2: Perform 6.1 or 6.2 or 6.3 or 6.4

- 6.1 If admin select to deactivate a user, a confirmation message will be prompted
  - 6.1.1 If admin select confirm, the status of the user will switch to deactivated.
  - 6.1.2 If admin select cancel, the process stops. Continue to flow 5.
- 6.2 If admin select to reactivate a user, a confirmation message will be prompted
  - 6.2.1 If admin select confirm, the status of the user will switch to active.
  - 6.2.2 If admin select cancel, the process stops. Continue to flow 5.
- 6.3 If admin selects Add New User, the system displays a registration form (name, email, role, password, etc.).
  - 6.3.1 The admin fills in details and submits.
  - 6.3.2 The system validates input and creates the new account.
  - 6.3.3 The new user appears in the user list.
- 6.4 If admin selects Edit Profile, the system displays an editable form with user details.
  - 6.4.1 The admin updates and submits changes.

6.4.2 The system validates and saves changes. 6.4.3 A success message is displayed, and the updated profile is shown.
Alternate/Exceptional Flows: A. If invalid inputs are entered when adding/editing a user, the system displays error messages and requests corrections. B. If an attempt is made to deactivate the last active system admin, the system blocks the action and shows a warning. Assumptions: A. Actor is an authenticated user.

Table 4.18: Use case description of Manage Own Profile

Use Case Name**:** Manage Own Profile	ID**:** UC-16	Importance Level**:** High
Primary Actor**:** Driver, Parking Operator, Admin	Use Case Type**:** Detail, Real	
Stakeholders and Interests**:** N/A		
Brief Description**:** Allows users of all roles to view and update their personal profile information, including name, contact details, and password. Ensures that user data stays accurate and current. Drivers can also extend this functionality to manage their payment methods.		
Trigger**:** The actors want to manage their own profile.		
Relationships: Association **:** Driver, Parking Operator, Admin Include **: -** Extend **:** UC-4 Manage Payment Methods (*only for Driver*) Generalization**: -**		
Normal Flow of Events**:** 1. The actor navigates to the “Profile” page.		

2. The system displays the actor's current profile information, including full name, email, phone number, and role. If actor is Driver, proceed to E-1.
3. Actor edits desired fields and submit changes.
4. The system validates the input. Continue to S-1.
5. System displays confirmation message.

Sub-flows:

S-1: Perform 4.1 or 4.2

4.1 If invalid input, the system displays an error message and prompts the actor to re-enter the required information. Continue to flow 2.

4.2 If valid input, the system saves the updated profile. Continue to flow 5.

Alternate/Exceptional Flows:

E-1: Perform 2.1 or 2.2

2.1 If the actor is a Driver, the system displays an additional option: "Manage Payment Methods."

2.1.1 The driver may choose to add, edit, or delete payment methods (UC-4).

2.1.1.1 On completion, control returns to the Manage Own Profile step 2.

Assumptions:

A. Actor is an authenticated user.

4.5 Interface Flow Diagram

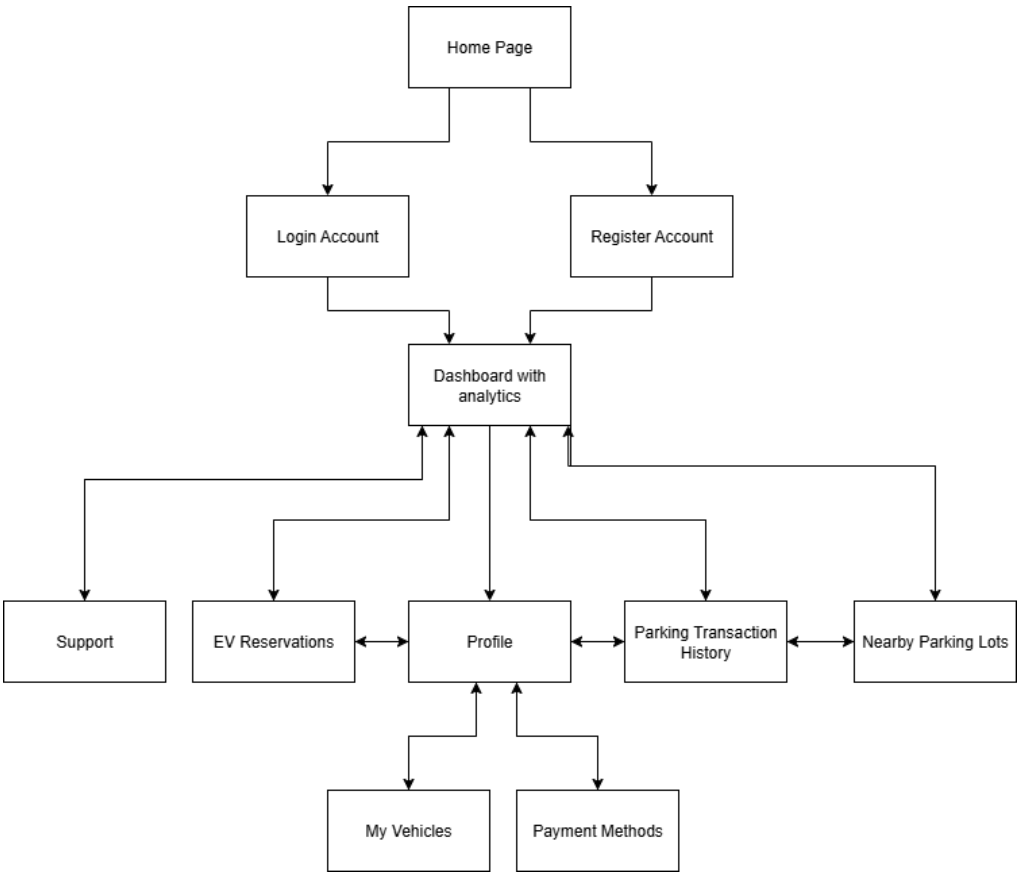


Figure 4.23: Interface flow diagram of the proposed system for drivers

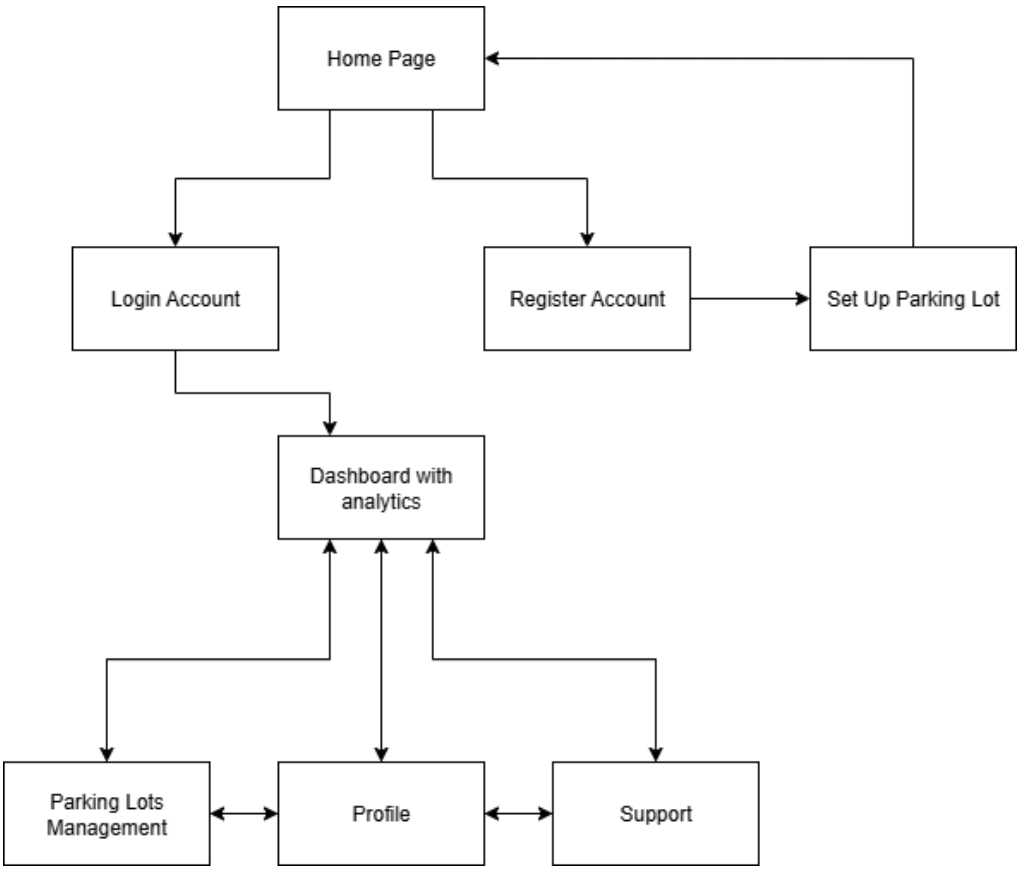


Figure 4.24: Interface flow diagram of the proposed system for parking operators

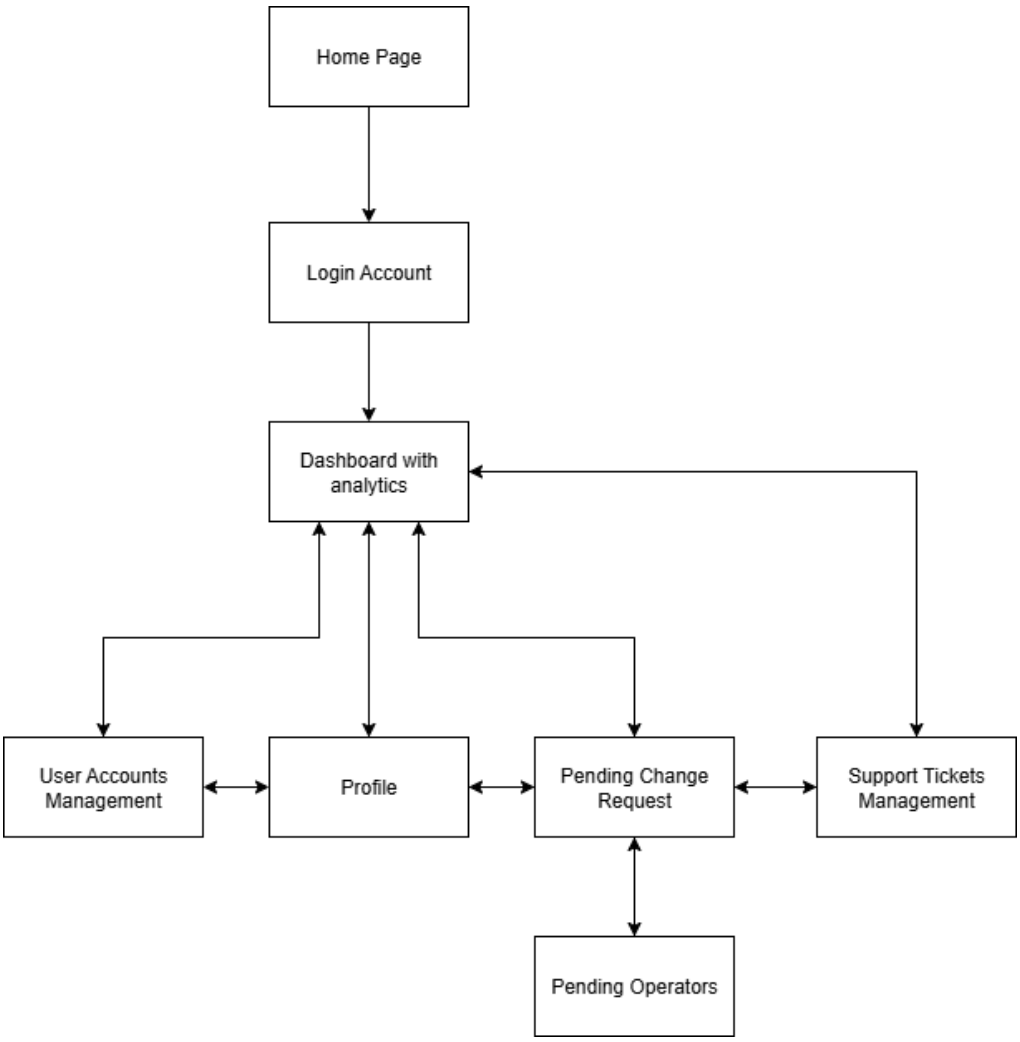


Figure 4.25: Interface flow diagram of the proposed system for admin

4.6 Initial Prototype

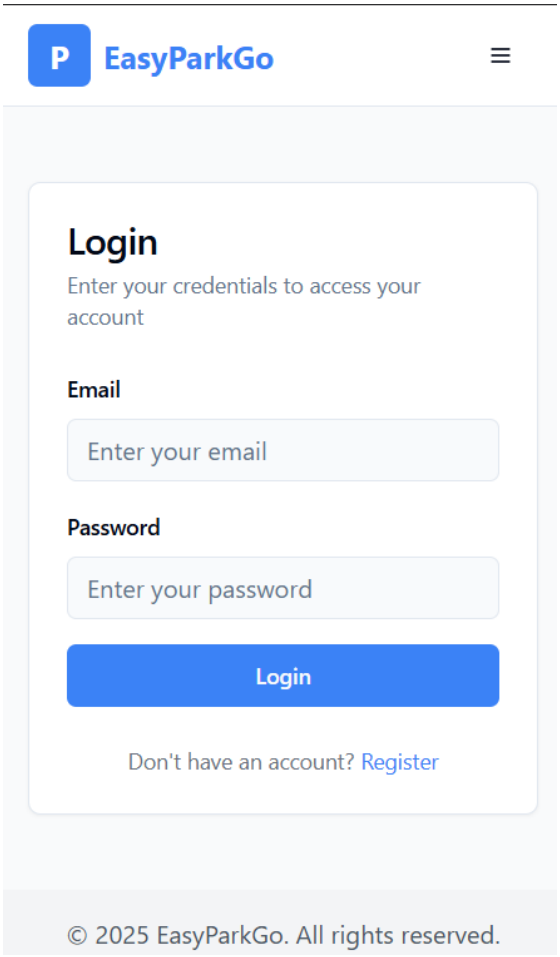


Figure 4.26: Login Page on mobile

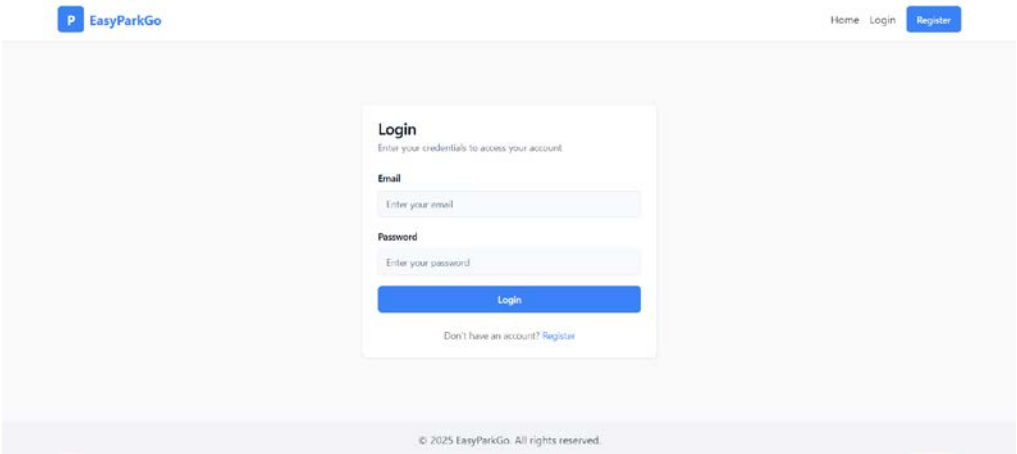


Figure 4.27: Login Page on desktop

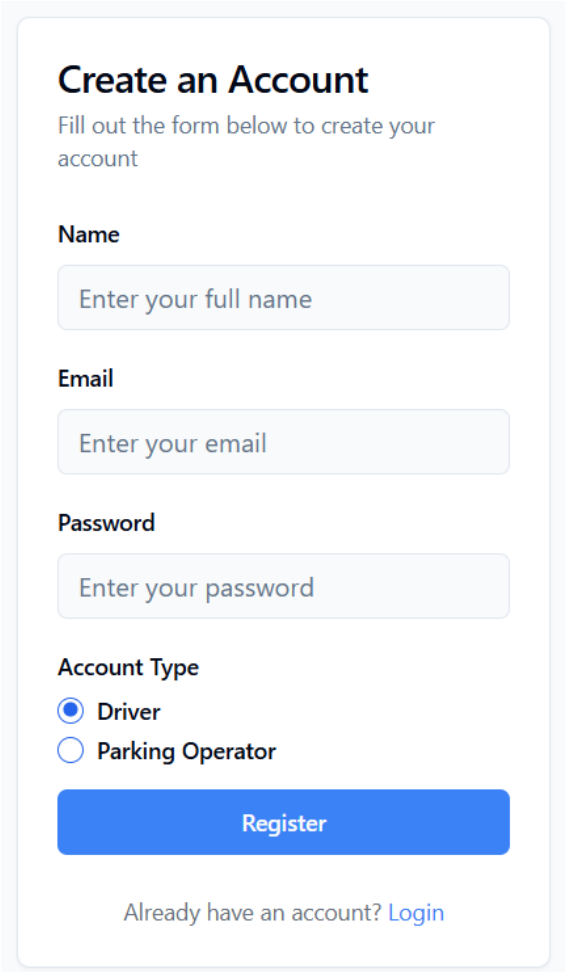


Figure 4.28: Register Page on mobile

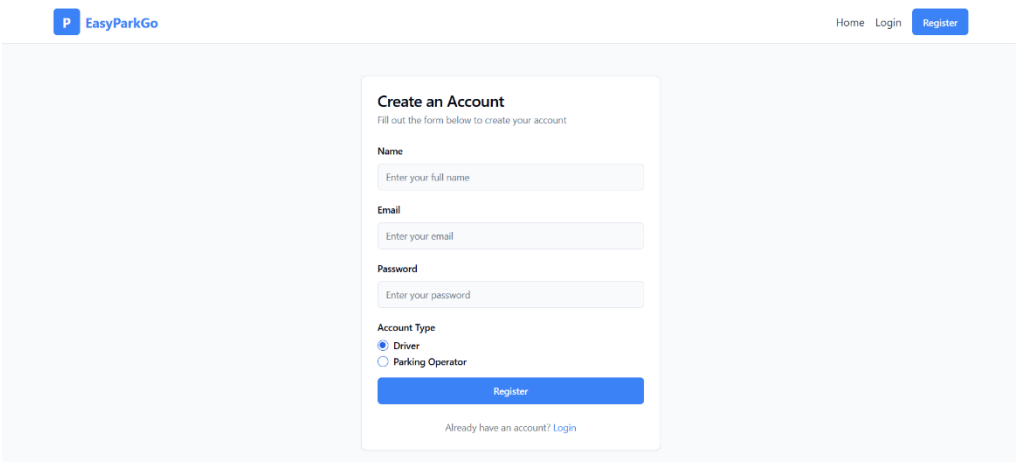


Figure 4.29: Register Page on desktop

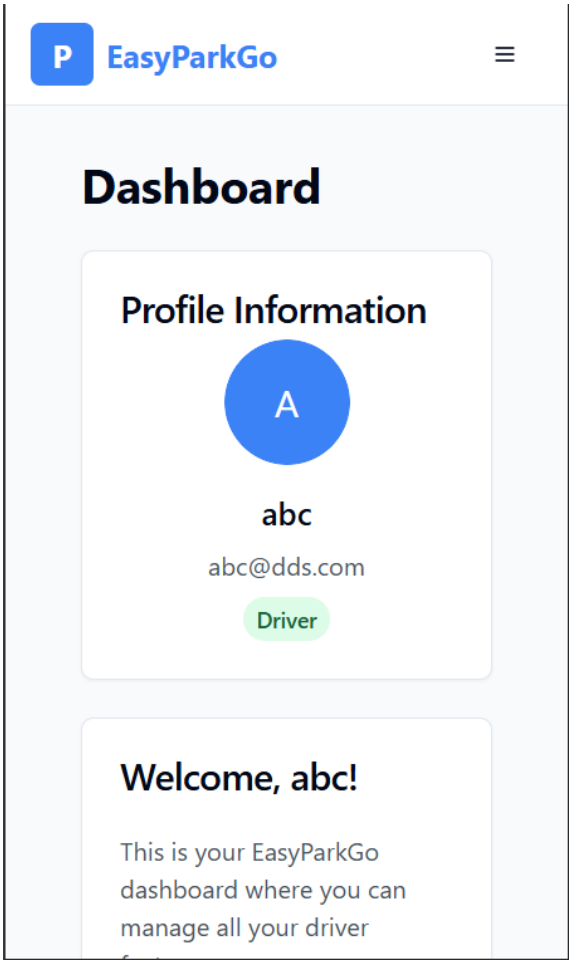


Figure 4.30: Driver Dashboard page on mobile

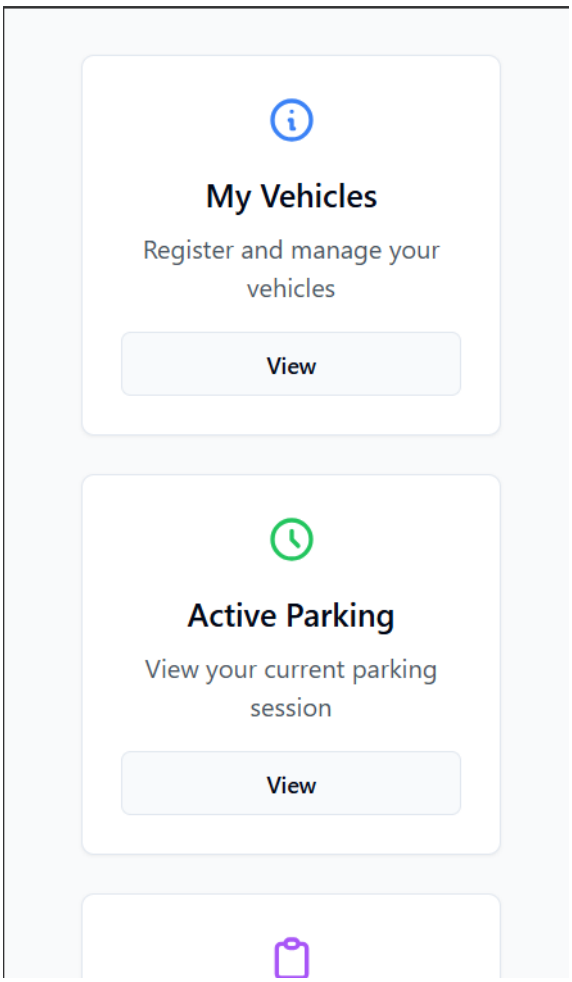


Figure 4.31: Continued Driver Dashboard page on mobile

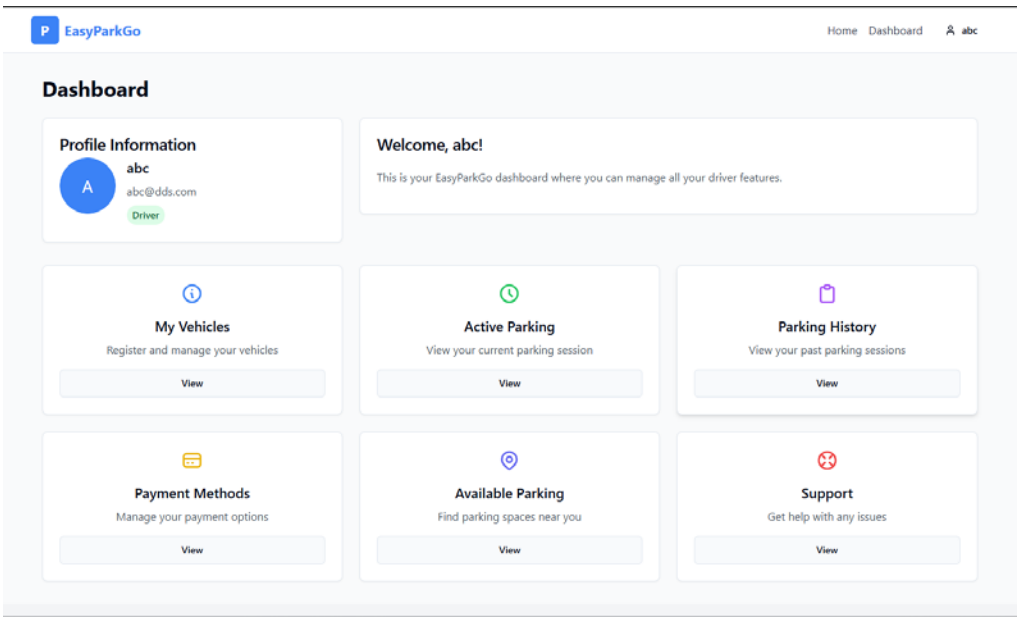


Figure 4.32: Driver Dashboard page on desktop

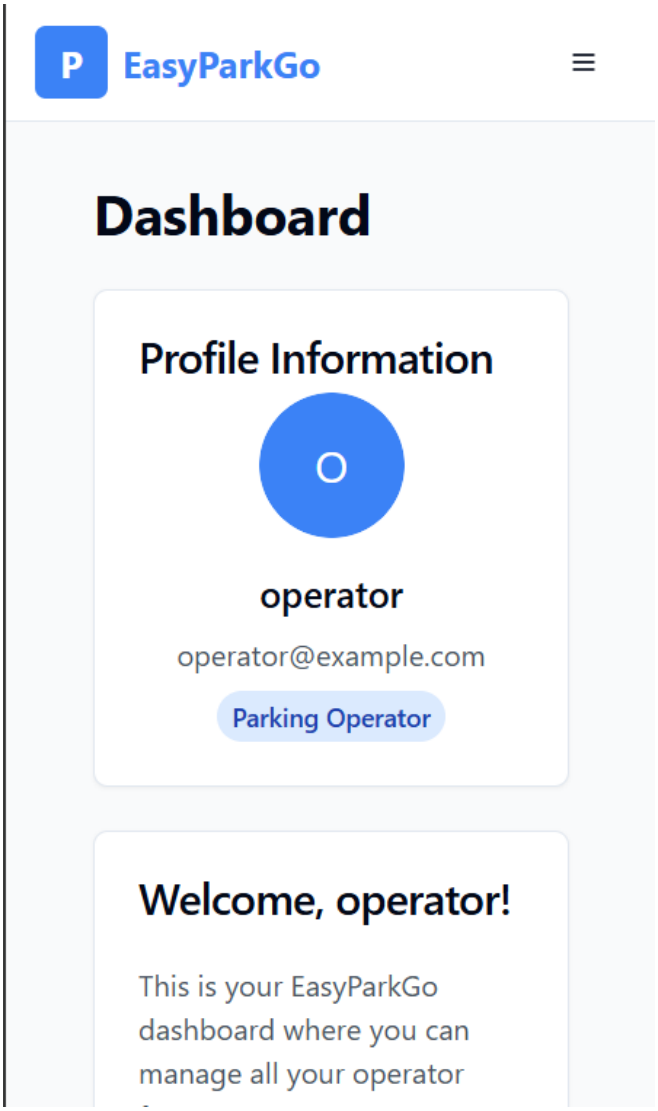


Figure 4.33: Operator Dashboard page on mobile

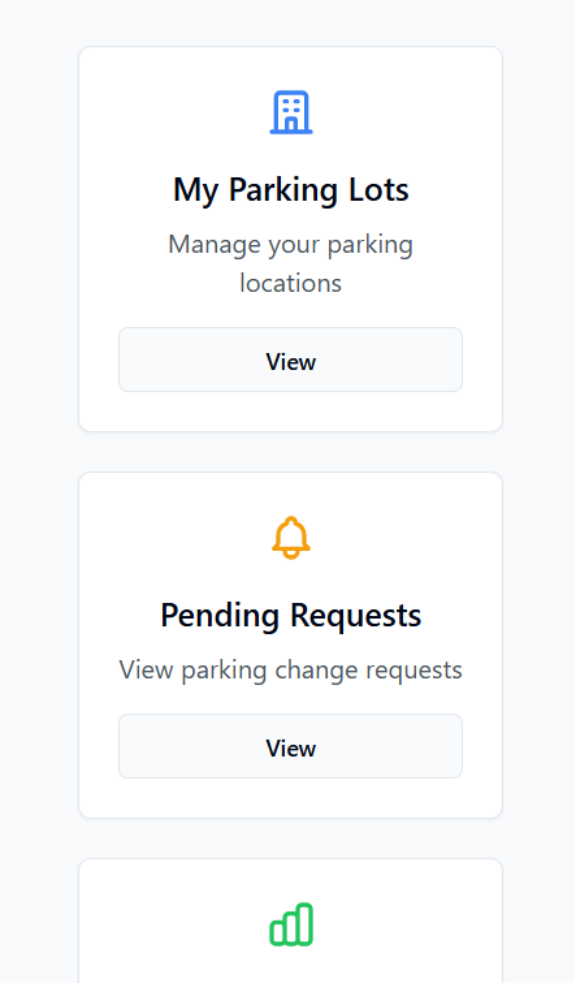


Figure 4.34: Continued Operator Dashboard page on mobile

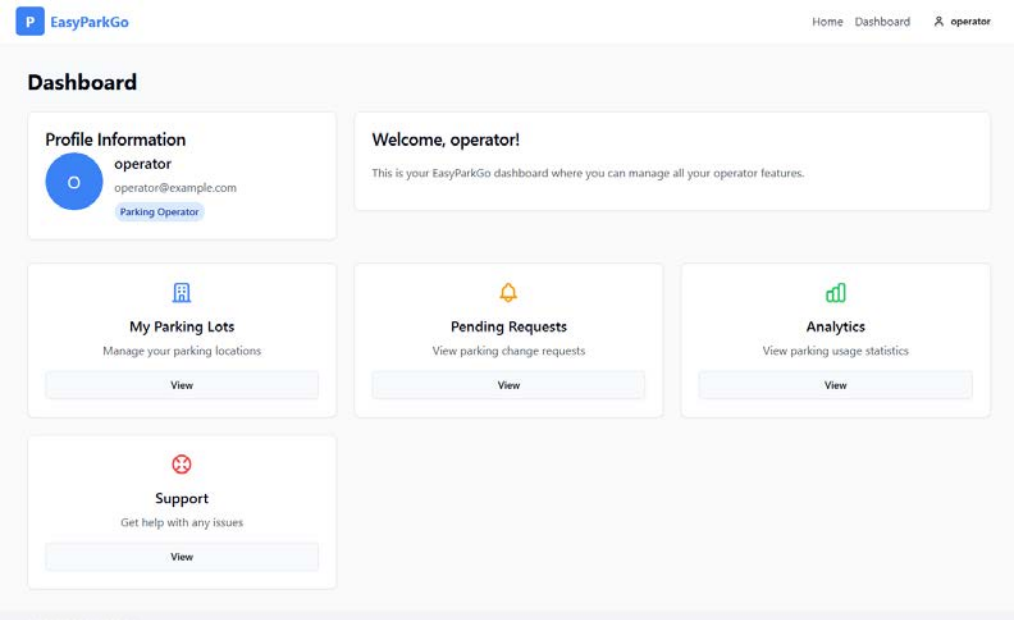


Figure 4.35: Operator Dashboard page on desktop

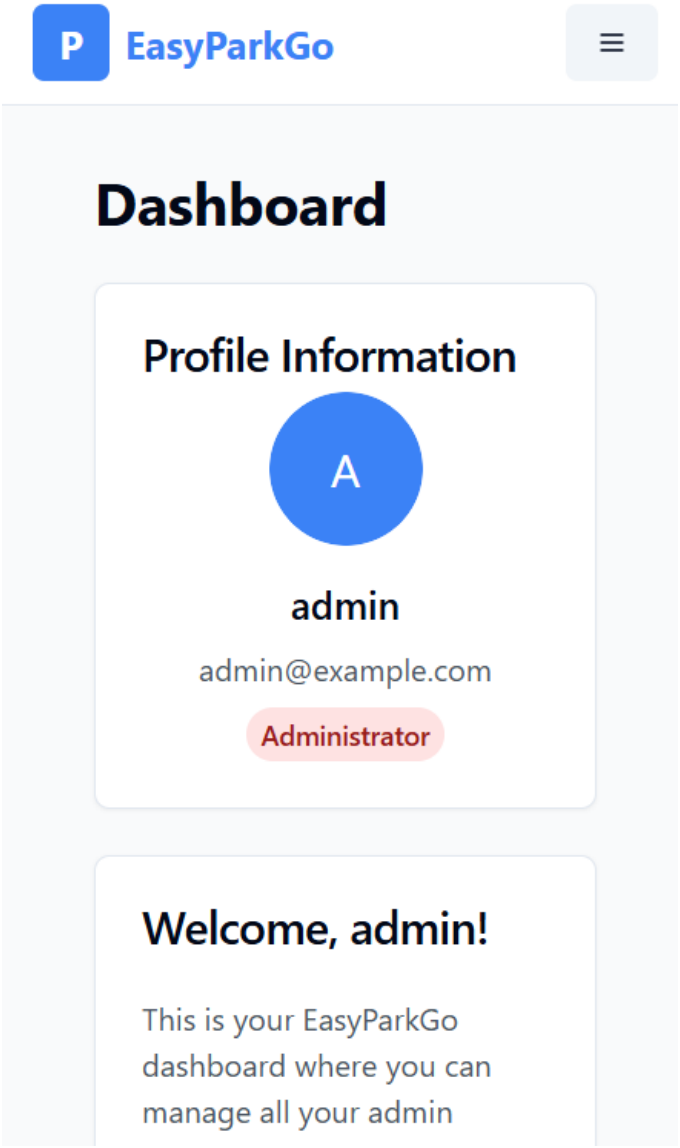


Figure 4.36: Admin Dashboard page on mobile

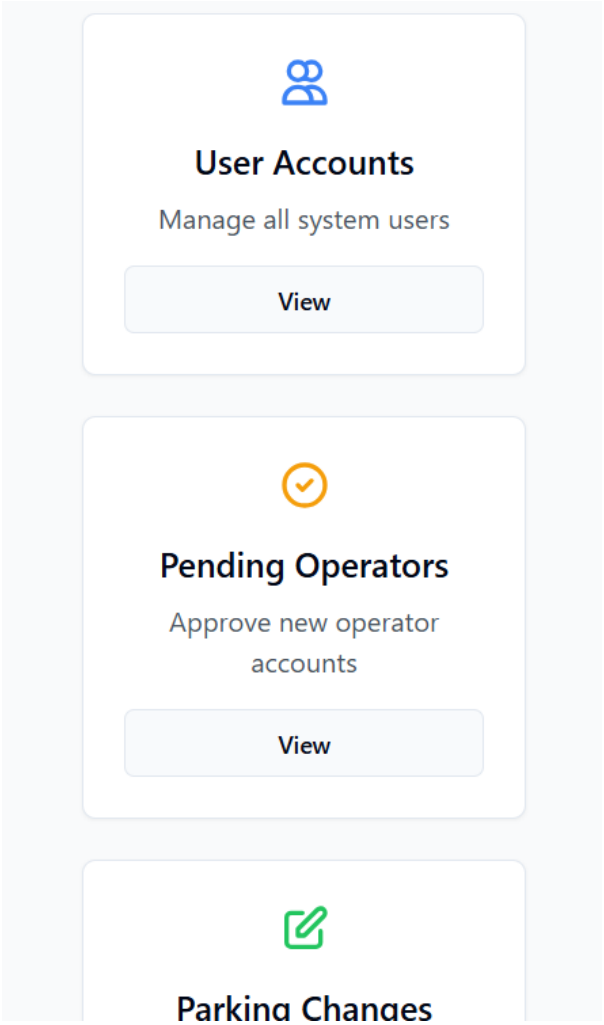


Figure 4.37: Continued Admin Dashboard page on mobile

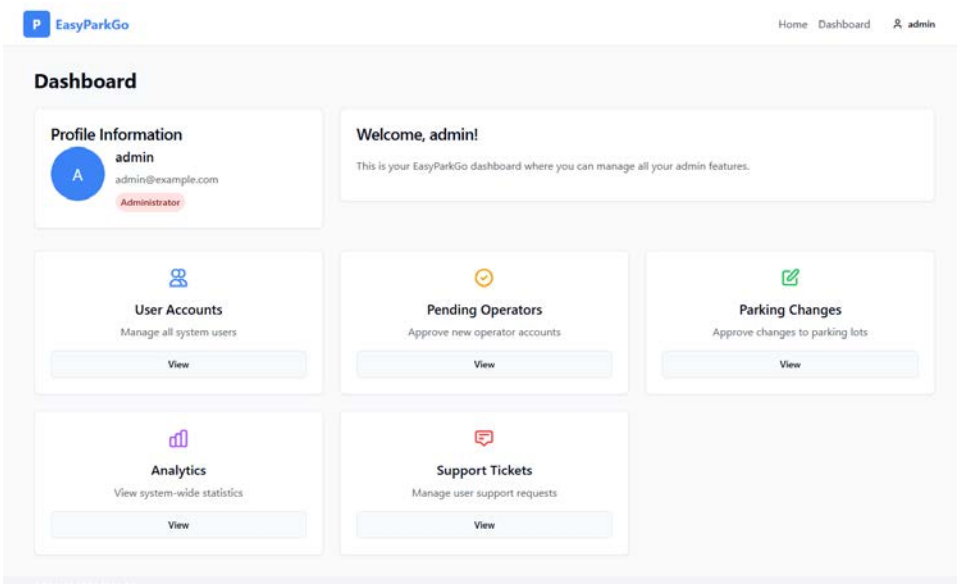


Figure 4.38: Admin Dashboard page on desktop

4.7 Preliminary run on Vehicle Detection and Segmentation

4.7.1 Overview

This section outlines a preliminary experiment conducted to assess the effectiveness of vehicle detection and segmentation using the YOLOv8 model family. Specifically, YOLOv8 and YOLOv8-Seg were employed to detect and segment vehicles in a custom dataset comprising 50 real-world images sourced online. The primary objective was to evaluate the models' capability to accurately identify and isolate vehicle instances under diverse conditions, including varying angles, lighting, and backgrounds. The results from this initial test serve as an early benchmark for detection accuracy, segmentation quality, and model responsiveness, guiding further refinement of the vehicle recognition pipeline for real-world applications.

4.7.2 Experimental Setup and Results

The preliminary experiment utilized the YOLOv8 and YOLOv8-Seg models to evaluate vehicle detection and segmentation performance. A total of 50 real-world images were manually collected from online sources, featuring various environments, lighting conditions, and vehicle angles to reflect practical usage scenarios. The models, pre-trained on the COCO dataset, were run using the official Ultralytics YOLOv8 implementation in Python. No additional fine-tuning was performed. Each image was processed individually to record several key metrics, which are the number of vehicles detected, the number of segmentation instances, the confidence scores for detections and segmentations, and the largest segment ratio, defined as the size of the largest segmented vehicle area relative to the image dimensions. Since there were no predefined or labeled data, the analysis focused on the models' raw output to evaluate their initial effectiveness in real-world conditions.

The models detected an average of 1.78 vehicles and produced 1.92 segmentations per image, indicating generally accurate detection with slight over-segmentation. The average largest segment covered 39.18% of the image area, suggesting effective focus on primary vehicle regions. Detection confidence averaged 63.42%, while segmentation confidence was slightly higher at 69.15%, reflecting moderate to strong model certainty. Overall, the

results indicate reliable initial performance, with room for further refinement through model tuning and dataset expansion.

```
Overall Statistics:
Total images processed: 50
Average detected vehicles per image: 1.78
Average segmentation count per image: 1.92
Average largest segment ratio: 39.18%
Average detection confidence: 63.42%
Average segmentation confidence: 69.15%
```

Figure 4.39: The result from the detection and segmentation.

CHAPTER 5
SYSTEM DESIGN

5.1 Introduction

This chapter describes the architecture design of the system, which includes the frontend, backend, and their communication.

5.2 System Architecture Design

The ParkPal system separates frontend and backend components for efficient operation. The frontend consists of mobile and web applications for drivers, parking operators, and admins, built with Expo, React Native, and React Native Web, allowing management of vehicles, parking sessions, payments, and support tickets.

The backend includes a Laravel PHP server, which handles CRUD operations on a PostgreSQL database, manages users, processes parking transactions, handles notifications, and integrates with Stripe for automatic payments. A Python/Uvicorn server performs AI-based license plate and vehicle attribute recognition using Gemini Flash 2.5, supporting driver vehicle registration and automatic parking fee processing.

NGROK is used for exposing backend servers during development but does not perform backend operations. In operation, frontend requests are processed by Laravel, interacting with the database, Stripe, and the Python server for AI tasks, then returning responses and notifications to users. This architecture ensures secure transactions, real-time updates, and automated parking fee collection.

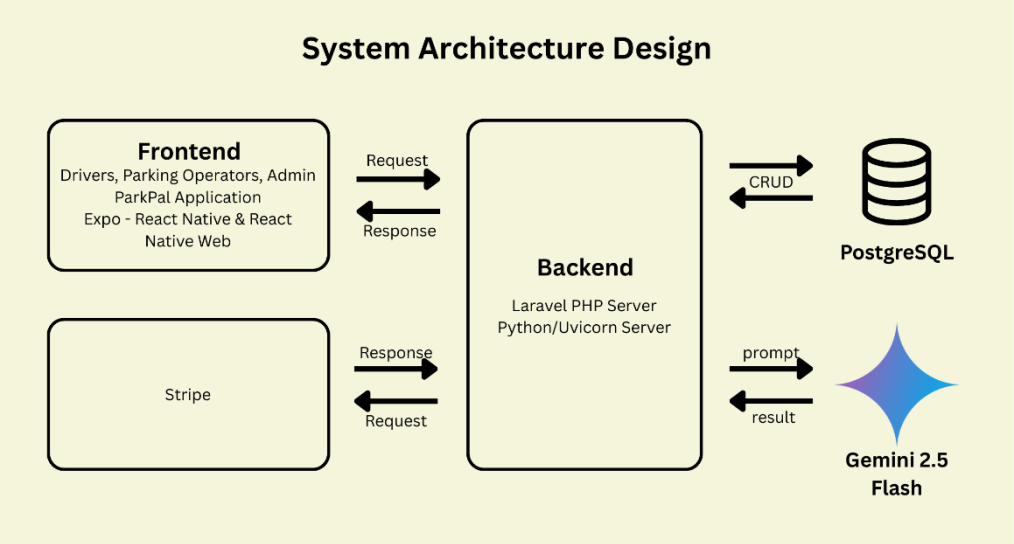


Figure 5.1: System Architecture Design.

5.2.1 Front-end Architecture

The frontend of the ParkPal application is developed using React Native and React Native Web, providing a consistent interface across mobile and web platforms. React Native components are dynamic elements that respond to state changes, user interactions, and component events, such as button clicks or text entry. The main entry point, index.js, registers the root component (App) using registerRootComponent from Expo, ensuring that the application environment is correctly initialized whether running in Expo Go or a native build. The App.js file contains the <NavigationContainer> and sets up the main stack navigator for routing across screens.

State management and authentication are handled via the <AuthProvider> context, while persistent navigation state is stored in AsyncStorage to restore user sessions across app launches. The app also integrates real-time notifications through Pusher and Echo, allowing drivers to receive updates on payment events immediately. Client-side rendering ensures that all UI components, such as <View>, <Text>, and <Image>, are executed on the device, providing a responsive and interactive experience. This architecture allows seamless interaction with backend services, efficient state handling, and real-time updates while maintaining a single codebase for multiple platforms.

In React Native, the UI is rendered on the client side, meaning it is executed directly on the mobile device. During installation, the necessary native code and bundled JavaScript components are deployed on the device. When the application is launched, the React Native runtime executes the JavaScript bundle, rendering UI components and enabling real-time interactions, such as vehicle registration, parking session management, and automatic payment transactions.

React Native communicates with the device's native components via a bridge, connecting the JavaScript thread (which executes the app logic) and the native thread (which handles rendering). Data is serialized in JSON format to pass between threads, providing a native-like experience while using a single codebase for multiple platforms. This architecture reduces the need to develop separate applications for iOS and Android and distributes rendering tasks to the client, lowering server workload.

The project uses Expo as a development framework to streamline bundling and deployment. Developers can connect a mobile device via a QR code or URL to load the JavaScript bundle directly from the development machine. Expo simplifies testing and reduces resource consumption by eliminating the need for virtual devices during development.

5.2.2 Back-end Architecture

The backend of ParkPal consists of a Laravel PHP server as the main backend, supported by a Python/Uvicorn server for AI processing. The Laravel server handles requests from both mobile and web applications, performing CRUD (Create, Read, Update, Delete) operations on a PostgreSQL database. It manages user accounts, parking sessions, notifications, and integrates with Stripe for payment processing.

The Python/Uvicorn server performs specialized AI tasks, including license plate and vehicle attribute recognition. There are two main flows for image processing. First, during driver vehicle registration, drivers upload vehicle images through the mobile app, which are sent to the Python server. The server communicates with Gemini Flash 2.5 to extract attributes such as license

plate, make, model, and color. The results are then returned to the Laravel backend for storage in PostgreSQL.

Second, for automatic parking fee processing, images of vehicles captured at parking lots are sent directly to the Python server for recognition via Gemini Flash 2.5. The processed results are returned to the Laravel backend to calculate parking fees, process payments through Stripe, and send notifications to drivers.

For development and mobile testing, NGROK is used to expose the backend servers to the internet, allowing devices to connect remotely. Overall, the backend processes HTTP requests from the frontend, interacts with the database, communicates with Stripe and the AI service, and returns responses to the frontend, ensuring real-time updates and efficient system operation.

5.3 Database Architecture

The system utilizes PostgreSQL as its primary RDBMS due to its robustness, scalability, and strong support for complex queries and data integrity. PostgreSQL is well-suited for handling structured data, relationships, and transactional operations, which are essential for managing users, vehicles, reservations, parking lots, and EV charger information in the application.

The database is designed following relational principles, ensuring that entities such as users, parking lots, EV chargers, and reservations are represented as separate tables with clearly defined relationships. For instance, a users table maintains personal and authentication details, while vehicles are linked to users through foreign key constraints. Similarly, reservations and ev_chargers tables are associated with users and parking locations to track bookings, availability, and occupancy in real time.

This relational design supports data consistency, integrity, and scalability. Constraints such as primary keys, foreign keys, and unique indices are applied to prevent invalid or duplicate data. Additionally, PostgreSQL’s support for advanced features like JSONB fields allows flexible storage for dynamic or semi-structured data, such as parking lot metadata or EV charger specifications, without compromising query performance.

