

**VALET - SMART PARKING MANAGEMENT SYSTEM FOR UTAR KAMPAR**

**BY**

Tan Chern Lynn

**A REPORT**

**SUBMITTED TO**

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

**BACHELOR OF COMPUTER SCIENCE (HONOURS)**

Faculty of Information and Communication Technology

(Kampar Campus)

**JANUARY 2025**

## **COPYRIGHT STATEMENT**

© 2025 Tan Chern Lynn. All rights reserved.

This Final Year Project report is submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Science (Honours) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project report represents the work of the author, except where due acknowledgment 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 sincere thanks and appreciation to my supervisor, Dr. Aun Yichiet, for providing me with the opportunity to engage in the project "VALET - Smart Parking Management System for UTAR Kampar." His continuous support, guidance, and valuable insights have been crucial throughout the development of this project.

I am also grateful to the faculty members and staff of the University Tunku Abdul Rahman (Kampar, Perak Campus) for their assistance and for providing the necessary resources and facilities to carry out this research effectively.

A special thanks to my family and friends for their encouragement and unwavering support during this journey. Their patience and understanding have been instrumental in helping me stay focused and motivated.

## **ABSTRACT**

This project aims to create a smart parking system for UTAR Kampar, leveraging AIoT and OCR technologies. The primary focus is on using AIoT cameras to conduct OCR for reading vehicle license plates. The system is designed to facilitate the process of finding parking and enhance the functionality of the accompanying application. The research aims to introduce an advanced parking system powered by OCR technology, which includes data collection, analysis of existing parking setups, and the development of a user-friendly app or dashboard. This system will provide real-time car park information, validate vehicle plate numbers, enforce parking regulations, and enhance campus security. Instead of manually searching for their parked car row by row, users will be able to track their car's location through the application. Additionally, users can check the availability of parking spaces through the app, making parking more efficient and convenient.

Area of Study: Artificial Intelligence of Things, Image Processing

Keywords: Automatic License Plate Recognition (ALPR), Optical Character Recognition (OCR), YOLOv8, Firebase Realtime Database, FlutterFlow, Mobile Application Development, Object Detection, Computer Vision, Real-Time Parking Monitoring, Smart Parking System



# TABLE OF CONTENTS

<b>TITLE PAGE</b>	<b>i</b>
<b>COPYRIGHT STATEMENT</b>	<b>ii</b>
<b>ACKNOWLEDGEMENTS</b>	<b>iii</b>
<b>ABSTRACT</b>	<b>iv</b>
<b>TABLE OF CONTENTS</b>	<b>v</b>
<b>LIST OF FIGURES</b>	<b>viii</b>
<b>LIST OF TABLES</b>	<b>xii</b>
<b>LIST OF SYMBOLS</b>	<b>xiii</b>
<b>LIST OF ABBREVIATIONS</b>	<b>xiv</b>
<b>CHAPTER 1 INTRODUCTION</b>	<b>1</b>
1.1 Problem Statement and Motivation	1
1.2 Objectives	2
1.3 Project Scope and Direction	4
1.4 Contributions	4
1.5 Report Organization	5
<b>CHAPTER 2 LITERATURE REVIEW</b>	<b>6</b>
2.1 Previous Works on OCR/ALPR	6
2.2 Convolutional Neural Networks (CNN)	13
2.3 Optical Character Recognition (OCR)	18
2.3.1 The Brief History of OCR Technology	18
2.3.2 Steps Involve in OCR	20
2.4 AIoT camera	21
2.5 Critical Remarks of Previous Works	23
<b>CHAPTER 3 SYSTEM METHODOLOGY/APPROACH (FOR DEVELOPMENT-BASED PROJECT)</b>	<b>24</b>
3.1 System Design Diagram	24
3.1.1 System Architecture Diagram	24

3.1.2	Use Case Diagram and Description	30
3.1.3	Activity Diagram	32
<b>CHAPTER 4</b>	<b>SYSTEM DESIGN</b>	<b>35</b>
4.1	System Block Diagram	35
4.2	System Components Specifications	36
4.2.1	Entrance Camera Modules	36
4.2.2	Parking Area Camera Modules	38
4.2.3	Data Integration and Processing	43
4.2.4	Mobile Application	44
4.3	Components Design in Application	45
4.4	System Components Interaction Operations	61
<b>CHAPTER 5</b>	<b>SYSTEM IMPLEMENTATION (FOR DEVELOPMENT- BASED PROJECT)</b>	<b>64</b>
5.1	Hardware Setup	64
5.2	Software Setup	65
5.3	Setting and Configuration	67
5.4	System Operation (with Screenshot)	81
5.5	Implementation Issues and Challenges	107
5.6	Concluding Remark	108
<b>CHAPTER 6</b>	<b>SYSTEM EVALUATION AND DISCUSSION</b>	<b>110</b>
6.1	System Testing and Performance Metrics	110
6.1.1	License Plate Detection and Recognition	110
6.1.2	Parking Area Module	113
6.2	Testing Setup and Result	118
6.2.1	Testing Setup	118
6.2.2	Testing Result	131
6.3	Project Challenges	147
6.4	Objectives Evaluation	149
6.5	Concluding Remark	151

<b>CHAPTER 7 CONCLUSION AND RECOMMENDATION</b>	<b>153</b>
7.1 Conclusion	153
7.2 Recommendation	155
<b>REFERENCES</b>	<b>156</b>
<b>POSTER</b>	<b>A-1</b>
<b>APPENDIX</b>	<b>A-2</b>

# LIST OF FIGURES

<b>Figure Number</b>	<b>Title</b>	<b>Page</b>
Figure 2.1	Conventional ALPR System	6
Figure 2.2	Flowchart of the Algorithm	7
Figure 2.3	Detailed Algorithm	8
Figure 2.4	Algorithm Applied to Detect the Characters	9
Figure 2.5	ALPR Workflow	10
Figure 2.6	Smart Gate Entrance System	11
Figure 2.7	The Architecture of a Typical CNN	13
Figure 2.8	Architecture of the CNNs Applied to Digit Recognition	14
Figure 2.9	Maximum Pooling	16
Figure 2.10	Fully Connected Layer	17
Figure 2.11	Major Steps Involve in Character Recognition	20
Figure 3.1	System Architecture Diagram	24
Figure 3.2	Use Case Diagram	30
Figure 3.3	Activity Diagram	32
Figure 4.1	Block Diagram of System Overview	35
Figure 4.2	Component for Entrance Camera Modules	36
Figure 4.3	Simplified Overview of the CNN Architecture in Yolov8	37
Figure 4.4	Component for Parking Area Camera Modules	38
Figure 4.5	Car Detection from One of the Camera Views	39
Figure 4.6	Car Detection in Labelled Bounding Space	40
Figure 4.7	Adjustment on Location of Circle Point	41
Figure 4.8	Integrate All Previous Modules and Display Parking Counter	42
Figure 4.9	Component for Entrance Camera Modules	43
Figure 4.10	Entry Record Tab	45
Figure 4.11	Comparison of plate_number with carPlate1	46
Figure 4.12	Comparison of plate_number with carPlate2	47
Figure 4.13	Comparison of plate_number with carPlate3	48
Figure 4.14	Authorization Text field for carPlate1	49

Figure 4.15	Authorization Text field for carPlate2	50
Figure 4.16	Authorization Text field for carPlate3	51
Figure 4.17	Zone Definitions	52
Figure 4.18	Zone B Tab Page	53
Figure 4.19	Zone C Tab Page	54
Figure 4.20	Zone D Tab Page	55
Figure 4.21	Active Camera View	56
Figure 4.22	Setting on ActiveView Widget	57
Figure 4.23	Setting on Text Widget Below Image	58
Figure 4.24	Sample Data Records in the User Collection	59
Figure 5.1	Raspberry Pi Imager	67
Figure 5.2	Connection of Pi Camera to Raspberry Pi	68
Figure 5.3	Connection of Raspberry Pi to a Laptop	69
Figure 5.4	Setup Raspberry Pi's Desktop on Monitor	71
Figure 5.5	Command in Terminal	72
Figure 5.6	Create New Firestore Database	73
Figure 5.7	Select Cloud Firestore Data's Location	74
Figure 5.8	Security Rules for Database	74
Figure 5.9	Firestore Database in FlutterFlow's Firebase Tab	75
Figure 5.10	Firestore Database in Firebase Console	75
Figure 5.11	Create New Realtime database	76
Figure 5.12	Select Realtime Database Location	77
Figure 5.13	Security Rules for RealtimeDatabase	77
Figure 5.14	Realtime Database Structure	78
Figure 5.15	Storyboard of the Smart Parking System Application	81
Figure 5.16	First Page	82
Figure 5.17	Login Page	83
Figure 5.18	Query Data in Cloud Firestore	84
Figure 5.19	Successfully Sign Up for First User	85
Figure 5.20	Successfully Sign Up for Second User	85
Figure 5.21	Successfully Sign Up for Third User	86
Figure 5.22	Test Case for Wrong Email	87
Figure 5.23	Snack Bar Message for Wrong Email	87

Figure 5.24	Test Case for Wrong Password	88
Figure 5.25	Snack Bar Message for Wrong Password	88
Figure 5.26	Test Case for Wrong Email and Password	89
Figure 5.27	Snack Bar Message for Wrong Email and Password	89
Figure 5.28	Multiple Conditions in Sign In Button	90
Figure 5.29	Home Page for First User in Test Mode	91
Figure 5.30	Home Page for Second User in Test Mode	92
Figure 5.31	Home Page for Third User in Test Mode	93
Figure 5.32	Home Page in Preview Mode	94
Figure 5.33	Realtime Database in Firebase Console	95
Figure 5.34	Python Script Involve Setting of Firebase	96
Figure 5.35	Service Accounts Tab in Firebase Console	97
Figure 5.36	URL of Firebase	98
Figure 5.37	Button Disable Options	99
Figure 5.38	If-else Condition Applied in Notification Button	100
Figure 5.39	Create API Call	100
Figure 5.40	Setting to Apply API Call Named Availability	101
Figure 5.41	Preview Mode	102
Figure 5.42	Code Snippet for Storing Encoded String	103
Figure 5.43	Create API Call to Get Encoded String	104
Figure 5.44	Setting to Apply API Call Named GetBaseImage	104
Figure 5.45	Refresh Now Button in Block N Page	106
Figure 6.1	License Plate Detection	110
Figure 6.2	License Plate Number Recognition	111
Figure 6.3	View 1 Captured by Raspberry Pi Camera Module 3	114
Figure 6.4	View 2 Captured by Raspberry Pi Camera Module 3	115
Figure 6.5	View 3 Captured by Raspberry Pi Camera Module 3	115
Figure 6.6	Carpark Lots 1 - 7 with Boundaries and Labels	116
Figure 6.7	Carpark Lots 8 - 13 with Boundaries and Labels	116
Figure 6.8	Carpark Lots 14 - 20 with Boundaries and Labels	117
Figure 6.9	Pi Camera Attached to a Power Bank	119
Figure 6.10	Raspberry Pi Attached to a Power Bank	119
Figure 6.11	Entire Setup Held by Phone Holder	121