By leveraging PostgreSQL, the system ensures reliable data management for real-time operations, complex filtering and searching, and future extensibility, enabling efficient handling of both transactional and analytical requirements for the parking and EV reservation platform.

5.3.1 Database Entity Relationship Diagram (ERD)

Figure 5.2 shows the Entity Relationship Diagram (ERD) for the system database. It illustrates the entities and their relationships, which form the foundation for the database schema implemented in PostgreSQL. The ERD defines how data is logically structured and interconnected within the system, ensuring referential integrity, data consistency, and efficient query performance.

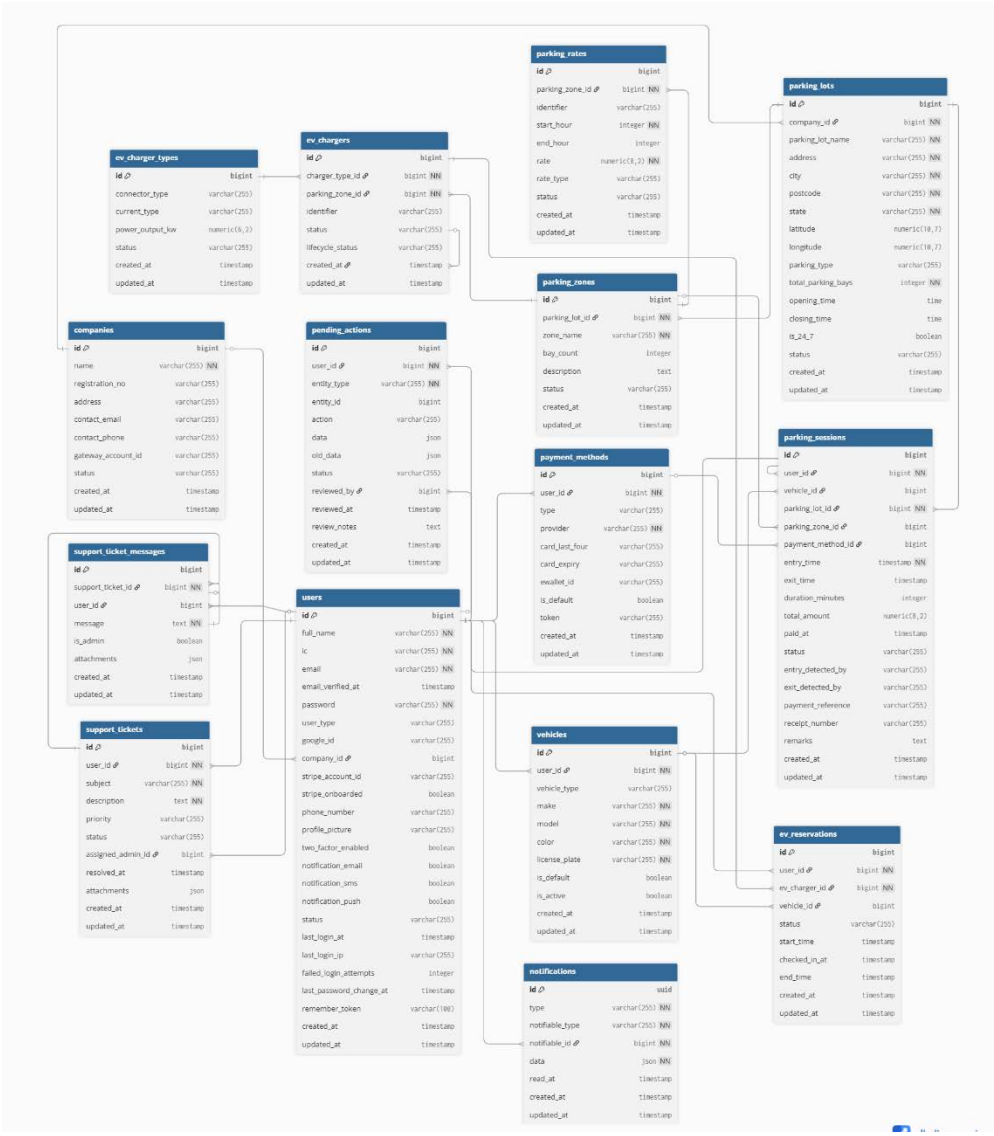


Figure 5.2: Entity Relationship Diagram for the System Database

The relationships among entities such as users, vehicles, parking sessions, and payments enable accurate tracking of parking activities, vehicle entries and exits, and corresponding financial transactions. Supporting entities such as notifications, support tickets, and pending actions provide functionality for user communication, issue management, and administrative approvals.

The database design follows normalization principles to minimize redundancy and improve data integrity, while maintaining flexibility for future system expansion. For example, modular entities such as EV chargers, companies, and payment gateways can be easily extended or modified without disrupting existing relationships. This relational structure ensures seamless integration between the Laravel backend and other system components, supporting reliable and scalable data operations throughout the parking management system.

5.3.2 Database Schema

The database schema defines the logical structure of the system's data as implemented in PostgreSQL. It specifies the tables, fields, data types, and relationships derived from the Entity Relationship Diagram (ERD).

Table 5.1: Users Schema

Column	**Type**	**Default**	**Constraints**	**Description**
id	bigint		NOT NULL	Primary key
full_name	varchar(255)		NOT NULL	User's full name
ic	varchar(255)		NOT NULL	Identity card number
email	varchar(255)		NOT NULL	Email address

email_verified_at	timestamp			Email verification timestamp
password	varchar(255)		NOT NULL	Hashed password
user_type	varchar(255)	`'driver'`	Must be `driver`, `parking_operator`, `admin`	Type of user
google_id	varchar(255)			Google OAuth ID
company_id	bigint			Linked company (if applicable)
stripe_account_id	varchar(255)			Stripe account ID
stripe_onboarded	boolean	`false`		Stripe onboarding status
phone_number	varchar(255)			Contact number
profile_picture	varchar(255)			Profile image URL
two_factor_enabled	boolean	`false`		2FA enabled flag
notification_email	boolean	`true`		Email notifications enabled
notification_sms	boolean	`false`		SMS notifications enabled

notification_push	boolean	`false`		Push notifications enabled
status	varchar(255)	`'pending_setup'`	Must be `pending_setup`, `pending_approval`, `active`, `inactive`, `rejected`, `suspended`, `deactivated`	Account status
last_login_at	timestamp			Last login timestamp
last_login_ip	varchar(255)			Last login IP address
failed_login_attempts	integer	`0`		Count of failed login attempts
last_password_change_at	timestamp			Timestamp of last password change
remember_token	varchar(100)			Remember me token
created_at	timestamp			Creation timestamp
updated_at	timestamp			Last update timestamp

Table 5.2: Companies Schema

Column	Type	Default	Constraints	Description
id	bigint		NOT NULL	Primary key
name	varchar(255)		NOT NULL	Company name
registration_no	varchar(255)			Registration number
address	varchar(255)			Company address
contact_email	varchar(255)			Contact email
contact_phone	varchar(255)			Contact phone
gateway_account_id	varchar(255)			Payment gateway account ID
status	varchar(255)	`'pending'`	Must be one of `pending`, `verified`, `rejected`, `suspended`, `deactivated`	Company account status
created_at	timestamp			Creation timestamp
updated_at	timestamp			Last update timestamp

Table 5.3: EV Charger Types Schema

Column	Type	Default	Constraints	Description
id	bigint		NOT NULL	Primary key
connector_type	varchar(255)		Must be `Type 1`, `Type 2`, `CHAdeMO`, `CCS`, `Tesla`	Type of charger connector
current_type	varchar(255)		Must be `AC` or `DC`	Type of electric current
power_output_kw	numeric(6,2)			Power output in kW
status	varchar(255)	`'active'`	Must be `active` or `inactive`	Charger type status
created_at	timestamp			Creation timestamp
updated_at	timestamp			Last update timestamp

Table 5.4: EV Chargers Schema

Column	Type	Default	Constraints	Description
id	bigint		NOT NULL	Primary key
charger_type_id	bigint		NOT NULL	Foreign key to `ev_charger_types`
parking_zone_id	bigint		NOT NULL	Foreign key to parking zone
identifier	varchar(255)			Charger identifier
status	varchar(255)	`'available'`	Must be `available`, `reserved`, `in_use`, `out_of_order`	Current status of charger
lifecycle_status	varchar(255)	`'active'`	Must be `active` or `inactive`	Lifecycle status
created_at	timestamp			Creation timestamp
updated_at	timestamp			Last update timestamp

Table 5.5: EV Reservations Schema

Column	Type	Default	Constraints	Description
id	bigint		NOT NULL	Primary key
user_id	bigint		NOT NULL	User making the reservation
ev_charger_id	bigint		NOT NULL	Reserved charger
vehicle_id	bigint			Vehicle being charged
status	varchar(255)	`'reserved'`	Must be `reserved`, `active`, `completed`, `cancelled`, `expired`	Reservation status
start_time	timestamp			Start time of reservation
checked_in_at	timestamp			Check-in timestamp
end_time	timestamp			End time
created_at	timestamp			Creation timestamp
updated_at	timestamp			Last update timestamp

Table 5.6: Notifications Schema

Column	Type	Default	Constraints	Description
id	uuid		NOT NULL	Primary key
type	varchar(255)		NOT NULL	Type of notification
notifiable_type	varchar(255)		NOT NULL	Model type being notified
notifiable_id	bigint		NOT NULL	ID of the notified entity
data	json		NOT NULL	Notification content
read_at	timestamp			Timestamp when notification was read
created_at	timestamp			Creation timestamp
updated_at	timestamp			Last update timestamp

Table 5.7: Parking Lots Schema

Column	Type	Default	Constraints	Description
id	bigint		NOT NULL	Primary key
company_id	bigint		NOT NULL	Owning company
parking_lot_name	varchar(255)		NOT NULL	Name of parking lot
address	varchar(255)		NOT NULL	Street address
city	varchar(255)		NOT NULL	City
postcode	varchar(255)		NOT NULL	Postal code
state	varchar(255)		NOT NULL	State
latitude	numeric(10,7)			Latitude coordinate
longitude	numeric(10,7)			Longitude coordinate
parking_type	varchar(255)	`'open-air'`	Must be `basement`, `multi-storey`, `open-air`, `valet`	Type of parking
total_parking_bays	integer		NOT NULL	Total number of bays
opening_time	time			Opening hour

closing_time	time			Closing hour
is_24_7	boolean	`false`		24/7 availability
status	varchar(255)	`'pending'`	Must be `pending`, `approved`, `rejected`, `suspended`	Lot status
created_at	timestamp			Creation timestamp
updated_at	timestamp			Last update timestamp

Table 5.8: Parking Zones Schema

Column	**Type**	**Default**	**Constraints**	**Description**
id	bigint		NOT NULL	Primary key
parking_lot_id	bigint		NOT NULL	Related parking lot
zone_name	varchar(255)		NOT NULL	Zone name
bay_count	integer	`0`		Number of bays in zone
description	text			Optional description
status	varchar(255)	`'active'`	Must be `active` or `inactive`	Zone status
created_at	timestamp			Creation timestamp
updated_at	timestamp			Last update timestamp

Table 5.9: Parking Rates Schema

Column	Type	Default	Constraints	Description
id	bigint		NOT NULL	Primary key
parking_zone_id	bigint		NOT NULL	Zone for this rate
identifier	varchar(255)			Rate identifier
start_hour	integer		NOT NULL	Start hour of rate
end_hour	integer			End hour of rate
rate	numeric(8,2)		NOT NULL	Rate amount
rate_type	varchar(255)	`'weekday'`	Must be `weekday`, `weekend`, `holiday`	Type of rate
status	varchar(255)	`'active'`	Must be `active` or `inactive`	Rate status
created_at	timestamp			Creation timestamp
updated_at	timestamp			Last update timestamp

Table 5.10: Parking Sessions Schema

Column	Type	Default	Constraints	Description
id	bigint		NOT NULL	Primary key
user_id	bigint		NOT NULL	User who parked
vehicle_id	bigint			Vehicle used
parking_lot_id	bigint		NOT NULL	Parking lot
parking_zone_id	bigint			Zone parked in
payment_method_id	bigint			Payment method used
entry_time	timestamp		NOT NULL	Entry timestamp
exit_time	timestamp			Exit timestamp
duration_minutes	integer			Duration of parking in minutes
total_amount	numeric(8,2)			Total paid
paid_at	timestamp			Payment timestamp
status	varchar(255)	`'ongoing'`	Must be `ongoing`, `completed`, `cancelled`, `failed`	Session status

entry_detected_by	varchar(255)			Sensor/method detecting entry
exit_detected_by	varchar(255)			Sensor/method detecting exit
payment_reference	varchar(255)			Payment reference ID
receipt_number	varchar(255)			Receipt number
remarks	text			Additional notes
created_at	timestamp			Creation timestamp
updated_at	timestamp			Last update timestamp

Table 5.11: Vehicles Schema

Column	Type	Default	Constraints	Description
id	bigint		NOT NULL	Primary key
user_id	bigint		NOT NULL	Owner user ID
vehicle_type	varchar(255)	`'car'`	Must be `car`, `motorcycle`, `truck`	Type of vehicle
make	varchar(255)		NOT NULL	Vehicle brand
model	varchar(255)		NOT NULL	Vehicle model
color	varchar(255)		NOT NULL	Vehicle color
license_plate	varchar(255)		NOT NULL	Registration plate
is_default	boolean	`false`		Default vehicle flag
is_active	boolean	`true`		Active vehicle flag
created_at	timestamp			Creation timestamp
updated_at	timestamp			Last update timestamp

Table 5.12: Payment Methods Schema

Column	Type	Default	Constraints	Description
id	bigint		NOT NULL	Primary key
user_id	bigint		NOT NULL	Owner user ID
type	varchar(255)		Must be `card`, `ewallet`, `bank`	Payment method type
provider	varchar(255)		NOT NULL	Provider name
card_last_four	varchar(255)			Last 4 digits of card (if card)
card_expiry	varchar(255)			Card expiry date
ewallet_id	varchar(255)			E-wallet identifier
is_default	boolean	`false`		Default payment method
token	varchar(255)			Tokenized payment info
created_at	timestamp			Creation timestamp
updated_at	timestamp			Last update timestamp

Table 5.13: Support Tickets Schema

Column	Type	Default	Constraints	Description
id	bigint		NOT NULL	Primary key
user_id	bigint		NOT NULL	User who created ticket
subject	varchar(255)		NOT NULL	Ticket subject
description	text		NOT NULL	Detailed description
priority	varchar(255)	`'medium'`	Must be `low`, `medium`, `high`, `urgent`	Ticket priority
status	varchar(255)	`'open'`	Must be `open`, `in_progress`, `resolved`, `closed`	Ticket status
assigned_admin_id	bigint			Admin assigned to ticket
resolved_at	timestamp			Resolution timestamp
attachments	json			Attached files
created_at	timestamp			Creation timestamp
updated_at	timestamp			Last update timestamp

Table 5.14: Support Ticket Messages Schema

Column	Type	Default	Constraints	Description
id	bigint		NOT NULL	Primary key
support_ticket_id	bigint		NOT NULL	Linked ticket
user_id	bigint			Sender user ID
message	text		NOT NULL	Message content
is_admin	boolean	`false`		Flag if admin sent
attachments	json			Optional attachments
created_at	timestamp			Creation timestamp
updated_at	timestamp			Last update timestamp

Table 5.15: Pending Actions Schema

Column	Type	Default	Constraints	Description
id	bigint		NOT NULL	Primary key
user_id	bigint		NOT NULL	Action initiator
entity_type	varchar(255)		NOT NULL	Model type affected
entity_id	bigint			ID of affected entity
action	varchar(255)		Must be `create`, `update`, `deactivate`	Action type
data	json			New data
old_data	json			Previous data
status	varchar(255)	`'pending'`	Must be `pending`, `approved`, `rejected`	Action status
reviewed_by	bigint			Reviewer ID
reviewed_at	timestamp			Review timestamp
review_notes	text			Reviewer notes
created_at	timestamp			Creation timestamp
updated_at	timestamp			Last update timestamp

5.3.3 Collection Description

This section provides detailed descriptions of the database tables used in the system. Each table is explained in terms of its purpose.

Table 5.16: Collections Description Table

Table Name	**Description / Use**
`users`	Stores all registered users, including drivers, parking operators, and admins, along with their authentication and notification preferences.
`companies`	Stores information about companies that manage parking lots, including contact info and account status.
`company_change_requests`	Tracks requests by users to update company information, with review and approval workflow.
`ev_charger_types`	Stores types of EV chargers, their connector type, power, current type, and status.
`ev_chargers`	Represents individual EV chargers in parking zones, linked to charger types and their current availability.
`ev_reservations`	Records reservations of EV chargers by users, including start/end times and status.
`notifications`	Stores notifications sent to users or other entities, including type, content, and read status.
`parking_lots`	Contains information about parking lots, including company ownership, location, type, capacity, and operational hours.
`parking_zones`	Subdivisions of parking lots (zones) with bay counts, descriptions, and status.

parking_rates	Stores rate information for parking zones, including time slots, rate amount, and type (weekday/weekend/holiday).
parking_sessions	Tracks parking usage by users, including entry/exit times, duration, payment info, and status.
vehicles	Stores details about user vehicles, including type, make/model, color, license plate, and active/default status.
payment_methods	Stores user payment methods, including card, e-wallet, or bank, with provider info and default flag.
payment_gateways	Stores company-linked payment gateway information, API credentials, webhook settings, and verification status.
support_tickets	Tracks support tickets submitted by users, including subject, description, priority, status, and assigned admin.
support_ticket_messages	Stores individual messages within support tickets, including sender, message content, and attachments.
pending_actions	Tracks user-initiated actions that require admin review, including create/update/deactivate operations.
password_reset_tokens	Stores tokens for resetting user passwords, linked to user email and creation timestamp.

5.4 Data Flow Diagram

5.4.1 Context Diagram

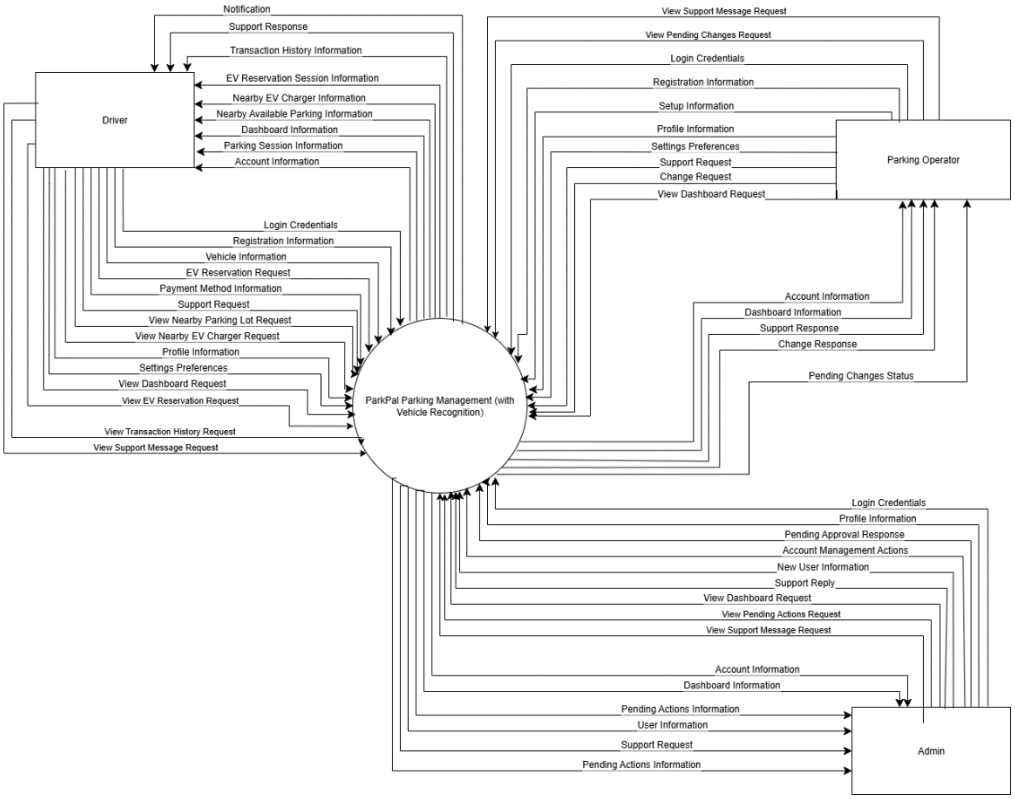


Figure 5.3: Context Diagram

5.4.2 DFD Level-0 Diagram

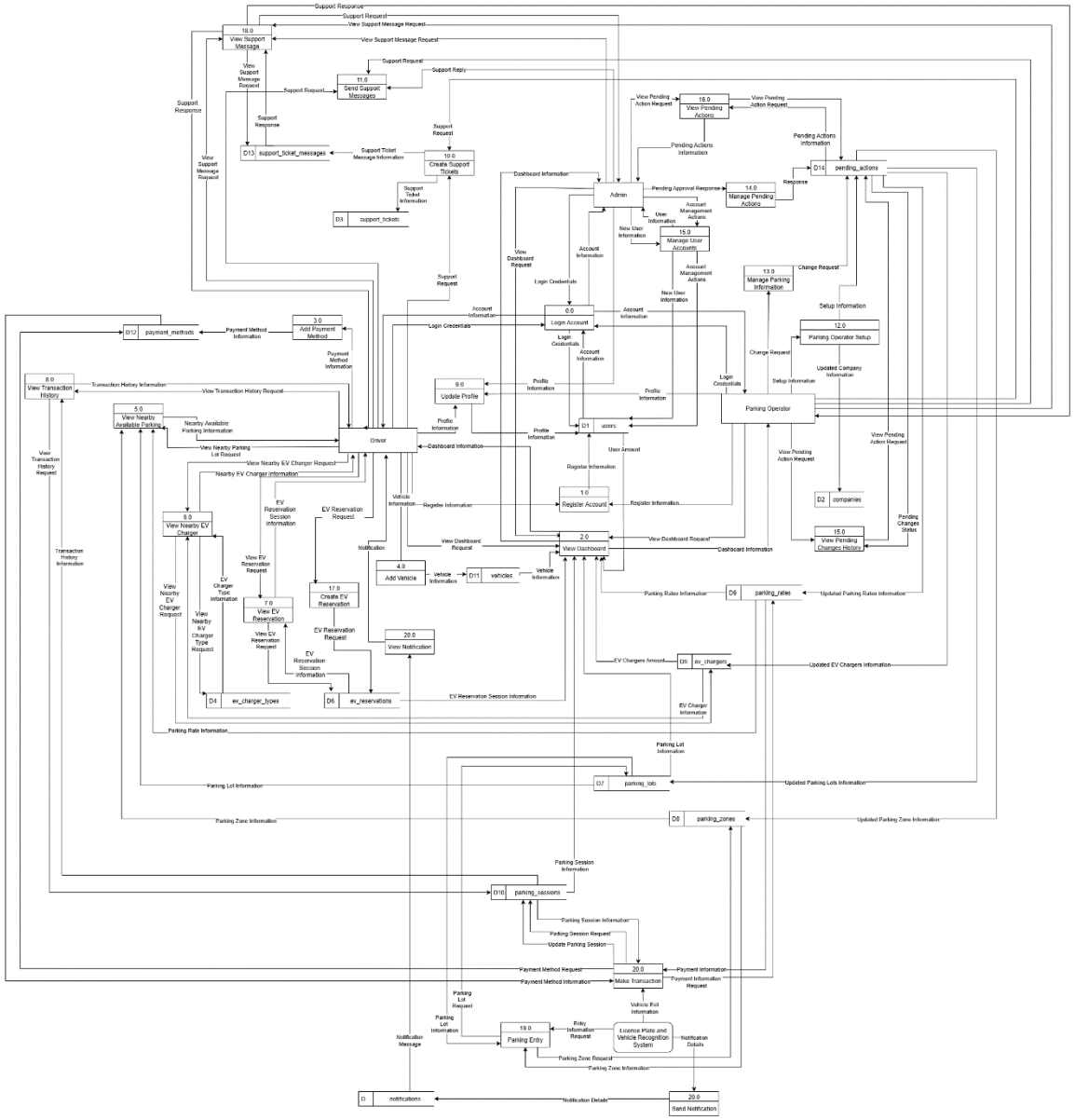


Figure 5.4: DFD Level 0 Diagram

5.5 Activity Diagram

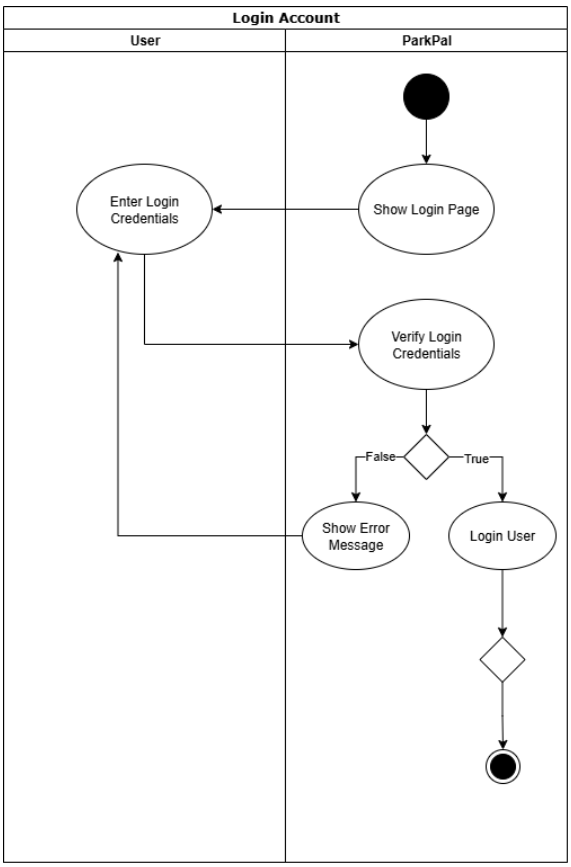


Figure 5.5: Activity Diagram of Login Account

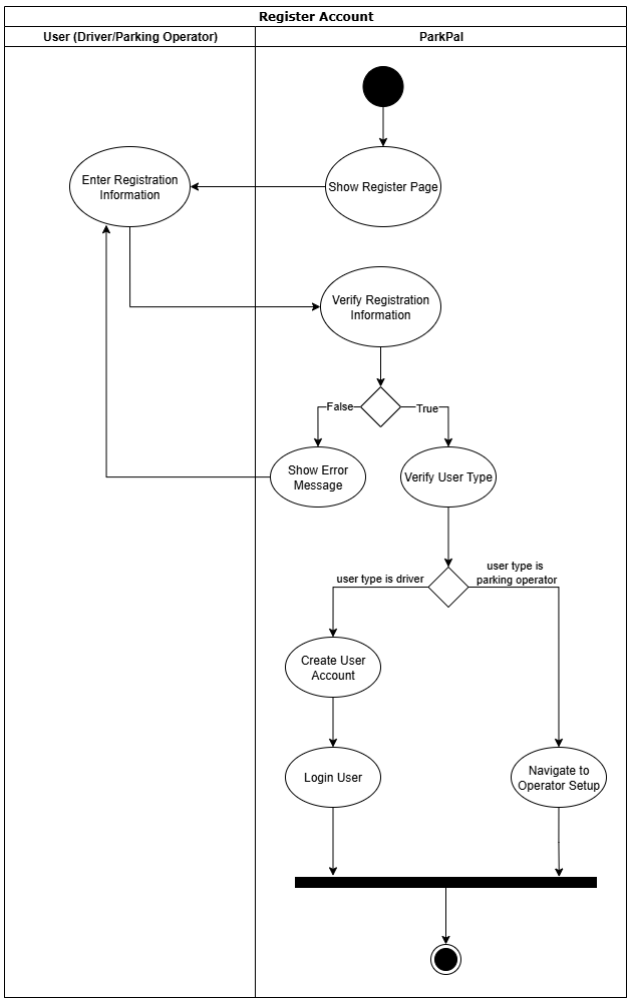


Figure 5.6: Activity Diagram of Register Account

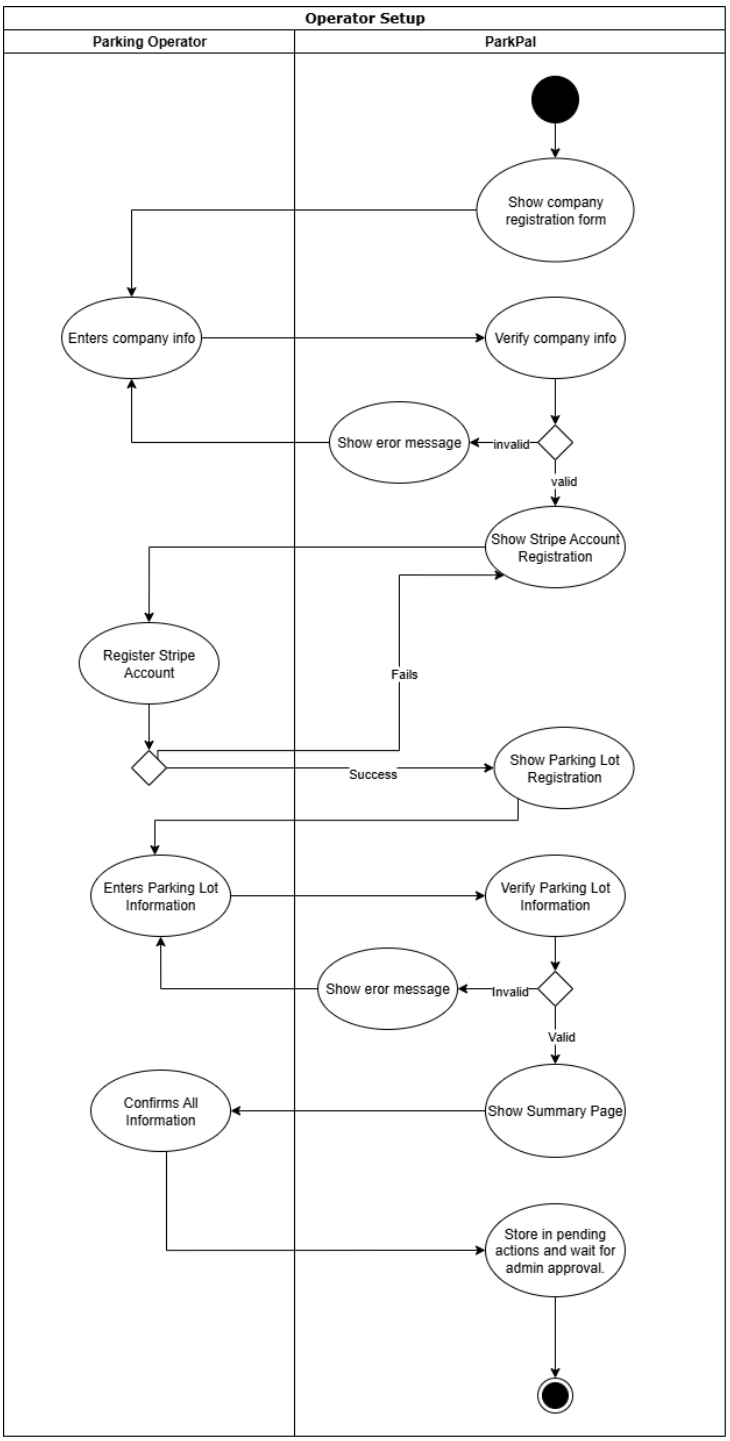


Figure 5.7: Activity Diagram of Operator Setup

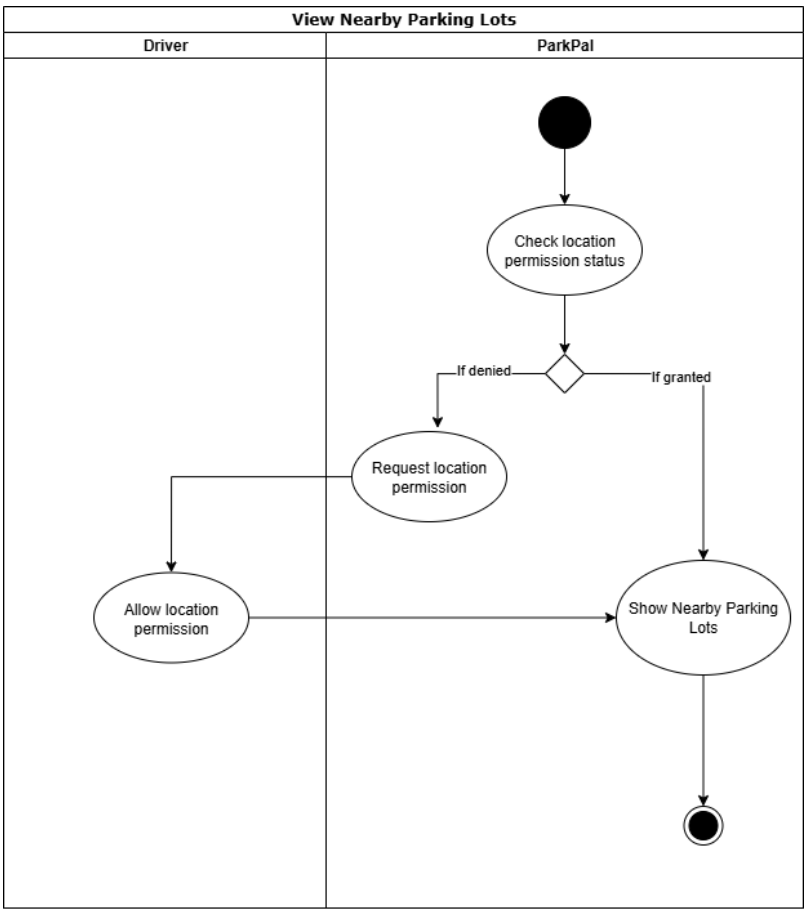


Figure 5.8: Activity Diagram of View Nearby Parking Lots

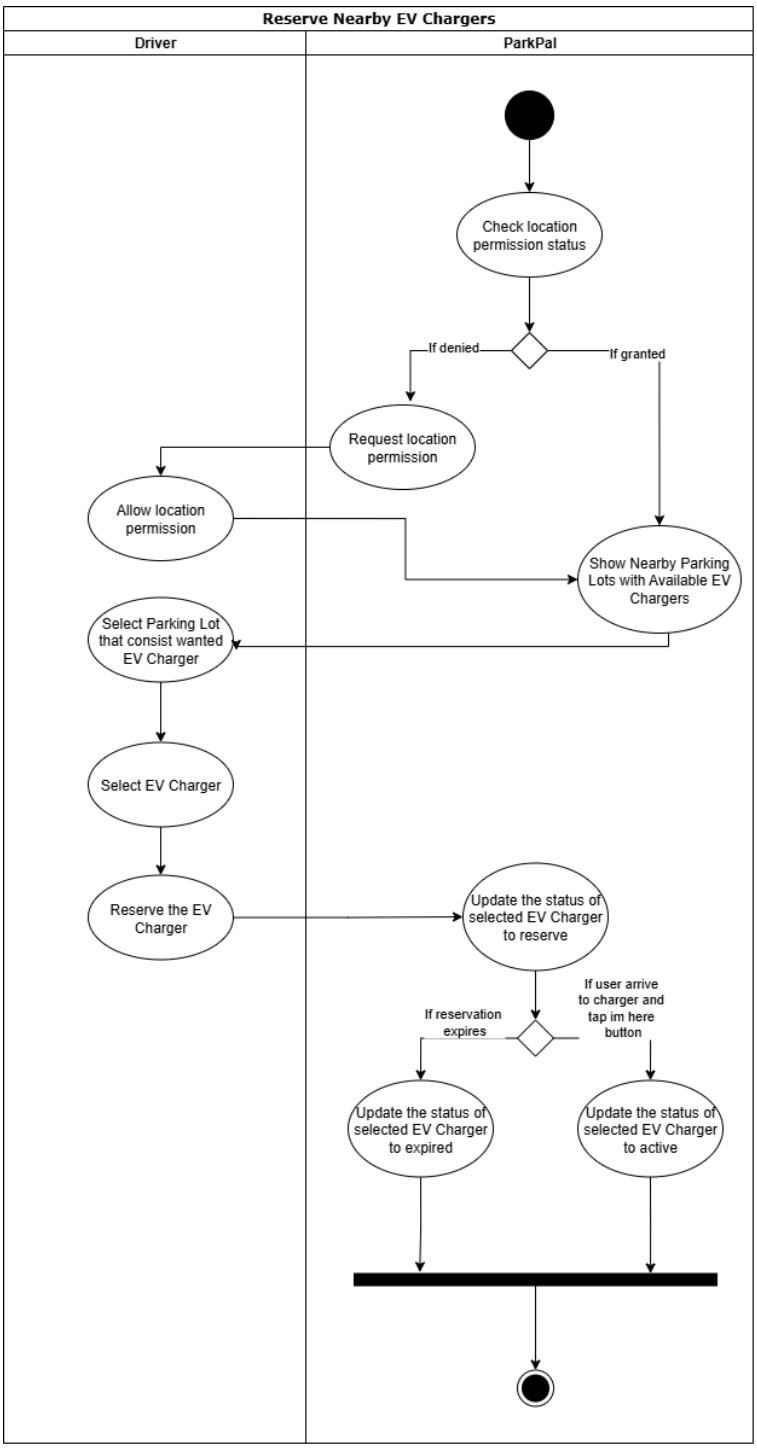


Figure 5.9: Activity Diagram of Reserve Nearcy EV Chargers

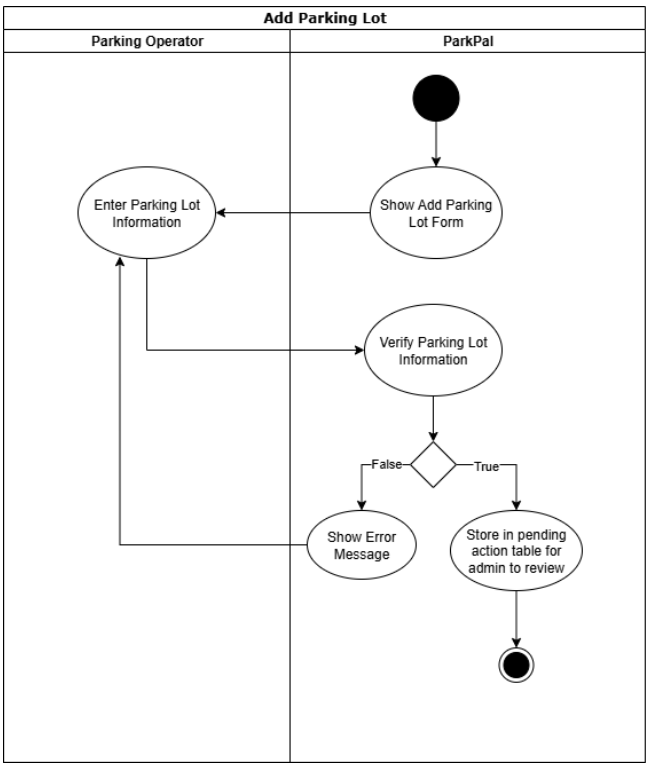


Figure 5.10: Activity Diagram of Add Parking Lot

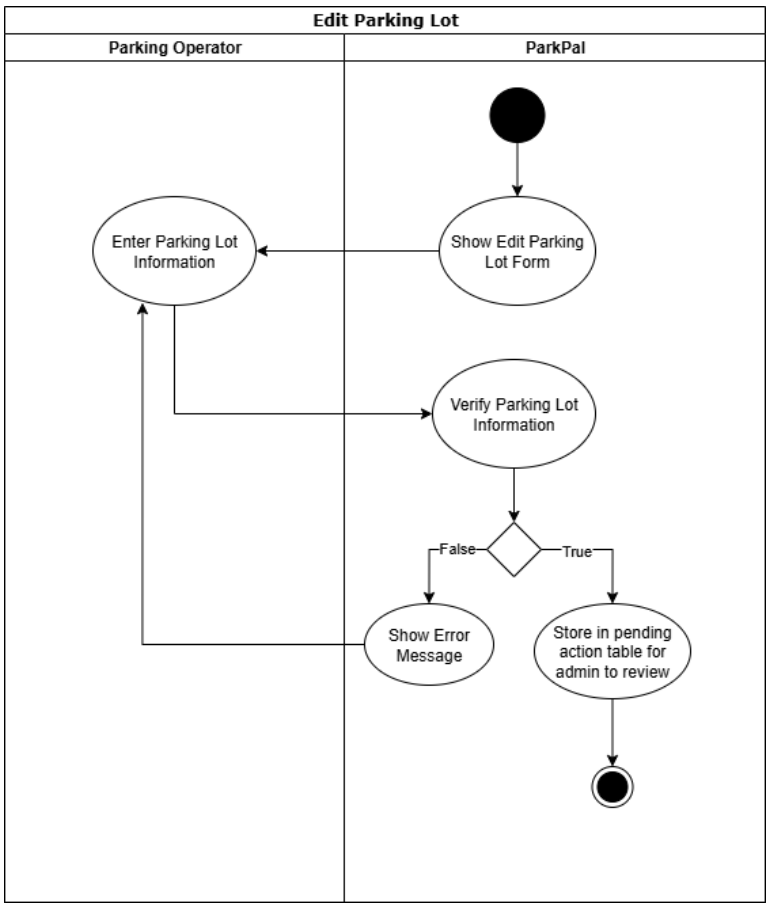


Figure 5.11: Activity Diagram of Edit Parking Lot

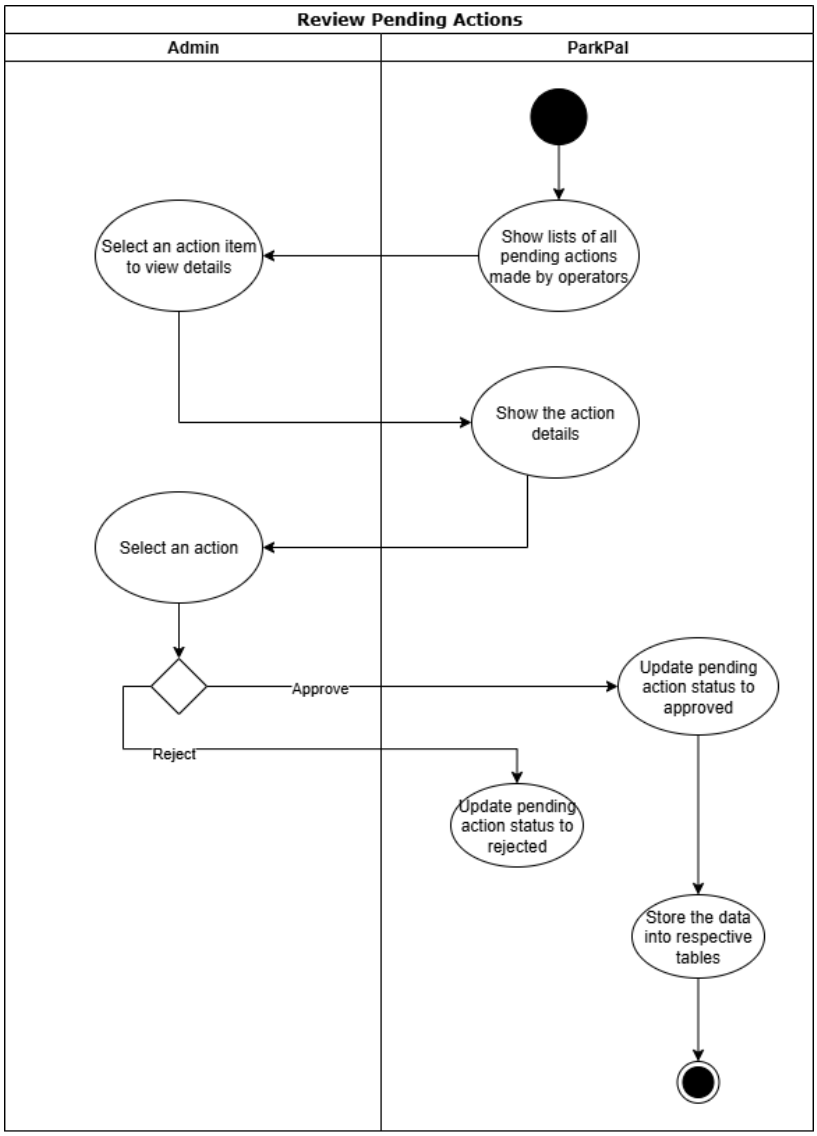


Figure 5.12: Activity Diagram of Review Pending Actions

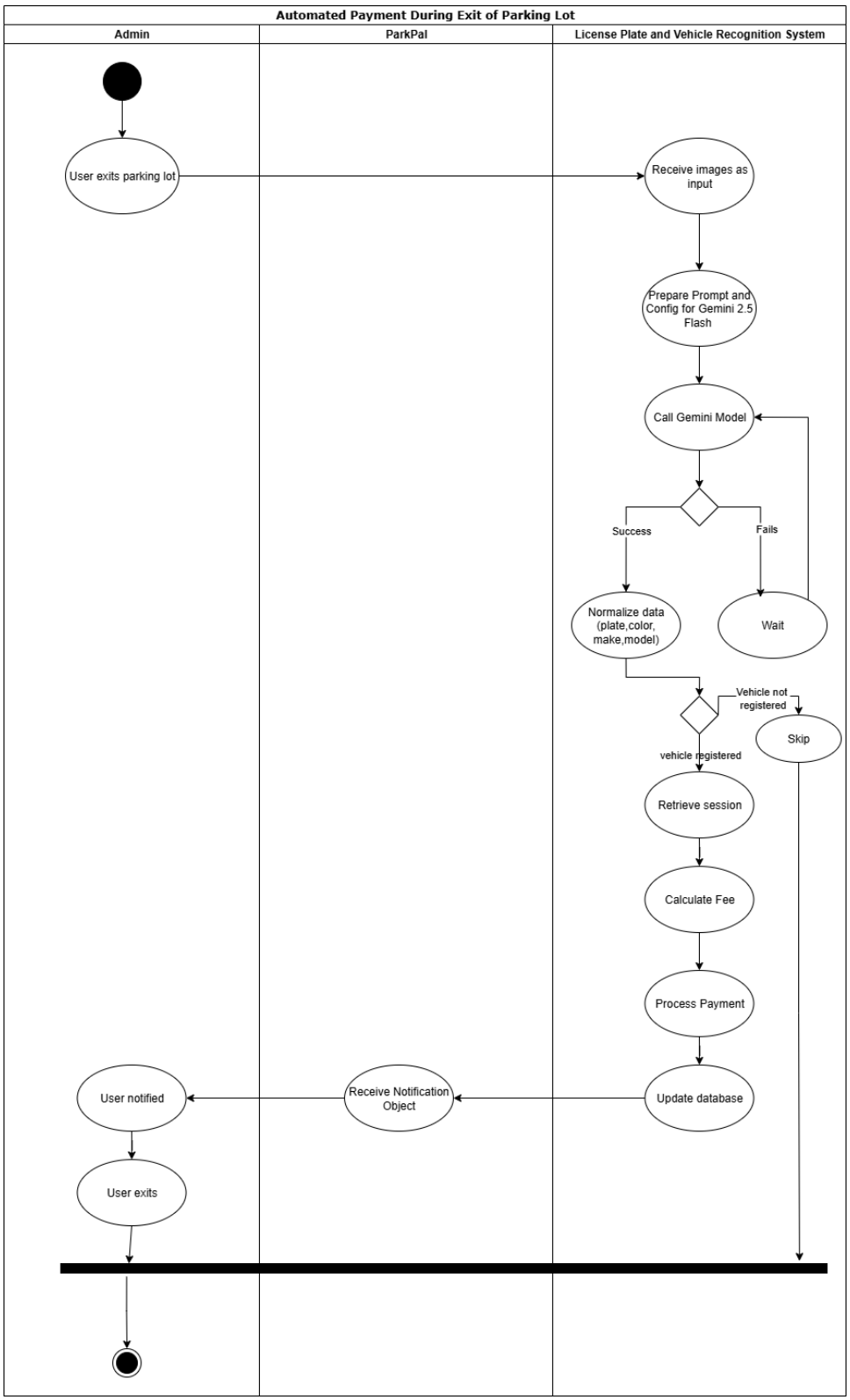


Figure 5.13: Activity Diagram of Automated Payment During Exit of Parking Lot

5.6 User Interface Design

5.6.1 Driver Mobile Interface

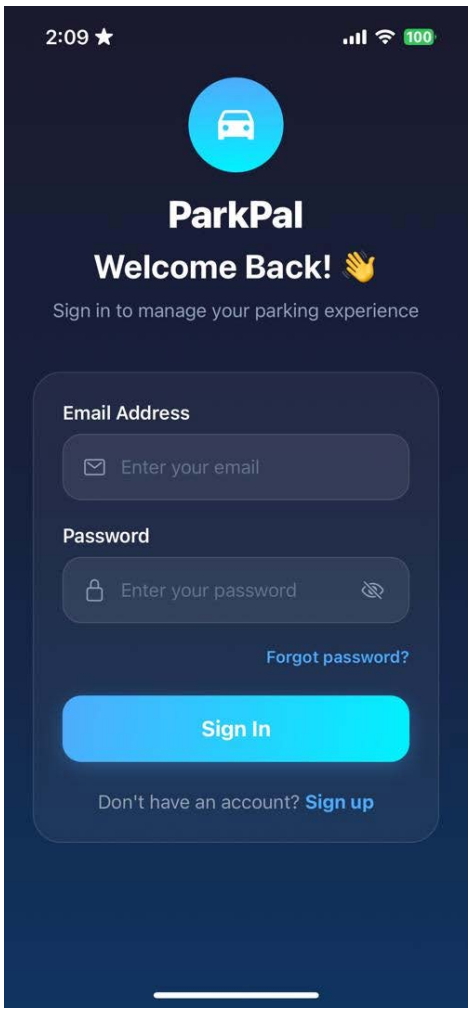


Figure 5.14: Login Page

This is the Login page of the ParkPal mobile application, where users (drivers) enter their email and password to access their account. Users who forgot their password can click “Forgot Password?”, and new users can tap “Sign Up” to create a new account.

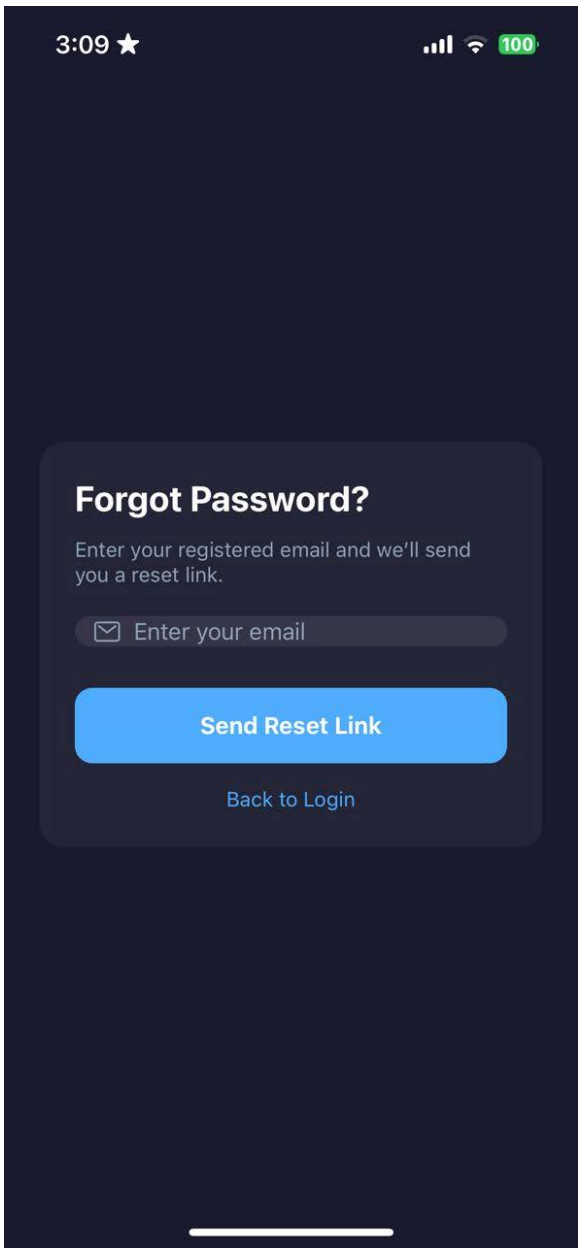


Figure 5.15: Forgot Password Page

This is the Forgot Password page, where users can enter their email to receive a password reset link. They can reset their password through the link, or tap “Back to Login” to cancel the operation.

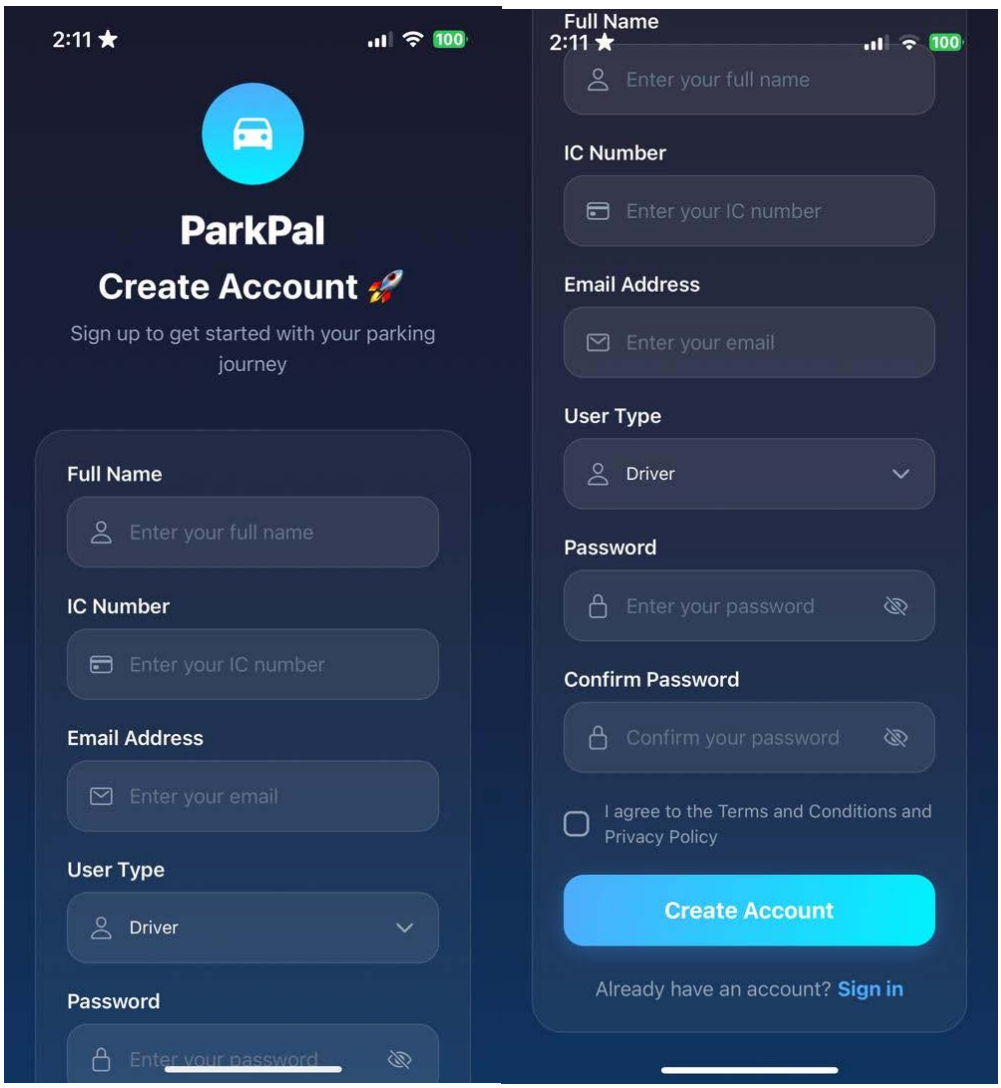


Figure 5.16: Register Page

This is the Registration page, where users can create an account by entering their personal information and agreeing to the terms and conditions and privacy policy. Users who already have an account can click the “Sign in” link to log in.

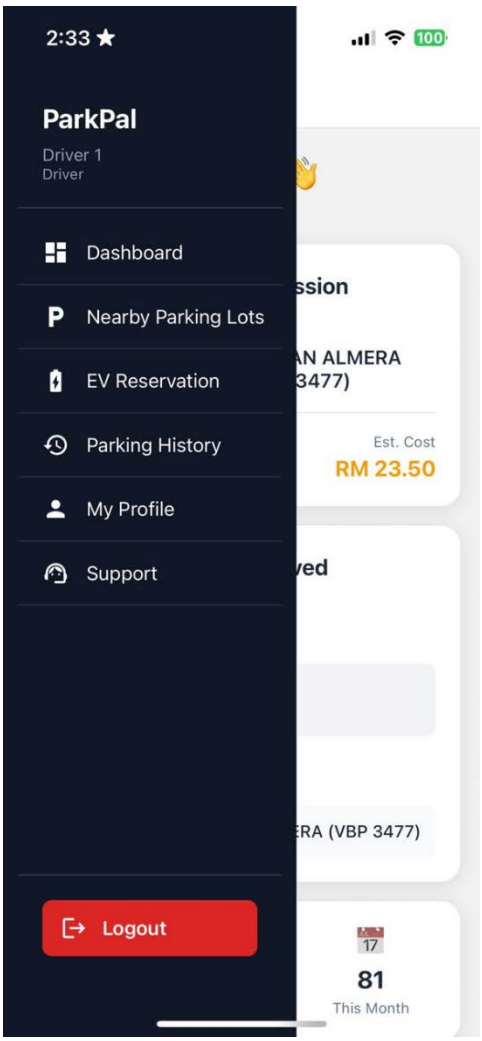


Figure 5.17: Drawer Navigation

This is the Drawer Navigation, which allows users to access other pages. Users can also view their name and role, and log out directly through the drawer.

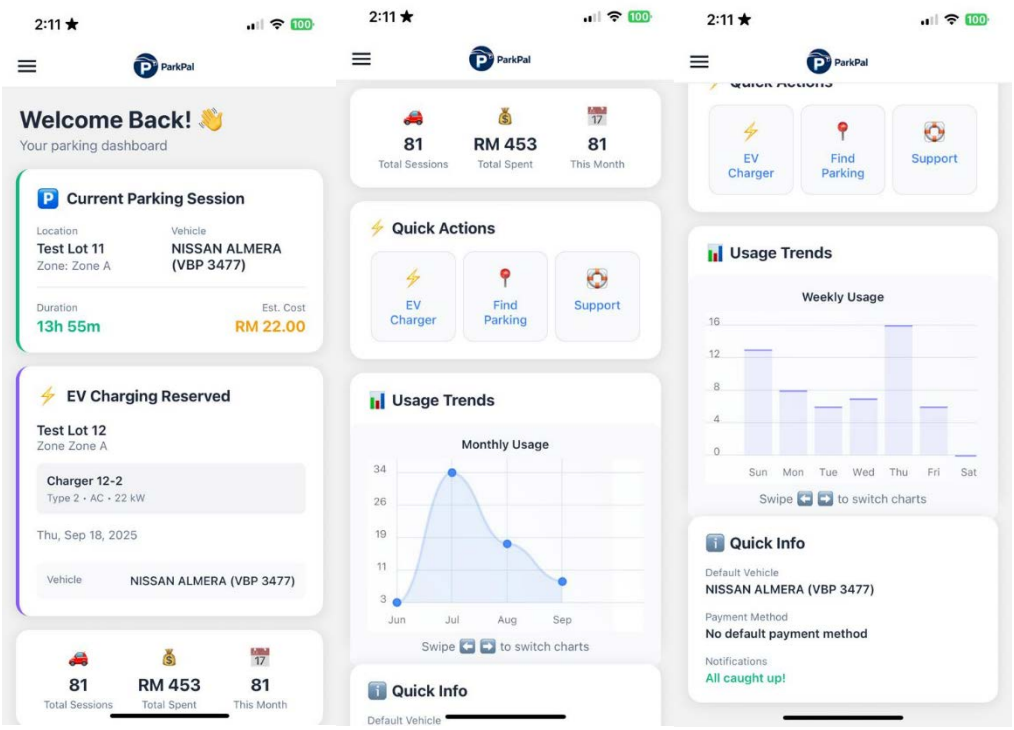


Figure 5.18: Driver Dashboard

The Driver Dashboard provides an overview of the driver's parking and EV activities. It displays the current parking session, reserved EV charging slots, and key statistics. Users can access quick actions for common tasks, view usage trends through interactive charts (switching between weekly and monthly views), and see quick informational summaries at a glance.

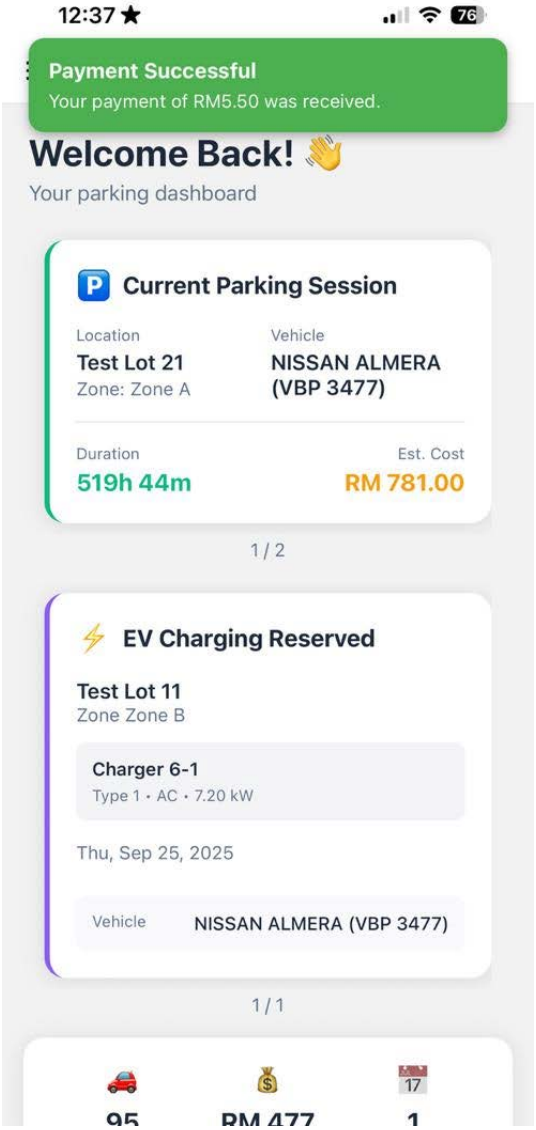


Figure 5.19: Toaster Notification after Payment

After a driver exits the parking lot and the simulated payment is successfully processed, the system displays a toaster notification to confirm completion. This notification appears instantly on the driver's screen, showing details such as the total parking fee, payment status, and session summary. It provides users with real-time feedback, ensuring they are informed of the successful transaction.

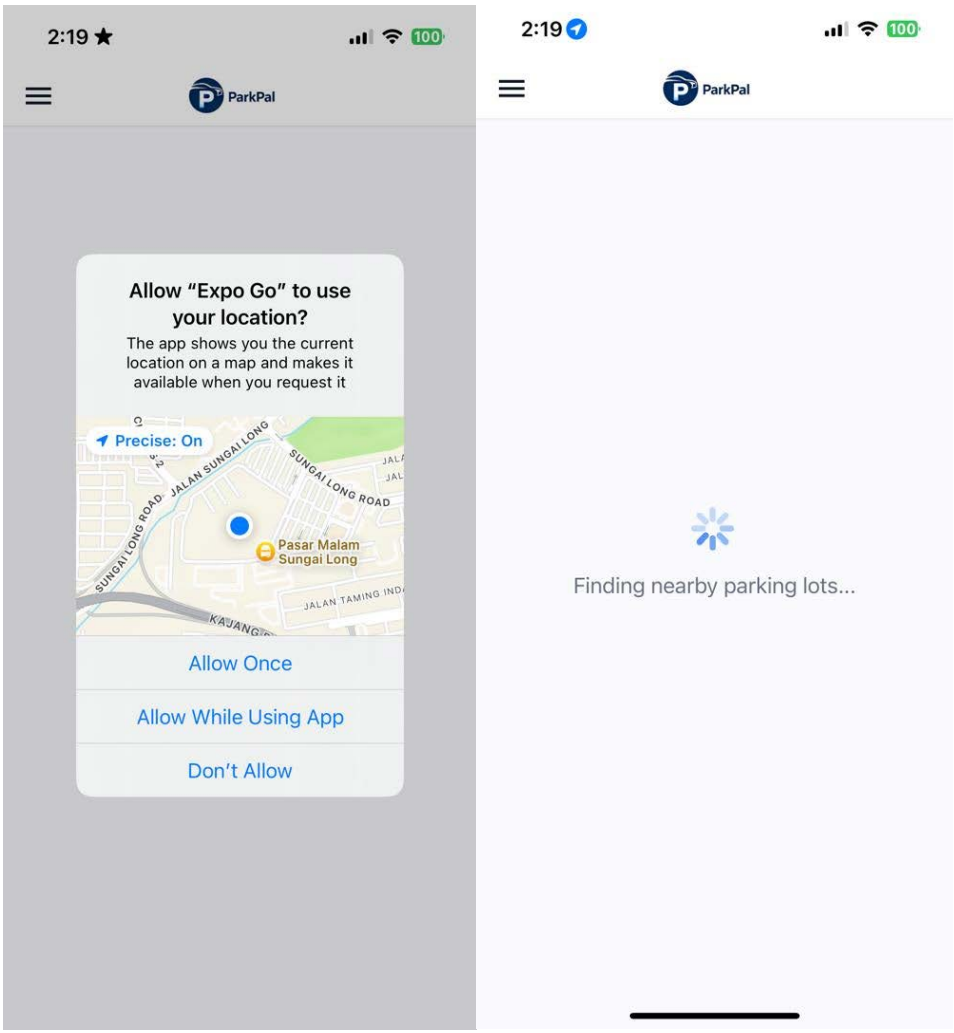


Figure 5.20: Nearby Parking Lot Location Permission and Loading Screen

Upon first use, the Nearby Parking Lots page prompts the user to grant location access. Once permission is granted, the system may display a loading screen while fetching nearby parking lot data.

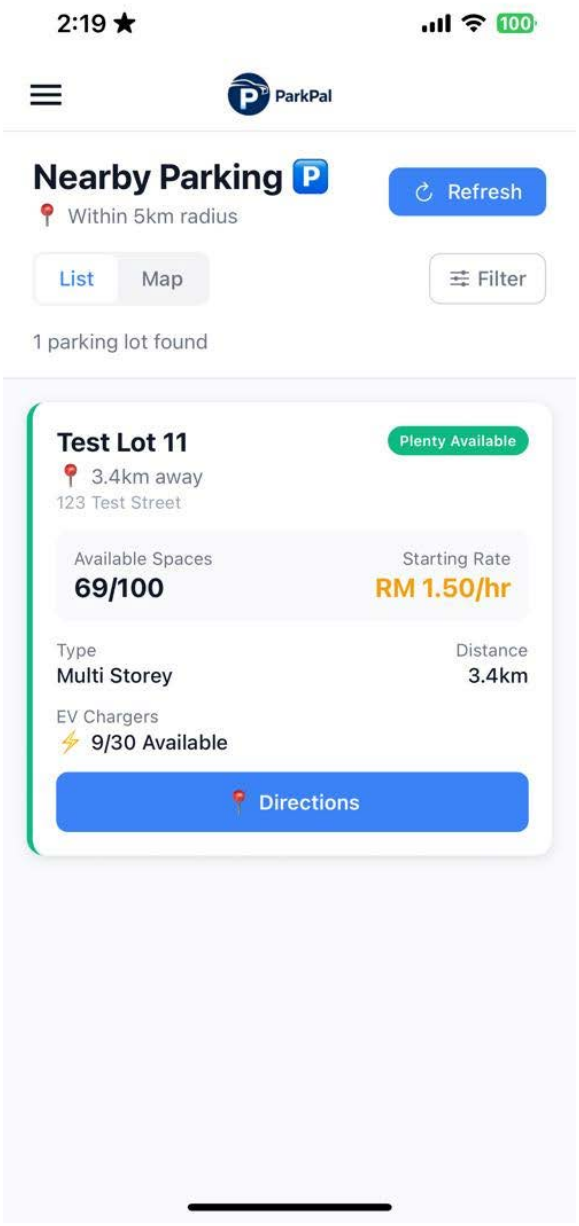


Figure 5.21: Nearby Parking Lots List View

The Nearby Parking Lots page displays a list of available parking lots near the user. Users can filter the results, switch to a map view, and view directions to their chosen parking lot.

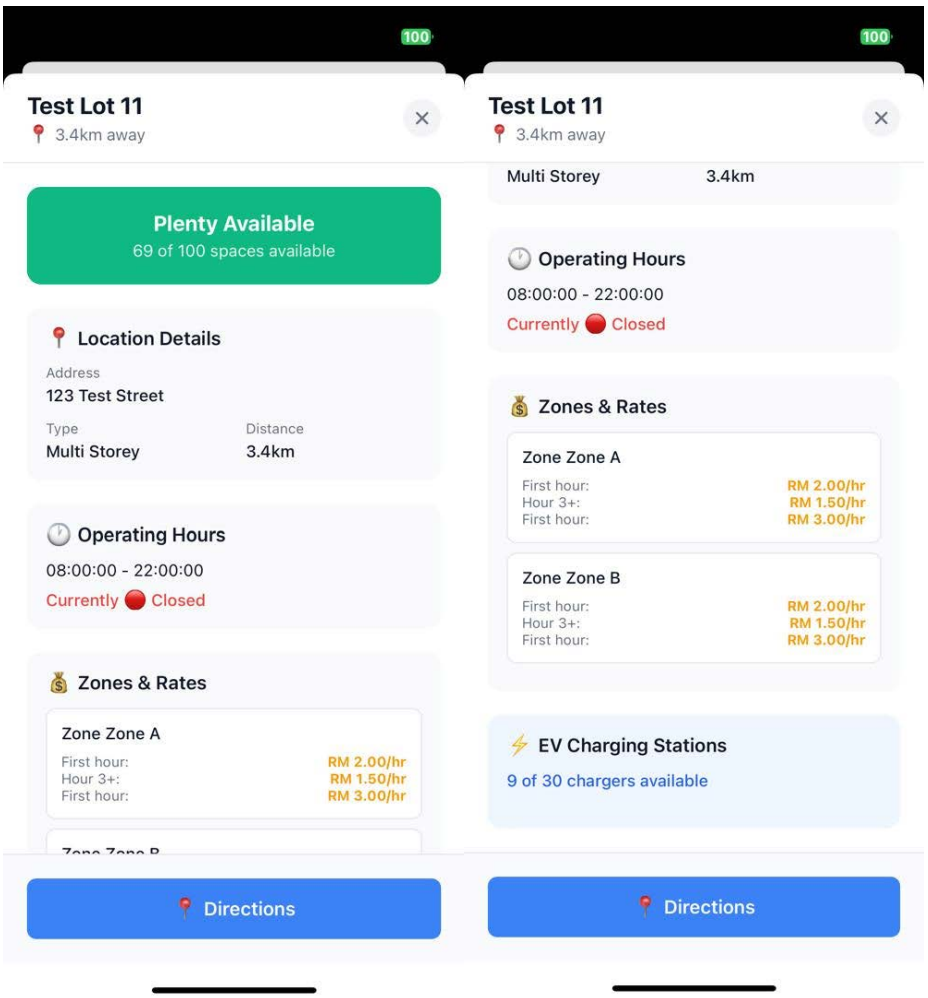


Figure 5.22: Nearby Parking Lots Details

Clicking on a parking lot card from the Nearby Parking Lot page opens the parking lot details, including available zones and their rates. If EV Chargers are present for the lot, the number of EV Chargers available is also displayed.

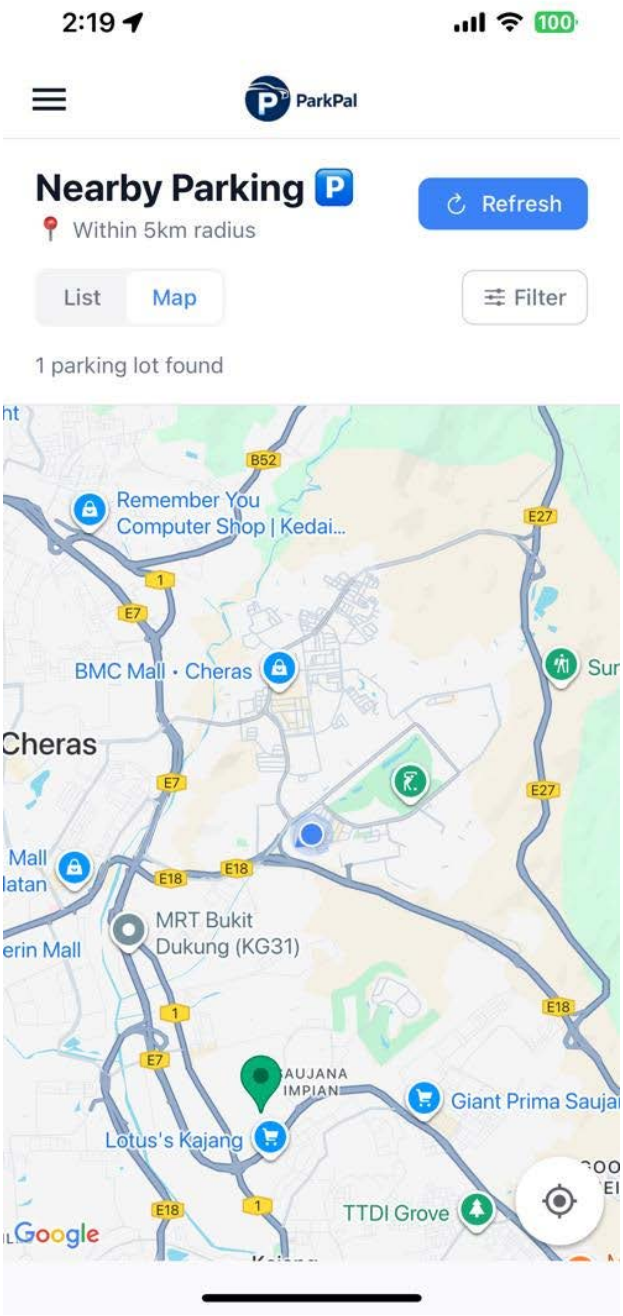


Figure 5.23: Nearby Parking Lots Map View

Switching to the map tab displays a map with pins marking the locations of nearby parking lots. Tapping a pin opens the parking lot details, including zones, rates, and available EV charging stations, as described previously.

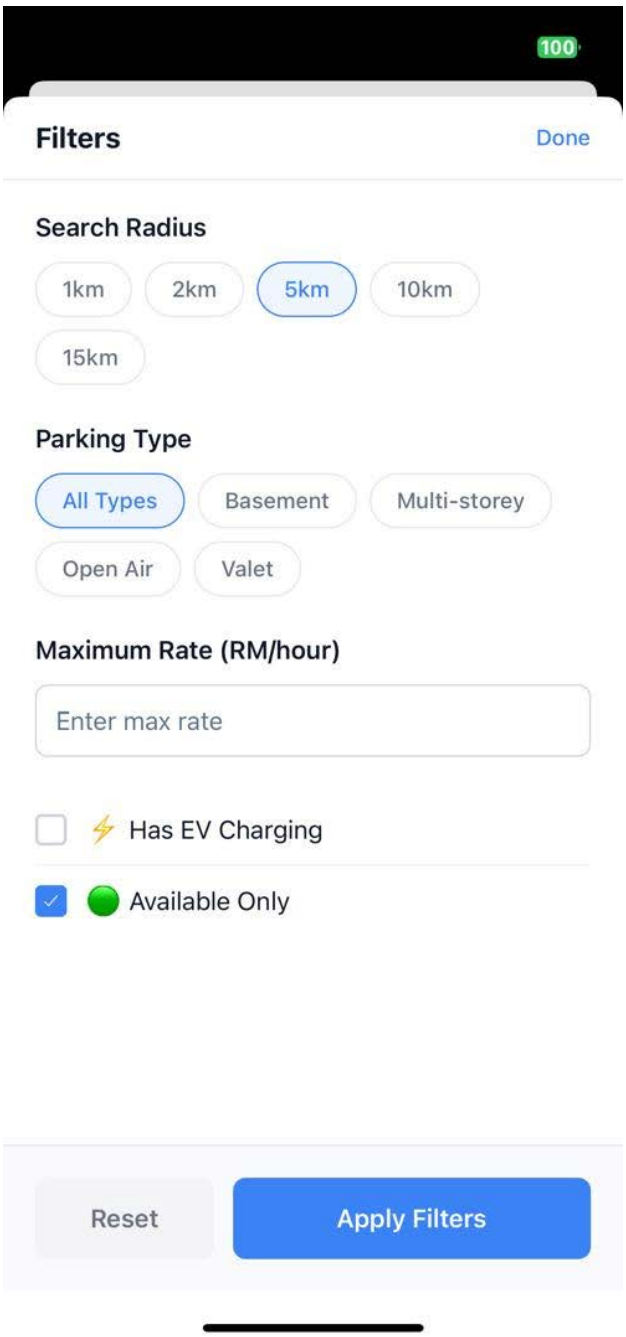


Figure 5.24: Nearby Parking Lot Filters

The filtering feature allows users to refine nearby parking lot results based on several criteria, such as search radius (1 km to 15 km), parking type, maximum rate, availability of EV charging, and currently available lots. Users can also reset all filters to return to the default view.

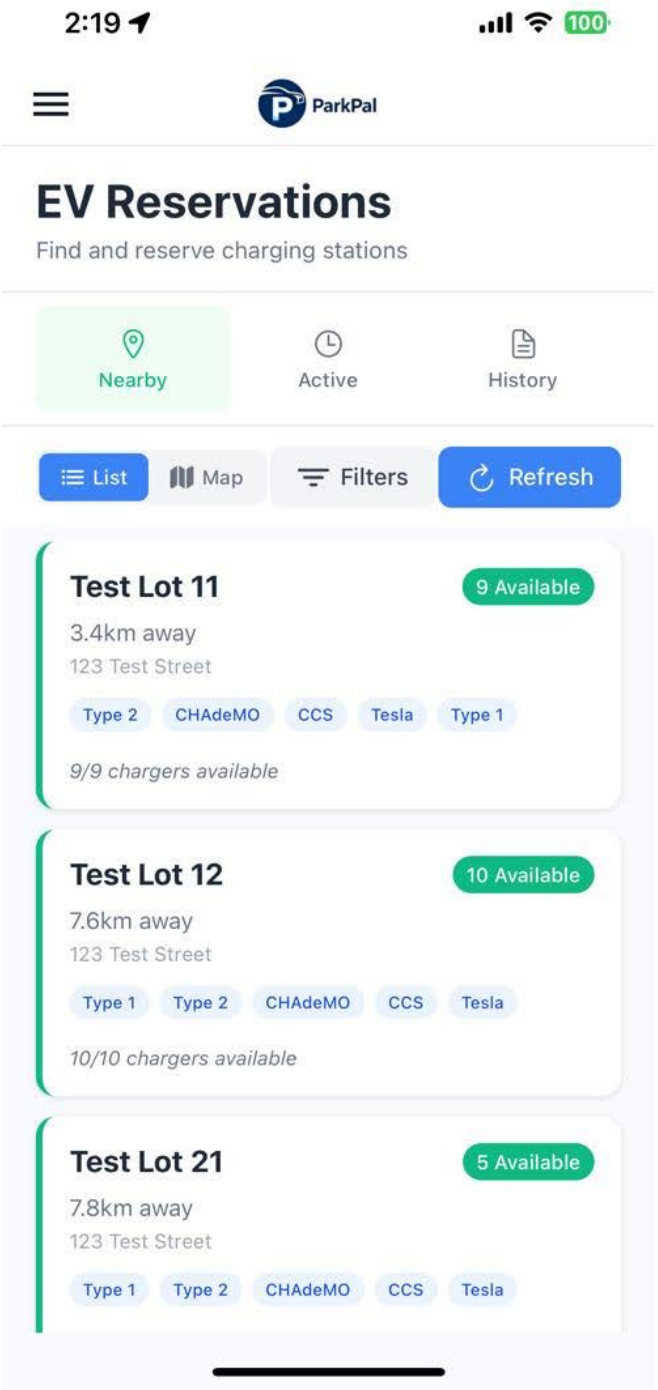


Figure 5.25: Nearby EV Reservation List View

The EV Reservation List page displays parking lots with available EV chargers. It shows the number of chargers available at each location and the types of chargers offered, allowing users to select a suitable option for reservation.

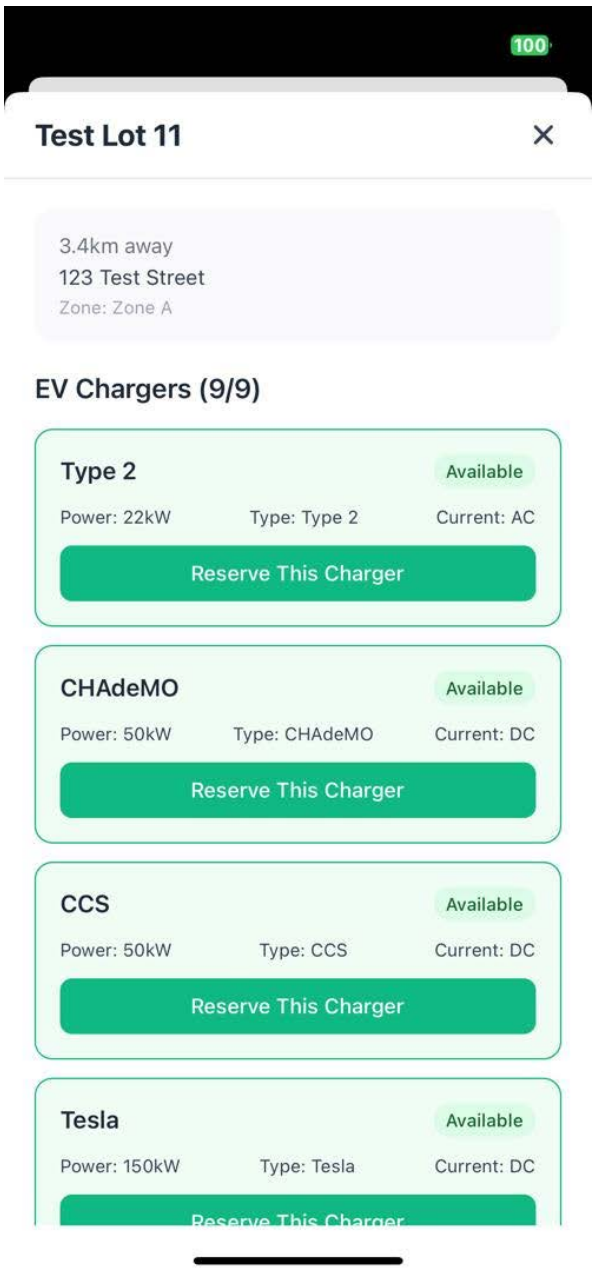


Figure 5.26: Available EV Reservation Details

Clicking on an EV reservation item opens a detailed view of all available chargers at that parking lot. Each charger includes detailed information, such as type, power output, and current availability, allowing users to choose the most suitable charger for reservation.

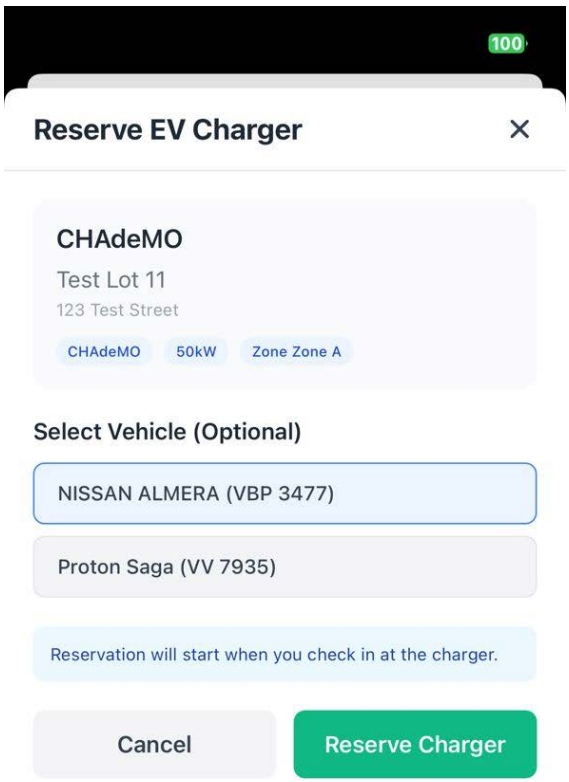


Figure 5.27: Vehicle Selection for EV Reservation

After clicking “Reserve This Charger,” the driver can select which of their registered vehicles to use for the reservation.

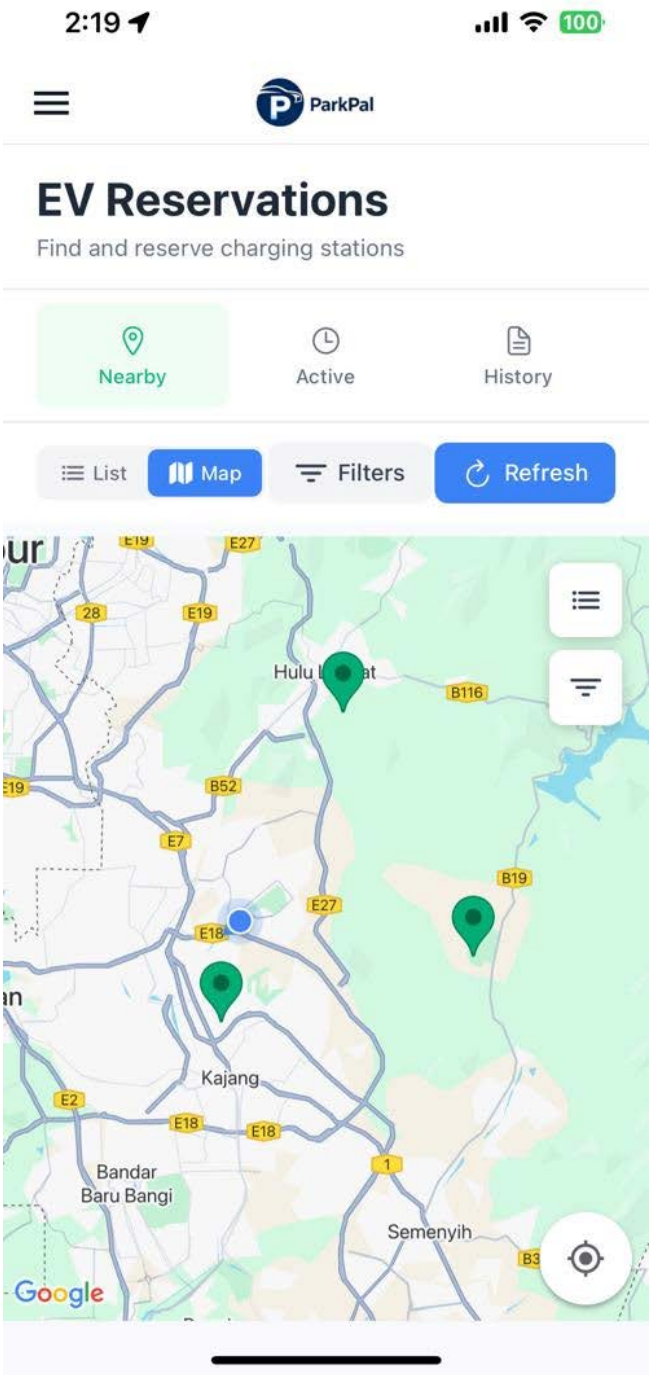


Figure 5.28: Nearby EV Reservation Map View

The EV Reservation map view displays pins for parking lots with available EV chargers. Tapping a pin opens the reservation details, showing all available chargers and their specifications, allowing users to make a reservation directly from the map.

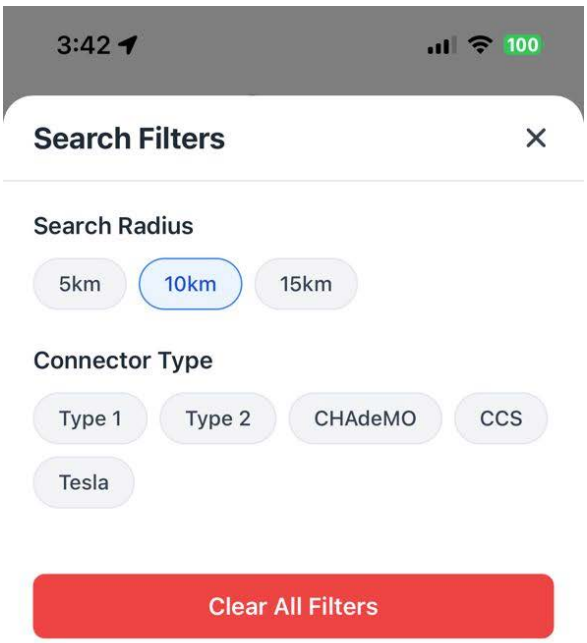


Figure 5.29: EV Reservation Filter

The EV Reservation filtering feature allows users to refine the list of available chargers based on search radius and connector types. This helps drivers quickly find chargers that are both nearby and compatible with their vehicle. Users can also clear all filters to return to the default view.

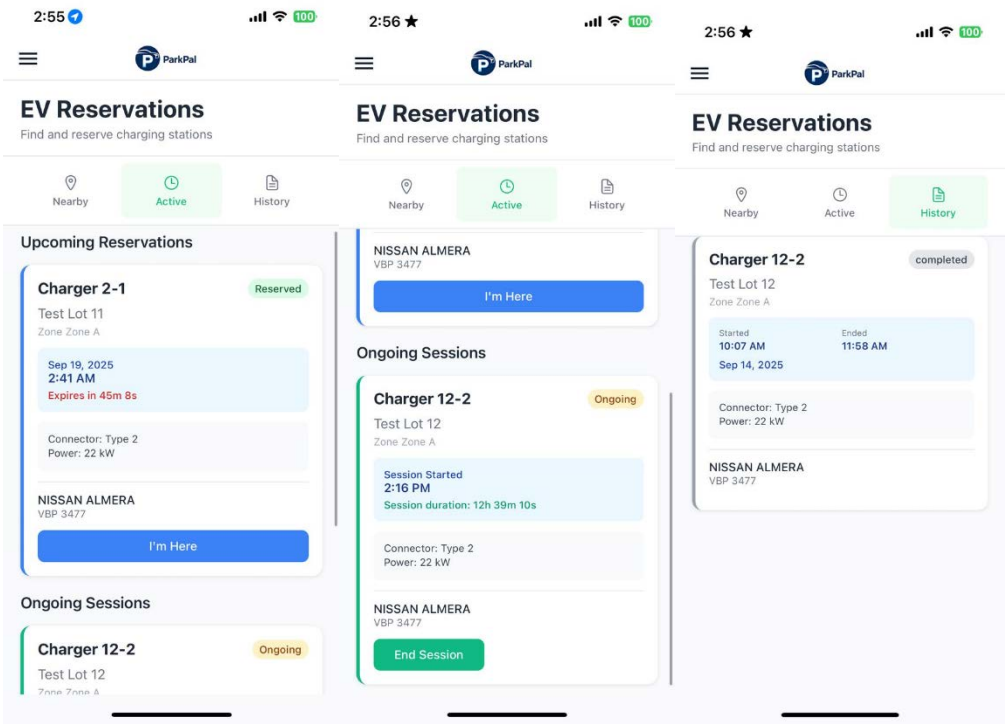


Figure 5.30: EV Reservation Active and History

The EV Reservation also shows the upcoming, ongoing, and past reservations. The Active tab displays both upcoming reservations, showing the time remaining until expiration (1-hour limit), and ongoing reservations, showing the current session duration. The History tab shows past reservations. All reservations display the associated vehicle, parking lot, and EV charger details.