Figure 6.12	Setup Positioned in the Carpark	121
Figure 6.13	Status LEDs Lighting Up	122
Figure 6.14	Personal Hotspot's Connected Devices Interface	123
Figure 6.15	PuTTY Configuration Window	124
Figure 6.16	Raspberry Pi Terminal	124
Figure 6.17	RealVNC Viewer	125
Figure 6.18	Raspberry Pi's desktop in RealVNC Viewer	125
Figure 6.19	Python Script in Thonny IDE	126
Figure 6.20	Script Execution in Terminal	127
Figure 6.21	Running Script	128
Figure 6.22	Test Mode in Flutterflow	129
Figure 6.23	Firebase Realtime Database	130
Figure 6.24	First Page's View from Mobile Phone	131
Figure 6.25	First Negative Test Case for Login Page	132
Figure 6.26	Second Negative Test Case for Login Page	133
Figure 6.27	Third Negative Test Case for Login Page	134
Figure 6.28	Home Page	136
Figure 6.29	Empty Spaces for each Zone and Block	137
Figure 6.30	Entry Record	139
Figure 6.31	Block N Page	140
Figure 6.32	Changes when Car Parked on Empty Parking Lot	142
Figure 6.33	Changes when Car leave Parking Lot	143
Figure 6.34	Test Case for Full Parking View	144
Figure 6.35	New Notification when Parking Full	145
Figure 6.36	Pop Up Content for the Notification	146

## LIST OF TABLES

<b>Table Number</b>	<b>Title</b>	<b>Page</b>
Table 2.1	Summary Table of Previous Work Review on ALPR/OCR	12
Table 2.2	Explanation of CNN Layers	15
Table 2.3	Timeline of OCR Technology	19
Table 2.4	Strength and Weakness of Previous Works	23
Table 5.1	Specifications of Laptop	64
Table 6.1	Accuracy Tests	112



# LIST OF SYMBOLS

$\in$  Element of

## LIST OF ABBREVIATIONS

<i>5G</i>	Fifth Generation
<i>API</i>	Application Programming Interface
<i>CPU</i>	Central Processing Unit
<i>GPIO</i>	General Purpose Input Output
<i>IOT</i>	Internet of Things
<i>IP</i>	Internet Protocol
<i>RAM</i>	Random Access Memory

# Chapter 1

## Introduction

In this chapter, a concise overview of the smart parking management system is provided, focusing on the key problems it aims to address and the motivation behind the project. The challenges of managing vehicle parking within the UTAR Kampar campus are discussed, highlighting the need for a more efficient and secure solution. The chapter outlines the objectives of the research, defines the scope and direction of the project, and presents the anticipated contributions to the field.

### 1.1 Problem Statement and Motivation

The parking situation at UTAR Kampar has become a significant issue due to the increasing number of students, staff, and visitors. Despite car sticker regulations intended to control vehicle access, unauthorized cars often enter the campus, leading to congestion and a shortage of parking spaces. This not only frustrates authorized users, who sometimes cannot find a place to park despite having paid for car sticker, but also results in inefficient manual enforcement by security guards, causing further delays and errors. The current system's limitations in effectively managing and monitoring parking spaces highlight the need for a more efficient solution.

To address these challenges, this project aims to develop a Smart Parking Management System that provides real-time updates on parking availability across different zones within the UTAR Kampar campus. The system focuses on improving the overall parking experience by delivering live information through a mobile application, allowing users to make informed decisions before entering the campus. As part of its features, the system includes vehicle license plate detection and recognition to verify whether a car is authorized to park within campus grounds. Instead, the system emphasizes automation, convenience, and enhanced security by enabling digital monitoring and timely in-app notifications. This aligns with UTAR's commitment to smart solutions that optimize resource use and improve user experience on campus.

### 1.2 Objectives

The aim of this project is to develop a smart parking system to address UTAR's parking issues while enhancing campus security. At its core, this initiative centers on five key objectives, each pivotal in reshaping the parking landscape within UTAR's campus.

#### **Building a UTAR Carpark Management App Using FlutterFlow**

The foundational objective of this project is to create a user-friendly mobile application for UTAR carpark management using FlutterFlow. This app stands as the primary interface for users, offering seamless access to various parking management functionalities. Leveraging FlutterFlow's cross-platform capabilities, the goal is to create an intuitive and user-friendly application interface. This app will enable efficient navigation and utilization of the smart parking system, inclusive of both a user-friendly mobile application and an intuitive dashboard interface. Through this, users can access parking information and effectively manage parking spaces within the Kampar campus.

#### **Developing a Robust OCR for Car Plate Recognition**

The core technological focus is on developing a robust OCR system capable of accurately recognizing and interpreting license plate information under various weather conditions, such as rain or heavy sunlight. The goal is to ensure consistent performance of the OCR technology across different environments, maintaining accuracy in license plate recognition. This reliability is essential for correctly detecting and displaying information, such as the car's authorization, regardless of weather conditions.

#### **Designing a Labelling System for Parking Bays in UTAR**

Implementing a structured labelling system for parking bays within the UTAR campus is crucial. This system involves categorizing and assigning distinct identifiers to each parking space. Each camera is assigned to capture a specific range of parking lots, ensuring that even if some parking lot views overlap between two cameras, each parking lot is assigned to only one camera. This prevents overlapping detection and ensures the accuracy of the parking counter. These labels facilitate the identification of available parking spots and enable the system or cameras to accurately detect occupancy status. This organized approach supports efficient parking space allocation, allowing users of the app to easily identify and navigate to available parking spots.

### **Providing Real-Time Parking Zone Monitoring with Live Availability and Visual Updates**

This feature provides users with live updates on parking availability across various campus zones, allowing them to monitor real-time parking conditions. The system displays both the number of available parking bays and the most recent visual feed from cameras placed in each parking zone. By presenting up-to-date occupancy data, users can quickly assess where parking spaces are available and make informed decisions, optimizing their parking search. This capability significantly enhances convenience by reducing the time spent searching for a vacant parking spot, particularly in large, crowded areas. The visual feed further aids users in confirming parking space availability and helps them navigate to the right areas with ease, improving the overall parking experience across campus.

### **Alerting Users on Full Parking Areas Through In-App Notifications**

This functionality keeps users informed by sending notifications when specific parking areas on campus are fully occupied. The system continuously monitors the status of each parking zone via connected cameras, tracking the availability of spaces in real time. When a parking zone reaches full capacity, users are immediately alerted through in-app notifications, which highlight the specific area that is no longer available. This timely alert system prevents users from wasting time driving through already full areas, especially during peak parking hours. By notifying users about full zones, the feature not only saves time but also helps reduce congestion and frustration, providing a smoother and more efficient parking experience. The real-time nature of these alerts makes the system highly responsive, ensuring that users are always aware of the latest parking status.

### 1.3 Project Scope and Direction

This project will deliver a smart parking management system that integrates both hardware and software components to efficiently manage parking at UTAR Kampar. The system will utilize a Raspberry Pi and Raspberry Pi Camera to detect parking space availability and recognize car license plates using Optical Character Recognition (OCR) technology. This enables real-time vehicle detection and authorization, ensuring efficient and secure management of parking spaces.

The final output will be a user-friendly application that displays a dashboard of the parking system, providing real-time information on parking availability and vehicle entry data. The application will not only allow users to easily locate available parking spots but will also enable administrators to monitor parking activities, identify unauthorized vehicles, and enforce parking regulations effectively. Additionally, the system will maintain a log of all vehicles entering and exiting the premises, which can be valuable for security purposes and incident tracking. By offering a centralized platform for both users and administrators, this application will simplify the management of parking operations, enhance campus security, and significantly improve the overall parking experience for all users.

### 1.4 Contributions

This project aims to significantly improve the UTAR community's experience by addressing the critical issue of parking management with a smart, technology-driven solution. By utilizing OCR technology and a real-time dashboard application, the project will enhance the efficiency of parking space utilization, reduce congestion, and increase security on campus. These advancements will create a more organized and stress-free environment for students, faculty, and visitors, ensuring that those with valid permits can easily access parking spaces. Beyond alleviating current parking challenges, this project provides a scalable model for other institutions facing similar issues, demonstrating the potential for technology to bring practical improvements to everyday campus life.

The significance of this project extends well beyond the immediate benefits to UTAR. By illustrating how modern technologies like AIoT and OCR can effectively address common logistical problems, this project sets a precedent for future innovations in smart campus management. The user-friendly application not only improves parking processes but also serves as a proof of concept for integrating advanced technologies into campus operations. This project has the potential to transform parking management into a more efficient, user-centric, and secure system, making it a valuable contribution to both academic research and practical applications. Readers will gain valuable insights into how targeted technological solutions can resolve persistent issues and improve the quality of life on campus, demonstrating the far-reaching impact of innovative problem-solving approaches.

### **1.5 Report Organization**

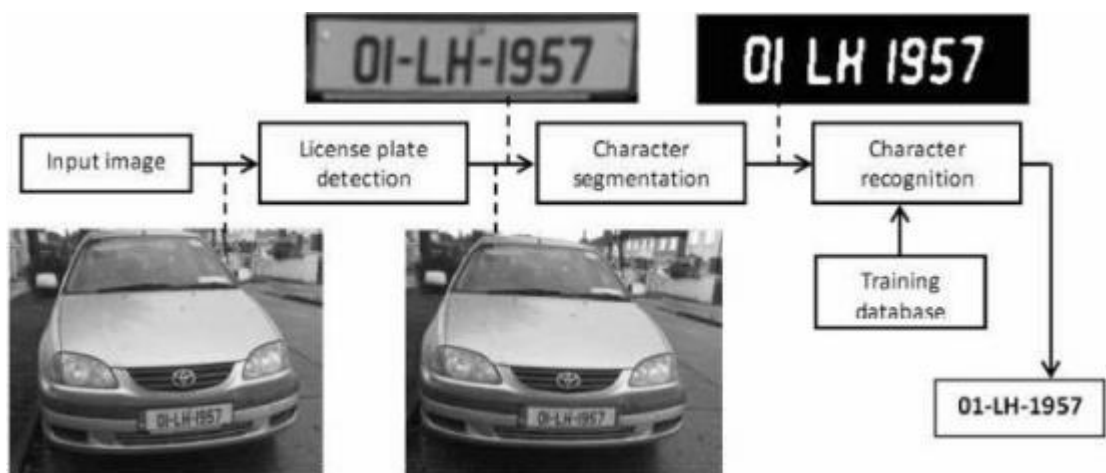
This report is structured into six chapters, each addressing a specific phase of the project. Chapter 1 introduces the Smart Parking Management System by outlining the background, problem statement, motivation, objectives, project scope, contributions, and the overall direction of the study. Chapter 2 presents the literature review, focusing on previous works related to OCR and Automatic License Plate Recognition (ALPR), the use of Convolutional Neural Networks (CNN), the evolution and technical steps of OCR technology, the implementation of AIoT cameras, and a critical analysis of existing systems. Chapter 3 discusses the system design, detailing both the hardware and software architecture, camera positioning, user interface layout, and the database structure used to support real-time monitoring and license plate recognition. Chapter 4 covers system implementation and testing, including the development of the OCR module, the integration of Raspberry Pi and cameras, and the construction of the mobile application using FlutterFlow, followed by a description of the testing procedures and performance evaluation. Chapter 5 presents the system outcomes and discussion, analyzing the results, identifying key contributions, and reflecting on the strengths, limitations, and real-world applicability of the proposed solution. Finally, Chapter 6 concludes the report by summarizing the key findings, highlighting the project's achievements, and suggesting potential improvements and future enhancements to further advance the smart parking system.

## Chapter 2

### Literature Review

#### 2.1 Previous Works on ALPR/OCR

ALPR is a technology that uses optical character recognition to automatically read and capture the license plate number of vehicles [2]. The development of a conventional ALPR system can be divided into four steps, as shown in Figure 2.1 [3].

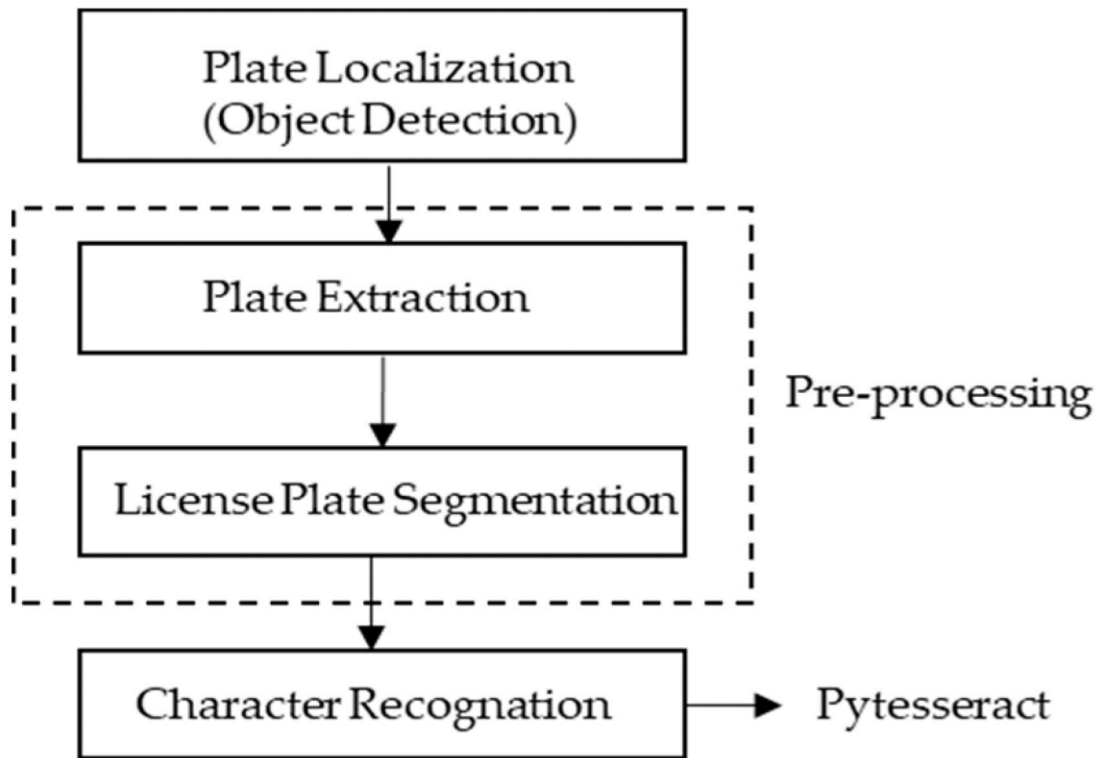


**Figure 2.1 Conventional ALPR System**

The initial functions of vehicle image capture and number plate detection involve identifying the vehicle and capturing an image to identify potential regions that may contain the license plate. Concerning character segmentation, its primary role is isolating foreground characters from the background within the detected license plate area. The character recognition step employs methods aimed at identifying and interpreting these characters.

In [4], a system was proposed to read frames of a camera and recognize the characters present on license plate in real-time. Figure 2.2 shows the flowchart of the algorithm.





**Figure 2.2 Flowchart of the Algorithm**

When a vehicle is detected by smart parking service, the algorithm initiates the plate localization process to extract the license plate from the vehicle image. This involves identifying rectangular objects within the vehicle image, which are likely to be license plates. Upon locating a plate, the algorithm isolates and creates a new image specifically containing the plate area. Subsequently, the algorithm engages in pre-processing the plate image to address environmental variations such as illumination, background discrepancies, and texture differences. To mitigate potential image noise, a bilateral filter [5] is applied as part of the pre-processing stage, aimed at smoothing the image. Following the pre-processing step, the algorithm utilizes Tesseract [6] to perform character recognition on the plate image. Refer to Figure 2.3 for a detailed algorithmic representation, and Figure 2.4 for the result of the algorithm using image taken from sensors.

```

1. Start Video Capture
2. Read Video_Frame
3. Resize Video_Frame
4. Gray_Frame = Transform to grayscale (Video_Frame)
5. Apply filter to remove the noise (Gray_Frame)
6. Frame_Edges = Detect edges (Gray_Frame)
7. Frame_Contours = Find contours (Frame_Edges)
8. Sort (Frame_Contours)
9. Declare Number_Plate_Contour
10. Declare Largest_Rectangle
11. for Contour in Frame_Contours do
12.     Perimeter_Rectangle = Calculate perimeter (Contour)
13.     Approx_Rectangle = Find the approximate rectangle (Perimeter_Rectangle)
14.     if (length (Approx_Rectangle) == 4) then
15.         Number_Plate_Contour = Approx_Rectangle
16.         Largest_Rectangle = Find area (Approx_Rectangle)
17.         break
18. x,y,w,h = Calculate up-right bounding rectangle (Largest_Rectangle)
19. Cropped_Frame = Crop using x,y,w,h (Video_Frame)
20. Draw Largest_Rectangle contours on Video_Frame
21. Transform to grayscale (Cropped_Frame)
22. Frame_Threshold = Binarize (Cropped_Frame)
23. Kernel = New square object of size 1x1
24. Image_Dilation = Dilates using Kernel (Frame_Threshold)
25. Dilated_Image_Contours = Find contours (Image_Dilation)
26. Sorted_Dilated_Contours = Sort (Dilated_Image_Contours)
27. for Dilated_Contour in Sorted_Dilated_Contours
28.     x,y,w,h = Calculate up-right bounding rectangle (Dilated_Contour)
29.     Draw a rectangle of dimensions x,y,w,h on Video_Frame
30. end for
31. Transform to binary (Gray_Frame)
32. License_Plate_Characters = Transform to string (Gray_Frame)
33. if length(License_Plate_Characters) > 0 then
34.     Get License_Plate_Characters
35. end if
36. return Video_Frame

```

Figure 2.3 Detailed Algorithm



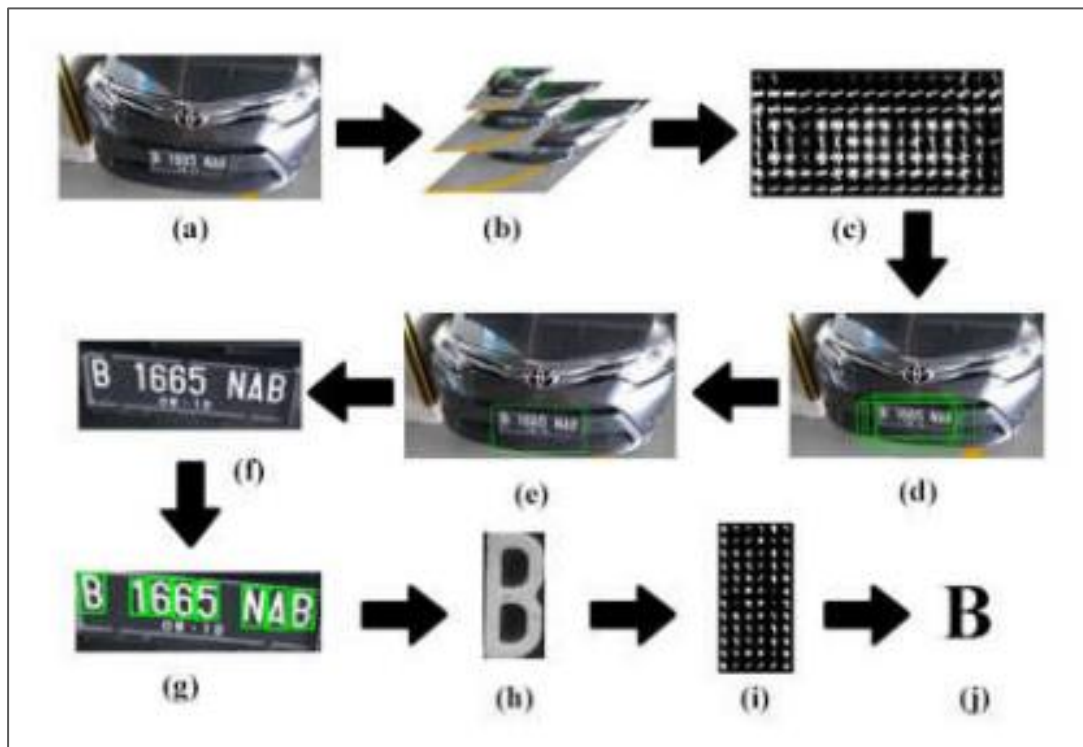
**Figure 2.4 Algorithm Applied to Detect the Characters**