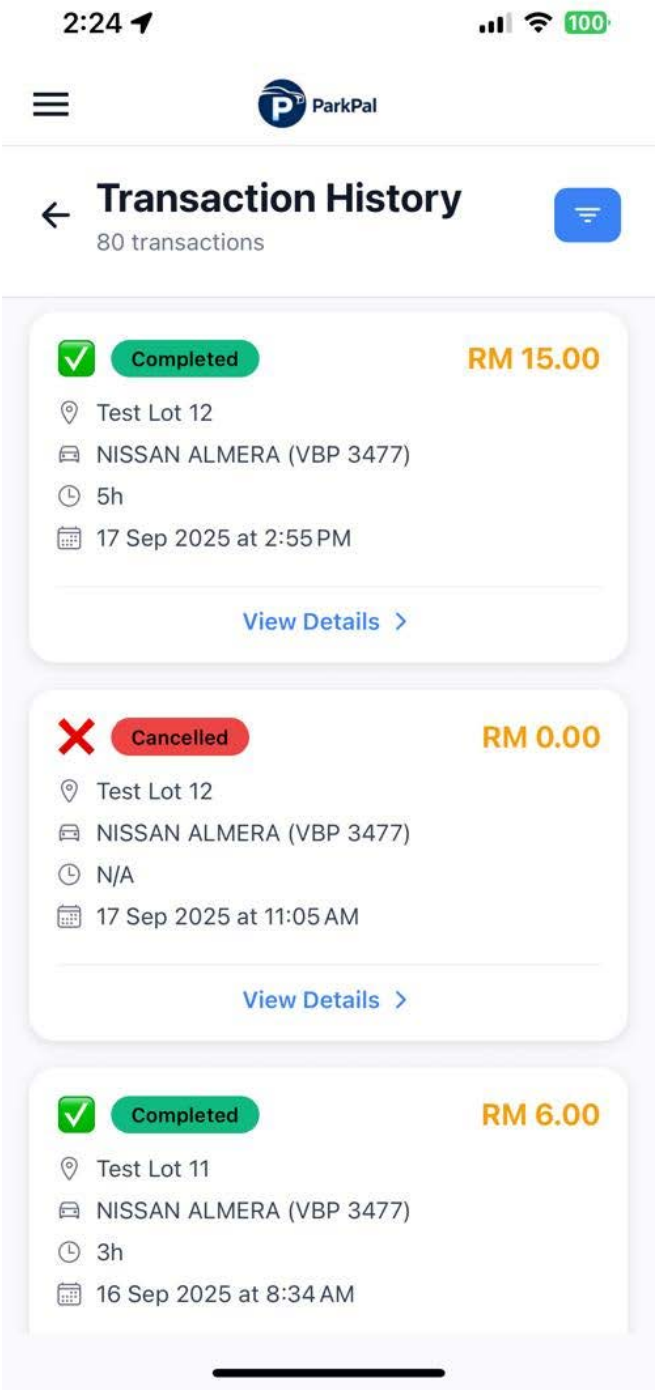


Figure 5.31: Parking Transaction History

The Parking Transaction History page displays a list of past parking transactions. Each entry includes the transaction status, parking lot, vehicle used, duration, date and time, and amount paid, giving users a clear record of their parking activities.

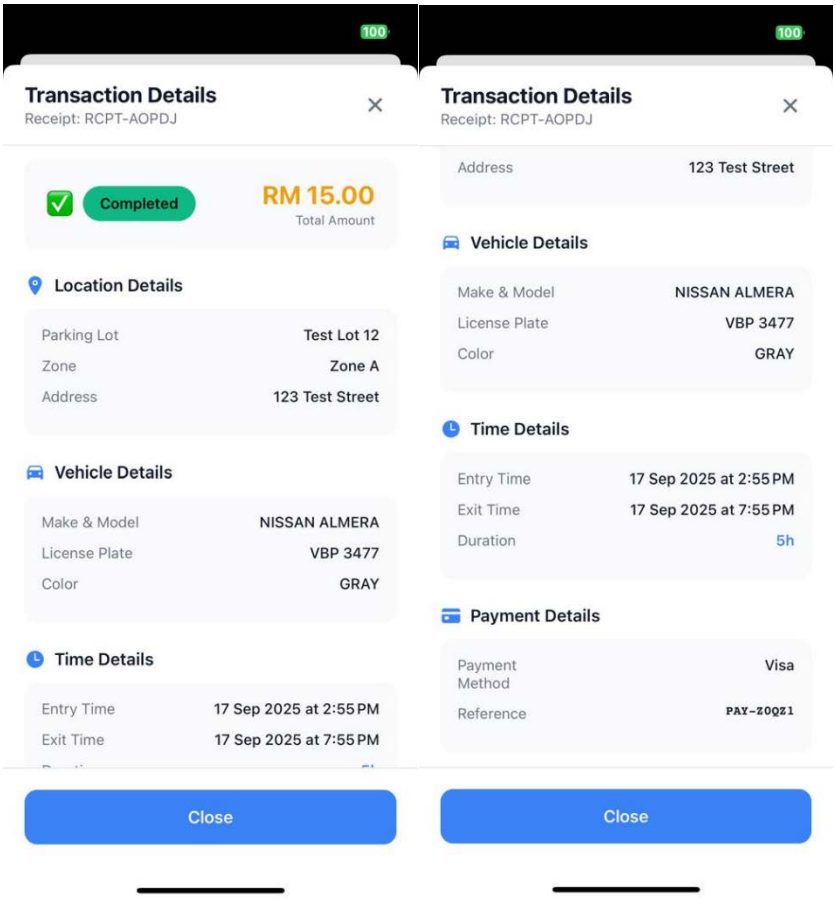


Figure 5.32: Parking Transaction History Details

Clicking “View Details” on a transaction opens an in-depth view, showing the parking lot location, vehicle information, time details, and payment information (if available), providing a comprehensive overview of the transaction.

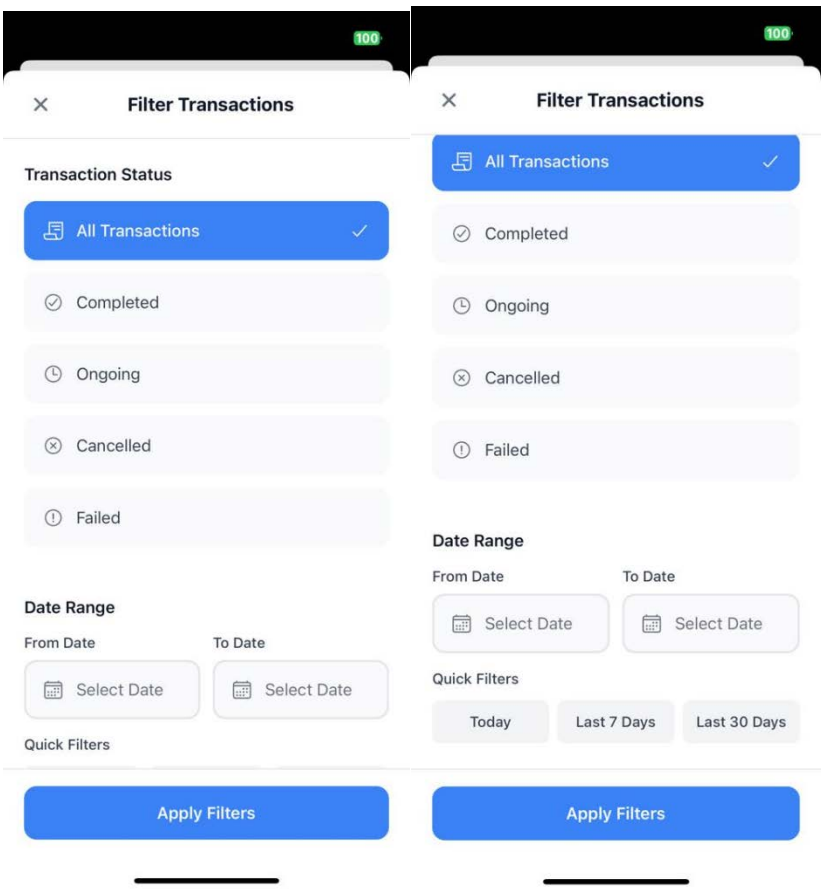


Figure 5.33: Parking Transaction Filter

The Transaction History page includes filters that allow users to refine transactions by status and date range. It also provides quick date range options for faster access to recent transactions.

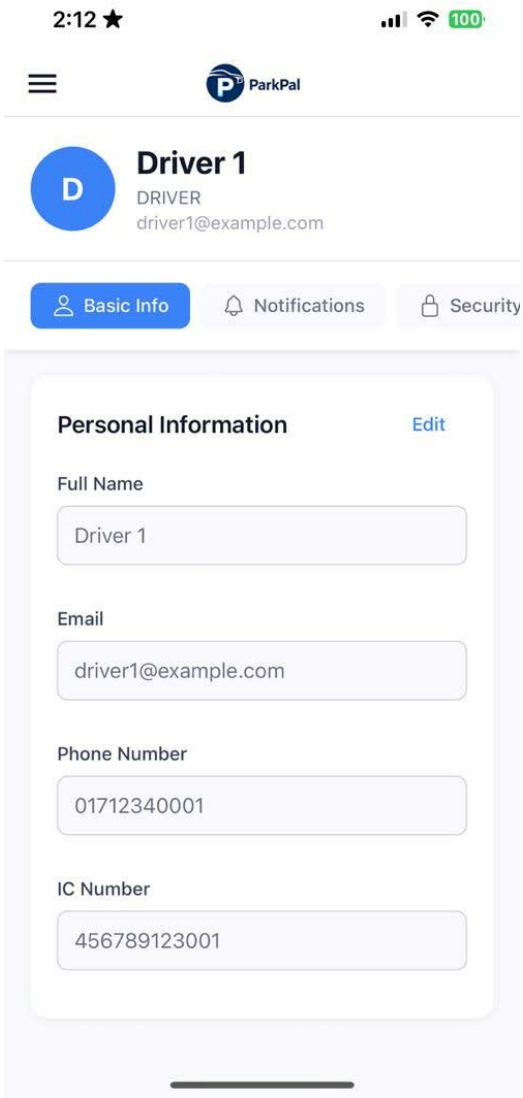


Figure 5.34: Profile Page Basic Info Tab

Under the Basic Info tab, users can view their personal information, including full name, email, phone number, and IC number. The IC number is displayed as read-only and cannot be edited, while the other fields can be updated as needed.

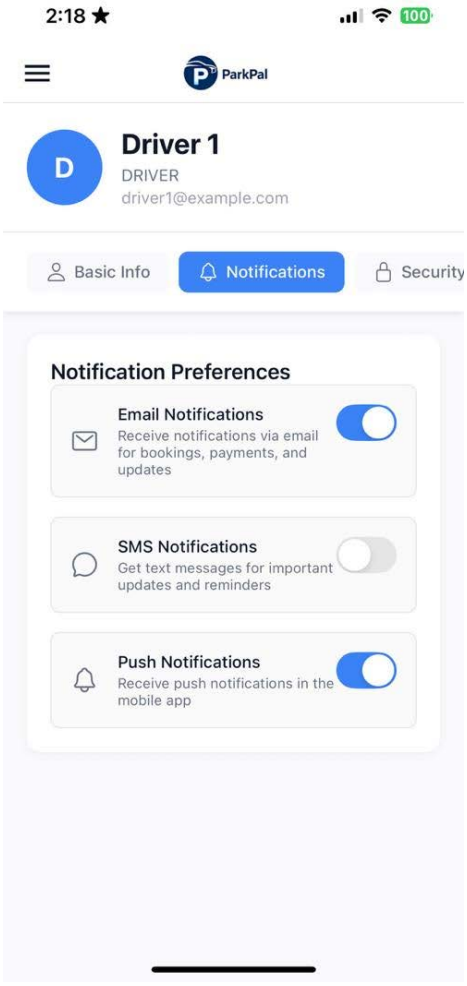


Figure 5.35: Profile Page Notifications Tab

Under the Notifications tab, users can manage their notification preferences. They can toggle options for receiving alerts via email, SMS, or push notifications according to their preferences.

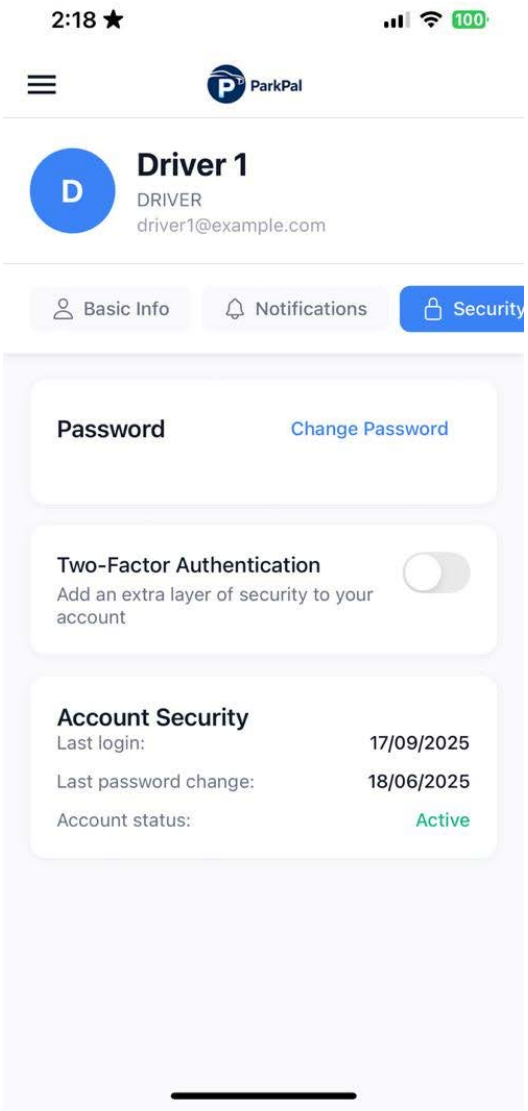


Figure 5.36: Profile Page Security Tab

Under the Security tab, users can change their password and toggle two-factor authentication for enhanced account security. This tab also displays relevant account security information to keep users informed about their account protection status.

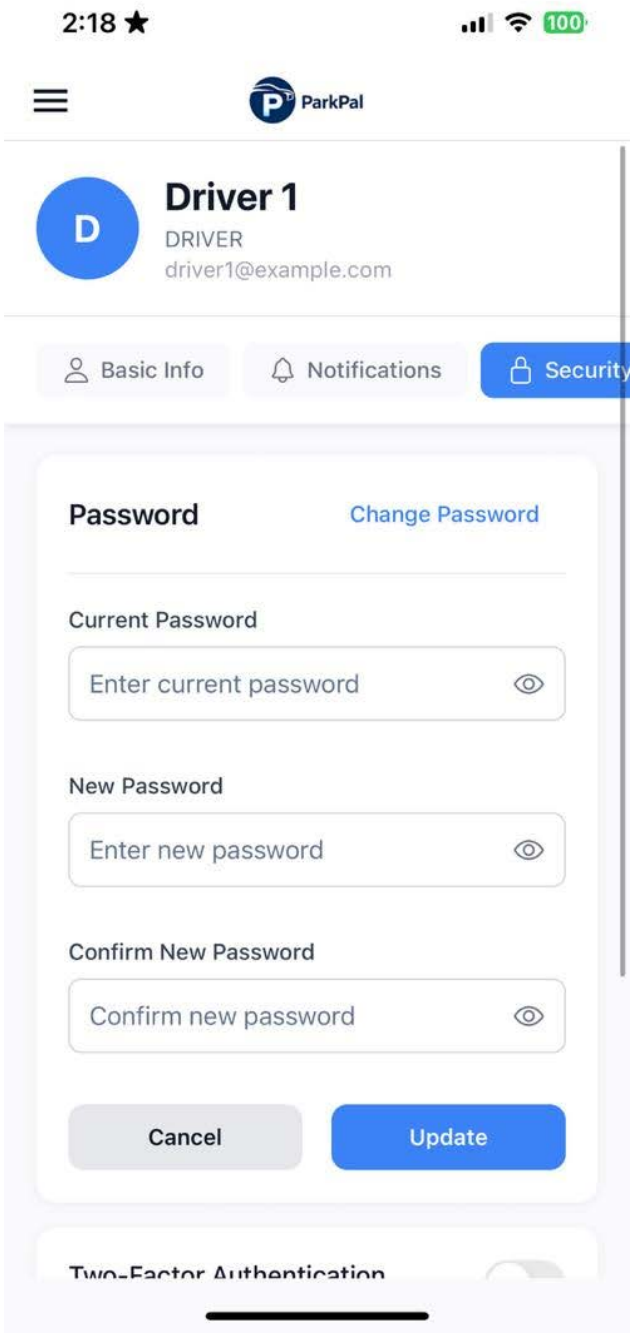


Figure 5.37: Profile Page Change Password

To update their password, users must enter their current password, choose a new password, and confirm the new password. This ensures that password changes are secure and verified.

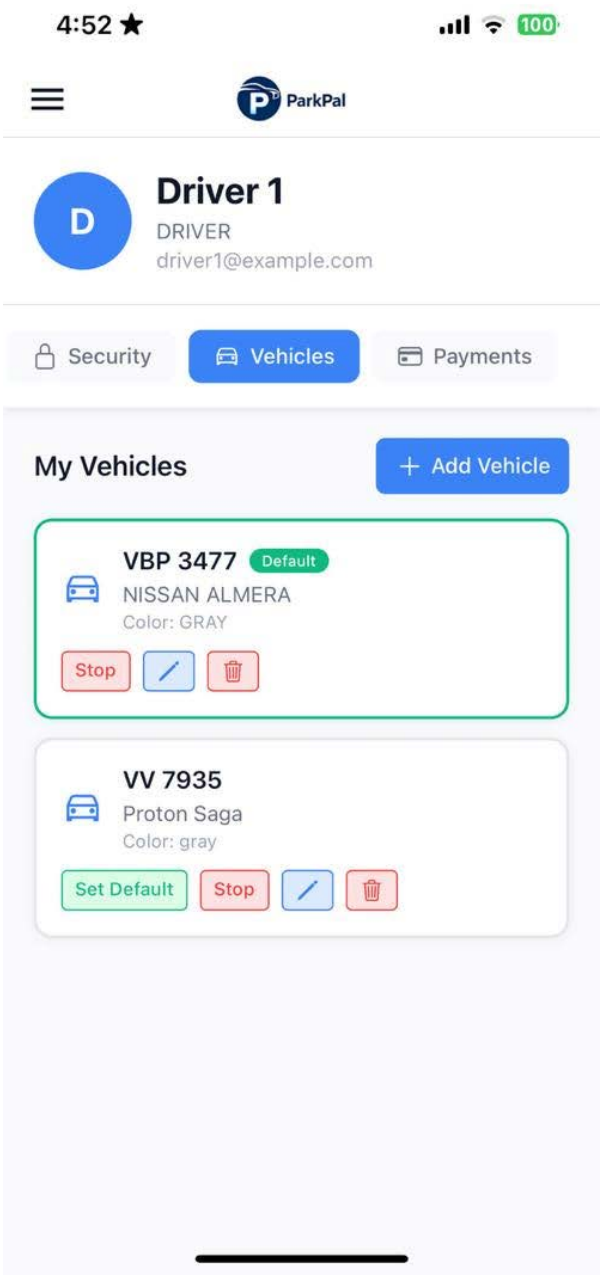


Figure 5.38: Profile Page Vehicles Tab

The Vehicles tab displays all vehicles registered by the user. Users can add new vehicles, edit or delete existing ones, mark a vehicle as default, or stop a vehicle in case of suspected fraud.

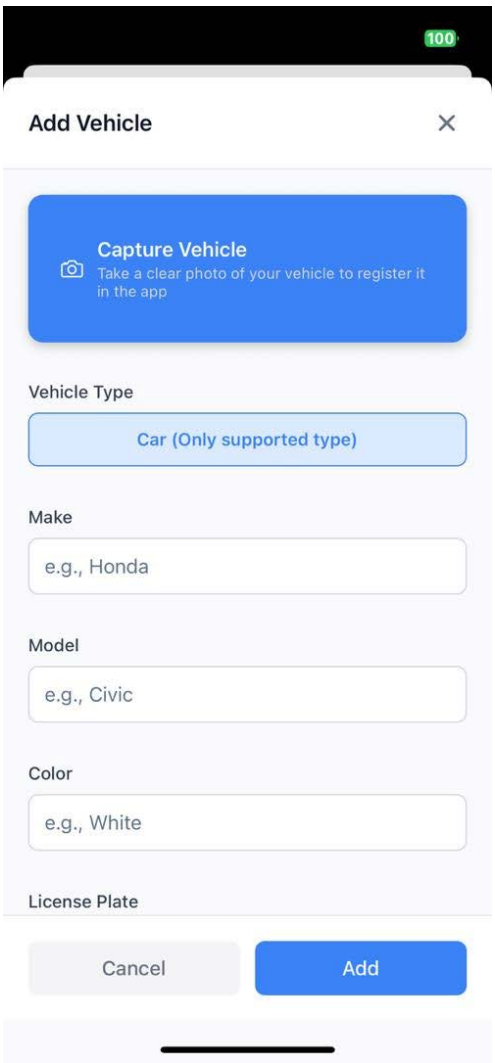


Figure 5.39: Profile Page Add Vehicles

When adding a vehicle, users can either manually enter the vehicle details or use the camera to capture a photo. The system will extract information such as make, model, color, and license plate from the image to simplify the process.

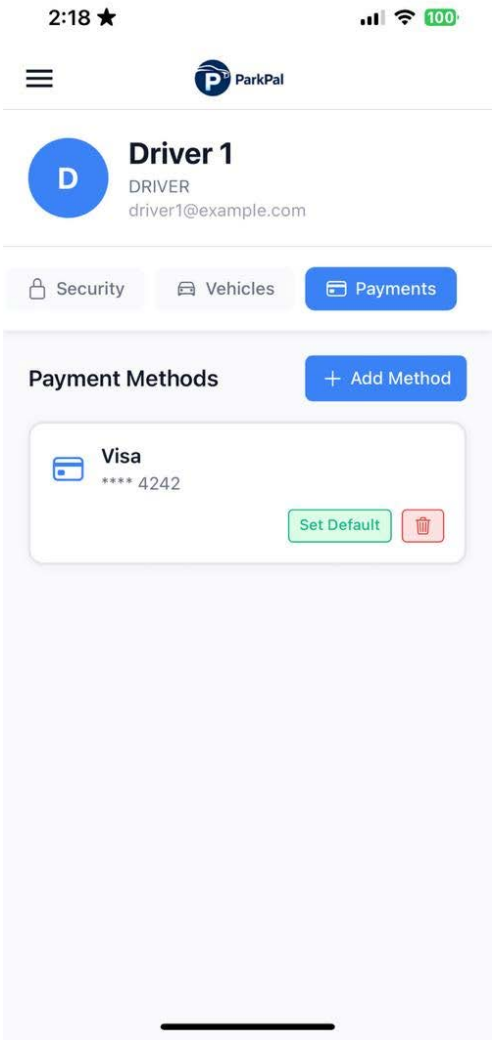


Figure 5.40: Profile Page Payments Tab

In the Payment Methods tab, users can add new payment methods, set a default method for transactions, and delete existing payment methods as needed.

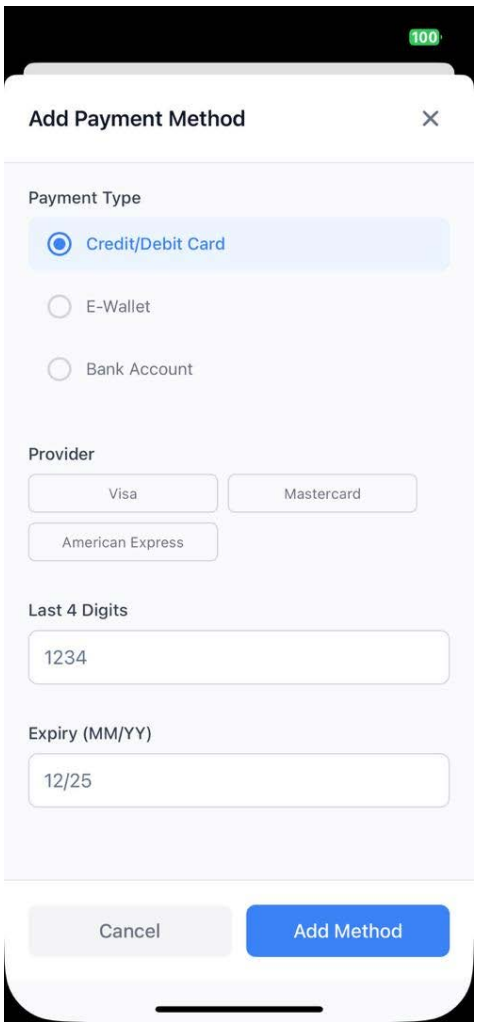


Figure 5.41: Profile Page Add Payment Method

When adding a payment method, users must provide the payment type, provider, last four digits of the card, and expiry date. This ensures the system has the necessary information to process transactions securely.

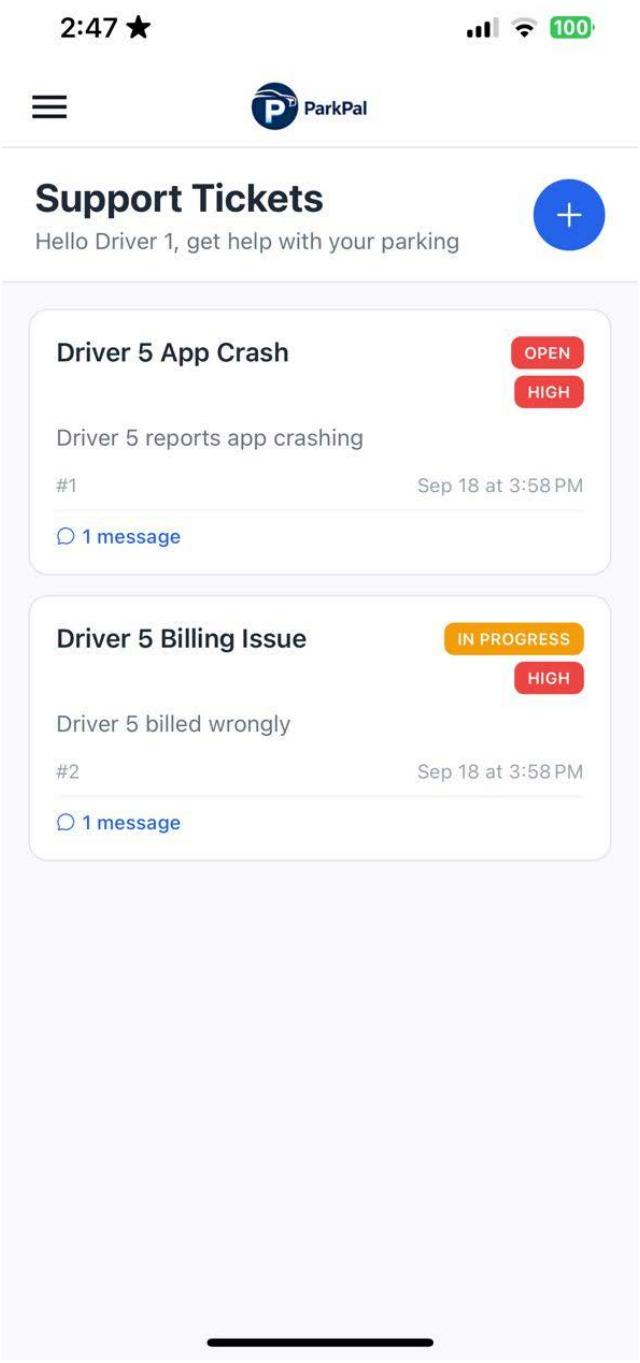


Figure 5.42: Support Tickets Page

The Support Ticket page displays a list of all tickets submitted by the driver. Users can view the status of existing tickets and create new tickets to report issues or request assistance.

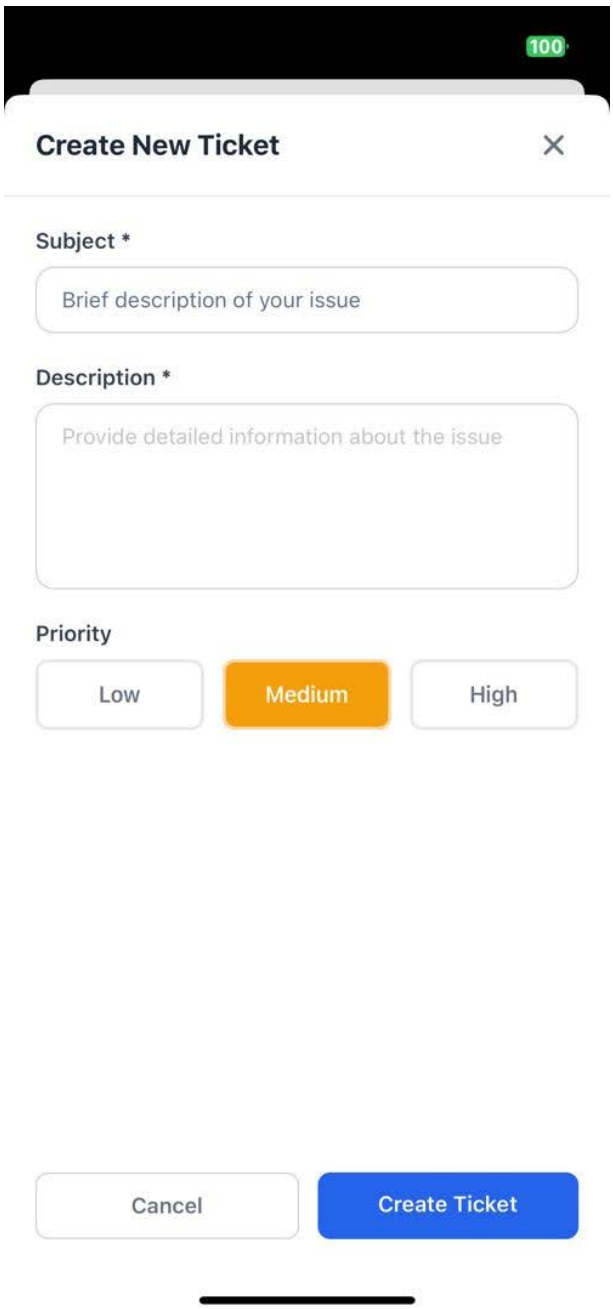


Figure 5.43: Create New Ticket

When creating a new support ticket, the driver is required to enter the ticket type, subject, detailed description, and select a priority level. This ensures the issue is properly categorized and addressed promptly.

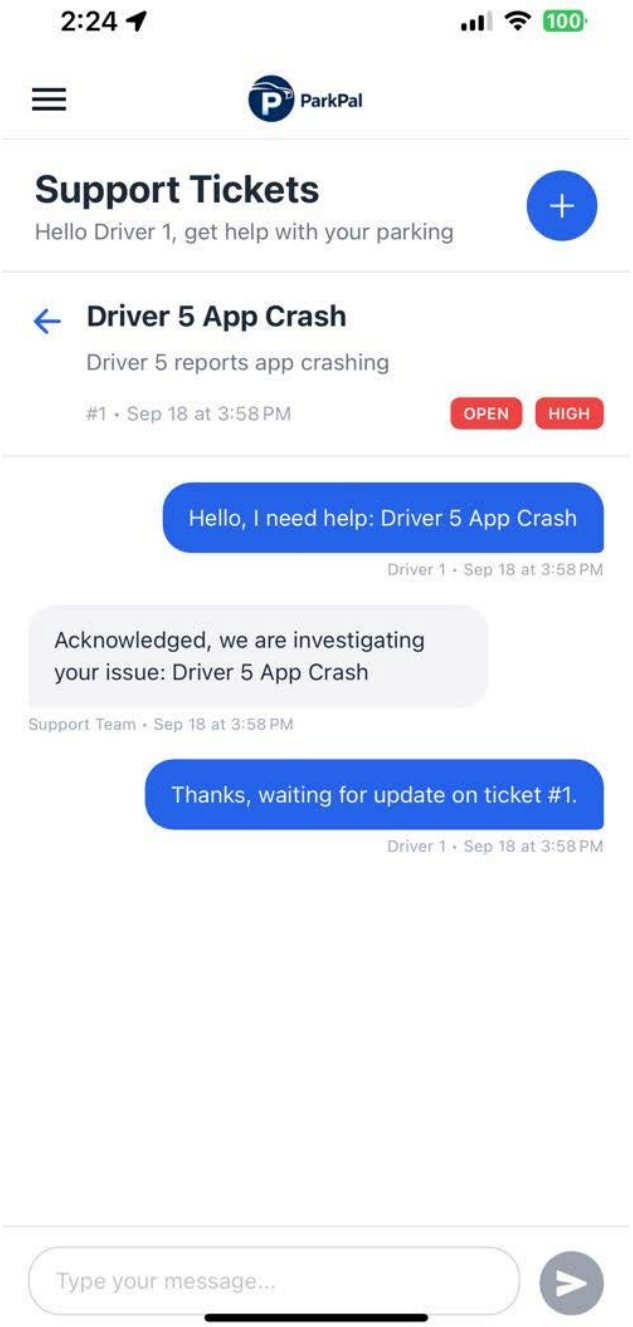


Figure 5.44: View and Send Support Ticket Message

The Support Ticket Messages feature functions like a text messaging system, allowing drivers and support staff to communicate within each ticket. Users can send and receive messages, providing updates, clarifications, and responses related to their submitted issues.

5.6.2 Web Interface

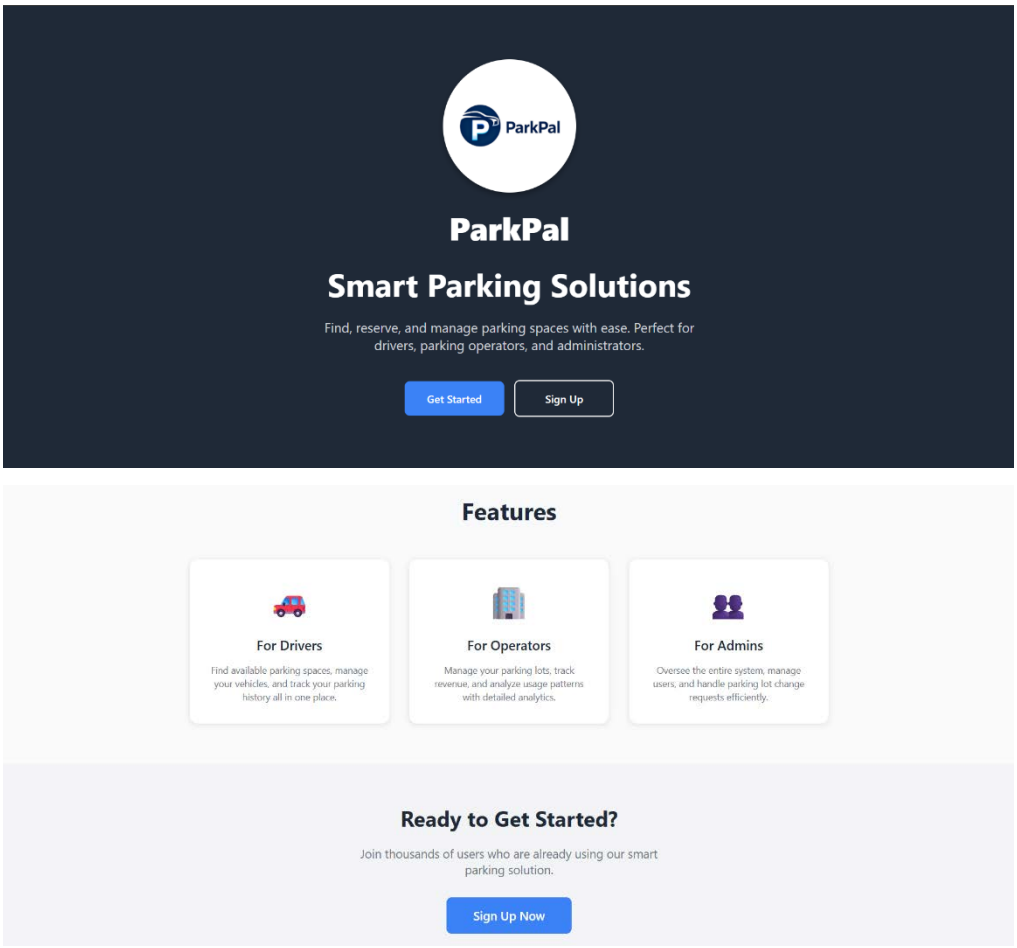


Figure 5.45: Landing Page

The Landing Page serves as the first screen of the ParkPal website, providing users with an overview of the app's main features. It offers quick navigation to key sections such as Login, Registration, and introductory information about the app's services.

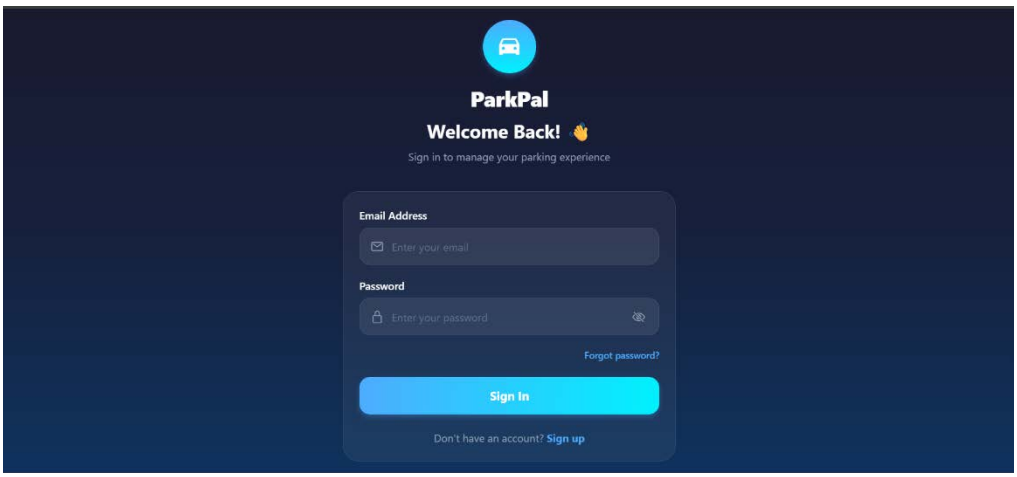


Figure 5.46: Login Page

This is the Login page of the ParkPal website, where users (parking operators and admins) enter their email and password to access their account. Users who forgot their password can click “Forgot Password?”, and new users can tap “Sign Up” to create a new account.

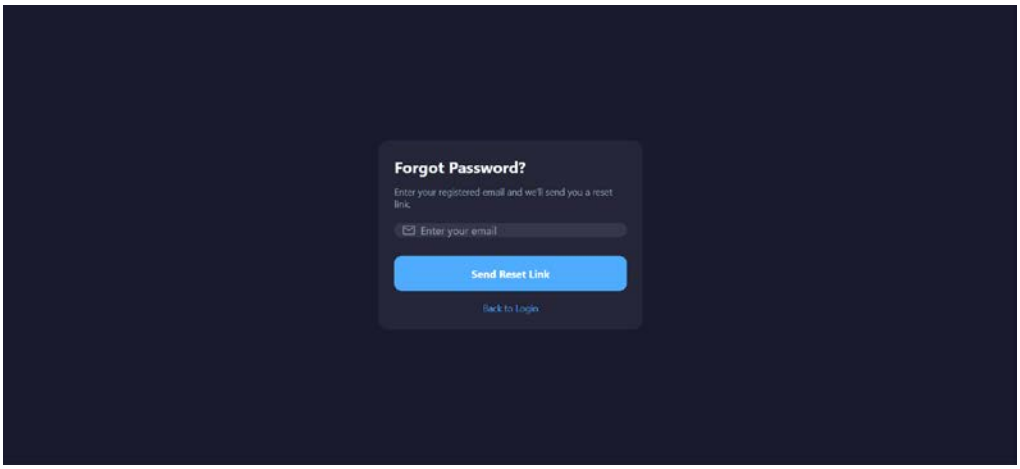


Figure 5.47: Forgot Password Page

This is the Forgot Password page, where users can enter their email to receive a password reset link. They can reset their password through the link, or tap “Back to Login” to cancel the operation.

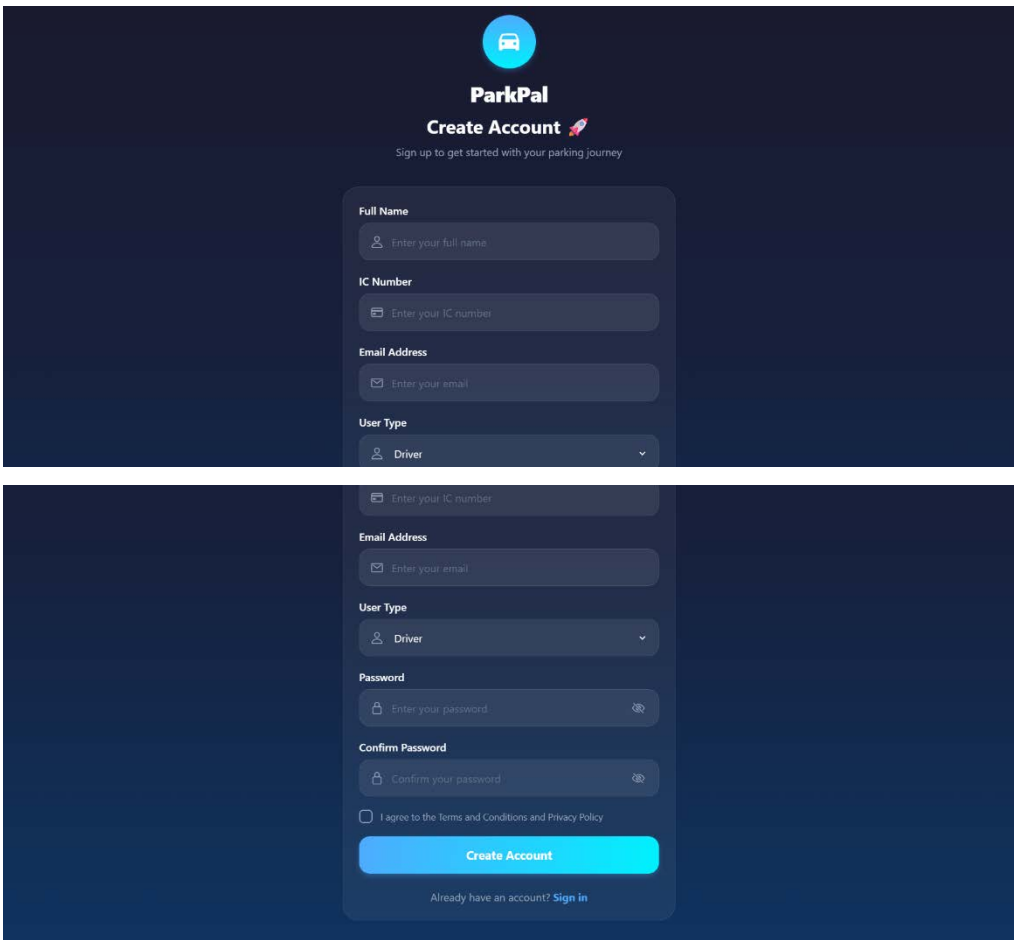


Figure 5.48: Register Page

This is the Registration page, where users can create an account by entering their personal information and agreeing to the terms and conditions and privacy policy. Users who already have an account can click the “Sign in” link to log in.

5.6.3 Parking Operator Web Interface

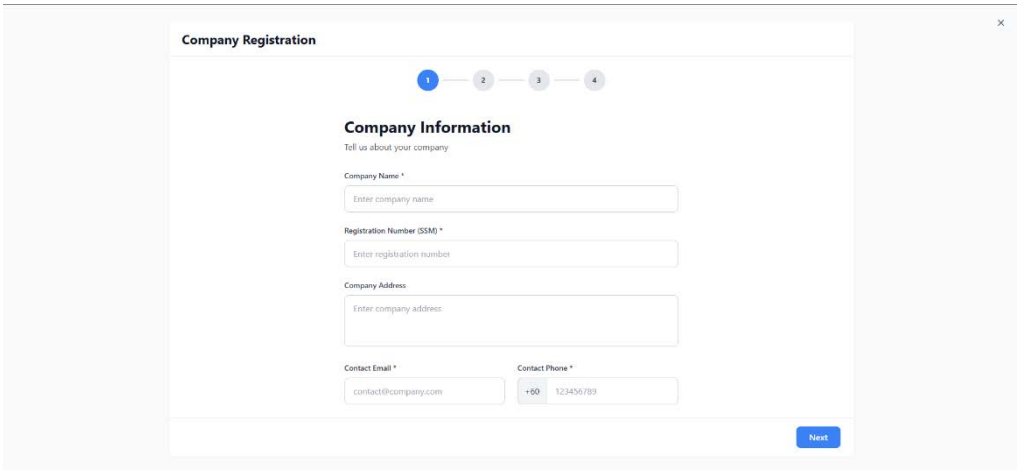


Figure 5.49: Operator Company Setup Page

There are 4 steps in the Operator Setup. In the first section of Operator Setup, parking operators enter their company details, including the company name, registration number, address, email, and phone number. This information is required to complete the initial setup of the operator account.

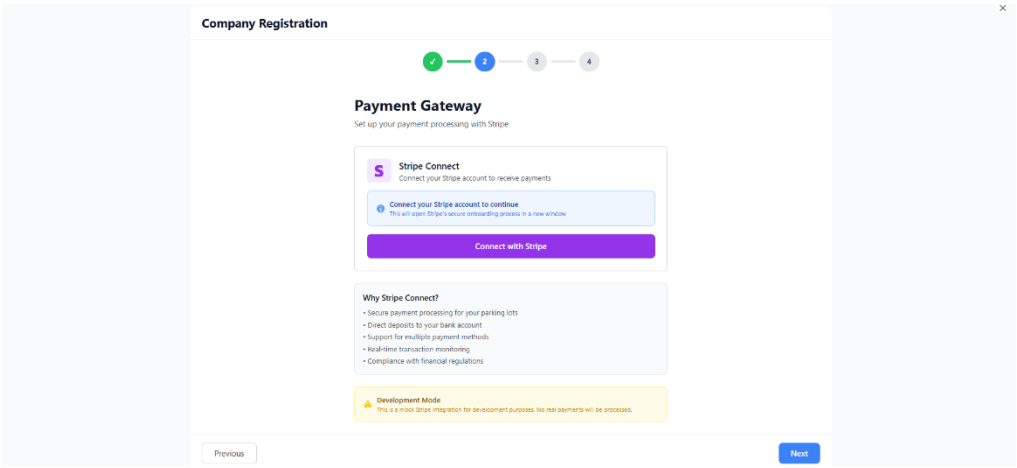


Figure 5.50: Operator Stripe Setup Page

In the second section, parking operators link their Stripe account to enable payment processing. This allows the system to handle transactions securely and ensures that parking fees can be collected and managed efficiently.