As can be seen in the proposed algorithm from previous work, the sequence initiates by activating the video camera. Subsequently, the camera reads and adjusts the frames' size (lines 2, 3), scaling them based on the camera's distance from the vehicle to enhance the plate's perspective. Following this, filters (lines 4 to 7) are employed to identify image contours. Initially, as shown in Figure 2.4(b), a grayscale filter is applied, succeeded by a bilateral filter to eliminate image noise while preserving edge details, as portrayed in Figure 2.4(c). The ensuing step involves edge detection, displayed in Figure 2.4(d). A loop commences at line 11, seeking the rectangle encompassing the plate numbers. Once identified, this rectangle is cropped (line 19) from the main frame. Subsequently, lines 21 to 31 illustrate the preprocessing steps applied to the cropped image, resulting in Figure 2.4(f). Finally, character recognition is conducted at line 34 within the algorithm.

In [7], the authors discuss the implementation of ALPR which involves capturing license plate images using cameras and processing them through various stages. The process begins by detecting possible license plate locations in an image, followed by

character identification through OCR to convert the plate image into text for further analysis. The workflow of ALPR, as depicted in Figure 2.5 included:

(a) Given image → (b) Detector Window → (c) HOG → (d) Detection → (e) NMS → (f) License Plate → (g) Segmentation → (h) Region of Interest → (i) HOG of ROI → (j) Result.



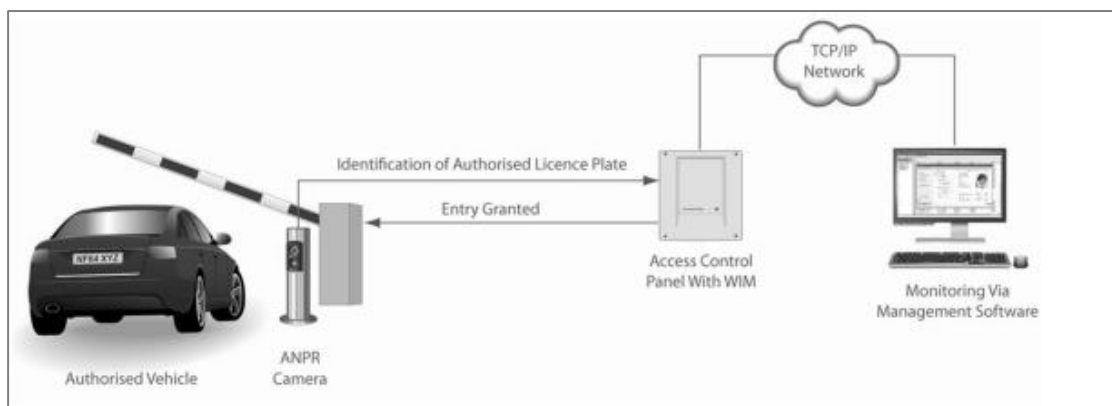
**Figure 2.5 ALPR Workflow**

To extract the license plate from an image, the authors employ a HOG Descriptor and a Linear SVM-based window classifier. The SVM-based window classifier uses a grid of overlapping blocks to extract HOG feature vectors representing object categories. This approach divides the frame into spatial regions, analysing gradient orientation and magnitude to differentiate between plate and non-plate areas [8].

The detection window scans the entire image at various positions and scales, employing NMS to identify object instances and generate license plate candidates. These candidates undergo several processes. Initially, they are converted to binary images and labelled using a method to identify interconnected pixels [9]. This step aims to isolate character-containing regions and filter out non-character elements. Subsequently, a

filtration process occurs based on given character boundaries, segmenting characters for recognition. Finally, character recognition occurs through SVM classification, where characters are extracted as Hog features and classified to produce readable strings.

In [10], the authors developed an ALPR system tailored for Indian license plates. Their system implemented the Tesseract LSTM OCR engine to accurately segment and recognize license plates, as shown in Figure 2.6.



**Figure 2.6 Smart Gate Entrance System**

To detect license plates, they utilized a Faster-RCNN network incorporating the RPN for object detection. Furthermore, the system employed pre-processing techniques like gray scaling, Gaussian blurring, and adaptive thresholding for binarization. As a result of these methods, the system achieved an impressive recognition accuracy of 95%.

The table 2.1 shows the summary table of the works on OCR for ALPR. This summary encapsulates the methodologies, main techniques, detection method, character recognition, pre-processing and reported recognition accuracy from various studies focusing on ALPR and OCR technologies for license plate recognition.

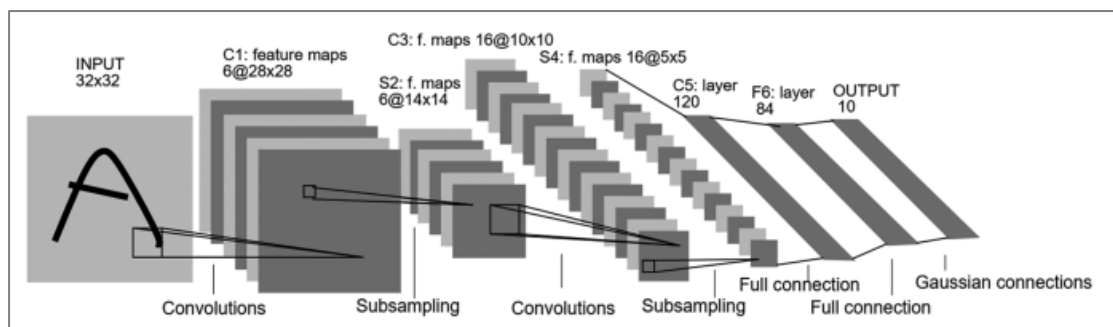
**Table 2.1 Summary Table of Previous Work Review on ALPR/OCR**

<b>Previous Work</b>	[2]	[3]	[6]	[9]
<b>Methodology</b>	Conventional ALPR system development	Real-time plate recognition system	ALPR with HOG Descriptor and Linear SVM	ALPR system for Indian license plates
<b>Main Techniques</b>	Vehicle image capture, number plate detection, character segmentation, character recognition	Plate localization, image pre-processing, character recognition using Tesseract	License plate location detection, pre-character identification using OCR	Faster-RCNN network for plate detection, pre-processing techniques, Tesseract LSTM OCR engine
<b>Detection Method</b>	-	Smart Parking Service	HOG Descriptor & Linear SVM	Faster-RCNN & RPN
<b>Character Recognition</b>	-	Tesseract	SVM classification	Tesseract LSTM
<b>Pre-processing</b>	-	Bilateral filter	Grayscale, blurring, thresholding	Grayscale, blurring, thresholding
<b>Recognition Accuracy</b>	Not specified	Not specified	Not specified	95% accuracy

## 2.2 Convolutional Neural Networks (CNN)

A CNN is a subset of Deep Neural Networks designed for image classification and analysis. In CNNs, each layer's output relies on a limited number of inputs, enhancing their ability to learn effectively from smaller datasets and reducing overfitting [11].

The typical CNN was proposed in 1998 [12]. In the Network as shown in Figure 2.7, the inputs pass through five convolutional layers and activate functions (e.g., the Sigmoid function) and eventually reach a fully connected layer, whose outputs are used as predicting factors in classification.



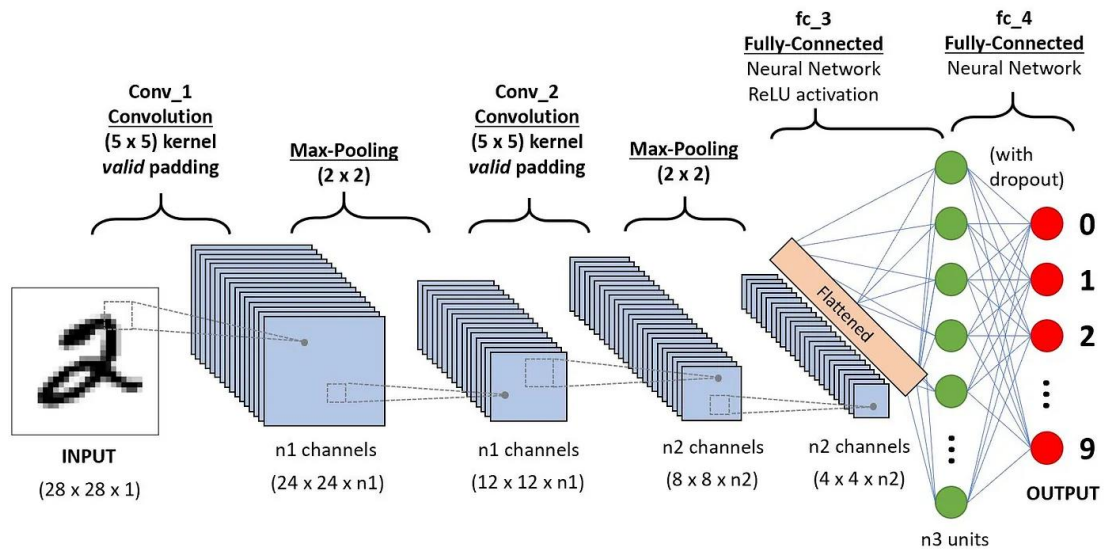
**Figure 2.7 The Architecture of a Typical CNN**

In the image encoding module, a Convolutional Neural Network (CNN) is utilized to encode the key features of an input text image, representing them as a sequence of feature vectors [13], with each vector corresponding to a specific local region of the input image. Unlike the sub-images used in segmentation-recognition methods, which are expected to contain complete letters or characters from the text image, the local regions here are simply vertical strips of the input image with a pre-defined width. This approach simplifies the process of obtaining these regions. A straightforward method for image encoding involves first slicing the input text image into vertical sections, and then applying a CNN of consistent structure and parameters to these sections, generating a sequence of feature vectors. This is known as the cutting-encoding approach.

Alternatively, a larger CNN can be applied to the entire input text image at once, and the resulting feature map can then be divided into a sequence of feature vectors, each representing one local region. The key advantage of this encoding-cutting approach is that it allows image information from adjacent regions to flow across the boundaries,

leading to more robust and informative feature vectors. In practice, the CNN can be configured with various architectures such as AlexNet [13], VGGNet [14], GoogleNet [15], ResNet [16], among others. Given the rapid advancements in computer vision and the continual introduction of new CNN architectures, the image encoding module can be updated to achieve state-of-the-art performance. However, as the network becomes deeper and more complex, the demands for training data and computational resources also increase. In some cases, the encoding module can be skipped by directly utilizing the features extracted by the text detection method from the earlier stages of the pipeline.

Figure 2.8 illustrates the functioning of CNN layers using an example of classifying handwritten digits [17]. Additionally, Table 2.2 provides a detailed explanation of the process where a 32x32-pixel handwritten digit is subjected to a series of CNN layers [18].



**Figure 2.8 Architecture of the CNNs Applied to Digit Recognition**



**Table 2.2 Explanation of CNN Layers**

Layer	Explanation
C1	The initial convolutional layer comprising six 5x5 kernels that scan the input image, producing six 28x28 output images. This layer typically detects fundamental features like edges and corners.
S2	Following C1, this layer is an average pooling or subsampling layer. It reduces each group of four pixels in the C1 output to a single pixel, effectively halving the size of the six 28x28 images to six 14x14 images.
C3	The second convolutional layer, housing 16 5x5 kernels that operate on the six 14x14 images from S2, generating 16 images of size 10x10.
S4	Another average pooling layer, scaling down the sixteen 10x10 images to sixteen 5x5 images.
C5	This fully connected convolutional layer contains 120 outputs. Each of the 120 output nodes connects to all 400 nodes (5x5x16) stemming from S4. The output here transforms from an image to a 1D array with a length of 120.
F6	Another fully connected layer, mapping the 120-array to a new array with a length of 10, where each element corresponds to one of the ten handwritten digits (0-9).
Output Layer	A softmax function applied to the output of F6, transforming it into a probability distribution of ten values summing up to 1, representing the likelihood of each digit class.

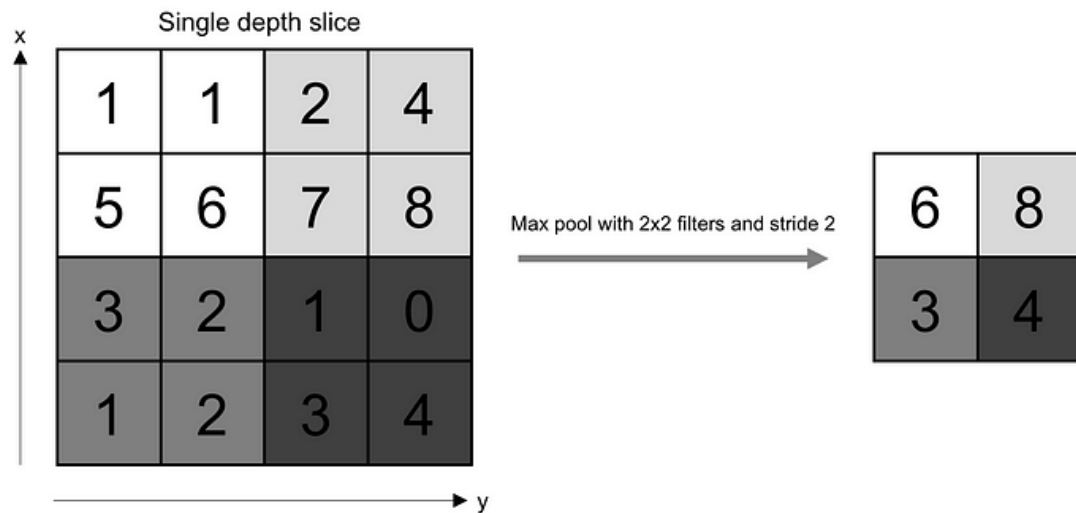
A CNN architecture typically contains a convolutional layer, a pooling layer, a fully connected layer and an output layer [19].

### Convolutional Layers

These layers employ various filters to extract features within their immediate proximity, acting as feature extractors. In a given convolutional layer  $l$ , assuming  $n^{[l]}$  filters of size  $3 \times 3$  represented as  $W^{[l]} \in \mathbb{R}^{3 \times 3 \times n^{[l]}}$  where  $n^{[l-1]}$  denotes the number of filters in the preceding layer. Employing a stride of [1,1], these filters traverse the entire input feature map. The resulting output characteristics are  $Y^{[l]} = f(\sum_{n=0}^n W^{[l]}[n] \otimes Y^{[l-1]} + b^{[l]})$  where  $f(\mathcal{A})$  signifies a non-linear activation function like ReLU or ELU. Additionally, for the initial layer in the network, if  $Y^{[l-1]}$  denotes the input image  $X \in \mathbb{R}^{w \times h \times 1}$ , zero-padding the matrix margins, known as same padding, is employed to maintain the invariant size of input matrices and retain maximal margin information.

### Pooling Layers

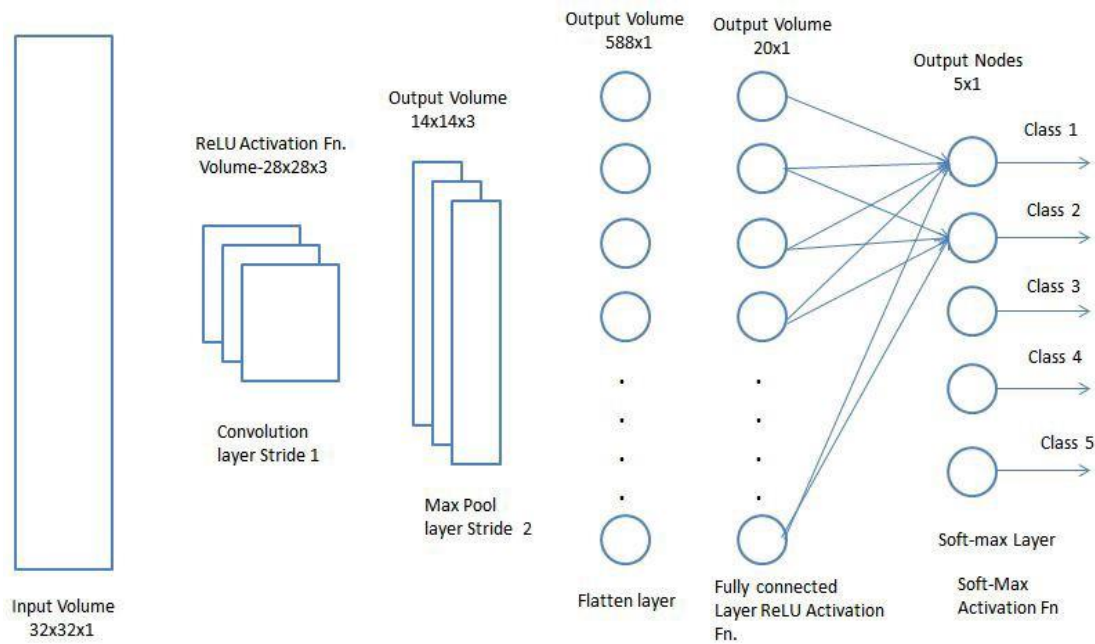
Responsible for feature selection and parameter reduction, pooling layers, also referred to as subsampling layers, execute a pooling function for every feature map patch. Typically, there are two kinds of pooling. The first pooling is the average pooling, computing the average value. Figure 2.9 shows the second process, maximum pooling, which calculating the largest value [20].



**Figure 2.9 Maximum Pooling**

### Fully Connected Layers

Similar to multilayer perceptrons, fully connected layers consist of numerous neurons in each layer. The feature maps from the final pooling layer are flattened into a vector for these layers. Various nonlinear activation functions like ReLU, tanh, and sigmoid can be utilized. Figure 2.10 show the fully connected layer [21].



**Figure 2.10 Fully Connected Layer**

### Output Layer

Serving as a fully connected layer, the output layer generates the final classification results by computing the input with a weight matrix, adding a bias vector, and then applying a softmax function [22]. Each prediction falls within the range  $[0,1]$ , computed as:

$$y_c = \frac{\exp(z_c)}{\sum_j \exp(z_j)}, \quad c=1, 2, \dots, C$$

where  $C$  represents the number of classes, and  $z_c$  denotes

the input of the  $c$ th output neuron corresponding to the  $c$ th class.

### **2.3 Optical Character Recognition (OCR)**

OCR involves extracting and converting handwritten or typed text from various sources like images, videos, or scanned documents into editable digital formats like txt or docx. This field within artificial intelligence is closely associated with computer vision and pattern recognition [23]. Integrating ML and AI has significantly enhanced OCR systems, elevating accuracy and performance levels to unprecedented heights [24]. These advancements enable systems to adapt to diverse layouts, fonts, and styles, boosting versatility and precision. Beyond research, OCR applications span numerous fields, from digitizing historical texts to automating license plate identification on vehicles. Its integration with cameras plays a crucial role in law enforcement, traffic monitoring, and parking enforcement, impacting various industries significantly.