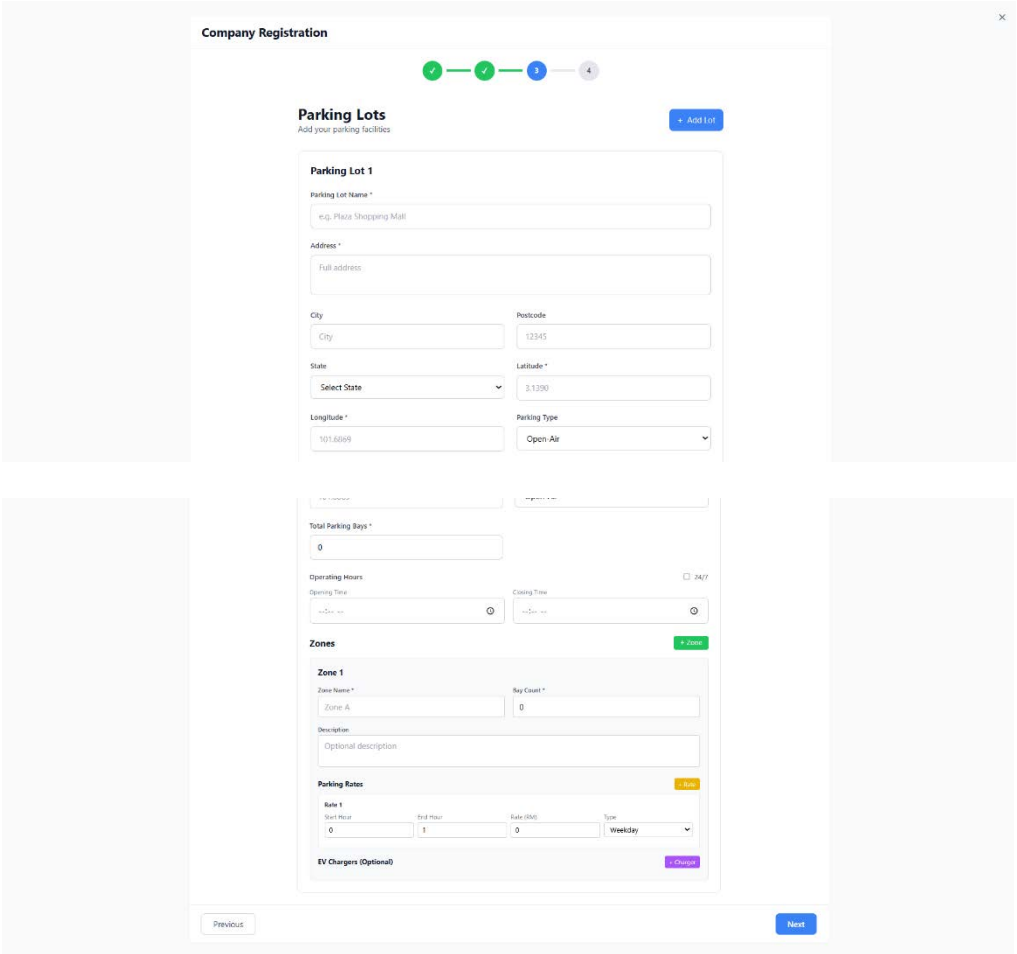


Figure 5.51: Operator Parking Lot Setup Page

In the third section, parking operators enter details about their parking lot, including the lot name, zones, and rates for each zone. Information about EV chargers can also be added if available. Operators can add multiple lots if they manage more than one location.

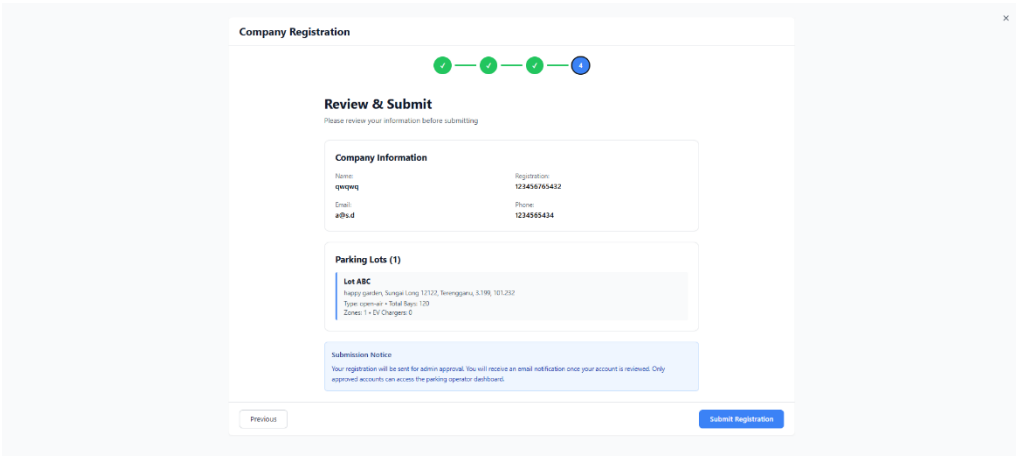


Figure 5.52: Operator Setup Review Page

In the final section, parking operators can review all the information they have entered, including company details, Stripe connection, and parking lot information. This allows them to verify and confirm the accuracy of their data before completing the setup process.

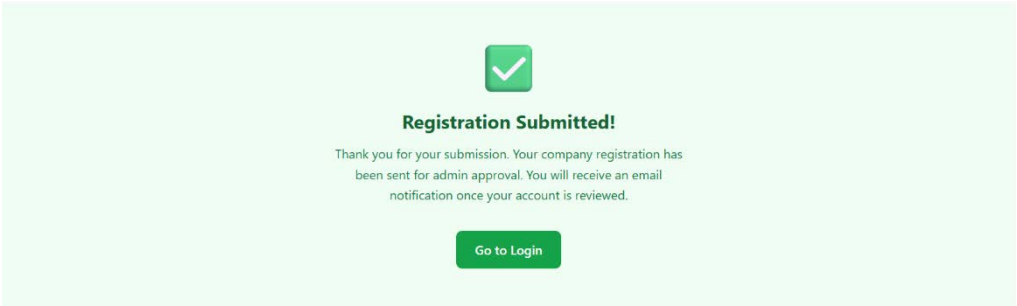


Figure 5.53: Operator Setup Successful Page

After submitting the setup, a confirmation page is displayed indicating that the registration has been submitted. The message informs the operator that admin approval is required before the account becomes active. Operator can then be redirected back to the login page.

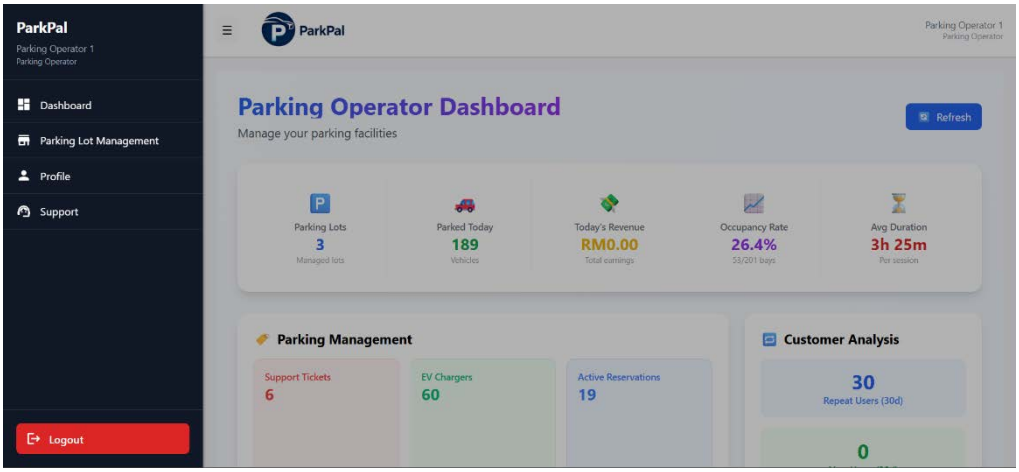


Figure 5.54: Drawer Navigation

This is the Drawer Navigation, which allows users to access other pages. Users can also view their name and role, and log out directly through the drawer.

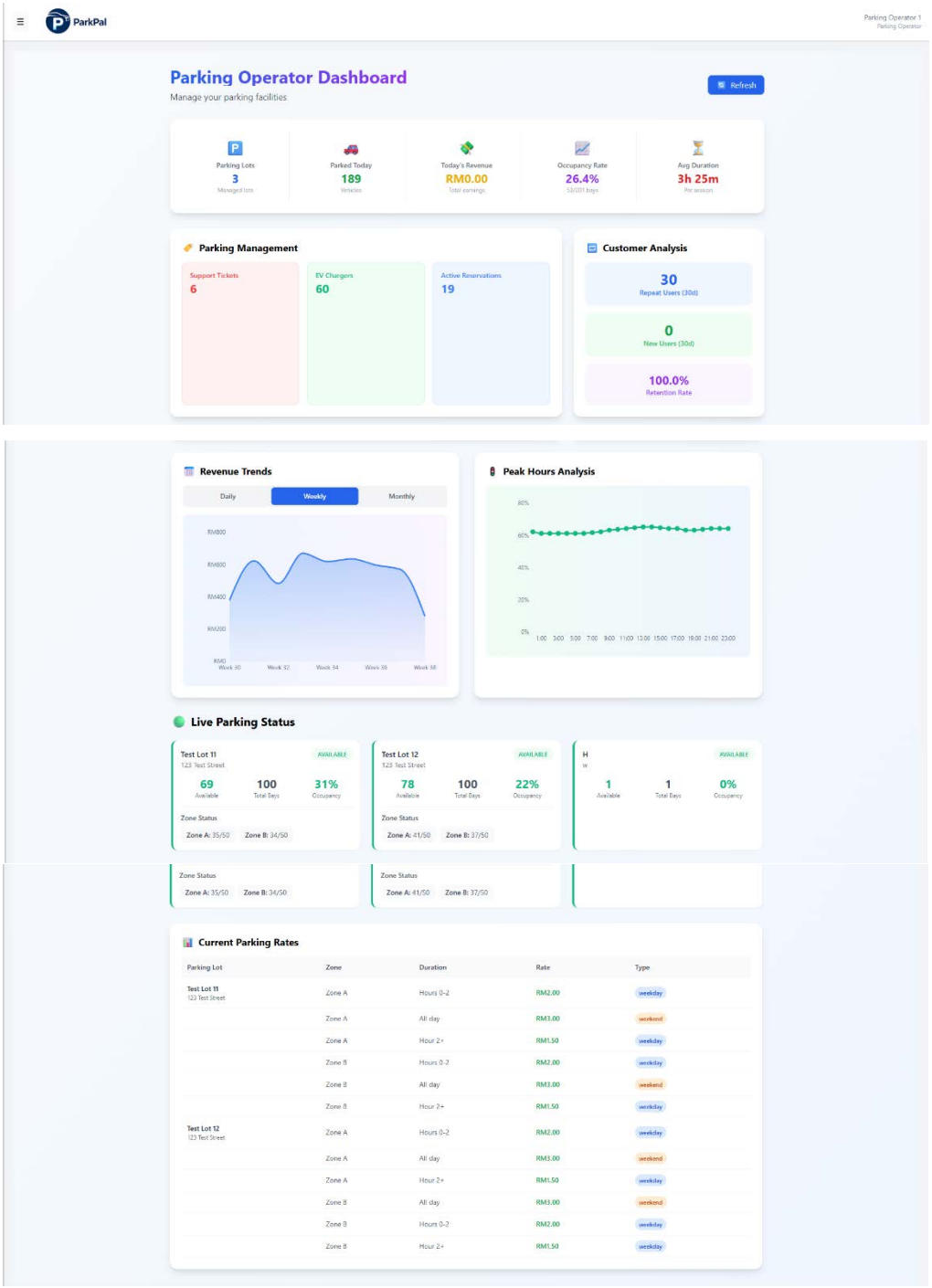


Figure 5.55: Parking Operator Dashboard

The Parking Operator Dashboard provides an overview of parking operations and performance. It includes statistics, customer analysis, revenue trends (with tabs for daily, weekly, and monthly views), peak hour analysis, live parking status, and current parking rates. This allows operators to monitor and manage their parking lots effectively.

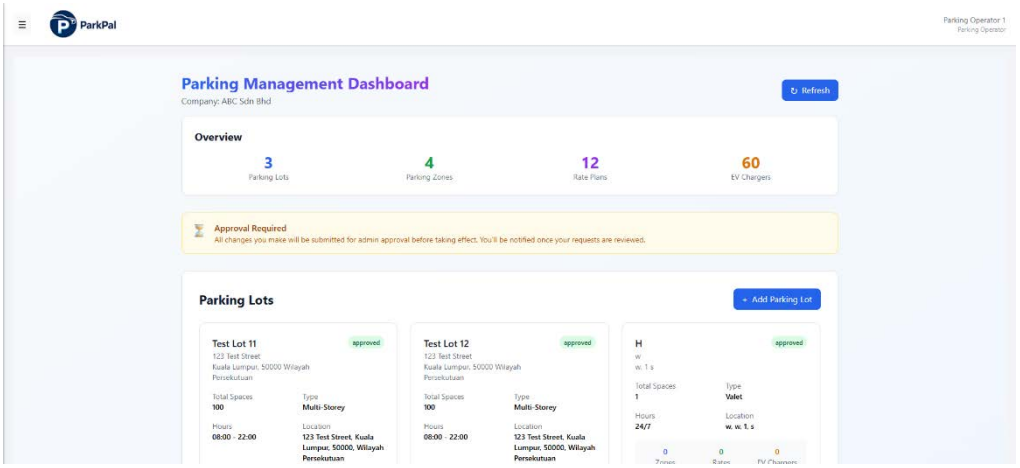


Figure 5.56: Parking Management Dashboard

The Parking Management Dashboard displays all parking lots managed by the operator, along with their details. Operators can click on a lot to view its zones, rates, and any associated EV chargers. An overview section provides summary insights for quick reference. All changes and actions made within this dashboard require admin approval before taking effect.

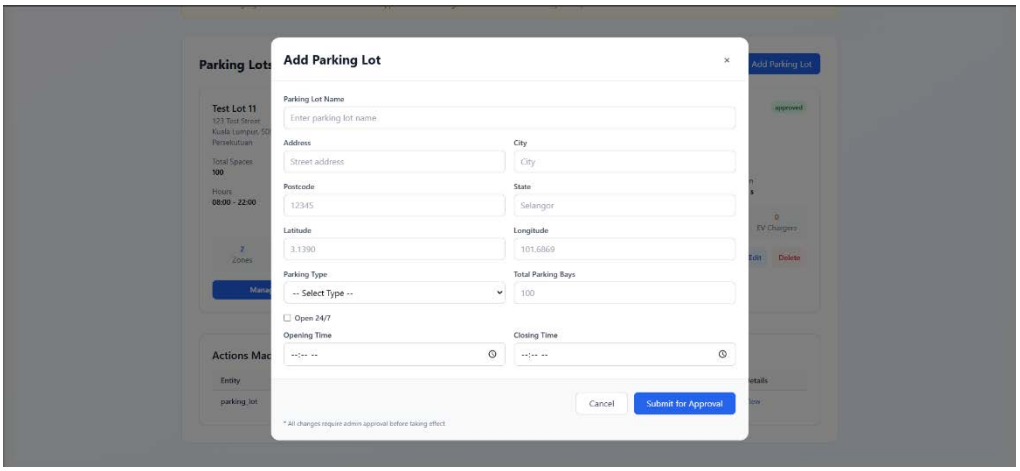


Figure 5.57: Parking Management Dashboard Add Parking Lot

Operators can add a new parking lot by entering its name, address, latitude and longitude, parking type, total number of parking bays, and opening and closing times. There is also an option to set the parking lot as open 24/7 for convenience. Once submitted, the new parking lot requires admin approval before it becomes active.

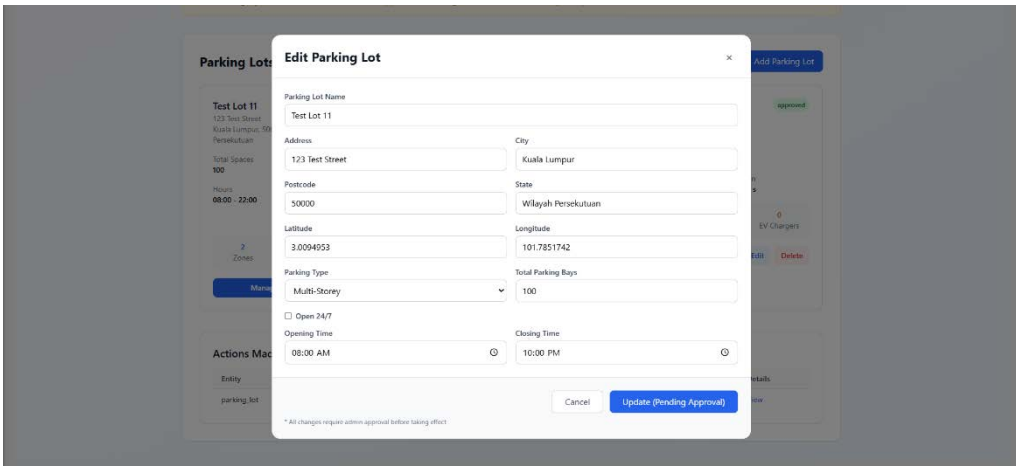


Figure 5.58: Parking Management Dashboard Edit Parking Lot

Operators can update an existing parking lot's details, including name, address, latitude and longitude, parking type, total parking bays, and opening and closing times, with the option to set it as open 24/7. All edits must be submitted for admin approval before the changes take effect.

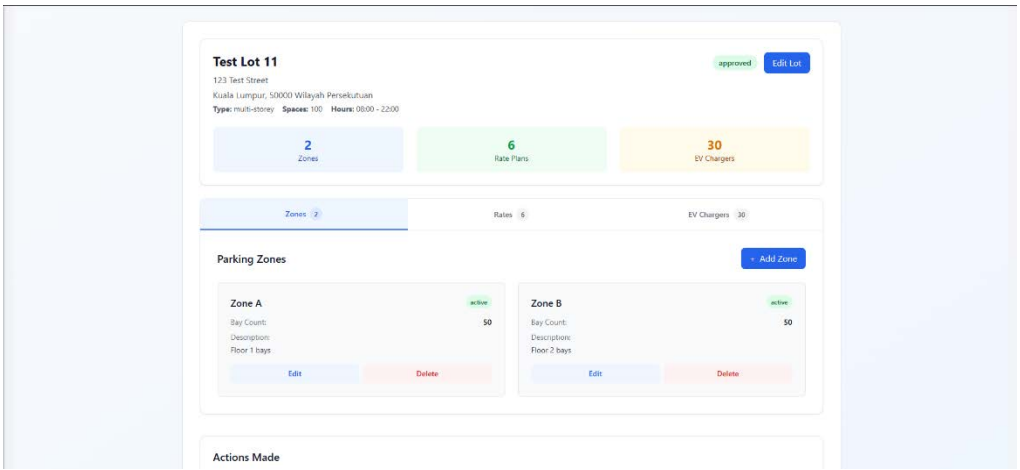


Figure 5.59: Parking Management Dashboard Zone Tab

Clicking on a parking lot opens its detailed view, organized into tabs for Zones, Rates, and EV Chargers. The Parking Zones tab displays a list of all zones within the lot, along with key details for each zone, allowing operators to manage and review their parking areas efficiently.

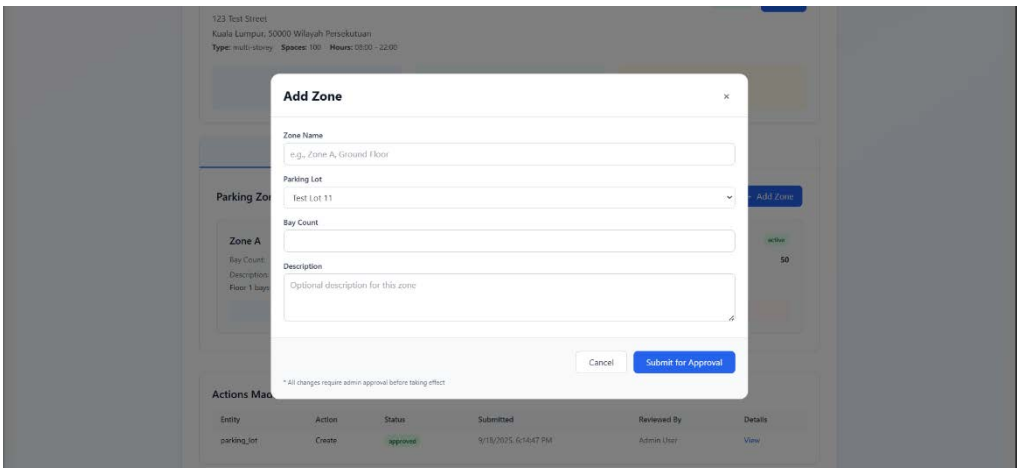


Figure 5.60: Parking Management Dashboard Add Zone

Operators can update the details of a parking zone, including its name, associated parking lot, number of parking bays, and a description. All changes must be submitted for admin approval before they take effect.

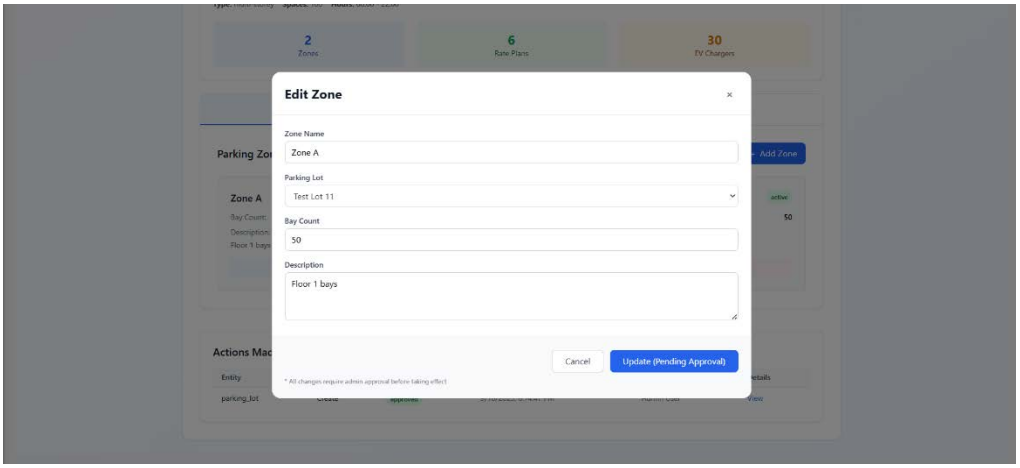


Figure 5.61: Parking Management Dashboard Edit Zone

Operators can edit a parking zone using the same fields, such as name, associated parking lot, number of bays, and description. All edits must be submitted for admin approval before taking effect.

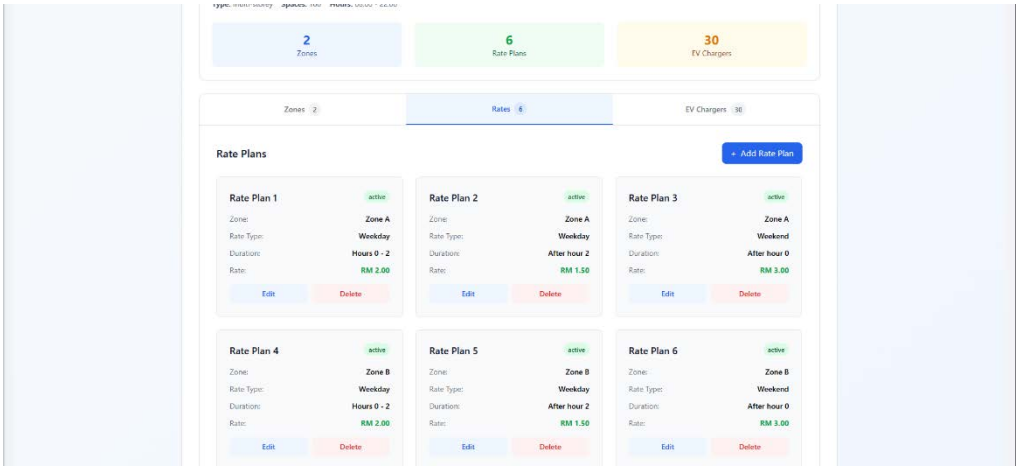


Figure 5.62: Parking Management Dashboard Rates Tab

The Rates tab displays all parking rates associated with the lot and their corresponding zones. Key details such as rate type, applicable zone, and pricing are shown. Operators can view, add, or edit rates, with all changes requiring admin approval before taking effect.

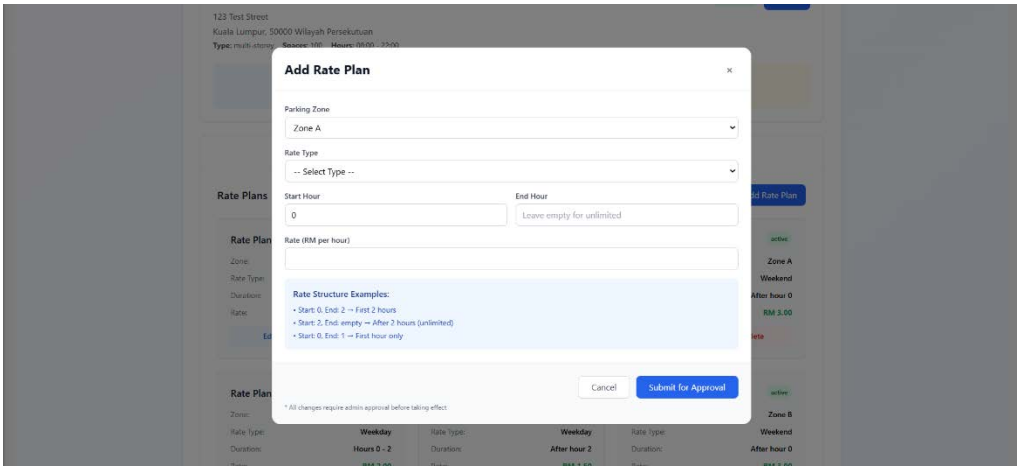


Figure 5.63: Parking Management Dashboard Add Rate Plan

Operators can add a new rate plan by specifying the associated zone, rate type, start and end hours, and the rate amount. Once submitted, the new rate plan requires admin approval before it becomes active.

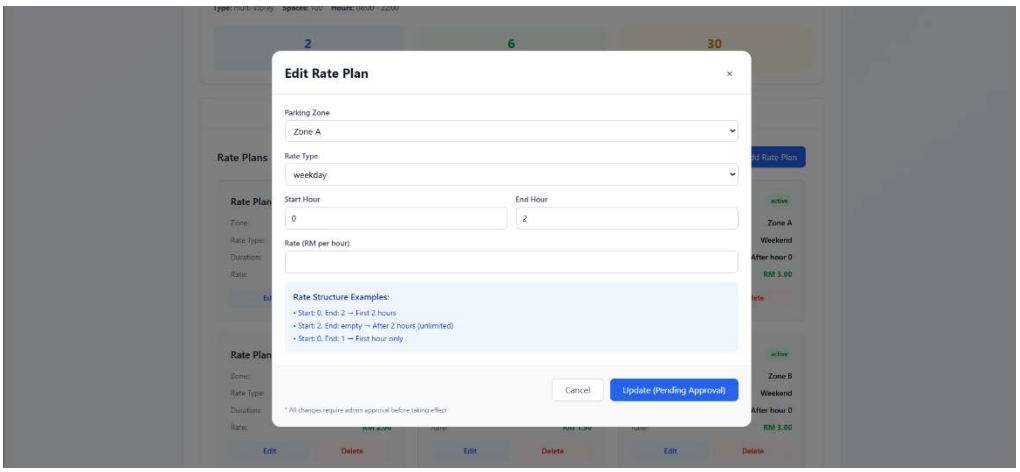


Figure 5.64: Parking Management Dashboard Edit Rate Plan

Operators can edit an existing rate plan using the same fields, including zone, rate type, start and end hours, and rate amount. All edits must be submitted for admin approval before they take effect.

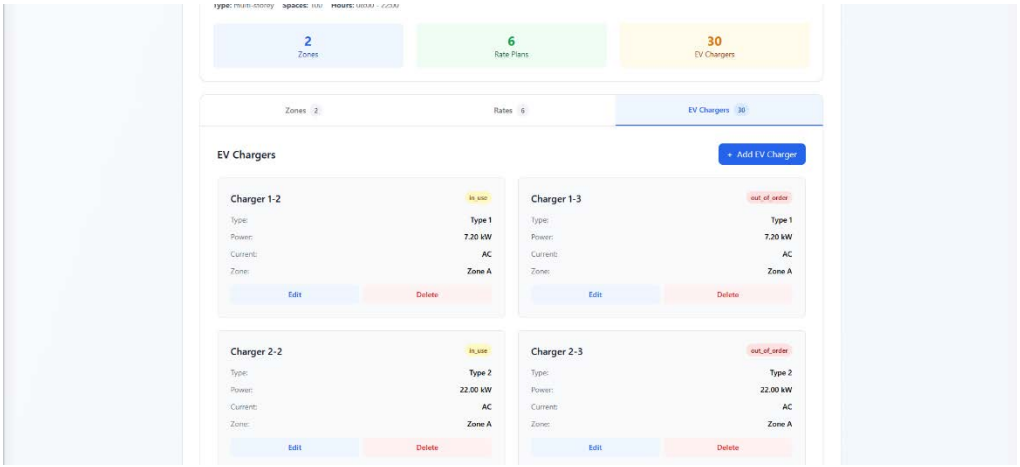


Figure 5.65: Parking Management Dashboard EV Chargers Tab

The EV Chargers tab displays all EV chargers associated with the parking lot, including key details such as charger type, power rating, availability, and associated zone. Operators can view, add, or edit chargers, with all changes requiring admin approval before taking effect.

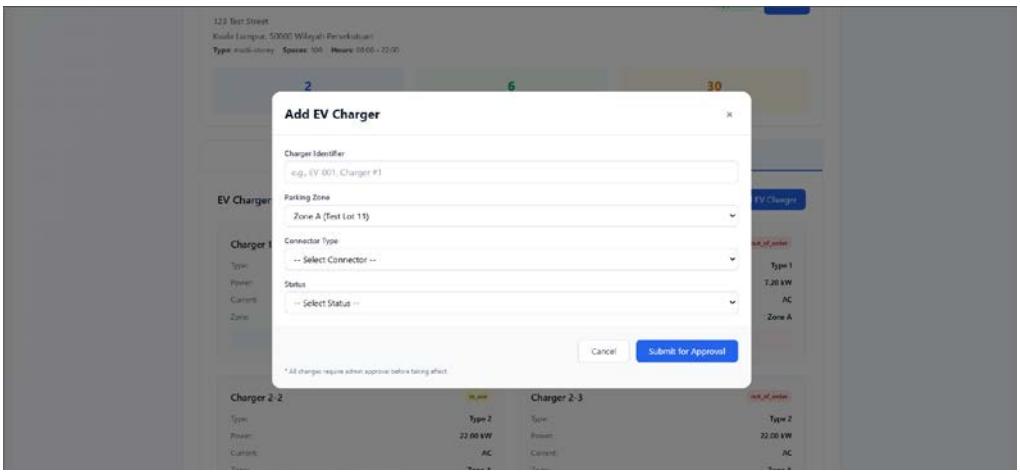


Figure 5.66: Parking Management Dashboard Add EV Charger

Operators can add a new EV charger by specifying the charger identifier, associated zone, type, and status. Once submitted, the new charger requires admin approval before it becomes active.

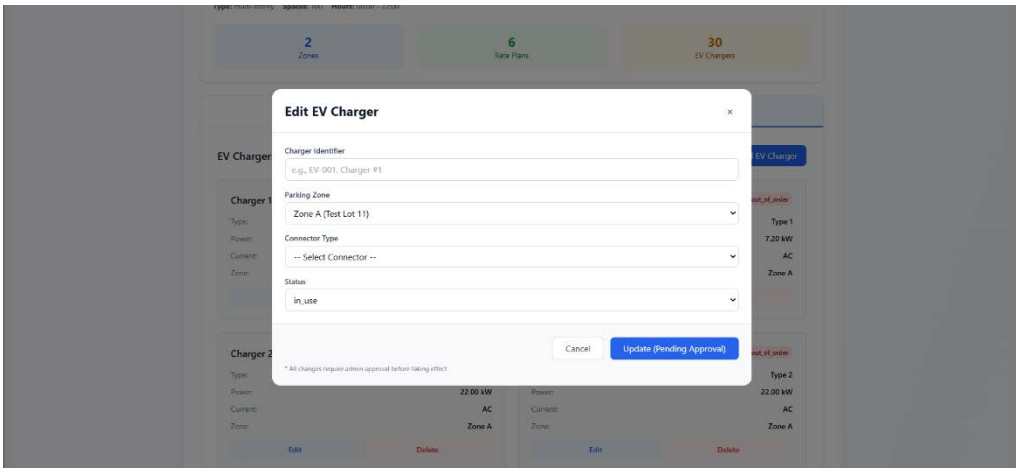


Figure 5.67: Parking Management Dashboard Edit EV Charger

Operators can edit an existing EV charger using the same fields, which are charger identifier, associated zone, type, and status. All edits must be submitted for admin approval before taking effect.

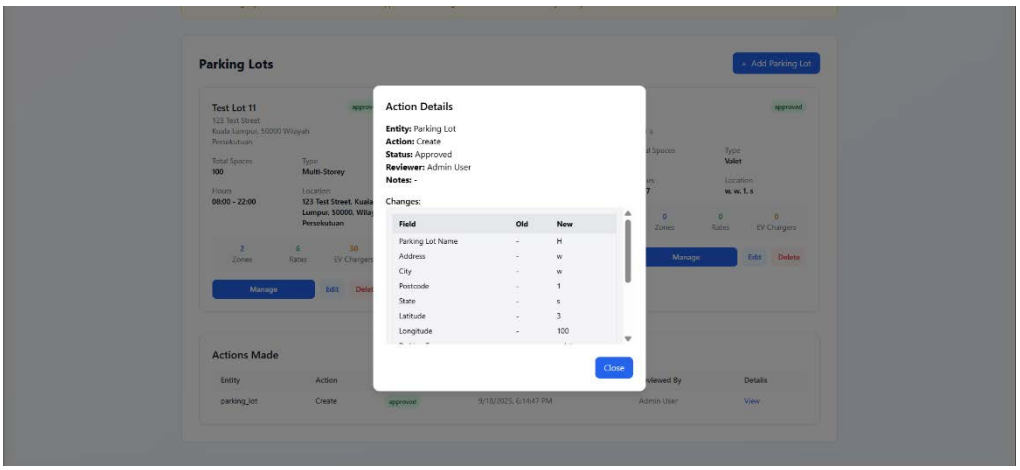


Figure 5.68: Parking Management Dashboard Actions Made

Operators can review the actions they have performed within the system and view detailed information for each action. This allows them to track changes they have made and monitor the status of submissions pending admin approval.

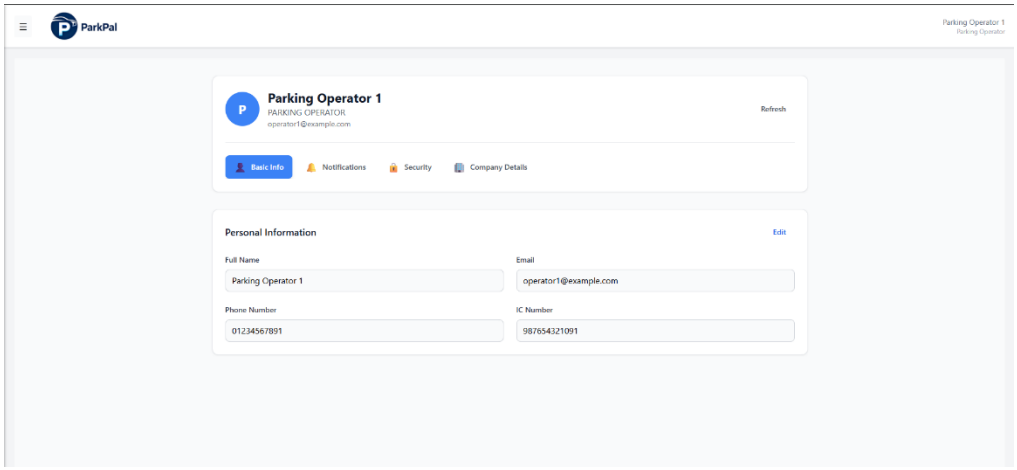


Figure 5.69: Operator Profile Page

The Basic Info tab displays the user's personal information, including full name, email, phone number, and IC number. While the name, email, and phone number can be updated, the IC number is read-only and cannot be changed.

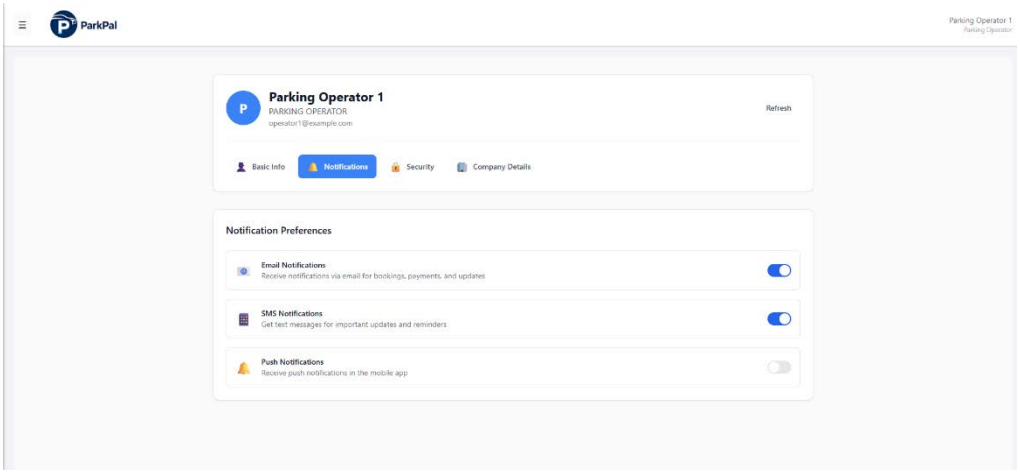


Figure 5.70: Operator Profile Page Notifications Tab

Under the Notifications tab, users can manage their notification preferences. They can toggle options for receiving alerts via email, SMS, or push notifications according to their preferences.

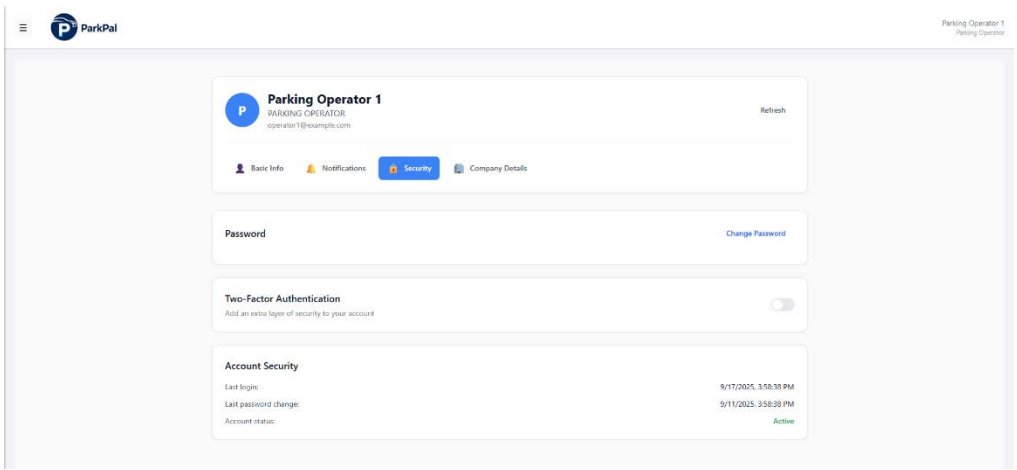


Figure 5.71: Operator Profile Page Security Tab

Under the Security tab, users can change their password and toggle two-factor authentication for enhanced account security. This tab also displays relevant account security information to keep users informed about their account protection status.

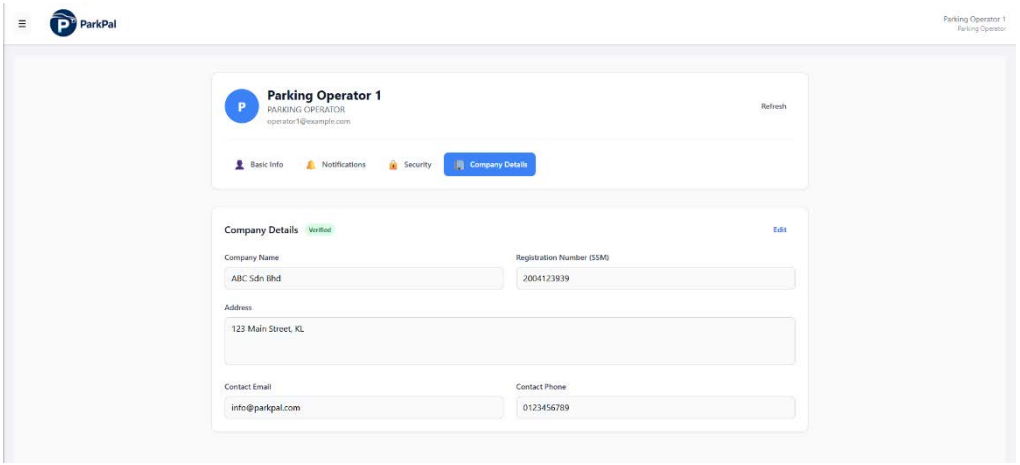


Figure 5.72: Operator Profile Page Company Tab

The Company tab displays the operator's company information, including company name, registration number, address, email, and phone number. Users can view and update these details as needed.

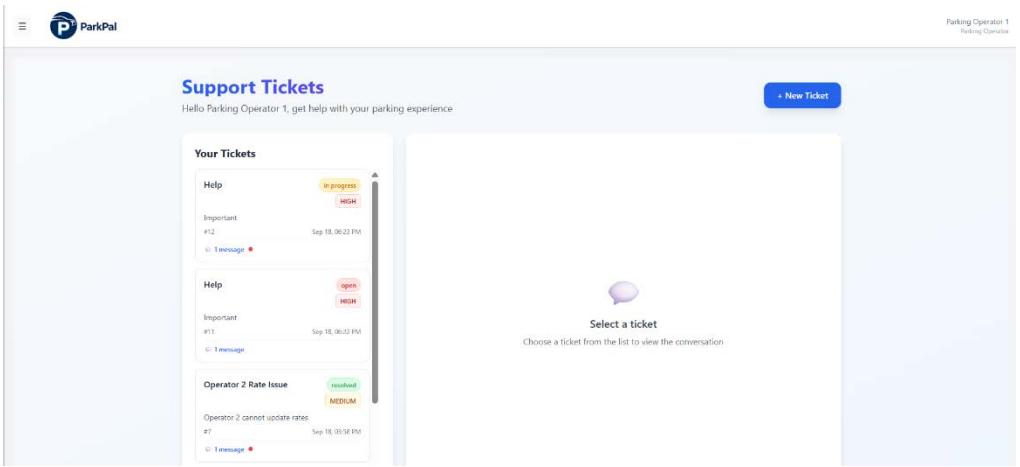


Figure 5.73: Support Tickets Page

The Support Ticket page displays a list of all tickets submitted by the driver. Users can view the status of existing tickets and create new tickets to report issues or request assistance.

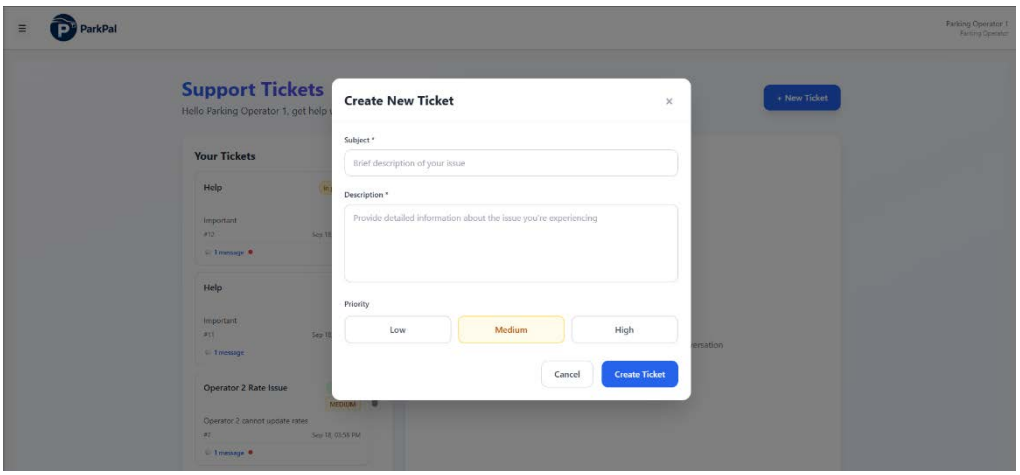


Figure 5.74: Create New Ticket

When creating a new support ticket, the driver is required to enter the ticket type, subject, detailed description, and select a priority level. This ensures the issue is properly categorized and addressed promptly.

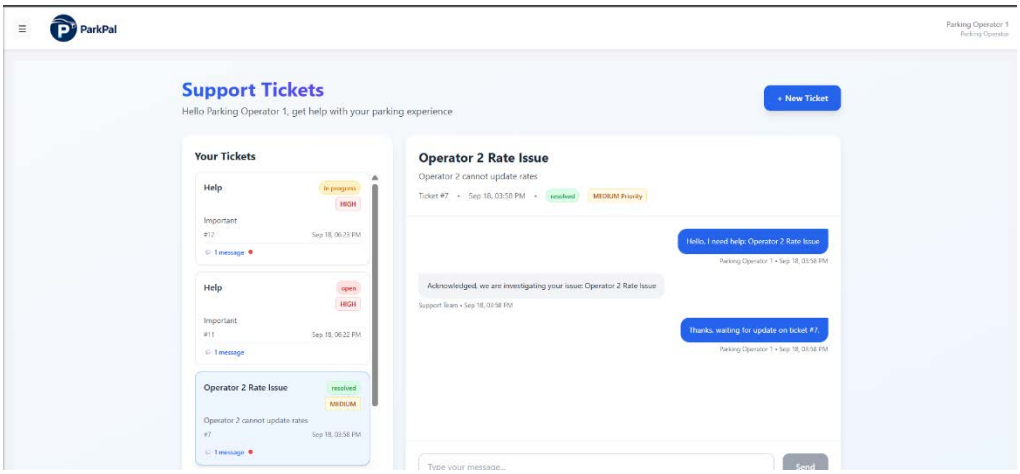


Figure 5.75: View and Send Support Ticket Message

The Support Ticket Messages feature functions like a text messaging system, allowing drivers and support staff to communicate within each ticket. Users can send and receive messages, providing updates, clarifications, and responses related to their submitted issues.