#### **2.3.1 The Brief History of OCR Technology**

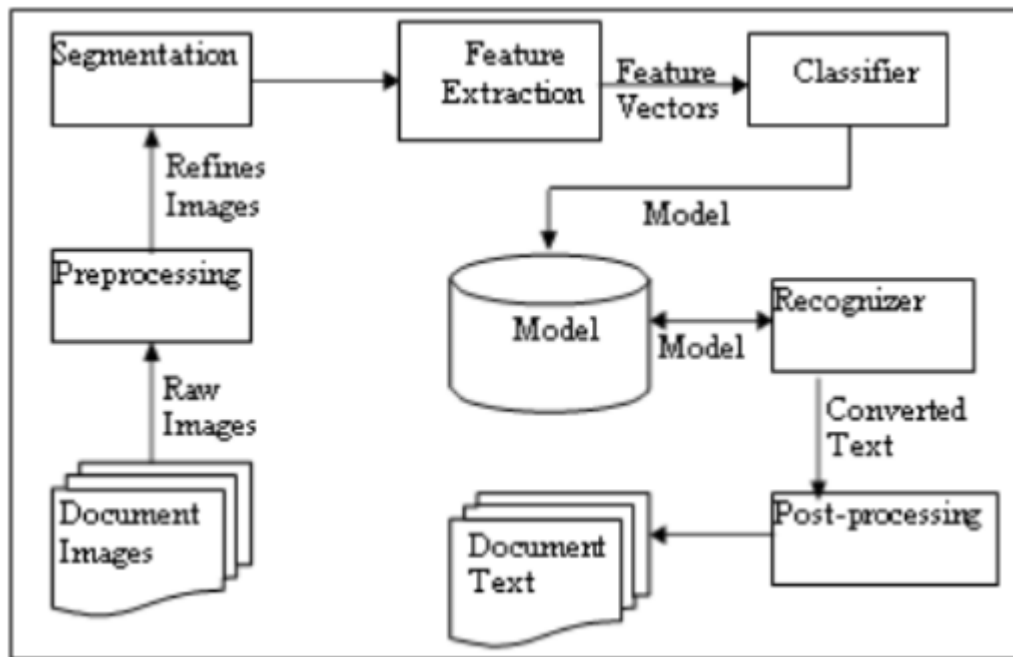
The table 2.3 displays the timeline outlining the brief history of OCR technology, encompassing various periods: early concepts (1920s-1930s), the advent of the first OCR machine (1950s), advancements in pattern recognition, ICR and MICR technologies (1960s-1970s), progress in digital OCR (1980s-1990s), emergence of commercial OCR software (1990s), the introduction of Tesseract OCR (2005), the revolution of deep learning (2010s), and the integration of OCR technology into everyday life (Present-day) [25].

**Table 2.3 Timeline of OCR Technology**

<b>Time Period</b>	<b>Summary</b>
1920s - 1930s	During this era, the foundations of OCR technology were laid. Emanuel Goldberg developed a machine for character recognition and document retrieval. Subsequently, Gustav Tauschek innovated with the Analog Reading Machine, advancing character recognition further.
1950s	The 1950s marked the emergence of the first official OCR machines. David Shepard and Harvey Cook Jr. introduced the GISMO, followed by the establishment of the first commercial OCR by Intelligent Machines Research Co. (IMR).
1960s - 1970s	MIT's research group focused on enhancing OCR's software capabilities, introducing the Hough Transform Technique for data capture. This period also witnessed the birth of ICR and MICR technologies, refining OCR for handwritten and magnetic ink characters.
1980s - 1990s	OCR shifted towards digital imaging, improving algorithms for diverse fonts, layouts, and multilingual support. Software advancements led to practical commercial systems, reducing reliance on hardware improvements.
1990s	The 1990s saw the widespread availability of commercial OCR software from companies like ABBYY, Adobe, and Nuance. These tools revolutionized document digitization and workflow efficiency.
2005	The open-source revival of Tesseract OCR by Google brought significant advancements, incorporating machine learning and demonstrating exceptional accuracy in text extraction from various sources.
2010s	The integration of neural networks, particularly CNNs and RNNs, revolutionized OCR technology, drastically increasing accuracy to nearly 99%. This era brought significant improvements in interpreting handwritten text and complex document structures.
Present-day	OCR technology has become integral to various industries, enabling automated document processing without bulky machines. Its applications span across real estate, finance, insurance, healthcare, and more, streamlining document digitization processes.

### 2.3.2 Steps Involve in OCR

Building an OCR engine involves several key steps that researchers commonly follow to ensure optimal character recognition. Refer to Figure 2.11 [26].



**Figure 2.11 Major Steps Involve in Character Recognition**

Optical scanning begins by capturing an image of the original document, ideally using a high-quality scanner with advanced sensing tools and a reliable transport mechanism. Pre-processing follows, where various operations like noise removal and character clarity enhancement are performed to prepare the image for segmentation. This includes converting the image to grayscale, binarization, thinning, skew correction, and normalization. The segmented image is then divided into subcomponents, starting with horizontally separating characters, then splitting words from sentences, and finally isolating individual characters for further processing. Feature extraction plays a key role by identifying distinguishing patterns from these characters, allowing the system to differentiate between them. For training and recognition, OCR pattern recognition methods such as template matching, statistical techniques, or neural networks are employed to ensure adaptability, especially when dealing with incomplete vocabulary. In the post-processing stage, tasks like grouping, error detection, and correction are carried out, with context-based error detection improving identification accuracy, even though achieving 100% accuracy remains challenging.

### 2.4 AIoT camera

ALPR cameras continually surveil specific areas such as roads or vehicle entrances. These cameras typically emit IR light, which reflects off license plates, aiding the traffic monitoring camera in locating and isolating them within the image. Utilizing OCR technology, ALPR cameras can decipher license plates in various weather conditions, eliminating the necessity for human involvement in the process. Moreover, ALPR cameras often gather multiple frames or images and autonomously determine the optimal ones for accurate license plate reading [27].

The integration of AI with IoT technologies has led to the development of innovative camera systems, particularly in the realm of LPR [28]. AIoT-based camera systems have evolved into two primary categories which include Fixed AIoT Camera Systems and Mobile AIoT Camera Systems.

Fixed AIoT Camera Systems leverage the synergy of AI and IoT technologies within a stationary setup. These systems are strategically installed at specific locations and are designed to remain stationary, providing precision in capturing license plates. Comprising advanced cameras and sophisticated software, they excel in real-time monitoring of large vehicle volumes, commonly employed in toll booths, parking lots, and other areas requiring continuous vehicle surveillance. Key considerations when selecting a fixed AIoT system involve computing capabilities, accuracy in diverse conditions, and scalability to link with multiple cameras. Essential features encompass robust CPUs, PoE support, and operational adaptability across varying temperature ranges.

In contrast, Mobile AIoT Camera Systems offer dynamic monitoring capabilities through mobile and flexible deployment. These systems typically incorporate cameras mounted on vehicles or handheld devices, granting mobility and adaptability. Law enforcement agencies leverage these systems for their versatility, enabling deployment across diverse locations and facilitating comprehensive coverage areas. With the ability to move and cover multiple sites, mobile AIoT systems excel in patrol applications. Crucial considerations in selecting these systems include computing capacity, accuracy in challenging environmental conditions, and the capacity to interface with multiple

## CHAPTER 2

cameras. Features such as wide temperature operational range, durable construction, and flexible installation options are pivotal in their utility.

Overall, AIoT-based camera systems, whether fixed or mobile, represent a significant technological advancement in License Plate Recognition. The integration of AI and IoT technologies enhances precision, scalability, and adaptability in monitoring and recognizing license plates, catering to diverse operational needs across surveillance, enforcement, and security applications. The selection of these systems necessitates careful consideration of their technological capabilities, environmental adaptability, and operational requirements, offering tailored solutions for efficient and effective license plate recognition.



## 2.5 Critical Remarks of Previous Works

**Table 2.4 Strength and Weakness of Previous Works**

Aspect	Strengths of Previous Works	Weaknesses of Previous Works	Proposed Solution
<b>Real-Time Processing</b>	Efficient real-time license plate recognition for immediate vehicle identification.	Sensitive to environmental changes like poor lighting or shadows, impacting real-time accuracy.	Enhances real-time processing with AIoT and OCR technologies, improving performance under various conditions.
<b>Accuracy</b>	High accuracy in controlled environments, using advanced OCR techniques for reliable character recognition.	Accuracy drops in adverse conditions such as bad weather, different lighting angles.	Uses adaptive algorithms to maintain high accuracy even in challenging environmental conditions.
<b>Scalability</b>	Capable of handling large volumes of vehicles, suitable for urban settings.	Small coverage area of camera increases costs and make scalability challenging.	Utilizes cost-effective components to reduce setup and maintenance costs, improving scalability.
<b>Implementation Complexity</b>	Established infrastructure and advanced algorithms provide robust performance.	High initial setup costs and complex maintenance due to need for specialized equipment and technical support.	Streamlined implementation with simpler hardware and efficient software, reducing complexity and costs.
<b>Adaptability to License Plate Formats</b>	Can handle standard license plate formats efficiently.	Limited adaptability to diverse license plate designs and formats, reducing effectiveness in different regions.	Incorporates machine learning models adaptable to various license plate formats, ensuring broader applicability.

## Chapter 3

### System Methodology/Approach

#### 3.1 System Design Diagram

This chapter outlines the design of the Smart Parking Management System by presenting its architecture, functionality, and user interactions through a series of visual models. These models include the system architecture diagram, use case diagram, and activity diagram, each serving to illustrate specific aspects of the system's structure and behavior.