5.6.4 Admin Web Interface

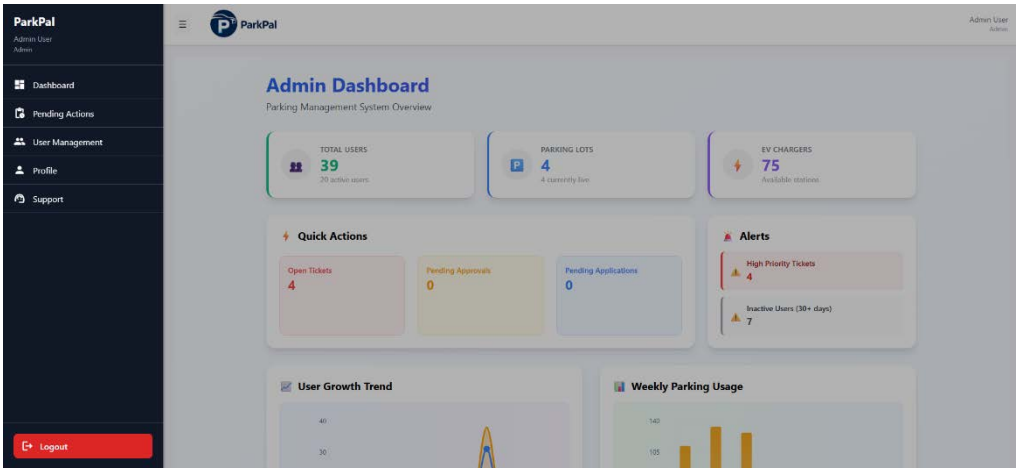


Figure 5.76: Drawer Navigation

This is the Drawer Navigation, which allows users to access other pages. Users can also view their name and role, and log out directly through the drawer.

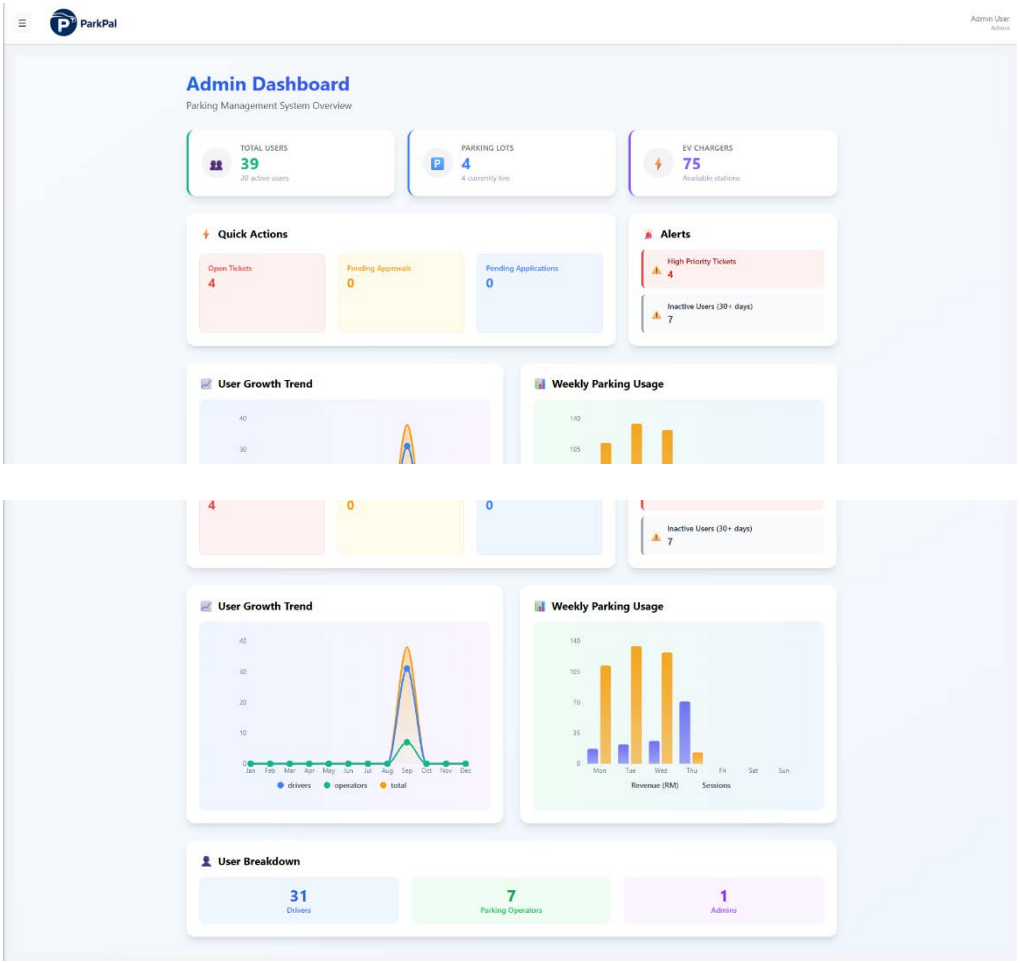


Figure 5.77: Admin Dashboard

The Admin Dashboard provides an overview of system performance and user activity. It displays key statistics, alerts, and user growth trends. Weekly parking usage is visualized to monitor demand patterns, while a user breakdown gives insights into different user types, helping admins manage and oversee the system effectively.

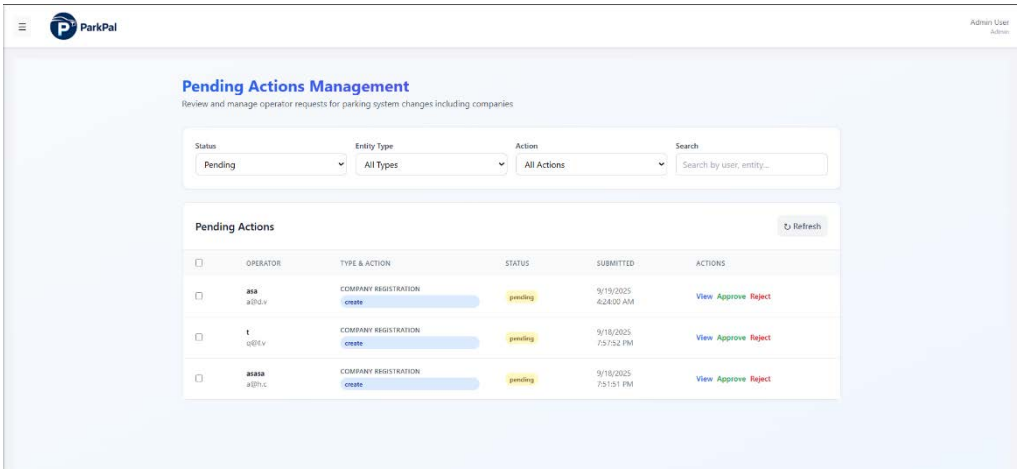


Figure 5.78: Admin Pending Actions Management

This section allows admins to view all pending actions submitted by operators, including additions, updates, or deletions related to companies, parking lots, zones, rates, and EV chargers. Admins can view details of each action and choose to approve or reject them. The interface also provides filtering options to easily locate specific actions.

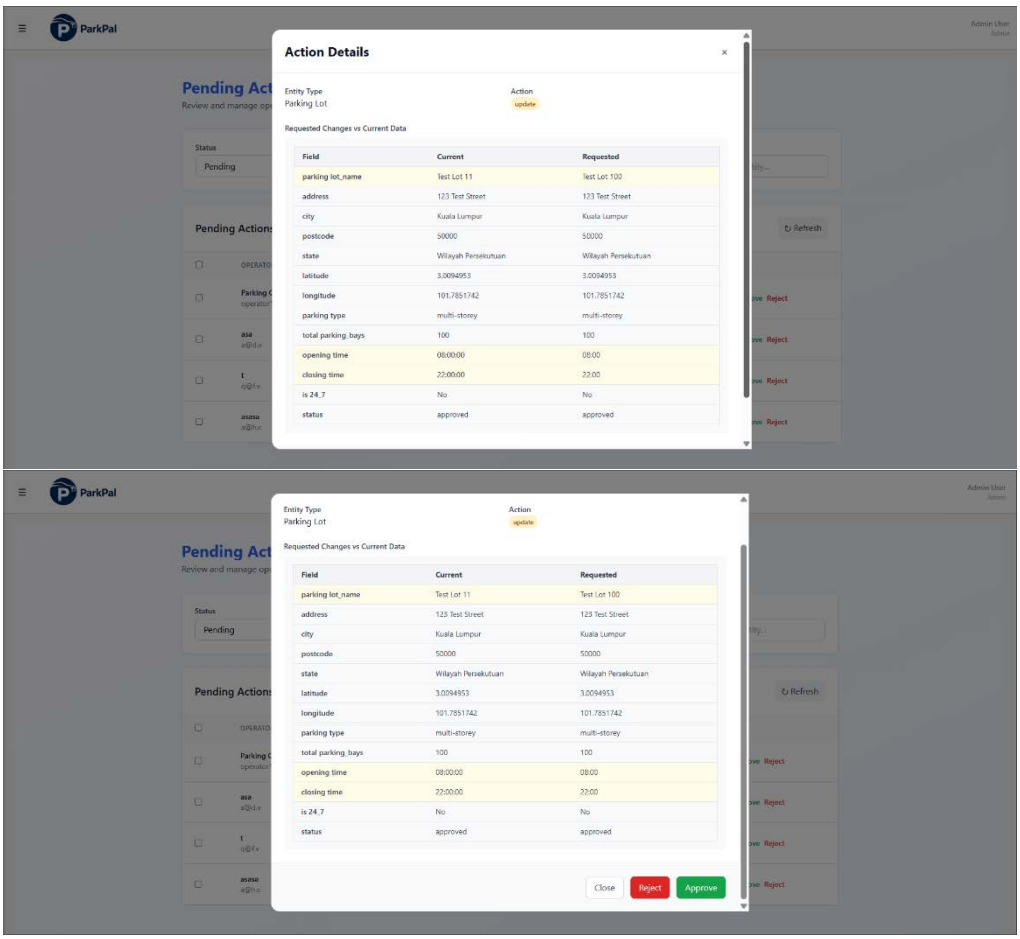


Figure 5.79: Admin Pending Actions Details

Admins can view the full details of each pending action submitted by operators. Any changes compared to the current data are highlighted, allowing admins to easily identify modifications before approving or rejecting the action. Admins can choose to either approve or reject the action directly from this view.

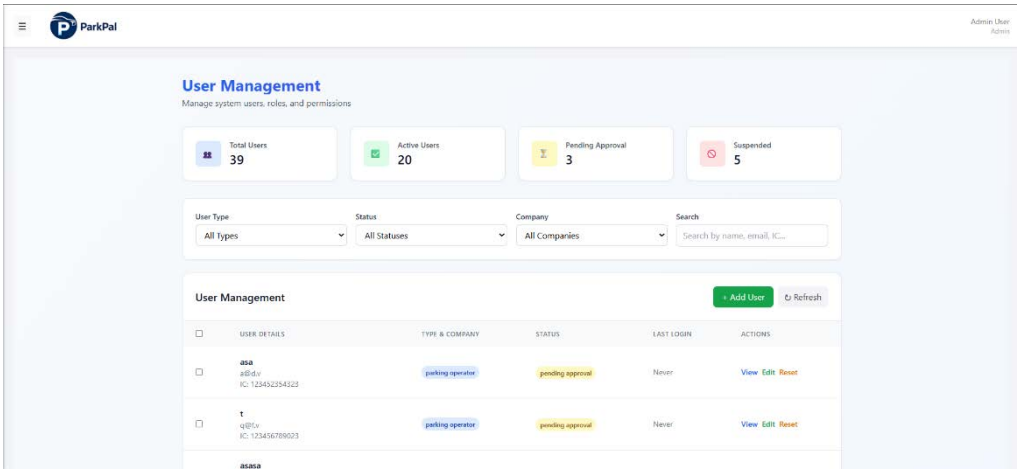


Figure 5.80: Admin User Management

This section allows admins to view all users in the system along with relevant statistics. Admins can filter users, add new accounts, edit existing ones, and view detailed information for each user to manage the system effectively.

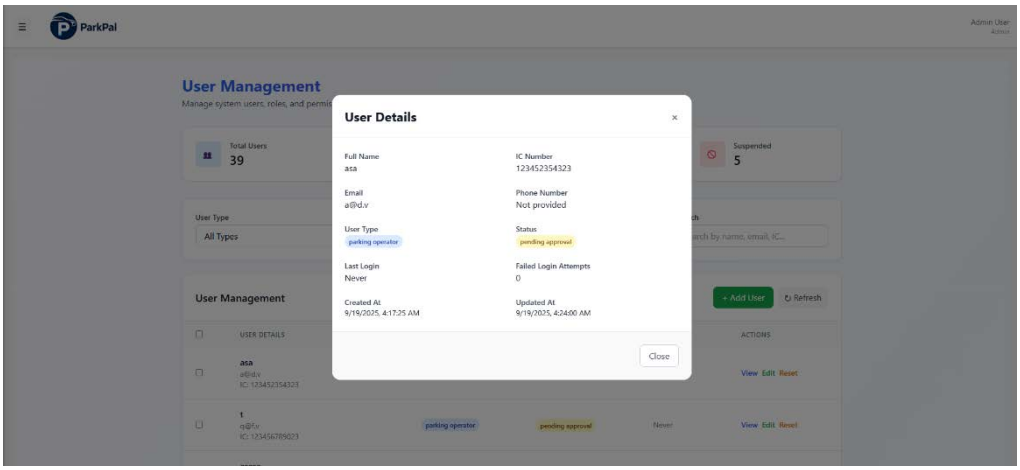


Figure 5.81: Admin User Management User Details View

Admins can access a detailed view of each user’s account. This comprehensive view helps admins monitor user activity and manage accounts effectively.

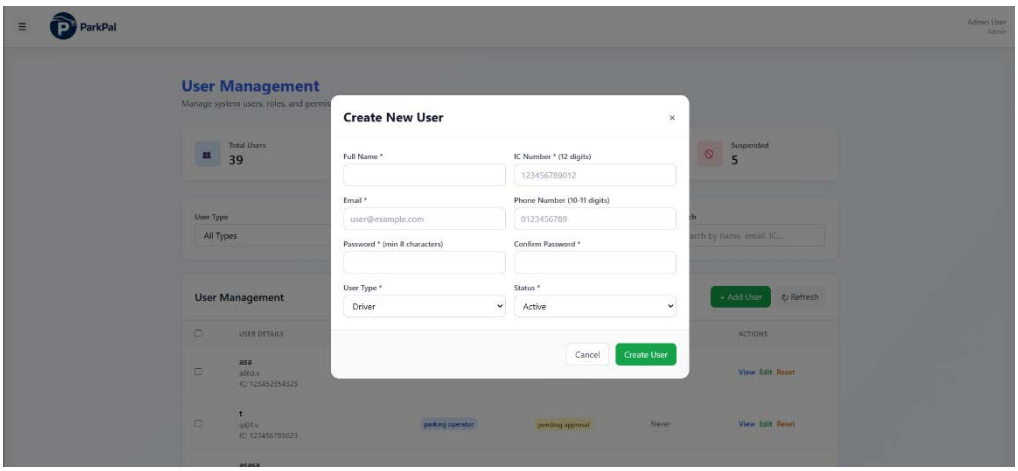


Figure 5.82: Admin User Management Add New User

In this section, admins can create new user accounts, including adding other admin accounts since admins cannot self-register. Required details are entered during account creation to ensure proper setup and access control.

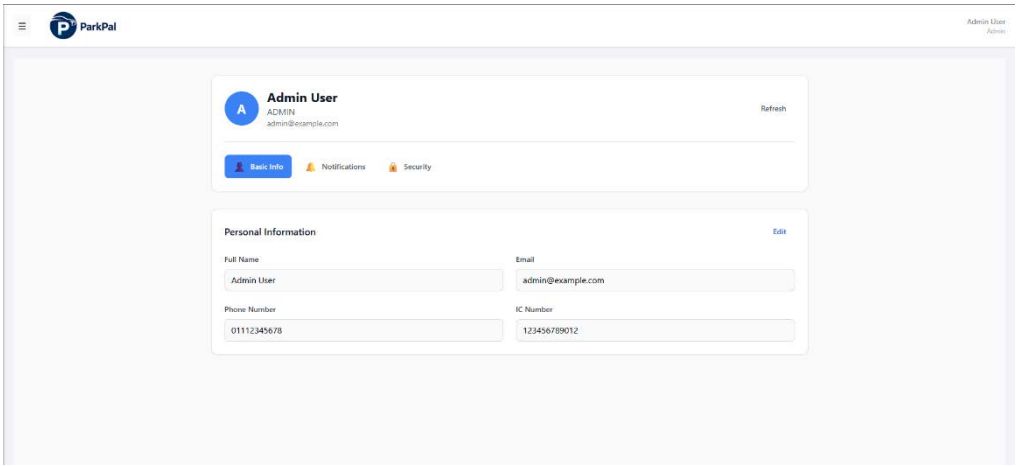


Figure 5.83: Admin Profile Page

The Basic Info tab displays the user's personal information, including full name, email, phone number, and IC number. While the name, email, and phone number can be updated, the IC number is read-only and cannot be changed.

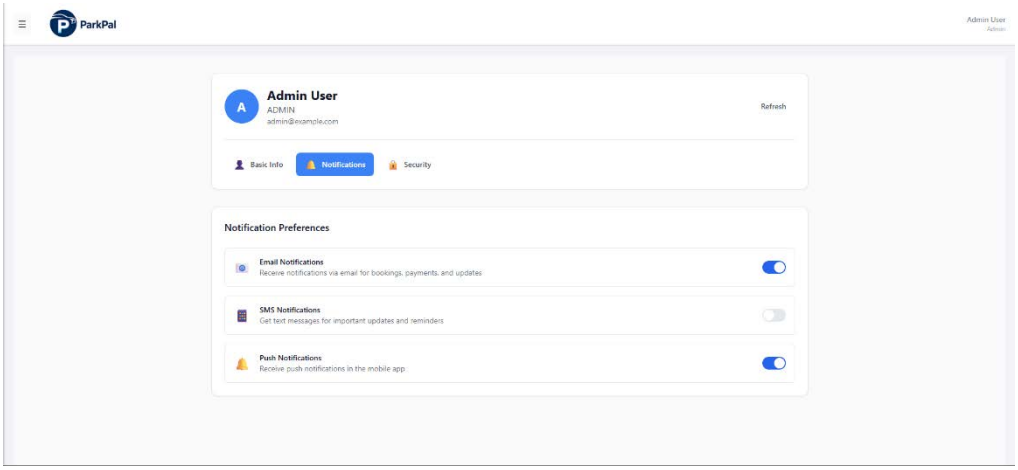


Figure 5.84: Admin Profile Page Notifications Tab

Under the Notifications tab, users can manage their notification preferences. They can toggle options for receiving alerts via email, SMS, or push notifications according to their preferences.

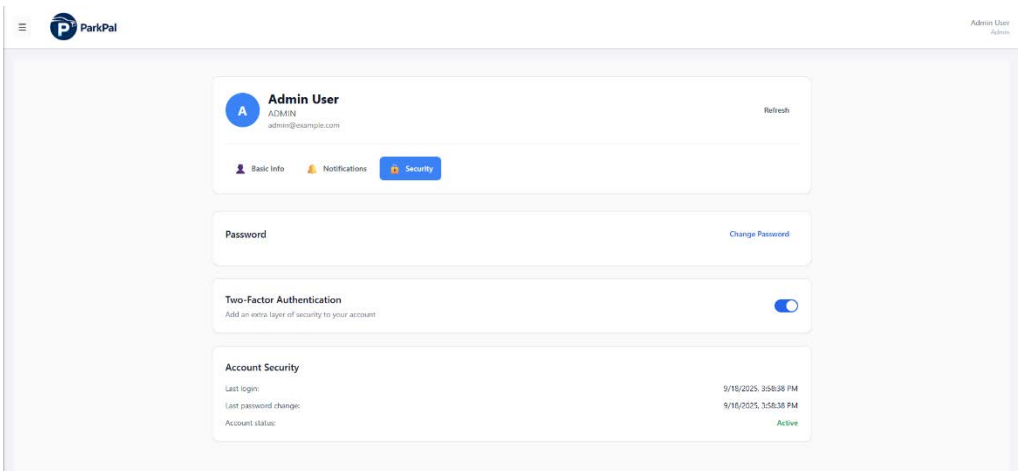


Figure 5.85: Admin Profile Page Security Tab

Under the Security tab, users can change their password and toggle two-factor authentication for enhanced account security. This tab also displays relevant account security information to keep users informed about their account protection status.

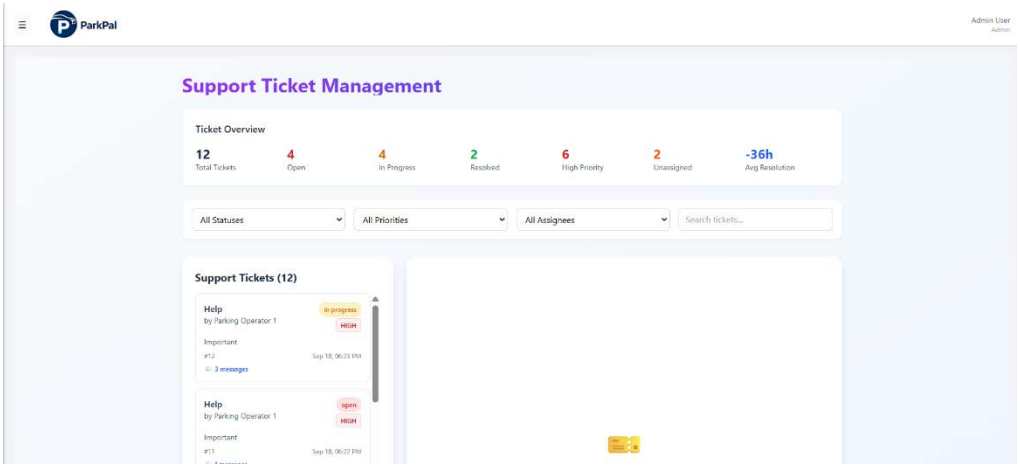


Figure 5.86: Admin Support Ticket Management

Admins can view all support tickets submitted by users, along with relevant statistics. The system provides filtering options to easily locate specific tickets and monitor the status of user-reported issues.

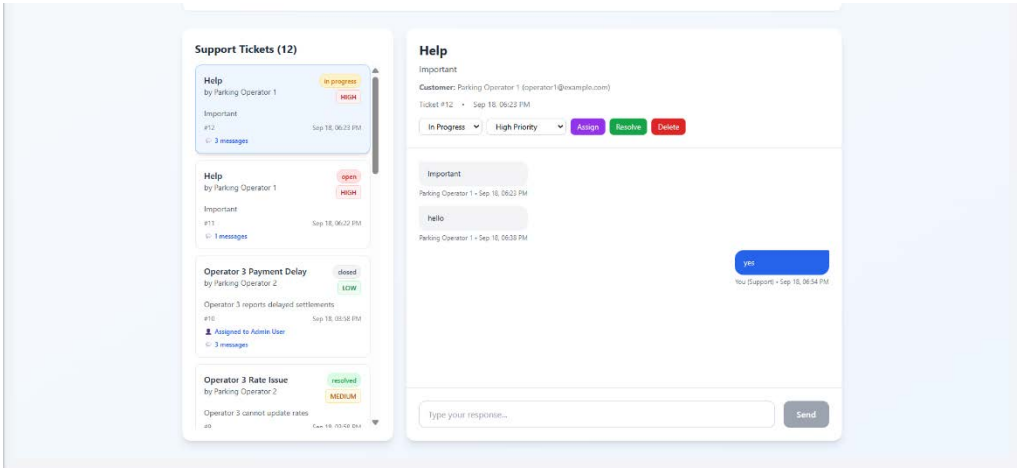


Figure 5.87: Admin View and Send Support Ticket Message

Within each support ticket, admins can send and receive messages with users, similar to a text messaging system. This allows admins to provide updates, request clarifications, and respond to user-reported issues efficiently.

CHAPTER 6
SYSTEM IMPLEMENTATION

6.1 Introduction

This chapter explains how the ParkPal system was implemented to create a working prototype. The implementation includes the backend, frontend, and AI-based vehicle recognition using the Gemini 2.5 Flash model. The backend was developed using Laravel and PostgreSQL to handle user authentication, data management, and system logic. The frontend, built with React Native, provides a user-friendly interface for drivers and administrators. A Python module was integrated to detect vehicle license plates and attributes automatically. Tools such as NGROK and Expo were also used to support testing and cross-platform development. Overall, this chapter describes how the system design was transformed into a functional and integrated solution.

6.2 Backend Implementation

6.2.1 Authentication and Authorization

The parking management system uses Bearer token authentication powered by Laravel Sanctum to ensure secure and scalable authentication. When a user successfully logs in or registers, the backend generates a unique Bearer token using Sanctum. This token is then sent to the client application and included in the Authorization header of all subsequent API requests in the following format: Authorization: Bearer <token>.

Passwords are hashed using Laravel's Hash facade (Bcrypt) before being stored in the database. This ensures that even if the database is compromised, plain-text passwords are never exposed. During login, the system verifies the provided password by comparing it with the hashed version stored in the database using Hash::check.

The system also supports token revocation. When a user logs out, their current access token is invalidated (currentAccessToken()->delete()), preventing further use of that token. Additionally, the system updates the last_login_at field on successful login to track account activity.

For authorization, each Bearer token contains user-specific information and is tied to a particular account. Access to different parts of the system is controlled using role-based middleware, including driver, parking_operator, and admin. This ensures that each type of user only has access to the features and data appropriate for their role. For example, parking operators must complete a company setup and obtain admin approval before being granted full access, while suspended or rejected operators are blocked from using the system.

Table 6.1: Key Methods Used in the Authentication and Authorization Flow

Method	Library / Middleware	Usage
Hash::make	Laravel Hash (Bcrypt)	Hashes plain-text passwords for secure storage in the database.
Hash::check	Laravel Hash (Bcrypt)	Compares a plain-text password with its hashed version.
createToken	Laravel Sanctum	Generates a new Bearer token for user authentication.
auth:sanctum middleware	Laravel Sanctum	Ensures only authenticated requests with valid tokens can access APIs.
role:driver/ operator/ admin	Custom Middleware	Restricts access to routes based on user role.
currentAccess Token()-> delete()	Laravel Sanctum	Revokes a user's current token during logout.

6.2.2 Database Integration (PostgreSQL)

To interact with PostgreSQL, the backend is implemented using the MVC (Model-View-Controller) architecture provided by Laravel. This design separates the application into three main components.

The models define the structure and relationships of database tables in PostgreSQL. Models represent entities such as users, parking lots, reservations, and transactions, and provide an interface to perform database operations. The

controllers handle the logic for processing requests, interacting with Models, and preparing responses for the frontend. Controllers manage CRUD (Create, Read, Update, Delete) operations and implement business rules such as reservation validation, payment processing, and user role management. In this system, the Views are represented by the React Native frontend, which retrieves and displays data provided by the backend.

To facilitate communication between the frontend and backend, Axios is used as a promise-based HTTP client. Axios sends HTTP requests (GET, POST, PUT, PATCH, DELETE) from the React Native application to the Laravel backend API. This enables the frontend to perform CRUD operations on PostgreSQL data, streamlining the interaction between the mobile/web interface and the server.

By combining PostgreSQL with Laravel's MVC architecture, the system achieves a structured, maintainable, and scalable design while ensuring efficient data storage, retrieval, and processing for all application modules. PostgreSQL's robust ACID compliance and indexing capabilities also ensure data consistency and efficient query performance, especially for large datasets such as transaction records and parking sessions.

6.2.3 Real-Time Communication (Reverb and Pusher-js)

The system employs Reverb and Pusher-js to enable real-time updates on the admin dashboard. Using Pusher's private channels, the dashboard subscribes to events such as DashboardUpdated, which triggers immediate updates to metrics without requiring manual page refreshes. This allows administrators to view up-to-date information on total users, active users, parking lot availability, EV charger status, and high-priority alerts. Reverb simplifies event handling and ensures smooth communication between the frontend and backend. In addition, cleanup routines are implemented to safely leave channels and stop event listeners, preventing memory leaks or duplicate updates during prolonged usage.

6.3 Frontend Implementation

6.3.1 Navigation Structure (React Navigation)

In the Park Pal mobile app for drivers, React Navigation is used to manage all navigation flows. The app implements a combination of stack and drawer navigators to create a nested navigation system.

The drawer navigator serves as the main navigation system. It provides access to primary screens such as the driver dashboard, available parking lots, EV reservations, transaction history, profile, and support tickets. The drawer is hidden by default to save screen space, and can be opened by tapping a menu button located in the top-left corner of the home screen header.

The stack navigator allows navigation between secondary pages and supports typical forward/backward navigation within the app. The drawer navigator is nested as the initial screen of the stack navigator, ensuring that users can access the main menu from anywhere in the app.

This combination of stack and drawer navigation creates a smooth and intuitive user experience, allowing drivers to move seamlessly between the dashboard, reservations, transaction history, and other key features.

6.3.2 Local Storage (Async Storage)

In the Park Pal mobile application, Async Storage is utilized as a local key-value storage system for maintaining essential user data on the device. This includes authentication tokens and user profile information, allowing the app to persist the user’s login state between sessions.

When a user logs in, the authentication token and user data are saved to Async Storage. This ensures that even if the app is closed or the device is restarted, the driver remains logged in. Upon reopening the app, the stored token and user data are retrieved automatically to authenticate the user without requiring manual login.

During logout, the token and user data are removed from Async Storage, effectively ending the session. Additionally, Async Storage is used throughout the app for temporarily storing other session-related data, such as selected chargers or reservation details.

By leveraging Async Storage, the app provides a seamless user experience, keeping the driver logged in and retaining necessary session data until the user explicitly logs out or the data is cleared.

6.3.3 Data Visualization

6.3.3.1 React Native Chart Kit

In the development of the Driver Dashboard, react-native-chart-kit is used to create visual representations of parking and vehicle usage data. The library provides an easy way to integrate charts such as line charts and bar charts directly into the mobile app, helping drivers understand their parking behavior and usage trends over time.

This react-native-chart-kit is chosen for its simplicity and pre-built components, allowing for rapid chart integration without extensive customization. While it may not offer as many advanced options as other charting libraries, it provides responsive and clear visualizations that meet the needs of the dashboard.

In this application, two primary chart types are implemented. The Monthly Usage Line Chart displays the total parking sessions per month, helping drivers see trends in their parking activity over the past months. Meanwhile, the Weekly Usage Bar Chart shows weekly parking activity, providing a quick overview of short-term usage patterns.

Both charts are encapsulated within reusable ChartCard components and receive usage data from custom hooks like useDriverDashboardData. These charts appear in the driver’s dashboard, allowing them to monitor their parking habits at a glance. The charts are interactive and scrollable horizontally, giving drivers an intuitive way to switch between monthly and weekly views.

By using react-native-chart-kit, the dashboard delivers visual insights that improve driver awareness of parking patterns, enabling better planning and efficient use of parking resources.

6.3.3.2 Recharts

Both the admin and parking operator dashboards use Recharts to render complex datasets into interactive visualizations. On the admin dashboard, area charts, line charts, and bar charts display trends such as user growth, weekly parking usage, and high-priority tickets. Custom tooltips and legends enhance data readability and highlight critical insights, while responsive containers ensure charts adapt to different screen sizes.

For parking operators, Recharts is employed to visualize revenue trends, occupancy rates, EV reservations, and peak-hour analyses. The dashboard provides interactive components such as period selectors, which allow users to switch between daily, weekly, or monthly data views. Live status cards summarize parking lot availability and occupancy, while quick action cards provide one-click access to ticket management, EV charger management, and reservation oversight. Together, these visualizations transform raw data into actionable insights, supporting efficient decision-making for both administrators and operators.

6.3.4 Location Services (Expo-Location)

The mobile application leverages Expo Location to capture and manage the driver's real-time geographic coordinates. Within custom hooks such as useNearbyParkingLots and useEvReservations, the system requests foreground location permissions on iOS and Android devices. Once granted, the app retrieves high-accuracy GPS coordinates using Location.getCurrentPositionAsync, which are then stored in the local state as userLocation. On the web, the system falls back to the browser's navigator.geolocation API for location access.

These coordinates are critical for dynamically querying nearby parking lots and EV chargers, allowing the application to calculate distances, filter results by radius, and sort by availability or proximity. The location data also supports reservation management, enabling the app to check whether a user can access a charger or parking space on time. Default coordinates (e.g., central Kuala Lumpur) are used as a fallback when permissions are denied or location retrieval fails, ensuring continuous functionality. Overall, Expo Location

provides a seamless and cross-platform method for integrating real-time geolocation into the driver experience.

6.4 AI & Detection Module

6.4.1 Python-Uvicorn Service Setup

Uvicorn serves as the ASGI server for running the Python backend that powers vehicle detection and recognition. It processes requests sent from the Laravel backend, such as analyzing parking lot images to extract vehicle license plates and attributes using the Gemini Flash 2.5 model. Uvicorn's high-performance, asynchronous capabilities make it well-suited for handling multiple AI inference requests efficiently, ensuring real-time responses during vehicle entry and exit.

6.4.2 Gemini 2.5 Flash Integration

The system integrates Gemini 2.5 Flash, an LLM, to perform vehicle detection from uploaded images. This module extracts key vehicle attributes, such as license plate number, make, model, and color, which are then used to facilitate parking reservation and verification.

The detection process is implemented in Python and interacts with the Gemini 2.5 Flash API. The Python service acts as a middleware between the LLM and the Laravel backend, ensuring smooth data flow between the AI inference output and database storage. Each uploaded vehicle image is sent to the model with a structured prompt requesting a JSON-formatted output. Once the response is received, the module performs normalization of the detected attributes as stated in the table below.

Gemini 2.5 Flash was chosen over the earlier Gemini 2.0 version, which has since been deprecated and is no longer actively supported. Tulsee Doshi (2025) mentioned that the newer release provides faster response times and cost-efficiency, excelling at high-volume, latency-sensitive tasks, which is suitable for our system. By adopting Gemini 2.5 Flash, the system benefits from improved stability, compatibility with the latest API features, and long-term reliability.

Table 6.2: Vehicle Attribute Normalization

Attribute	Normalization Method	Example
License plate	Cleans invalid characters and converts text to uppercase.	"abc 123" → "ABC 123"
Make and model	Capitalizes and formats strings consistently.	"toyota corolla" → "Toyota Corolla"
Color	Maps different color names to a standardized set of colors.	"burgundy" → "red"

The vehicle detection module integrates with the backend to automatically capture vehicle details during entry and exit at the parking lot. This reduces manual entry errors and ensures accurate tracking of vehicles, while detection times are logged for performance monitoring. This integration enhances the app's capabilities, allowing automated recognition of vehicles.

6.4.3 Vehicle Detection Setup and Workflow

The vehicle recognition component was implemented using the Gemini 2.5 Flash model integrated with a Python–Uvicorn service. The purpose of this module is to automatically extract vehicle information (license plate, make, model, and color) from captured images at the parking lot entry point.

6.4.3.1 Data Collection and Preparation

A total of 20 real vehicle images were collected from Roboflow, representing various makes, models, and colors under different lighting conditions. To evaluate detection performance, a ground truth dataset was manually prepared, recording the actual license plate number, make, model, and color for each image. It serves as a benchmark to assess the system's accuracy.

6.4.3.2 Detection Workflow

The detection and matching process follows the steps below.

1. Capture vehicle image at entry (simulated using Roboflow dataset images).
2. Send image to the Python server hosting Gemini 2.5 Flash API.
3. Extract attributes which are license plate, make, model, and color.
4. Normalize attributes for consistency (e.g., uppercase plates, standardized colors).
5. Match results with the registered user and vehicle records in the PostgreSQL database.
6. Create a new record in the parking_sessions table.
7. On exit, the system updates session details, calculates the parking fee, processes payment (simulated), and sends a driver notification.

6.4.3.3 System Integration Overview

The mobile frontend (React Native) allows users to initiate or view parking sessions. It communicates with the Laravel backend via HTTP requests (Axios), which interacts with PostgreSQL for all CRUD operations. For AI-based detection, the Laravel backend forwards captured images to the Python service for inference, receives JSON responses with detected attributes, and stores them in the database.

6.5 Development and Deployment Environment

6.5.1 NGROK for Local Testing

NGROK is used during development to expose the locally running backend servers (Laravel and Python) to the internet through secure tunnels. This makes it possible for the Expo-based mobile app to communicate with the backend services in real time, even when they are hosted on local machines. NGROK provides temporary public URLs for both the Laravel API and the Python AI server, ensuring seamless testing across physical devices without manual server deployment.

6.6 Conclusion

In conclusion, the implementation of the ParkPal system successfully combines web, mobile, and AI technologies into a cohesive platform for smart parking management. The integration of Laravel and PostgreSQL ensures secure and reliable backend operations, while React Native provides a unified and responsive frontend experience. Real-time communication through Reverb and Pusher-js enhances system interactivity, and the AI detection module powered by Gemini 2.5 Flash automates vehicle recognition with satisfactory accuracy. The use of NGROK, Expo, and modular architecture also facilitated efficient testing and deployment across devices. Overall, the implementation demonstrates the feasibility and practicality of an intelligent parking management system capable of improving user convenience, operational efficiency, and automation through modern cross-platform and AI-driven technologies.

CHAPTER 7
SYSTEM TESTING

7.1 Introduction

This chapter presents the testing results for the ParkPal Parking Management System. The system testing involves API testing, usability testing, and user acceptance testing (UAT) to ensure the overall functionality, usability, and reliability of the application. Additionally, a traceability matrix is produced, linking the use cases, functional requirements, and test cases to ensure that all functionalities are properly validated.

7.2 Traceability between Use Cases, Functional Requirements, and Test Cases

Testing is a crucial stage of the software development lifecycle as it ensures that the system meets the expectations of end-users while maintaining functionality and quality. To achieve this, a traceability matrix is developed to establish the relationship between use cases, which describe how users interact with the system, functional requirements that specify what the system must deliver, and test cases, which verify whether the requirements and use cases are satisfied. This mapping ensures that all intended features are covered during testing, improving maintainability, consistency, and reliability across the application.

7.2.1 Use Case Table

The following table displays the use cases, including their IDs and names.

Table 7.1: Use Case Table

Use Case ID	**Use Case Name**
UC1	Login
UC2	Register
UC3	Manage Vehicles
UC4	Manage Payment Methods
UC5	View Dashboard
UC6	View Nearby Parking Lot Details
UC7	View Nearby EV Chargers

UC8	View Parking Transaction History
UC9	View EV Reservations
UC10	Auto-Transaction of Parking Fee
UC11	Submit Support Tickets
UC12	Request Change to Parking Lot Details
UC13	Manage Support Tickets
UC14	Approve Pending Requests from Operators
UC15	Manage User Accounts
UC16	Manage Own Profile

7.2.2 Functional Requirements Table

The following table outlines the functional requirements that describe the expected functionalities of the system.

Table 7.2: Funstional Requirements Table

Functional Requirement ID	**Description**
FR1	The system shall allow drivers and parking operators to register user accounts.
FR2	The system shall allow drivers, parking operators, and admins to log in securely.
FR3	The system shall allow drivers, parking operators, and admins to manage their own profile information.
FR4	The system shall allow drivers to manage one or more vehicles.
FR5	The system shall allow drivers to manage the auto-transaction settings for parking through license plate and vehicle attribute recognition using multimodal AI.
FR6	The system shall allow drivers, parking operators, and admins to view dashboards personalized to their roles.
FR7	The system shall allow drivers to view nearby parking information.

FR8	The system shall allow drivers to view their parking history.
FR9	The system shall allow drivers to view nearby EV charger information.
FR10	The system shall allow drivers to make EV reservations.
FR11	The system shall allow drivers to view their EV reservations.
FR12	The system shall allow drivers to manage their payment methods.
FR13	The system shall process automatic parking fee transactions when drivers exit a parking lot.
FR14	The system shall provide drivers with notifications and alerts related to transactions.
FR15	The system shall require parking operators to set up parking rates, parking lot details, and all relevant information during the registration process.
FR16	The system shall allow parking operators to update and manage parking lot details, which will be reviewed and approved by the admin.
FR17	The system shall allow drivers and parking operators to submit support tickets for system-related issues.
FR18	The system shall allow admins to manage accounts for drivers and parking operators.
FR19	The system shall require admin approval for new parking operator accounts before they can start managing parking lots, rates, and other features.
FR20	The system shall require admin approval for any changes made to parking rates, parking zones, or other features requested by parking operators.
FR21	The system shall allow admins to manage support tickets submitted by drivers or parking operators.

7.3 API Testing

API testing is an essential part of this system's validation process. It ensures that the application programming interfaces (APIs) work correctly in terms of functionality, reliability, performance, and security. Since the system relies heavily on APIs to manage user authentication, parking transactions, EV reservations, and administrative operations, testing these endpoints is critical to achieving a reliable application.

For this project, Postman was used to perform manual API testing. Postman allows developers to send HTTP requests to the server and verify the responses. Additionally, during front-end development, the network tab of browser developer tools (or React Native debugging tools) was used to monitor API calls in real time. This allows verification of request payloads, response data, status codes, and error handling directly from the client side.

The backend server was developed in Laravel and exposes endpoints that interact with the database to perform CRUD operations and business logic for drivers, parking operators, and administrators. All APIs were tested according to their intended functional requirements. Each test case includes the test case ID, description, endpoint, request method, test scenario, test data, expected result, and actual result.

By combining Postman testing with network tab inspection, the system's API responses were validated both from the server perspective and from the client-side interaction, ensuring correctness, consistency, and reliability of the application.

7.3.1 Summary of API Test Cases

The table below provides a summary of the API test cases executed for this project.

Table 7.3: Summary of API Test Cases and Results

Test Case ID	Test Case Name	Status
TC001	User Registration	Pass
TC002	User Login (Successful)	Pass

TC003	User Login (Failed – Wrong Credentials)	Pass
TC004	View Profile Information	Pass
TC005	Update Profile Information	Pass
TC006	View Vehicles	Pass
TC007	Add Vehicle	Pass
TC008	Update Vehicle Information	Pass
TC009	Delete Vehicle	Pass
TC010	View Payment Methods	Pass
TC011	Add Payment Method	Pass
TC012	Delete Payment Method	Pass
TC013	View Driver Dashboard	Pass
TC014	View Parking Operator Dashboard	Pass
TC015	View Admin Dashboard	Pass
TC016	View Nearby Parking Lots	Pass
TC017	View Parking History	Pass
TC018	View EV Charger Information	Pass
TC019	Create EV Reservation	Pass
TC020	View EV Reservations	Pass
TC021	Operator Setup	Pass
TC022	Operator Manage Parking Lot Details	Pass
TC023	Submit Support Ticket	Pass
TC024	View Support Ticket	Pass
TC025	Send Support Ticket Messages	Pass
TC026	View Support Ticket Messages	Pass
TC027	Admin View Support Ticket	Pass
TC028	Admin Send Support Ticket Messages	Pass
TC029	Admin View Support Ticket Messages	Pass

TC030	Admin View User Accounts	Pass
TC031	Admin Edit User Account	Pass
TC032	Admin Add User Account	Pass
TC033	Admin View Operator Requests	Pass
TC034	Admin Approve Operator Requests	Pass

Table 7.4: Test Case of User Registration

Test Case ID	TC001	**Actual Result**	Pass
Test Case Title	Test Case of User Registration		
Model	User		
Controller	AuthController		
Method	POST		
Endpoints Involved	http://${API_BASE_URL}/register		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
User completes and submits the registration form with personal information.	1. full_name 2. ic 3. email 4. password 5. password_confirmation 6. user_type	1. JSON object with message 2. User created.	1. JSON object with message 2. User created.

Table 7.5: Test Case of User Login (Successful)

Test Case ID	TC002	**Actual Result**	Pass
Test Case Title	Test Case of User Login (Successful)		
Model	User		
Controller	AuthController		
Method	POST		

Endpoints Involved	http://${API_BASE_URL}/login		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
User logs in with valid credentials.	1. email 2. password	1. JSON object with success message. 2. Authentication token returned.	1. JSON object with success message. 2. Authentication token returned.

Table 7.6: Test Case of User Login (Failed – Wrong Credentials)

Test Case ID	TC003	**Actual Result**	Pass
Test Case Title	Test Case of User Login (Failed – Wrong Credentials)		
Model	User		
Controller	AuthController		
Method	POST		
Endpoints Involved	http://${API_BASE_URL}/login		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
User attempts login with invalid credentials.	1. email 2. password	1. JSON object with error message. 2. Authentication denied.	1. JSON object with error message. 2. Authentication denied.

Table 7.7: Test Case of View Profile Information

Test Case ID	TC004	**Actual Result**	Pass
Test Case Title	Test Case of View Profile Information		
Model	User		
Controller	ProfileController		
Method	GET		
Endpoints Involved	http://${API_BASE_URL}/profile		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**

User retrieves their profile information.	-	1. JSON object with user profile details.	1. JSON object with user profile details.

Table 7.8: Test Case of Update Profile Information

Test Case ID	TC005	**Actual Result**	Pass
Test Case Title	Test Case of Update Profile Information		
Model	User		
Controller	ProfileController		
Method	PUT		
Endpoints Involved	http://${API_BASE_URL}/profile		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
User updates profile details.	1. full_name 2. email 3. phone_number	1. JSON object with success message and updated profile.	1. JSON object with success message and updated profile.

Table 7.9: Test Case of View Vehicles

Test Case ID	TC006	**Actual Result**	Pass
Test Case Title	Test Case of View Vehicles		
Model	Vehicle		
Controller	ProfileController		
Method	GET		
Endpoints Involved	http://${API_BASE_URL}/vehicles		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
Driver retrieves a list of registered vehicles.	-	1. JSON array of vehicles associated with the driver.	1. JSON array of vehicles associated with the driver.

Table 7.10: Test Case of Add Vehicles

Test Case ID	TC007	**Actual Result**	Pass
Test Case Title	Test Case of Add Vehicles		
Model	Vehicle		
Controller	ProfileController		
Method	POST		
Endpoints Involved	http://${API_BASE_URL}/vehicles		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
Driver adds a new vehicle to their account.	1. vehicle_type 2. make 3. model 4. color 5. license_plate	1. JSON object of vehicle details.	1. JSON object of vehicle details.

Table 7.11: Test Case of Update Vehicle Information

Test Case ID	TC008	**Actual Result**	Pass
Test Case Title	Test Case of Update Vehicle Information		
Model	Vehicle		
Controller	ProfileController		
Method	PUT		
Endpoints Involved	http://${API_BASE_URL}/vehicles/${id}		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
Driver updates details of an existing vehicle.	1. vehicle_id 2. vehicle_type 3. make 4. model 5. color 6. license_plate	1. JSON object of updated vehicle.	1. JSON object of updated vehicle.

Table 7.12: Test Case of Delete Vehicle

Test Case ID	TC009	**Actual Result**	Pass

Test Case Title	Test Case of Delete Vehicle		
Model	Vehicle		
Controller	ProfileController		
Method	DELETE		
Endpoints Involved	http://${API_BASE_URL}/vehicles/${id}		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
Driver deletes a registered vehicle.	1. vehicle_id	1. JSON object with success message, vehicle mark as inactive.	1. JSON object with success message, vehicle mark as inactive.

Table 7.13: Test Case of View Payment Methods

Test Case ID	TC010	**Actual Result**	Pass
Test Case Title	Test Case of View Payment Methods		
Model	PaymentMethod		
Controller	ProfileController		
Method	GET		
Endpoints Involved	http://${API_BASE_URL}/payment-methods		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
Driver views their saved payment methods.	-	1. JSON array of saved payment methods.	1. JSON array of saved payment methods.

Table 7.14: Test Case of Add Payment Method

Test Case ID	TC011	**Actual Result**	Pass
Test Case Title	Test Case of Add Payment Method		
Model	PaymentMethod		
Controller	ProfileController		
Method	POST		

Endpoints Involved	http://${API_BASE_URL}/payment-methods		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
Driver adds a new payment method.	1. type 2. provider 3. cast_last_four 4. card_expiry 5. ewallet_id	1. JSON object of added payment method.	1. JSON object of added payment method.

Table 7.15: Test Case of Delete Payment Method

Test Case ID	TC012	**Actual Result**	Pass
Test Case Title	Test Case of Delete Payment Method		
Model	PaymentMethod		
Controller	ProfileController		
Method	DELETE		
Endpoints Involved	http://${API_BASE_URL}/payment-methods/${id}		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
Driver deletes a payment method.	1. payment_id	1. JSON object with success message, payment method set to inactive.	1. JSON object with success message, payment method set to inactive.

Table 7.16: Test Case of View Driver Dashboard

Test Case ID	TC013	**Actual Result**	Pass
Test Case Title	Test Case of View Driver Dashboard		
Model	ParkingSession, EvReservation, Vehicle, PaymentMethod, ParkingLot		
Controller	DriverDashboardController		
Method	GET		

Endpoints Involved	http://${API_BASE_URL}/driver/dashboard http://${API_BASE_URL}/monthly-usage http://${API_BASE_URL}/weekly-usage		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
Driver views dashboard tailored to their role.	-	1. JSON object with dashboard data.	1. JSON object with dashboard data.

Table 7.17: Test Case of View Parking Operator Dashboard

Test Case ID	TC014	**Actual Result**	Pass
Test Case Title	Test Case of View Parking Operator Dashboard		
Model	ParkingLot, ParkingZone, ParkingRate, ParkingSession, EvCharger, EvReservation		
Controller	ParkingOperatorDashboardController		
Method	GET		
Endpoints Involved	http://${API_BASE_URL}/operator/dashboard http://${API_BASE_URL}/revenue-trends http://${API_BASE_URL}/occupancy-trends http://${API_BASE_URL}/live-lot-status http://${API_BASE_URL}/parking-rates		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
Parking Operator views dashboard tailored to their role.	-	1. JSON object with dashboard data.	1. JSON object with dashboard data.

Table 7.18: Test Case of View Admin Dashboard

Test Case ID	TC015	**Actual Result**	Pass
Test Case Title	Test Case of View Admin Dashboard		
Model	User, ParkingLot, SupportTicket, EvCharger, ParkingSession		

Controller	AdminDashboardController		
Method	GET		
Endpoints Involved	http://${API_BASE_URL}/admin/dashboard http://${API_BASE_URL}/dashboard/user-growth http://${API_BASE_URL}/parking-usage		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
Admin views dashboard tailored to their role.	-	1. JSON object with dashboard data.	1. JSON object with dashboard data.

Table 7.19: Test Case of View Nearby Parking Lots

Test Case ID	TC016	**Actual Result**	Pass
Test Case Title	Test Case of View Nearby Parking Lots		
Model	ParkingLot, ParkingSession, EvReservation		
Controller	ParkingLotController		
Method	GET		
Endpoints Involved	http://${API_BASE_URL}/nearby-parking-lots		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
Driver views nearby parking lot details.	1. latitude 2. longitude 3. radius 4. available_only 5. sort_by	1. JSON array of nearby parking lots with details.	1. JSON array of nearby parking lots with details.

Table 7.20: Test Case of View Parking History

Test Case ID	TC017	**Actual Result**	Pass
Test Case Title	Test Case of View Parking History		
Model	ParkingSession		
Controller	TransactionHistoryController		
Method	GET		

Endpoints Involved	http://${API_BASE_URL}/transaction-history		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
Driver views their parking transaction history.	-	1. JSON array of past transactions.	1. JSON array of past transactions.

Table 7.21: Test Case of View EV Charger Information

Test Case ID	TC018	**Actual Result**	Pass
Test Case Title	Test Case of View EV Charger Information		
Model	EvCharger, EvReservation, ParkingLot		
Controller	EvReservationController		
Method	GET		
Endpoints Involved	http://${API_BASE_URL}/nearby-ev-chargers		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
Driver views details of nearby EV chargers.	-	1. JSON array of EV chargers with availability status	1. JSON array of EV chargers with availability status

Table 7.22: Test Case of Create EV Reservation

Test Case ID	TC019	**Actual Result**	Pass
Test Case Title	Test Case of Create EV Reservation		
Model	EvCharger, EvReservation, ParkingLot, Vehicle		
Controller	EvReservationController		
Method	POST		
Endpoints Involved	http://${API_BASE_URL}/ev-reservations		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**

Driver creates a reservation for an EV charger.	1. ev_charger_id 2. vehicle_id	1. JSON object with success message and reservation details.	1. JSON object with success message and reservation details.

Table 7.23: Test Case of View EV Reservations

Test Case ID	TC020	**Actual Result**	Pass
Test Case Title	Test Case of View EV Reservations		
Model	EvReservation		
Controller	EvReservationController		
Method	GET		
Endpoints Involved	http://${API_BASE_URL}/ev-reservations		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
Driver views their existing EV reservations.	-	1. JSON array of reservations.	1. JSON array of reservations.

Table 7.24: Test Case of Operator Setup

Test Case ID	TC021	**Actual Result**	Pass
Test Case Title	Test Case of Operator Setup		
Model	PendingAction		
Controller	CompanyRegirstrationController		
Method	POST		
Endpoints Involved	http://${API_BASE_URL}/company-registration		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
Driver views their existing	company: 1. address	2. JSON object confirming	2. JSON object confirming

EV reservations.	2. contact_email 3. contact_phone 4. name 5. registration_no gateway: 6. provider 7. stripe_account_id 8. stripe_onboarded parking_lots: 9. address 10. city 11. closing_time 12. is_24_7 13. latitude 14. longitude 15. opening_time 16. parking_lot_name 17. parking_type 18. postcode 19. state 20. total_parking_bays zones: 21. bay_count 22. description 23. ev_chargers 24. zone_name ev_chargers: 25. end_hour	that the company, parking lot, zones, and EV chargers have been successfully registered and submitted for admin approval.	that the company, parking lot, zones, and EV chargers have been successfully registered and submitted for admin approval.

	26. rate 27. rate_type 28. start_hour		

Table 7.25: Test Case of Operator Manage Parking Lot Details (Create, Update, Delete)

Test Case ID	TC022	**Actual Result**	Pass
Test Case Title	Test Case of Operator Manage Parking Lot Details (Create, Update, Delete)		
Model	PendingAction		
Controller	PendingActionsController		
Method	POST		
Endpoints Involved	http://${API_BASE_URL}/pending-actions		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
Operator updates existing parking lot details.	1. entity_type 2. entity_id 3. action 4. data	1. JSON object with success message, pending admin approval.	1. JSON object with success message, pending admin approval.

Table 7.26: Test Case of Submit Support Ticket

Test Case ID	TC023	**Actual Result**	Pass
Test Case Title	Test Case of Submit Support Ticket		
Model	SupportTicket, SupportTicketMessage		
Controller	SupportTicketController		
Method	POST		
Endpoints Involved	http://${API_BASE_URL}/support-tickets		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
User submits a support ticket	1. subject 2. description	1. JSON object with success	1. JSON object with success

for system-related issue.	3. priority	message and ticket ID.	message and ticket ID.

Table 7.27: Test Case of View Support Ticket

Test Case ID	TC024	**Actual Result**	Pass
Test Case Title	Test Case of View Support Ticket		
Model	SupportTicket, SupportTicketMessage		
Controller	SupportTicketController		
Method	GET		
Endpoints Involved	http://${API_BASE_URL}/support-tickets		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
User retrieves the list of their submitted support tickets, including messages and status.	1. user_id	1. JSON object containing a list of support tickets submitted by the user.	1. JSON object containing a list of support tickets submitted by the user.

Table 7.28: Test Case of Send Support Ticket Messages

Test Case ID	TC025	**Actual Result**	Pass
Test Case Title	Test Case of Send Support Ticket Messages		
Model	SupportTicketMessage		
Controller	SupportTicketController		
Method	POST		
Endpoints Involved	http://${API_BASE_URL}/support-tickets/${ticketId}/messages		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
User sends a message to a	1. ticketId 2. is_admin 3. message	1. JSON object containing the message that	1. JSON object containing the message

specific support ticket.	4. user_id	was added to the support ticket.	that was added to the support ticket.

Table 7.29: Test Case of View Support Ticket Messages

Test Case ID	TC026	**Actual Result**	Pass
Test Case Title	Test Case of View Support Ticket Messages		
Model	SupportTicketMessage		
Controller	SupportTicketController		
Method	GET		
Endpoints Involved	http://${API_BASE_URL}/support-tickets/${ticketId}/messages		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
User retrieves all messages associated with a specific support ticket.	1. ticketId	1. JSON object containing a list of messages for the specified support ticket.	1. JSON object containing a list of messages for the specified support ticket.

Table 7.30: Test Case of Admin View Support Ticket

Test Case ID	TC027	**Actual Result**	Pass
Test Case Title	Test Case of Admin View Support Ticket		
Model	SupportTicket, SupportTicketMessage		
Controller	SupportTicketController		
Method	GET		
Endpoints Involved	http://${API_BASE_URL}/admin/support-tickets		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
Admin retrieves the list of all submitted	-	1. JSON object containing a list of all support	1. JSON object containing a list of all

support tickets along with their messages and statuses.		tickets in the system.	support tickets in the system.

Table 7.31: Test Case of Admin Send Support Ticket Messages

Test Case ID	TC028	**Actual Result**	Pass
Test Case Title	Test Case of Admin Send Support Ticket Messages		
Model	SupportTicketMessage		
Controller	SupportTicketController		
Method	POST		
Endpoints Involved	http://${API_BASE_URL}/support-tickets/${ticketId}/messages		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
Admin sends a message to a specific support ticket.	1. ticketId 2. is_admin 3. message	1. JSON object containing the message that was added to the support ticket.	1. JSON object containing the message that was added to the support ticket.

Table 7.32: Test Case of Admin View Support Ticket Messages

Test Case ID	TC029	**Actual Result**	Pass
Test Case Title	Test Case of Admin View Support Ticket Messages		
Model	SupportTicketMessage		
Controller	SupportTicketController		
Method	GET		
Endpoints Involved	http://${API_BASE_URL}/admin/support-tickets/${ticketId}/details		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**

Admin retrieves all messages associated with a specific support ticket.	1. ticketId	1. JSON object containing a list of messages for the specified support ticket.	1. JSON object containing a list of messages for the specified support ticket.

Table 7.33: Test Case of Admin View User Accounts

Test Case ID	TC030	**Actual Result**	Pass
Test Case Title	Test Case of Admin View User Accounts		
Model	User, Company		
Controller	UserManagementController		
Method	GET		
Endpoints Involved	http://${API_BASE_URL}/admin/users?${params}		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
Admin retrieves and filters user accounts, with the ability to view details, update, or deactivate accounts.	1. page 2. per_page 3. user_type 4. status 5. company_id 6. search	1. JSON object containing a paginated list of user accounts filtered according to the provided parameters	1. JSON object containing a paginated list of user accounts filtered according to the provided parameters

Table 7.34: Test Case of Admin Edit User Account

Test Case ID	TC031	**Actual Result**	Pass
Test Case Title	Test Case of Admin Edit User Account		
Model	User, Company		
Controller	UserManagementController		
Method	PUT		

Endpoints Involved	http://${API_BASE_URL}/admin/users/${selectedUser.id}		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
Admin updates details of a specific user account.	1. full_name 2. ic 3. email 4. phone_number 5. user_type 6. status 7. company_id 8. password 9. password_confirmation	1. JSON object of the updated new details.	1. JSON object of the updated new details.

Table 7.35: Test Case of Admin Add User Account

Test Case ID	TC032	**Actual Result**	Pass
Test Case Title	Test Case of Admin Add User Account		
Model	User, Company		
Controller	UserManagementController		
Method	POST		
Endpoints Involved	http://${API_BASE_URL}/admin/users		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
Admin creates a new user account by providing all necessary	1. full_name 2. ic 3. email 4. phone_number	1. JSON object confirming the new user account was successfully	1. JSON object confirming the new user account was successfully

personal, role, and company details.	5. password 6. user_type 7. company_id 8. status	created with the provided details.	created with the provided details.

Table 7.36: Test Case of Admin View Operator Requests

Test Case ID	TC033	**Actual Result**	Pass
Test Case Title	Test Case of Admin View Operator Requests		
Model	PendingAction		
Controller	AdminPendingActionController		
Method	GET		
Endpoints Involved	http://${API_BASE_URL}/admin/pending-actions?${params}		
Test Scenario	**Test Data**	**Expected Result**	**Actual Result**
Admin retrieves a list of pending operator requests, optionally filtered by status, type, or search criteria.	1. page 2. per_page 3. status 4. entity_type 5. action 6. search	1. JSON object containing a paginated list of pending operator requests	1. JSON object containing a paginated list of pending operator requests

Table 7.37: Test Case of Admin Approve Operator Requests

Test Case ID	TC034	**Actual Result**	Pass
Test Case Title	Test Case of Admin Approve Operator Requests		
Model	PendingAction		
Controller	AdminPendingActionController		
Method	PATCH		
Endpoints Involved	http://${API_BASE_URL}/admin/pending-actions/${actionId}/review		

Test Scenario	Test Data	Expected Result	Actual Result
Admin reviews and approves or rejects a pending operator request.	1. review _notes 2. status	1. JSON object confirming that the pending operator request has been updated with the new status and notes.	1. JSON object confirming that the pending operator request has been updated with the new status and notes.

7.4 Traceability Matrix

To clearly illustrate the relationship between the test cases executed, the functional requirements, and the use cases, a traceability matrix has been created. This matrix links each test case with its corresponding functional requirement and use case. The traceability matrix is presented in the table below.

Table 7.38: Traceability Matrix Table

Use Case ID	Functional Requirement ID	Test Case ID
UC1	FR2	TC002, TC003
UC2	FR1	TC001
UC3	FR4	TC006, TC007, TC008, TC009
UC4	FR12	TC010, TC011, TC012
UC5	FR6	TC013, TC014, TC015
UC6	FR7, FR15, FR16	TC016, TC021, TC022
UC7	FR9	TC018
UC8	FR8	TC017
UC9	FR10, FR11	TC019, TC020
UC11	FR17	TC023
UC12	FR20	TC033, TC034
UC13	FR21	TC027, TC028, TC029
UC14	FR19	TC033, TC034
UC15	FR18	TC030, TC031, TC032

UC16	FR3	TC004, TC005

7.5 Performance Evaluation of Vehicle Recognition

The purpose of this test is to validate the system's ability to accurately identify vehicle information, including license plate, make, model, and color, using image inputs. The test also evaluates the system's response time and overall reliability for real-world usage.

7.5.1 Evaluation Methodology

A dataset of real vehicle images was used as input for the system. The system processed each image to extract vehicle attributes, and recognition results were compared against ground truth. Metrics recorded included detection time per image and accuracy for each vehicle attribute. The detailed results are provided in a table in Appendix A.

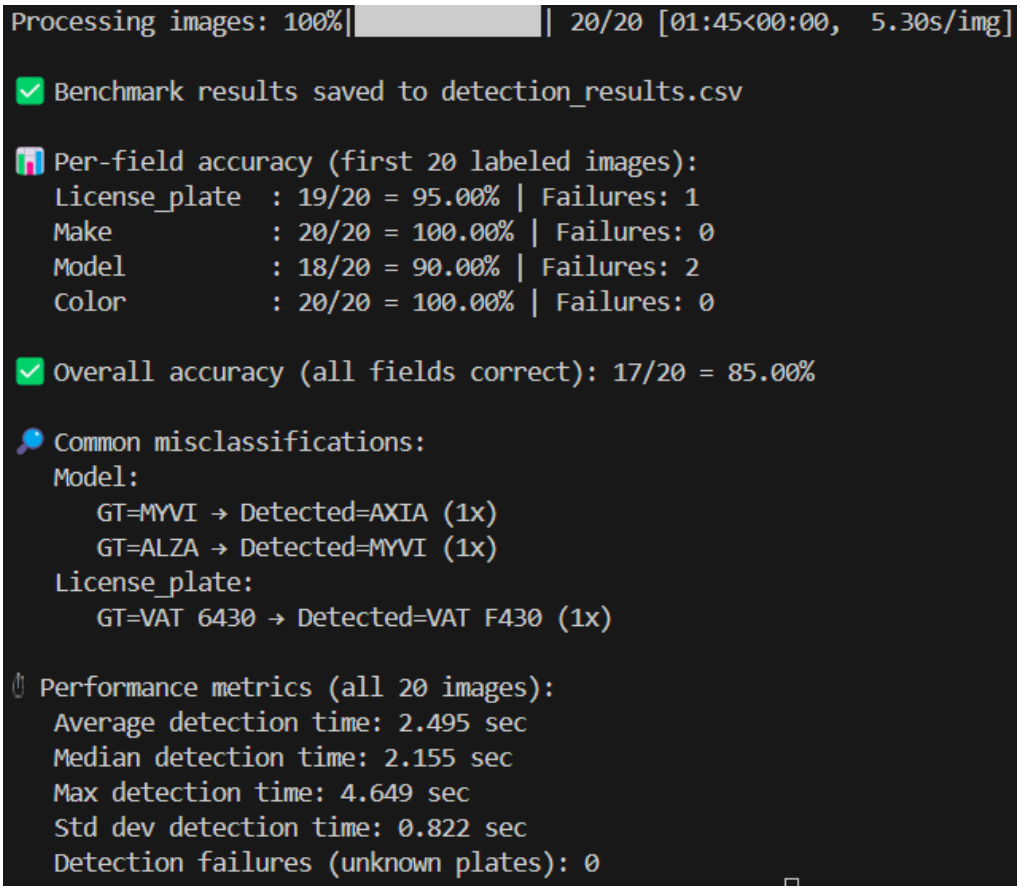


Figure 7.1: Vehicle Recognition System Benchmark Results

The image above presents the benchmark results of the vehicle recognition system tested on 20 images. Key observations include per-field accuracy, such as license plate, make, model, and color, overall accuracy, common misclassifications, and performance metrics. These results demonstrate that the system is accurate in detecting vehicle attributes and performs efficiently in terms of processing time.

7.6 Evaluation of Auto Payment via Vehicle Recognition

The automatic parking fee system, based on license plate and vehicle recognition, was tested to demonstrate end-to-end functionality, from vehicle detection to payment processing. The system successfully scanned vehicle image, identified the license plate, make, model, and color, matched the vehicle to a registered user, calculated the parking fee based on duration and rate, processed the payment automatically, and stored a notification for the driver. The following log content illustrates a complete parking session, including entry, duration calculation, fee computation, payment processing, and notification delivery.

```
2025-09-17 12:58:23 [INFO] Scanning vehicle from image: proto_image\016667cb-5915-4440-9031-efae7700de8f_jpg.rf.c3500bd80a69ff22a92940e992b29ccc.jpg
2025-09-17 12:58:23 [INFO] AFC is enabled with max remote calls: 10.
2025-09-17 12:58:26 [INFO] HTTP Request: POST https://generativelanguage.googleapis.com/v1beta/models/gemini-2.5-flash:generateContent "HTTP/1.1 200 OK"
2025-09-17 12:58:26 [INFO] AFC remote call 1 is done.
2025-09-17 12:58:26 [INFO] Detected Plate: VBP 3477, Make: Nissan, Model: Almera, Color: gray
2025-09-17 12:58:26 [INFO] Plate VBP 3477 belongs to user Driver 1 (User ID 5)
2025-09-17 12:58:26 [INFO] Vehicle VBP 3477 entered parking lot 1, zone 1
2025-09-17 12:58:26 [INFO] VBP 3477 will stay parked for 5 seconds (simulated 3 hours)
2025-09-17 12:58:31 [INFO] Processing exit for vehicle VBP 3477
2025-09-17 12:58:31 [INFO] Calculating fee for 3 hours parked (weekday)...
2025-09-17 12:58:31 [INFO]   - From hour 0 to 2 (2h @ RM2.00/h) = RM4.00
2025-09-17 12:58:31 [INFO] VBP 3477 parked for 180 min (~3h), total fee: RM5.50
2025-09-17 12:58:31 [INFO] Charging Visa card ****4242 (exp 12/26) for VBP 3477...
2025-09-17 12:58:31 [INFO] VBP 3477 exit completed, Paid RM5.50, Payment ID: pi_mock_1758085111302
2025-09-17 12:58:31 [INFO] Notification stored for VBP 3477
```

Figure 7.2: Vehicle Recognition and Auto Payment Results

7.7 Usability Test

Usability testing was conducted as a critical phase of system evaluation to ensure that users can navigate the app and perform essential tasks with ease and minimal confusion. The primary objective of this testing was to gather direct feedback from users and assess overall usability using the System Usability

Scale (SUS). The testing was conducted online following a structured methodology to ensure consistent and reliable results.

Participants were divided into three main roles: drivers, parking operators, and administrators, representing the primary users of the system. Each participant completed predefined tasks simulating real-world interactions relevant to their role.

After completing these tasks, participants provided feedback via a structured user satisfaction survey. The detailed survey results are provided in Appendix B, and the collected data were analyzed to calculate the average SUS score, reflecting the system's overall usability and user satisfaction across all user roles.

7.7.1 Test Scenarios of Usability Test

Table 7.39: Test Scenarios of Usability Test

No	Test Name	Test Description
1	Account Registration	Test that drivers and parking operators can successfully register an account and receive confirmation.
2	Secure Login	Verify that all users (drivers, operators, admins) can log in securely with valid credentials.
3	Profile Management	Test that users can view and update their profile information without errors.
4	Vehicle Management	Verify that drivers can add, edit, or remove vehicle details in the system.
5	Payment Method Management	Test that drivers can add, update, or delete payment methods seamlessly.
6	Dashboard Access	Verify that users can view a personalized dashboard displaying relevant information for their role.
7	Nearby Parking Info	Test that drivers can view nearby parking lot locations and details accurately.

8	Parking History	Verify that drivers can view a complete history of their parking transactions.
9	EV Charger Info	Test that drivers can view nearby EV chargers with relevant details.
10	EV Reservations	Verify that drivers can make and view EV reservations successfully.
11	Auto-Transaction	Test that the system can process automatic parking fee transactions based on license plate and vehicle recognition.
12	Notifications & Alerts	Verify that drivers receive correct notifications related to parking transactions.
13	Submit Support Tickets	Test that drivers and parking operators can submit support tickets for system-related issues.
14	Manage Parking Lot Details	Verify that parking operators can update parking lot information, which is sent for admin approval.
15	Approve Operator Requests	Test that admins can review and approve new parking operator accounts and change requests.
16	Manage User Accounts	Verify that admins can manage driver and parking operator accounts.
17	Manage Support Tickets	Test that admins can view, update, and resolve support tickets submitted by users.

7.7.2 Results of Usability Test

After completing the usability testing, the user satisfaction survey forms were collected from all participants and are included in Appendix B. The responses from each form were analyzed and summarized. Each of the ten usability questions was rated on a scale from 1 (Strongly Disagree) to 5 (Strongly Agree).

Table 7.40: Test Scenarios of Usability Test

ID	Score for each test										Total SUS Score
	1	2	3	4	5	6	7	8	9	10	
P1	5	3	3	4	4	3	3	4	4	3	55
P2	5	1	5	3	5	5	5	1	5	3	80
P3	3	2	5	2	4	1	3	2	4	4	70
P4	5	2	4	1	4	1	4	1	4	2	85
P5	5	1	5	1	4	1	5	1	5	1	97.5
Average SUS Score											77.5

The average SUS score across all participants is 77.5, showing that the system is generally easy to use, though there is still room for improvement. Some participants mentioned that the app's loading time can be a bit slow, which may affect the user experience. It should be noted that the test involved only five participants, so the results provide a preliminary view of usability. Future usability testing with a larger group would give more reliable and representative feedback. Additionally, the helpful comments provided in the open-ended survey questions have been reviewed and noted for future improvements.

7.8 User Acceptance Testing (UAT)

During the User Acceptance Testing (UAT), participants provided feedback on various aspects of the system, including functionality, interface design, and usability. Most of the issues reported were minor and focused on improving user experience rather than correcting critical system errors. The key feedback and the actions taken to address them are summarized.

Table 7.41: Summary of UAT Feedback and Actions

Feedback Given	**Action Taken**
"It would be useful if vehicle recognition could be integrated into the vehicle registration process."	Added vehicle recognition feature to streamline vehicle registration.
"Some parts of the UI are not consistent, such as the drawer background color."	Updated UI components to maintain consistent colors and styles.
"Some icons could be improved to look nicer and more intuitive."	Replaced selected icons with improved designs for better appearance and clarity.

In summary, the UAT was successful, requiring minimal intervention from the developers. Testers completed all UAT scenarios and effectively validated the core functionality of ParkPal. While a few minor suggestions were provided to improve user experience, these did not affect the overall performance or usability of the system.

7.9 Conclusion

This report presents the testing and evaluation of ParkPal, covering functionality, performance, usability, and user acceptance. The traceability analysis showed that all use cases and functional requirements were properly tested. API testing confirmed that the system's endpoints work correctly. The vehicle recognition system performed accurately and responded quickly enough for practical use. Usability testing showed that drivers, parking operators, and administrators can navigate the app and complete tasks easily, with a high average SUS score. User Acceptance Testing (UAT) confirmed that the system meets its intended purpose, with all test scenarios completed successfully. Overall, ParkPal has been tested thoroughly and is ready for deployment, with some suggestions for future improvements to enhance user experience.

CHAPTER 8
CONCLUSION AND RECOMMENDATIONS

8.1 Introduction

This chapter provides the conclusion of the project by reviewing how the objectives were achieved, the limitations that were encountered, and the possible improvements for future development. It highlights the progress made in building the parking management system while also pointing out areas that can be enhanced to ensure greater efficiency, accuracy, and usability in future versions.

8.2 Objectives Achievement

The objectives of this project were successfully achieved through the development and evaluation of the proposed parking management system mentioned in Chapter 1.

The first objective, which focused on examining license plate and vehicle attribute recognition approaches and reviewing similar applications, was fulfilled through a comprehensive study of existing methods. Traditional computer vision techniques were compared with modern LLMs, and related parking and mobility applications were analyzed to identify common practices, limitations, and opportunities for improvement. This provided the foundation for designing a system that leverages advanced recognition technologies while addressing practical challenges observed in current solutions.

The second objective, to develop an automated parking payment system that integrates multimodal LLMs for license plate and vehicle attribute recognition, was also achieved. The project implemented the Gemini 2.5 Flash model to automatically detect license plates and extract key vehicle attributes such as make, model, and color. These detections were integrated into the parking workflow to enable automated validation and fee calculation during vehicle entry and exit. While the current implementation relies on the free-tier version of the model, which introduces certain limitations, it successfully demonstrated the feasibility of incorporating AI-powered recognition into the automation of parking payments.

The third objective, to develop a parking management application, was realized through the creation of a complete system with role-based functionality for drivers, parking operators, and administrators. Drivers are able to register, manage vehicles, view nearby parking lots, and make EV reservations. Operators can manage parking lots and view parking statistics to monitor performance and usage trends, while administrators are provided with a dashboard to monitor system usage and manage users and pending actions. The application was developed using Laravel, React Native, PostgreSQL, and supporting frameworks, ensuring a scalable, secure, and user-friendly design.

In summary, the project achieved its intended objectives by delivering a functional prototype of an AI-assisted parking management system. The system integrates multimodal recognition for automation, role-based dashboards for management, and scalable technologies for implementation, thereby demonstrating the viability of intelligent parking solutions in real-world contexts.

8.3 Project Limitations

Despite the successful implementation, the project has several limitations. One key limitation lies in the AI model constraints. The vehicle recognition system relies on the free version of the Gemini 2.5 Flash model, which restricts its processing capacity. As a result, handling large volumes of images in real time becomes challenging. In testing, the license plate recognition achieved an accuracy of 85%, which is slightly below the 90% target set in the non-functional requirements. This difference is mainly due to the limited dataset of 20 real vehicle images used for evaluation and the constraints of the free API tier. An upgrade to a higher-tier model or a larger, more diverse dataset would likely improve accuracy, processing speed, and overall reliability of the system.

Another limitation is the scope of analytics and reporting. The current features are fairly general, focusing mainly on basic usage statistics and payment summaries. While these provide some insights, more detailed and customizable reports, such as those filtered by time range, parking lot performance, or driver behavior, would significantly enhance decision-making for operators and administrators.

Finally, the system still requires manual setup by parking operators. At present, operators need to manually key in parking lot details, which can be time-consuming and prone to error. More efficient methods, such as bulk data import or configuration cloning, and maybe even integration with external APIs, would reduce this burden and streamline onboarding for new operators.

8.4 Recommendations for Future Work

While the current version of the parking management app provides essential features, there are several opportunities to enhance its functionality and user experience in future iterations.

One potential improvement is the integration of third-party authentication methods, such as Google, Apple, or social media logins. This would simplify access for users and reduce the friction associated with creating a new account.

While the system currently supports real-time parking management, a parking reservation module could be added too, which works similarly to the current EV Reservation. This would allow drivers to book specific parking spots in advance, ensuring availability upon arrival and reducing uncertainty in busy areas.

A significant improvement would be to enable payments for electric vehicle (EV) parking reservations. Currently, the app allows users to reserve EV charging spots but does not support payment for these reservations. Integrating a secure payment feature for EV spots would make the service more convenient and complete, encouraging greater adoption among EV users.

To improve transparency and accountability, the system could include receipt printing and automated receipt generation. Beyond individual receipts, administrators and operators could benefit from advanced analysis reports, with options to export data in PDF format for auditing, financial tracking, or decision-making purposes.

Another area for enhancement is the incorporation of additional services. By analyzing existing parking and mobility apps, features such as vehicle insurance options, promotional offers, and loyalty programs could be

added. These services would provide added value beyond basic parking management and help engage users more effectively.

In addition, expanding the app's geographical scope is also recommended. The app could be extended to include toll payment systems across Malaysia, offering a more comprehensive mobility solution. With appropriate localization, it could potentially be adapted for use in other countries, reaching a wider audience and increasing its utility.

Finally, the app can improve through pre-processing techniques such as image enhancement and noise reduction before sending data to the model. For scenarios requiring deeper customization, such as adapting recognition to specific local license plate formats, vehicle types, or environmental conditions, a fine-tunable open-source model could be integrated alongside Gemini. This hybrid approach would allow developers to retrain or fine-tune the open-source model on domain-specific datasets while still using Gemini for broader recognition tasks.

These improvements would not only strengthen the app's functionality but also create a seamless and satisfying user experience, positioning it as a competitive solution for both local and international markets.

.

REFERENCES

Abdullah, M. *et al.* (2021) "LICENSE PLATE RECOGNITION TECHNIQUES: COMPARATIVE STUDY," *Malaysian Journal of Computer Science*, 2021(Special Issue 1), pp. 94–105. Available at: https://doi.org/10.22452/mjcs.sp2021no1.9.

A.K.M Zahidul Islam and Ferworn, A. (2020) "A Comparison between Agile and Traditional Software Development Methodologies," *Global Journal of Computer Science and Technology: C Software & Data Engineering*, 20(2).

AlDahoul, N. *et al.* (2024) "Advancing Vehicle Plate Recognition: Multitasking Visual Language Models with VehiclePaliGemma." Available at: http://arxiv.org/abs/2412.14197.

Alexander Gillis (no date) *enterprise architecture (EA), TechTarget*. Available at: https://www.techtarget.com/searchcio/definition/enterprise-architecture (Accessed: April 27, 2025).

Aly, M. (2008) *Real time Detection of Lane Markers in Urban Streets, IEEE Intelligent Vehicles Symposium, Proceedings*. Available at: https://doi.org/10.1109/IVS.2008.4621152.

Apoorva Srivastava, Sukriti Bhardwaj and Shipra Saraswat (2017) *SCRUM Model for Agile Methodology*. IEEE.

Apple (no date a) *Flexi Parking on the App Store, Apple*. Available at: https://apps.apple.com/cn/app/flexi-parking/id1466897086?l=en-GB (Accessed: May 2, 2025).

Apple (no date b) *JomParking on the App Store, Apple*. Available at: https://apps.apple.com/my/app/jomparking/id990623185 (Accessed: April 12, 2025).

Bay, H., Tuytelaars, T. and Van Gool, L. (2006) *SURF: Speeded up robust features, Computer Vision-ECCV 2006*. Available at: https://doi.org/10.1007/11744023_32.

Burtescu, E. *et al.* (2014) "Database Systems Journal," *Database Systems Journal*, 5(3).

Cao, J. *et al.* (2020) "Front vehicle detection algorithm for smart car based on improved SSD model," *Sensors (Switzerland)*, 20(16), pp. 1–21. Available at: https://doi.org/10.3390/s20164646.

Carranza-García, M. *et al.* (2021) "On the performance of one-stage and two-stage object detectors in autonomous vehicles using camera data," *Remote Sensing*, 13(1), pp. 1–23. Available at: https://doi.org/10.3390/rs13010089.

Caruana, R., Pratt, L. and Thrun, S. (1997) *Multitask Learning* *. Kluwer Academic Publishers.

Chiang, J. (2024) *GPT-4o: what do we know so far*, *Medium*. Available at: https://medium.com/@tsunhanchiang/gpt-4o-what-do-we-know-so-far-6672e85c4de5 (Accessed: April 30, 2025).

Cipolla, R., Gal, Y. and Kendall, A. (2018) "Multi-task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, pp. 7482–7491. Available at: https://doi.org/10.1109/CVPR.2018.00781.

Dalal, N. and Triggs, B. (2005) *Histograms of Oriented Gradients for Human Detection*, *Comput. Vision Pattern Recognit.* Available at: https://doi.org/10.1109/CVPR.2005.177.

Erickson, J. (2024) *MySQL: Understanding What It Is and How It's Used*, *Oracle*. Available at: https://www.oracle.com/mysql/what-is-mysql/ (Accessed: April 30, 2025).

Gayen, S. *et al.* (2024) "Two decades of vehicle make and model recognition – Survey, challenges and future directions," *Journal of King Saud University - Computer and Information Sciences*. King Saud bin Abdulaziz University. Available at: https://doi.org/10.1016/j.jksuci.2023.101885.

H. Meybodi, Z. *et al.* (2021) *TEDGE-Caching: Transformer-based Edge Caching Towards 6G Networks*. Available at: https://doi.org/10.48550/arXiv.2112.00633.

Haire, A. (no date) *What is Ionic: Learn the essentials of what you can do with Ionic and how it works.*, *ionic*. Available at: https://ionic.io/resources/articles/what-is-ionic (Accessed: April 29, 2025).

Han, K. *et al.* (2023) "A Survey on Vision Transformer," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1), pp. 87–110. Available at: https://doi.org/10.1109/TPAMI.2022.3152247.

He, K. *et al.* (2016) "Deep residual learning for image recognition," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, pp. 770–778. Available at: https://doi.org/10.1109/CVPR.2016.90.

JomParking (no date a) *Documentation*, *JomParking*. Available at: https://web.jomparking.com/documentation (Accessed: April 12, 2025).

JomParking (no date b) *JomParking*, *JomParking*. Available at: https://web.jomparking.com/features (Accessed: April 12, 2025).

Kang, S. *et al.* (2025) “Object Detection YOLO Algorithms and Their Industrial Applications: Overview and Comparative Analysis,” *Electronics (Switzerland)*. Multidisciplinary Digital Publishing Institute (MDPI). Available at: https://doi.org/10.3390/electronics14061104.

Kattenborn, T. *et al.* (2021) “Review on Convolutional Neural Networks (CNN) in vegetation remote sensing,” *ISPRS Journal of Photogrammetry and Remote Sensing*. Elsevier B.V., pp. 24–49. Available at: https://doi.org/10.1016/j.isprsjprs.2020.12.010.

Kongyoung, S., Macdonald, C. and Ounis, I. (2020) “Multi-Task Learning using Dynamic Task Weighting for Conversational Question Answering,” in J. Dalton et al. (eds.) *Proceedings of the 5th International Workshop on Search-Oriented Conversational AI (SCAI)*. Online: Association for Computational Linguistics, pp. 17–26. Available at: https://doi.org/10.18653/v1/2020.scai-1.3.

Korkut, T. (2023) *Excelling in Software Development with Scrum Methodology Part 2*, *Medium*. Available at: https://blog.stackademic.com/excelling-in-software-development-with-scrum-methodology-part-2-e2d0b29437ce (Accessed: April 16, 2025).

Kwiatkowski, K. (2024) *What is Expo?*, *Pagepro*. Available at: https://pagepro.co/blog/what-is-expo-js/ (Accessed: May 2, 2025).

Laravel (no date) *Installation*. Available at: https://laravel.com/docs/12.x (Accessed: April 29, 2025).

Manzoor, M.A., Morgan, Y. and Bais, A. (2019) “Real-Time Vehicle Make and Model Recognition System,” *Machine Learning and Knowledge Extraction*, 1(2), pp. 611–629. Available at: https://doi.org/10.3390/make1020036.

Miller, G. (2001) *The Characteristics of Agile Software Processes.* Available at: https://doi.org/10.1109/TOOLS.2001.10035.

Montazzolli, S. and Jung, C. (2018) *License Plate Detection and Recognition in Unconstrained Scenarios*.

Nadira Muda *et al.* (2007) “Optical Character Recognition By Using Template Matching (Alphabet),” in.

NestJS (no date) *Introduction*, *NestJS*. Available at: https://docs.nestjs.com/ (Accessed: April 30, 2025).

Nuradzimmah Daim (2023) *36.3 million vehicles in Malaysia*, *New Straits Times* . Available at: https://www.nst.com.my/news/nation/2023/12/987062/363-million-vehicles-malaysia (Accessed: May 1, 2025).

ParkEasy (2016) *ParkEasy, park safely*, *ParkEasy*. Available at: https://www.parkeasy.co/blog/tag/parking (Accessed: May 2, 2025).

ParkEasy (no date) *Add Membership*, *ParkEasy*. Available at: https://www.parkeasy.co/add-membership (Accessed: May 2, 2025).

Phung, V.H. and Rhee, E.J. (2018) "A Deep Learning Approach for Classification of Cloud Image Patches on Small Datasets," *Journal of Information and Communication Convergence Engineering*. 2018/09/30, 16(3), pp. 173–178. Available at: https://doi.org/10.6109/jicce.2018.16.3.173.

Ranjan, R. *et al.* (2016) "An All-In-One Convolutional Neural Network for Face Analysis." Available at: https://doi.org/10.48550/arXiv.1611.00851.

Ren, S. *et al.* (2017) "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6), pp. 1137–1149. Available at: https://doi.org/10.1109/TPAMI.2016.2577031.

Romanowski, J. (2020) *Major Companies Using PostgreSQL: Purposes & Examples*, *LearnSQL.com*. Available at: https://learnsql.com/blog/companies-that-use-postgresql-in-business/ (Accessed: April 30, 2025).

Rozlan, I. (2023) *Touch 'n Go eWallet Enables QR Code Scanning To Pay Parking*, *Lowyat.NET*. Available at: https://www.lowyat.net/2023/296185/touch-n-go-ewallet-qr-code-parking/ (Accessed: April 12, 2025).

Ruder, S. (2017) "An Overview of Multi-Task Learning in Deep Neural Networks." Available at: http://arxiv.org/abs/1706.05098.

Saeed, S. *et al.* (2019) "Analysis of software development methodologies," *International Journal of Computing and Digital Systems*, 8(5), pp. 445–460. Available at: https://doi.org/10.12785/ijcds/080502.

Sakshi Sachdeva (2016) "Scrum Methodology," *International Journal Of Engineering And Computer Science* [Preprint]. Available at: https://doi.org/10.18535/ijecs/v5i6.11.

Sehouli, H. (2025) *Protecting Your Laravel App Against SQL Injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF)*, *Medium*. Available at: https://medium.com/@sehouli.hamza/protecting-your-laravel-app-against-sql-injection-cross-site-scripting-xss-and-cross-site-ea3a05260afe (Accessed: April 29, 2025).

Shi, B., Bai, X. and Yao, C. (2017) "An End-to-End Trainable Neural Network for Image-Based Sequence Recognition and Its Application to Scene Text Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(11), pp. 2298–2304. Available at: https://doi.org/10.1109/TPAMI.2016.2646371.

Sochor, J., Špaňhel, J. and Herout, A. (2017) "BoxCars: Improving Vehicle Fine-Grained Recognition using 3D Bounding Boxes in Traffic Surveillance,"

IEEE Transactions on Intelligent Transportation Systems, PP. Available at: https://doi.org/10.1109/TITS.2018.2799228.

Sporici, D., Cușnir, E. and Boiangiu, C.A. (2020) "Improving the accuracy of Tesseract 4.0 OCR engine using convolution-based preprocessing," *Symmetry*, 12(5). Available at: https://doi.org/10.3390/SYM12050715.

Suhasini Gadam (2023) *What is iterative and incremental development? Process, examples*. Available at: https://blog.logrocket.com/product-management/what-is-iterative-incremental-development-process-examples/ (Accessed: April 15, 2025).

Tabani, H. *et al.* (2021) "Improving the Efficiency of Transformers for Resource-Constrained Devices." Available at: http://arxiv.org/abs/2106.16006.

Tan, M. and Le, Q. (2019) "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in K. Chaudhuri and R. Salakhutdinov (eds.) *Proceedings of the 36th International Conference on Machine Learning*. PMLR (Proceedings of Machine Learning Research), pp. 6105–6114. Available at: https://proceedings.mlr.press/v97/tan19a.html.

Tao, L. *et al.* (2024) "A Real-Time License Plate Detection and Recognition Model in Unconstrained Scenarios," *Sensors*, 24(9). Available at: https://doi.org/10.3390/s24092791.

Tesseract Documentation (no date) *Tesseract User Manual*. Available at: https://tesseract-ocr.github.io/tessdoc/ (Accessed: May 1, 2025).

The PostgreSQL Global Development Group (no date) *About, The PostgreSQL Global Development Group*. Available at: https://www.postgresql.org/about/ (Accessed: April 30, 2025).

Touch 'n Go (no date a) *LPR Parking (License Plate Recognition), Touch 'N Go*. Available at: https://support.tngdigital.com.my/hc/en-my/sections/20991708013081-LPR-Parking-License-Plate-Recognition (Accessed: April 12, 2025).

Touch 'n Go (no date b) *Our Story, Touch 'n Go*. Available at: https://www.touchngo.com.my/ewallet/about-us/our-story/ (Accessed: April 12, 2025).

Touch 'n Go (no date c) *Scan QR in/out Parking, Touch 'n Go*. Available at: https://support.tngdigital.com.my/hc/en-my/sections/16266968986265-Scan-QR-in-out-Parking (Accessed: April 12, 2025).

Touch 'n Go (no date d) *Street Parking, Touch 'n Go*. Available at: https://support.tngdigital.com.my/hc/en-my/sections/360008669294-Street-Parking?page=1#articles (Accessed: April 12, 2025).

Touch ’n Go (no date e) *What is TNG ParkInsure?*, *Touch ’n Go*. Available at: https://support.tngdigital.com.my/hc/en-my/articles/21505528982041-What-is-TNG-ParkInsure (Accessed: April 12, 2025).

Tsimpoukelli, M. *et al.* (no date) *Multimodal Few-Shot Learning with Frozen Language Models*.

Tulsee Doshi (2025) *Start building with Gemini 2.5 Flash - Google Developers Blog*. Available at: https://developers.googleblog.com/en/start-building-with-gemini-25-flash/ (Accessed: September 19, 2025).

Ultralytics (2023a) *Instance Segmentation , Ultralytics*. Available at: https://docs.ultralytics.com/tasks/segment/ (Accessed: May 1, 2025).

Ultralytics (2023b) *Segment Anything Model (SAM)*, *Ultralytics*. Available at: https://docs.ultralytics.com/models/sam/#introduction-to-sam-the-segment-anything-model (Accessed: May 1, 2025).

Vaswani, A. *et al.* (2017) “Attention Is All You Need.” Available at: http://arxiv.org/abs/1706.03762.

Vishal Chandra (2015) “Comparison between Various Software Development Methodologies,” *International Journal of Computer Applications*, 131(9), pp. 975–8887.