##### 3.1.1 System Architecture Diagram

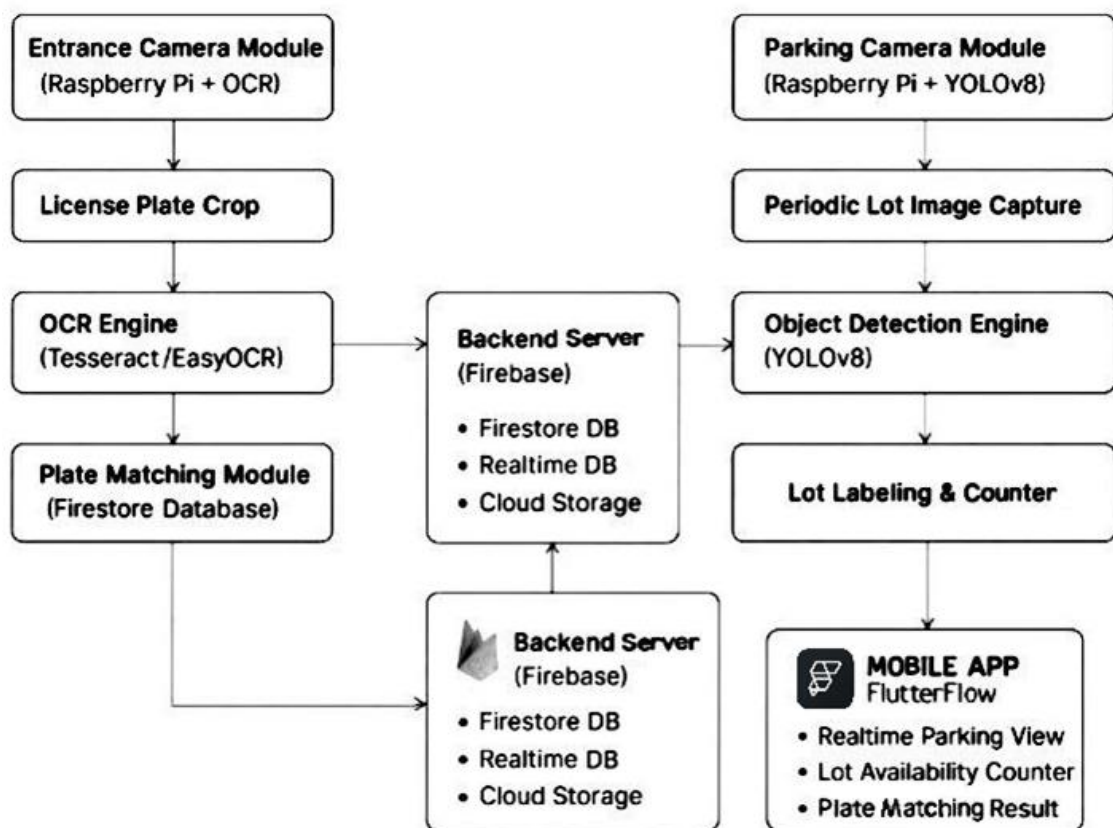


Figure 3.1 System Architecture Diagram

### **Entrance Camera Module**

The Entrance Camera plays a pivotal role in the conceptual design of the vehicle recognition process, as it is intended to capture the license plate of each vehicle entering the parking area. Ideally, the camera would be strategically positioned at the parking lot entrance to ensure it captures all incoming vehicles, with a focus on obtaining clear, high-resolution images of license plates. However, at this stage of the project, this feature remains unimplemented due to hardware constraints, specifically the limited availability of Pi cameras, and permission-related issues that restrict physical installation and testing. As an alternative, the system concept was validated using static image input to simulate the license plate detection and recognition process.

After an image of a vehicle is obtained, the system proceeds to extract the license plate number using Optical Character Recognition (OCR). OCR algorithms, such as EasyOCR, are employed to convert the text on the license plate from an image into machine-readable characters. This extracted text, which serves as the vehicle's license plate number, becomes a key identifier for verification. The recognized number can be cross-checked against a predefined database of authorized vehicles. If a match is found, the vehicle is marked as "authorized"; otherwise, it is classified as "unauthorized." Both the recognition result and the authorization status are communicated to the backend system for record-keeping and possible follow-up actions.

Although the real-time hardware deployment could not be realized in this stage, the entire detection and recognition process was effectively demonstrated using image-based input in a cloud environment. This simulation confirms the feasibility and robustness of the proposed approach for automated vehicle verification and access control in a smart parking system.

### **Parking Camera Module**

The Parking Camera uses a Raspberry Pi Camera 3 to monitor the parking lot, capturing images every 30 seconds to track the availability of parking spaces. The camera provides a real-time visual representation of the parking lot, focusing on the parking spaces where vehicles can park. Unlike the Entrance Camera, which deals with license plate recognition, the Parking Camera focuses on the visual detection of vehicles within predefined parking spaces.

Once an image is captured, the system applies boundary boxes to highlight each individual parking space in the lot. These boundary boxes are defined in advance and are used to demarcate the physical parking spaces, allowing the system to track occupancy in each area. The next step involves detecting the presence of vehicles using object detection techniques like YOLOv8, a state-of-the-art deep learning model for real-time object detection. YOLOv8 identifies objects in the image, but if a car is detected in a specific parking space, the system classifies that space as occupied.

Upon detecting a vehicle, the system marks the parking space as occupied by changing its visual representation, typically turning the parking space red. This change in colour visually indicates to users that the parking space is no longer available. Additionally, the system labels each parking space with an identifier, such as 1, 2, 3, and so on, which makes it easier to reference specific parking spaces in both the mobile app and backend system. These labelled parking spaces are then stored in the backend along with their current status (whether they are occupied or available). The processed image, which includes the boundary boxes, detected vehicles, and parking space labels, is also sent to the backend for further use.

### **Backend Server**

The backend server is the central hub of the system, responsible for storing and managing the data generated by the Entrance Camera and Parking Camera, and for serving that data to the mobile app. Firebase, a powerful and scalable cloud-based platform, is used as the backend in this system due to its robust real-time database capabilities, authentication services, and ease of integration with mobile applications.

First, the backend should be able to store the results from the OCR process carried out by the Entrance Camera in the future. Alternatively, a list of the vehicle's license plate numbers is predefined as authorized, allowing the app to check authorization by matching with the data in the database. This data is critical because it helps the system control access to the parking lot. For example, security or admin can view unauthorized vehicle entry records, and further action can be taken immediately to ensure campus safety.

Secondly, the backend server stores the parking status of each parking space, which includes whether a parking space is occupied or available. The status of each space is updated in real-time as the Parking Camera detects vehicles entering or leaving the parking lot. These updates are stored in the Firebase database, where they can be retrieved by the mobile app to provide users with real-time parking availability.

Additionally, the backend stores the processed images sent by the Parking Camera, which show the parking lot's current state with boundary boxes, vehicle detection, and space occupancy marked. These images are not only used for reference but can also be sent to the mobile app, allowing users to visually confirm the current status of the parking lot. The backend ensures that all this data is readily accessible to the mobile app in a timely manner, providing a seamless user experience.

### **Mobile Application**

The mobile application is the front-facing component of the system. It is designed to allow users to view real-time information about the parking lot, including the availability of parking spaces and the authorization status of vehicles. The app is built using FlutterFlow, which is a low or no-code platform that allows for rapid development of mobile apps with an intuitive user interface.

The app continuously communicates with the backend to display the current status of the parking lot. Users can view a real-time visual representation of the parking lot, which shows each parking space's status. Available parking spaces are typically marked in green, while occupied spaces are highlighted in red, making it easy for users to identify open spaces. Additionally, the app provides a count of empty spaces for each zone or block's carpark, allowing users to quickly find an available parking space based on their location.

The app also displays the authorization result. When a car plate number is recorded, it will be compared with the predefined data in the database. Once it matches the plate number in the database, it will show as 'authorized'; otherwise, it will display as 'unauthorized.' This authorization status can be viewed in the Entry Record tab.

Furthermore, the app sends notifications to users when the parking lot is nearing full capacity or when certain sections of the lot are completely occupied. These notifications help users avoid unnecessary trips to a full parking lot, making the parking experience more efficient. The mobile app is, therefore, the interface through which users interact with the system in real-time, making it an essential component of the overall parking management solution.

The system is designed for efficient and seamless data flow between its components. The Entrance Camera captures the vehicle license plate, extracts it using OCR, and matches it with the authorized vehicle database. The Parking Camera continuously captures images of the parking lot, detects vehicle presence, and updates the occupancy status of each parking space. Both cameras send their respective data (OCR results for

the entrance camera and parking space status for the parking camera) to the backend server.

The backend server stores all the data and serves it to the mobile app in real-time. The app then provides a user-friendly interface that displays the parking lot's current status, vehicle authorization information, and notifications about parking availability. This setup ensures that the system is dynamic, providing up-to-date information at all times and making it easy for users to navigate the parking lot efficiently.

This system architecture is designed to be highly efficient, with real-time data processing and seamless integration across the components. Each component plays a critical role in ensuring that the parking lot operates smoothly, from vehicle entry authorization to monitoring parking space occupancy. The use of Raspberry Pi cameras, OCR technology, and object detection models ensures accurate data capture, while the backend server ensures this data is stored and served efficiently to the mobile app. The mobile app, in turn, provides users with an intuitive interface for real-time parking management, making the entire system both practical and user-friendly.