Visual Studio Code (no date) *Visual Studio Code documentation*, *Visual Studio Code*. Available at: https://code.visualstudio.com/docs (Accessed: April 27, 2025).

Wong, A. (2025) *TNG eWallet now supports LPR parking, available in 12 locations*, *SoyaCincau*. Available at: https://soyacincau.com/2025/03/16/tng-ewallet-lpr-licence-plate-recognition-parking/ (Accessed: April 12, 2025).

Wu, J. (2018) “Complexity and accuracy analysis of common artificial neural networks on pedestrian detection,” in *MATEC Web of Conferences*. EDP Sciences. Available at: https://doi.org/10.1051/matecconf/201823201003.

Xia, Y., Feng, J. and Zhang, B. (2016) “Vehicle Logo Recognition and Attributes Prediction by Multi-task Learning with CNN,” in *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*. IEEE.

Yang, Z. *et al.* (2018) “Learning to Navigate for Fine-grained Classification,” in *Proceedings of the European Conference on Computer Vision (ECCV)*.

Zherzdev, S. and Gruzdev, A. (2018) “LPRNet: License Plate Recognition via Deep Neural Networks.” Available at: http://arxiv.org/abs/1806.10447.

APPENDICES

APPENDIX A: Vehicle Recognition Benchmark Result

Table A-1: Vehicle Recognition Benchmark Results

image	license_plate	make	model	color	detection_time_sec	accuracy_license_plate	accuracy_make	accuracy_model	accuracy_color
016667cb-5915-4440-9031-efae7700de8f_jpg.rf.c3500bd80a69ff22a92940e992b29ccc.jpg	VBP 3477	Nissan	Almera	gray	3.734	TRUE	TRUE	TRUE	TRUE
0324bca4-2650-43e3-b883-1dafa98eb2fe_jpg.rf.cfdc7599673636ca0e949d548e0408ed.jpg	JSM 2306	Perodua	Axia	gray	1.966	TRUE	TRUE	TRUE	TRUE
15c8f5f9-6856-4375-9aa3-a3aafd0e43a7_jpg.rf.57527c87c5958a4591510add9510799f.jpg	TBU 5553	Honda	City	red	1.733	TRUE	TRUE	TRUE	TRUE

1b584696-1322-461d-83e0-1a23a3bf82ef_jpg.rf.1309cfe57e2290c747febe4362827672.jpg	VCF 2025	Honda	City	white	1.828	TRUE	TRUE	TRUE	TRUE
2156eaea-9c31-4838-a9b8-8d8fec29900e_jpg.rf.09d85642cbd9ec9da8841c58bb91aa30.jpg	PMJ 5716	Perodua	Axia	white	1.909	TRUE	TRUE	FALSE	TRUE
248b04ae-38f2-4ece-be6f-216aa6218c2e_jpg.rf.86968a1efca8c5c1eb23751d79e8dd0a.jpg	VCT9264	Perodua	Alza	blue	1.866	TRUE	TRUE	TRUE	TRUE
29650796-36d3-468b-8cda-eb6859e053a3_jpg.rf.86977d4ce69c14fb8f510d89f75abf28.jpg	PMJ 5716	Perodua	Myvi	white	2.904	TRUE	TRUE	TRUE	TRUE
2a352357-3848-48cd-a423-6241f80d7b16_jpg.rf.83e34e6014c8e1138e6e0afc609d47b7.jpg	VAT F430	Perodua	Bezza	gray	2.103	FALSE	TRUE	TRUE	TRUE
2eb92e30-4f79-4ee7-9f95-1e291ac2fcda_jpg.rf.0bc95aa562ebc40ef3a694691eeeea98.jpg	PMP 9680	Perodua	Myvi	white	2.346	TRUE	TRUE	FALSE	TRUE

319e49f8-181f-4288-aa94-0d38d33f77c1_jpg.rf.c6d52ea6db320a485e9004df7044c8ff.jpg	VCF 1804	Nissan	Almera	gray	2.071	TRUE	TRUE	TRUE	TRUE
334c72e5-b22b-4341-a6ab-e65fe684ace2_jpg.rf.8023c0e493b6a5ab9d3645bb110bca68.jpg	SU 5805 D	Perodua	Bezza	red	3.987	TRUE	TRUE	TRUE	TRUE
33f48b10-37ec-4e83-b9d9-c25a83da84b0_jpg.rf.8d480063f77f44487804699ac3c97958.jpg	VDK 8639	Nissan	Leaf	white	1.696	TRUE	TRUE	TRUE	TRUE
35aff9e2-44ec-4d82-872b-3575b6863f73_jpg.rf.8b6155475e14eae5ad00a53ab51965b3.jpg	AKD 9878	Toyota	Vios	gray	4.649	TRUE	TRUE	TRUE	TRUE
37a46b75-847a-4860-becd-dfc7337138fe_jpg.rf.0ee0d210e016c8e0c8b201421f192dca.jpg	AKQ 206	Perodua	Axia	yellow	2.012	TRUE	TRUE	TRUE	TRUE
3b66b3ed-349f-4d1c-9861-f6a7c2065a80_jpg.rf.8895dbfb45569e75deb85ad30b00a5cb.jpg	VBR 2625	Renault	Captur	blue	3.153	TRUE	TRUE	TRUE	TRUE

3d7577dd-9427-45c0-92f3-e8f35543346a_jpg.rf.9b41d4448d9767a952e4b06e09085602.jpg	VK 641	Proton	Persona	brown	1.844	TRUE	TRUE	TRUE	TRUE
3ed72906-efff-40d0-afae-a52e3ba3f46f_jpg.rf.a86e374ef5498520a2d3202b27cb6561.jpg	VAT 6458	Perodua	Bezza	blue	2.208	TRUE	TRUE	TRUE	TRUE
3f7bbcb9-9c47-4950-8e53-012e36bb2acc-1-_jpg.rf.c906c79156d109434841658808dc2843.jpg	JRC 5492	Perodua	Axia	black	2.707	TRUE	TRUE	TRUE	TRUE
3fe7af30-1573-4658-b576-3607cbe9951f_jpg.rf.7cd5508c76abdaa0afe5417d8ab360ba.jpg	KEE 1096	Perodua	Axia	red	2.531	TRUE	TRUE	TRUE	TRUE
408a8b3f-89b0-431d-b409-ed8264e258a9_jpg.rf.45dd12f3440a6caf52ce602f78a29b09.jpg	WC 5763 R	Nissan	Almera	gray	2.65	TRUE	TRUE	TRUE	TRUE

APPENDIX B: System Usability Test Results

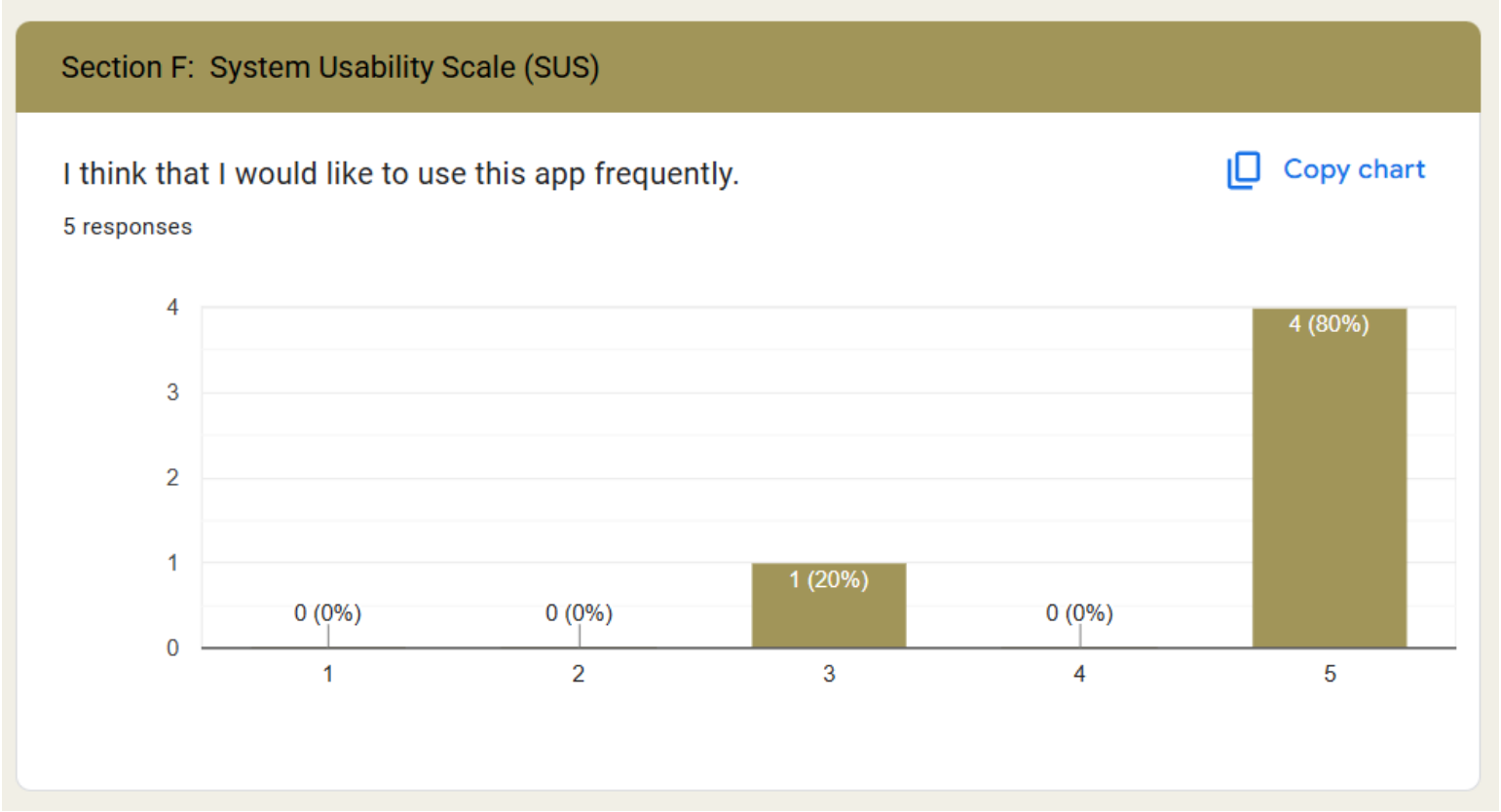


Figure B-1: SUS Result Question 1

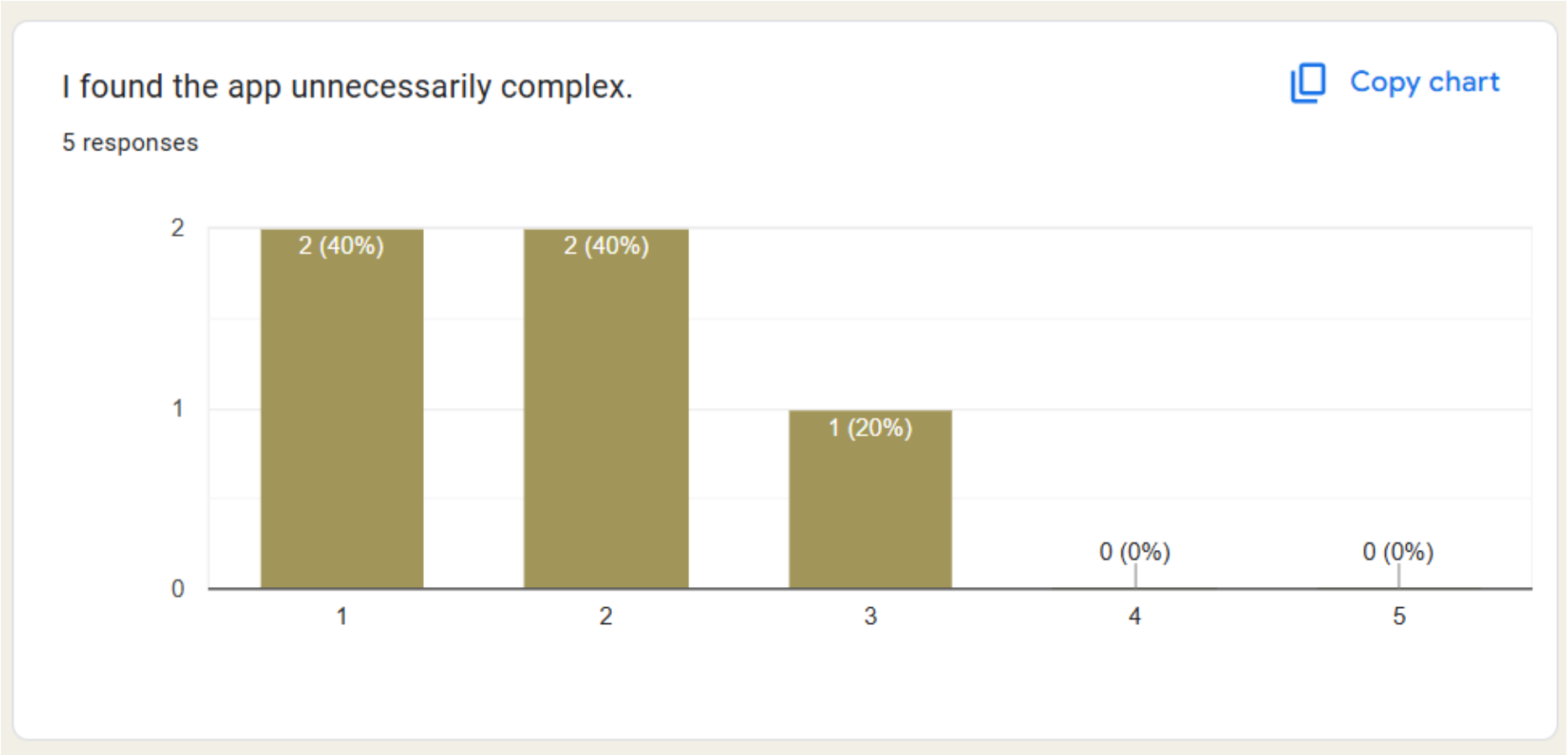


Figure B-2: SUS Result Question 2

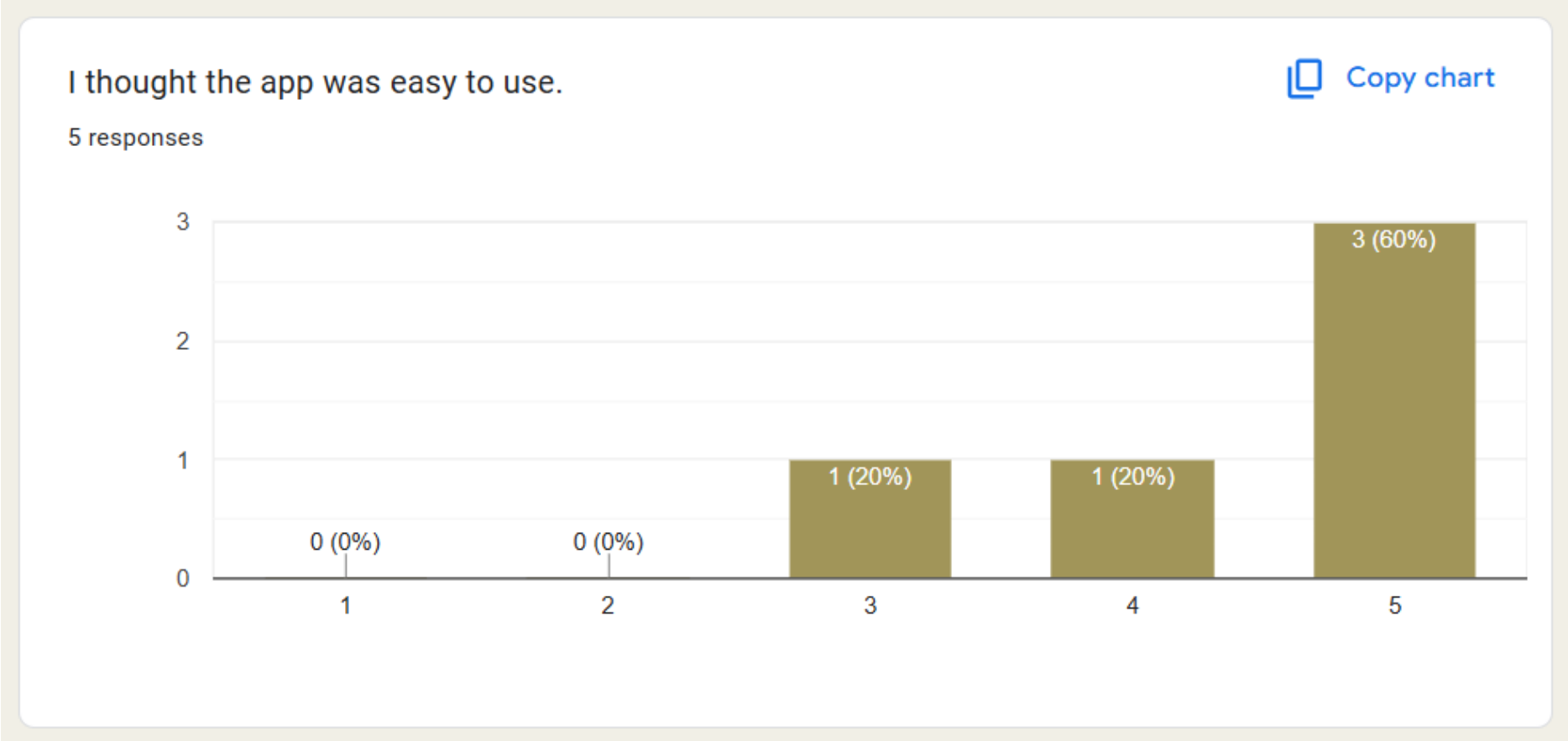


Figure B-3: SUS Result Question 3

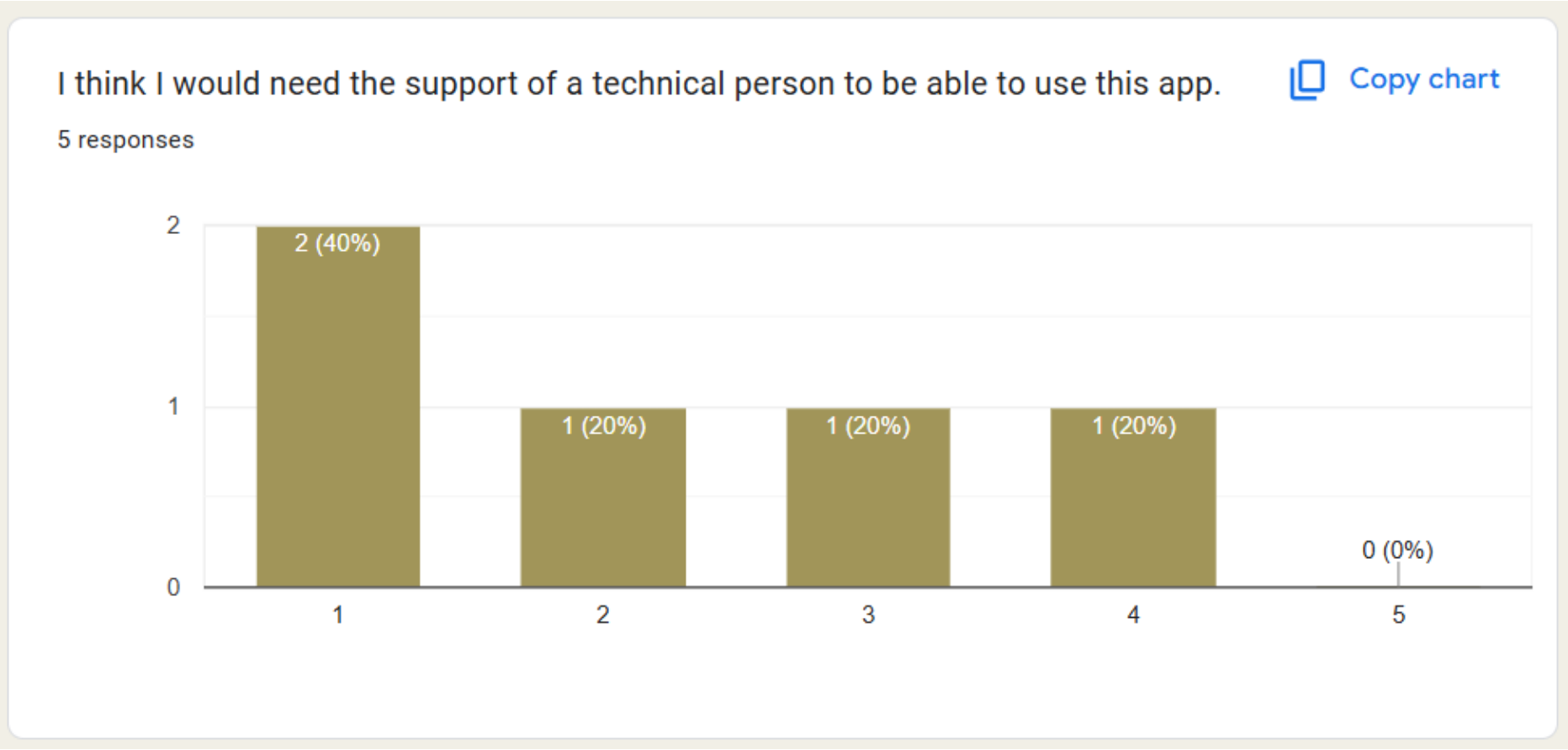


Figure B-4: SUS Result Question 4

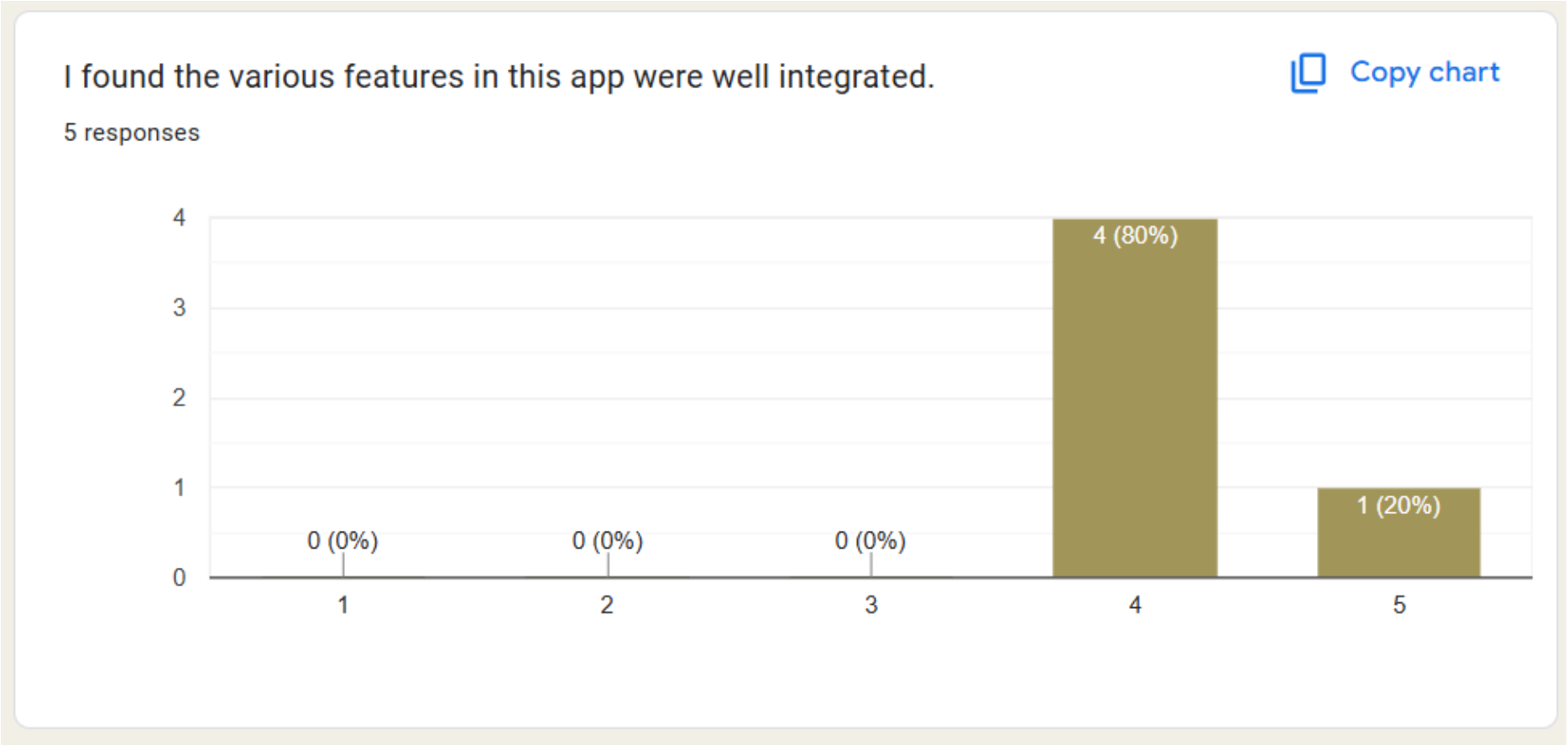


Figure B-5: SUS Result Question 5

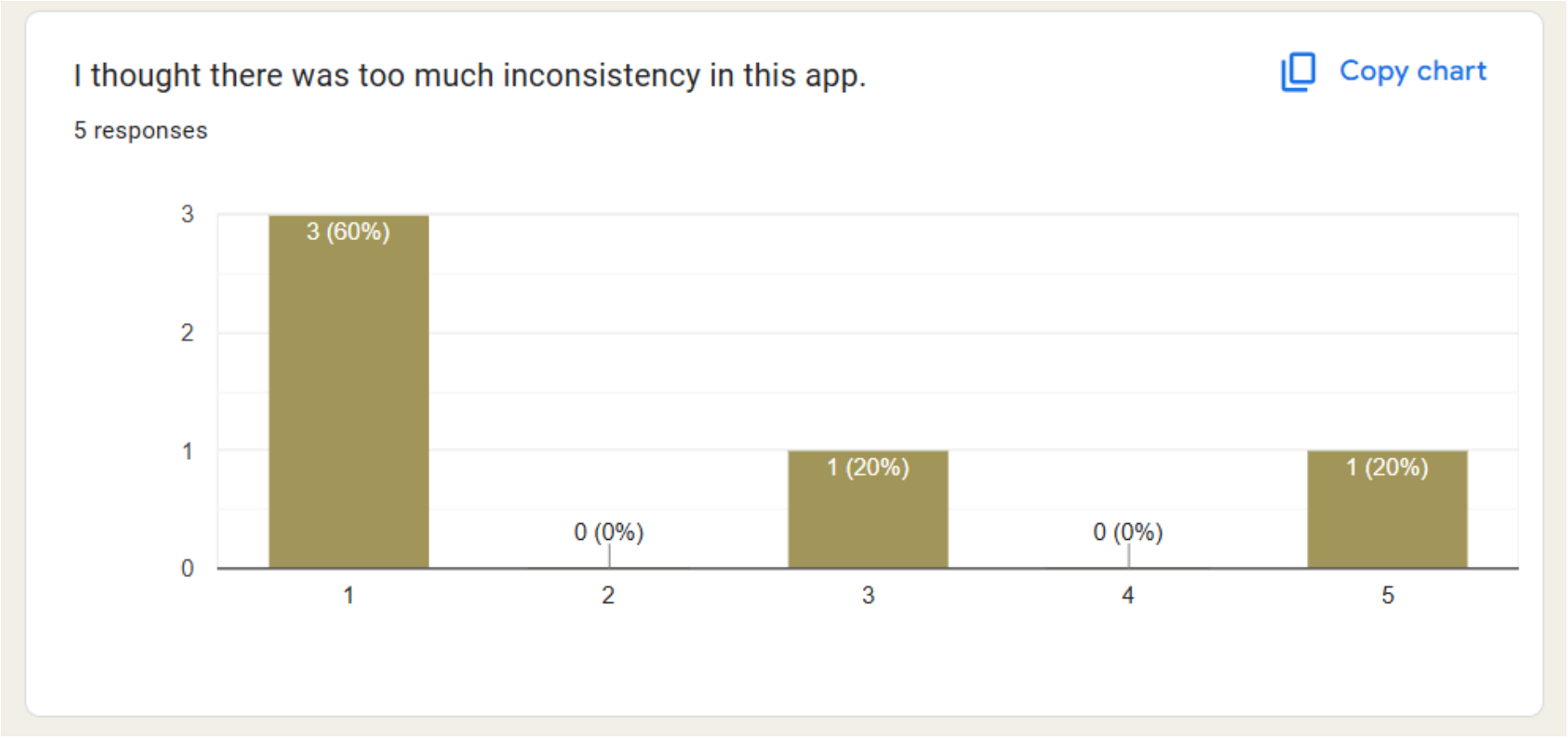


Figure B-6: SUS Result Question 6

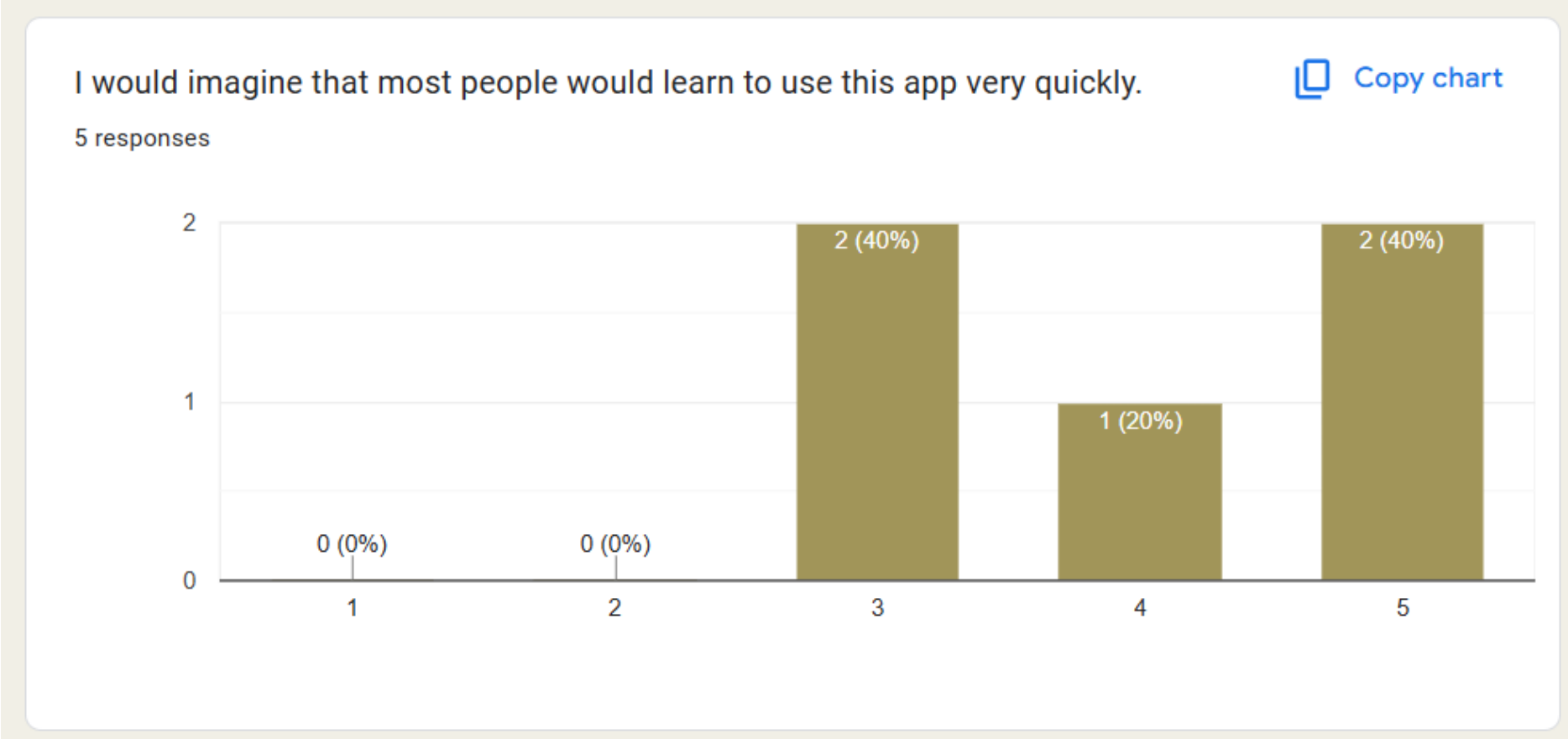


Figure B-7: SUS Result Question 7

I found the app very cumbersome to use.

Copy chart

5 responses

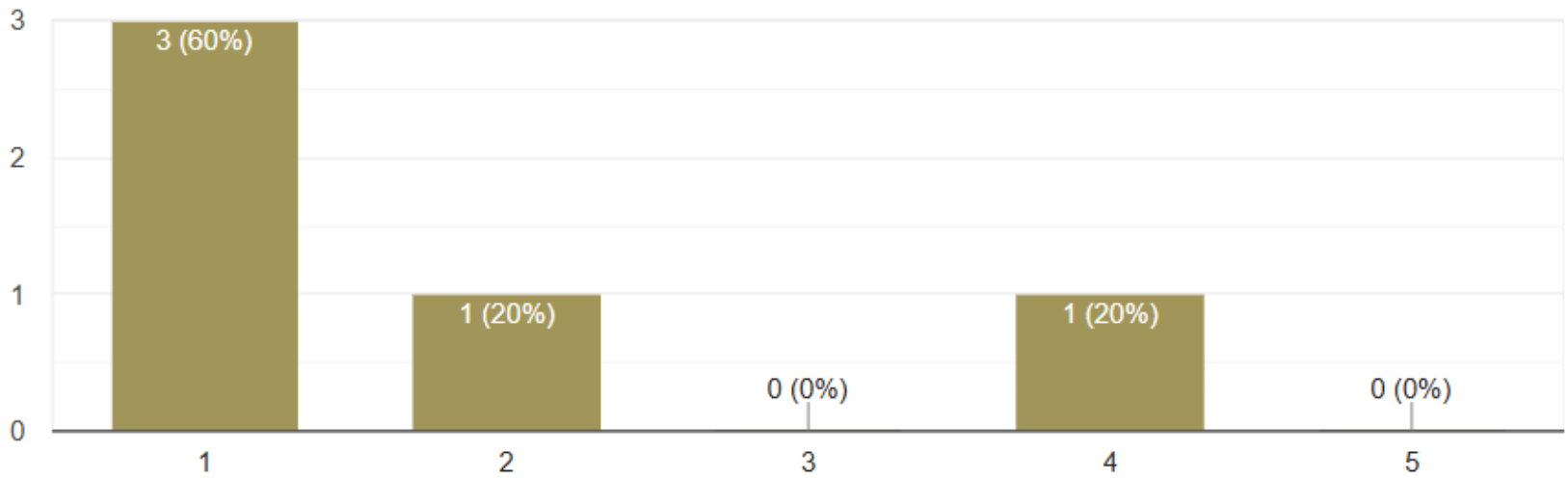


Figure B-8: SUS Result Question 8

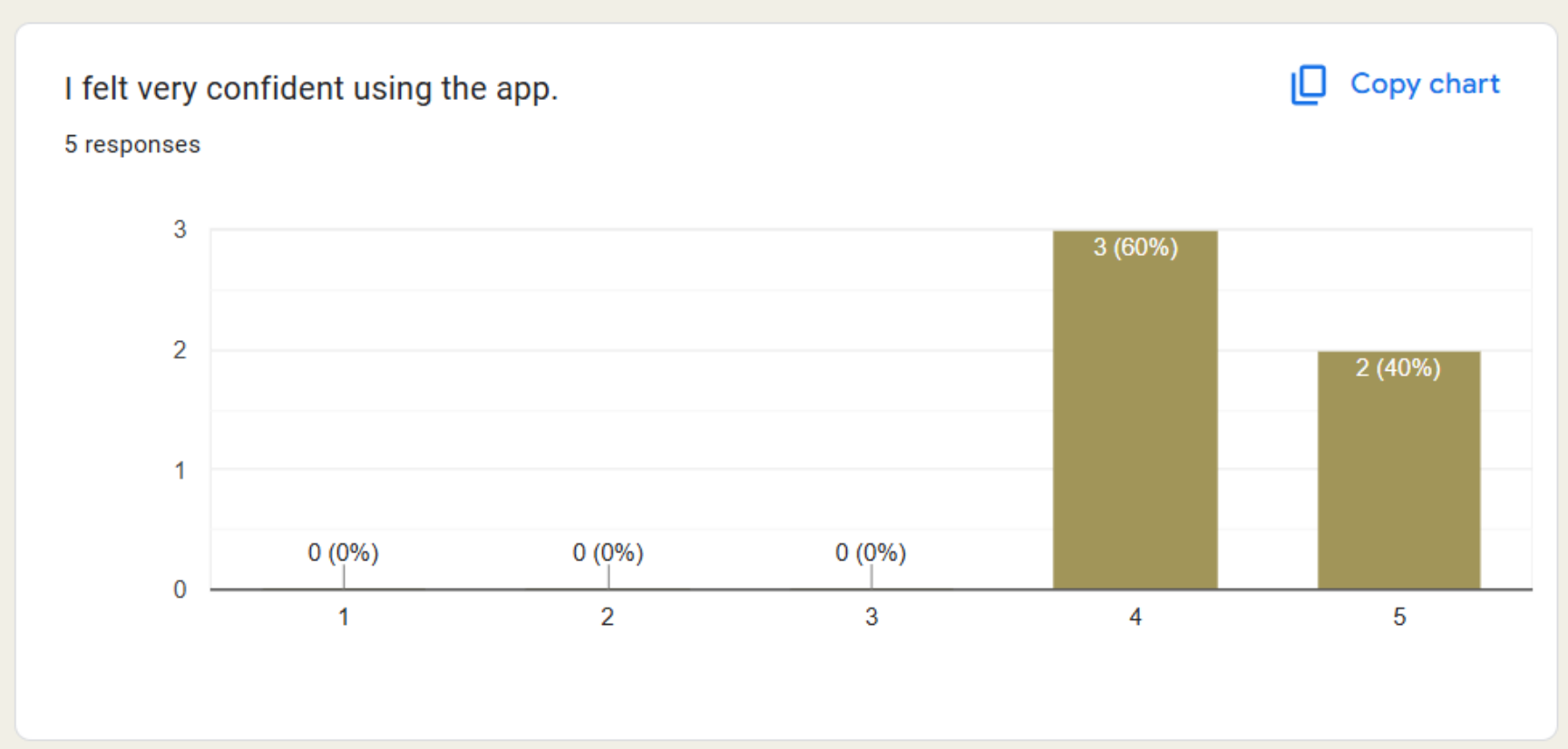


Figure B-9: SUS Result Question 9

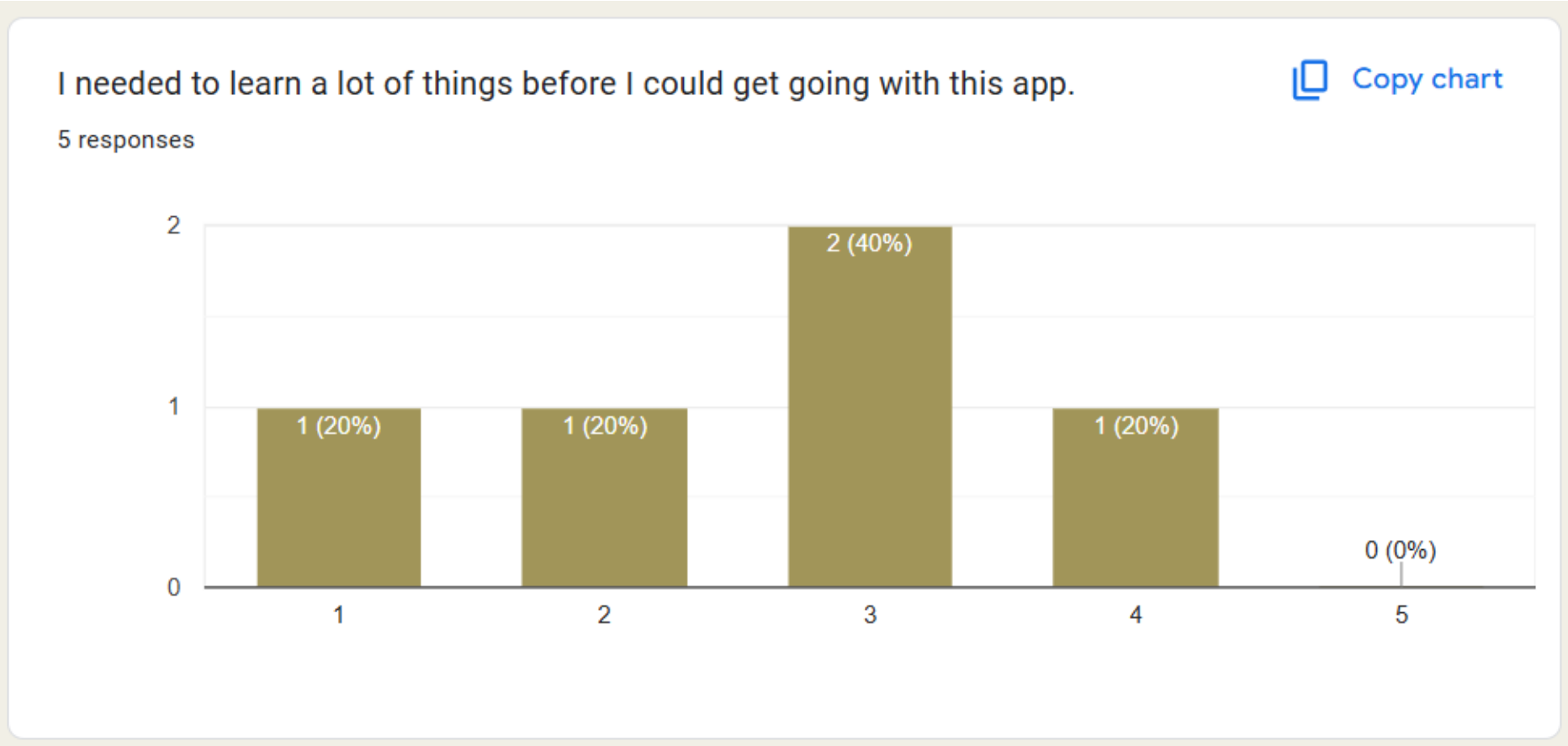


Figure B-10: SUS Result Question 10

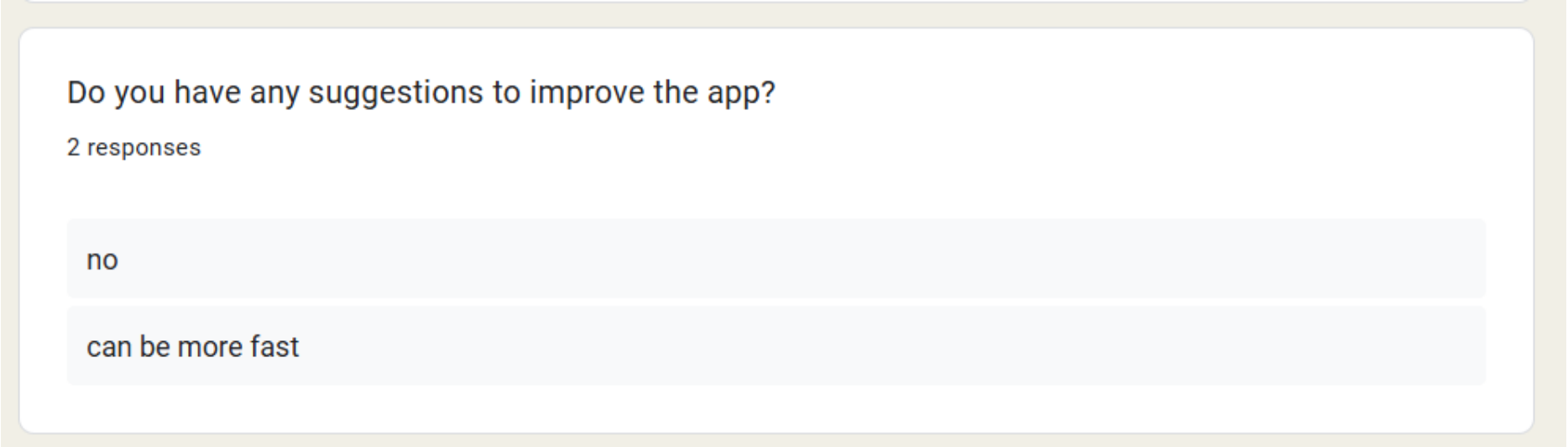


Figure B-11: UAT Suggestion

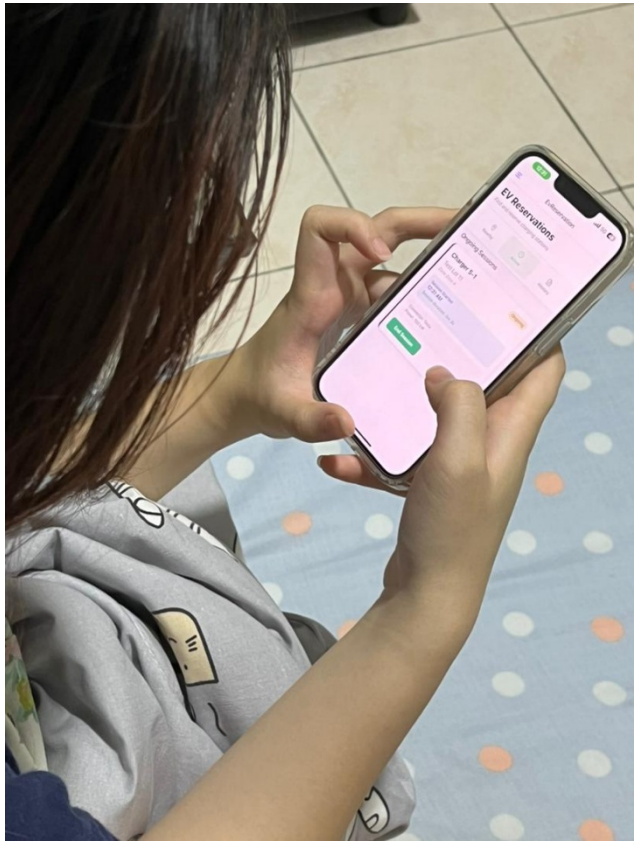


Figure B-12: Respondents testing ParkPal Application