## 3.1.2 Use Case Diagram and Description

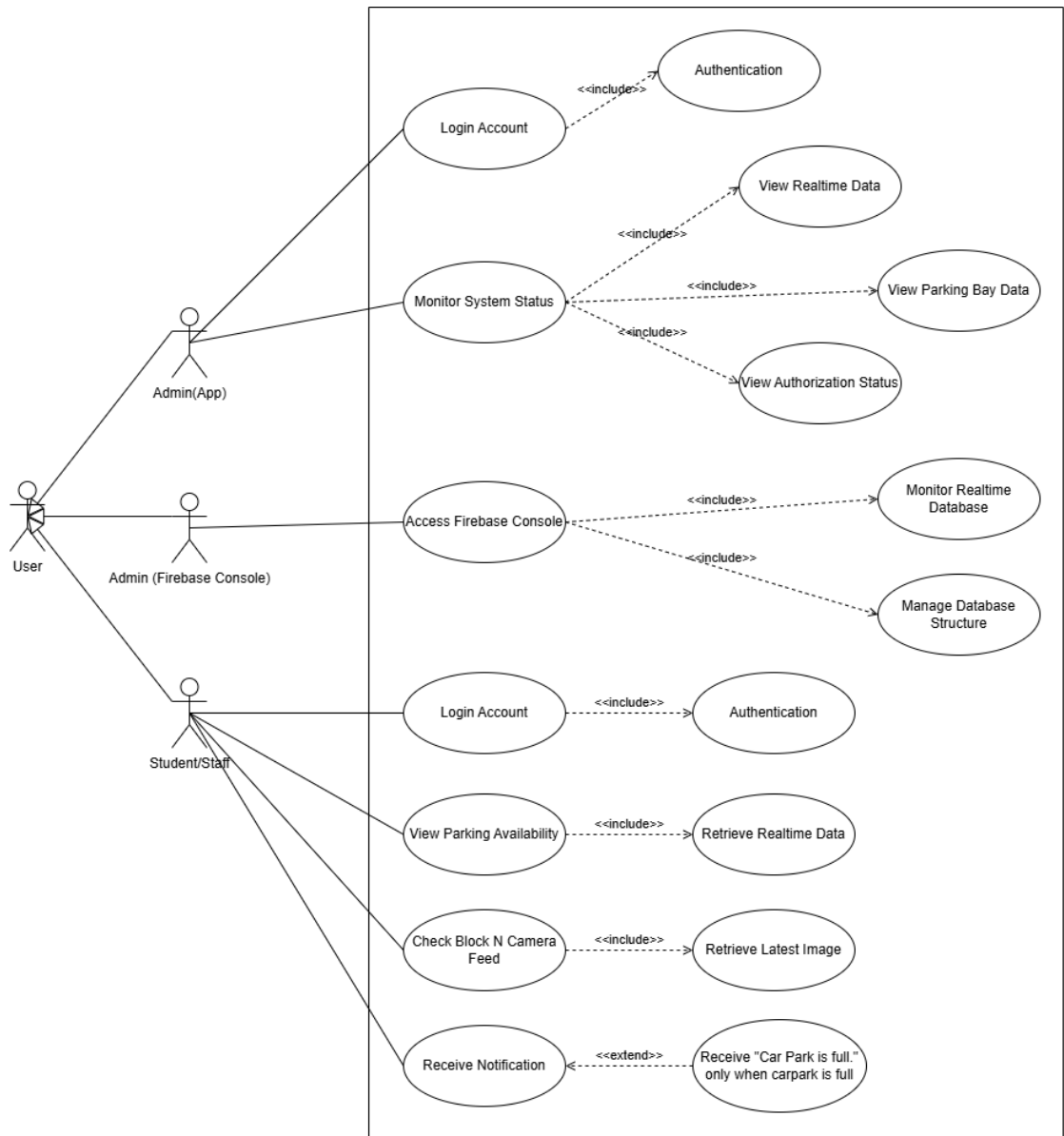


Figure 3.2 Use Case Diagram



## CHAPTER 3

The system involves three key user roles: Admin using the mobile application, Admin accessing the Firebase Console, and Student or Staff users. Each role interacts with the system through specific features based on their permissions and responsibilities.

For the Admin who accesses the system through the mobile application, the process begins with logging in to the app. This login process includes an authentication step, ensuring that only authorized personnel can gain access. Once authenticated, the admin can monitor the status of the system through the app interface. This monitoring function allows the admin to view real-time data on parking activities, access detailed information about individual parking bays, and check the authorization status of vehicles, such as whether they are registered or allowed to park. These features provide essential insights for maintaining order in the parking system and ensuring its smooth operation.

The second type of admin interacts with the system through the Firebase Console, focusing more on backend management tasks. This admin can access the Firebase platform to monitor the real-time database that stores all relevant parking data. In addition, they are responsible for managing the structure of the database. This includes adding, modifying, or organizing data collections to support consistent and accurate system behavior. These responsibilities are crucial in ensuring that the backend of the parking system functions reliably and can scale as needed.

Student and Staff users primarily use the mobile application to access services related to parking. When they open the app, they are required to log in using their credentials. This action includes an authentication step to verify their identity. Upon successful login, they can view the current availability of parking spaces, which is made possible by retrieving real-time data from the database. Additionally, users have the option to check the live camera feed of Block N, enabling them to observe the parking condition before making decisions. The system also includes a notification feature designed to inform users when the car park is full. This notification is only triggered if the system detects that all spaces are occupied, ensuring users receive timely and relevant alerts. Overall, this smart parking system enhances convenience for users while enabling efficient monitoring and management for administrators.

## 3.1.3 Activity Diagram

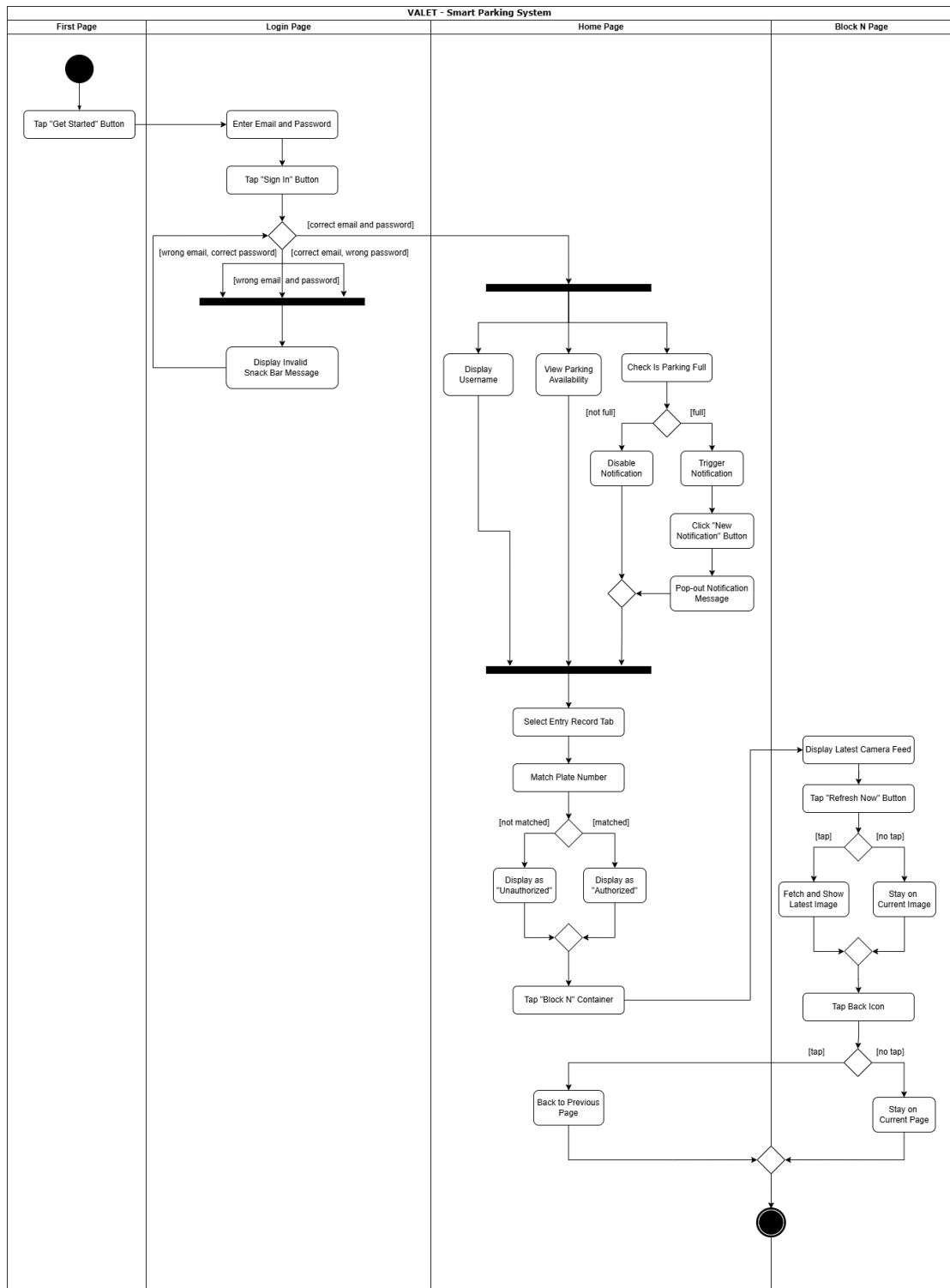


Figure 3.3 Activity Diagram

## CHAPTER 3

The activity begins with the first page of the mobile application, which acts as the welcome screen. When the application is launched, the user is presented with a "Get Started" button. This action serves as the initial point of engagement and directs the user to the login interface. It provides a smooth transition from the introductory screen to the core functionality of the application.

Upon reaching the login page, the user is prompted to enter their email and password credentials. After filling in the required fields, the user proceeds by tapping the "Sign In" button. At this stage, the system performs a validation check to ensure that both the email and password are correct. If the email is invalid or the password does not match, or if both entries are incorrect, the system displays a SnackBar message "Invalid email/password". The user remains on the login page and is encouraged to re-enter valid credentials. Only when both the email and password are confirmed to be valid does the system allow the user to proceed to the next interface.

Once the login is successful, the user is directed to the home page. This serves as the central dashboard where essential functions are made available. On this page, the system greets the user by displaying their name. One of the key features on the home page is the ability to view real-time parking availability. The system updates this information to reflect the current status of parking bays across zones, helping users to quickly identify available spaces.

In the background, as the user accesses the home page, the system automatically checks whether the specific parking area, particularly within Block N, is full. If the parking area is detected to be full, the system triggers a notification to inform the user. Upon tapping the notification, a pop-up message appears to alert the user that no parking bays are currently available. If the parking area is not full, the system disables this notification to avoid sending unnecessary alerts. This real-time feedback mechanism ensures that users are kept informed about parking conditions as soon as they access the application.

Additionally, the home page includes a tab for viewing entry records. When the user selects this tab, the system compares license plate numbers detected with records stored in the authorization database. If a match is found, the vehicle is labelled as authorized. If there is no match, the system displays the vehicle as unauthorized. This functionality supports effective monitoring of vehicle entry and enhances overall security within the parking area.

The home page also provides an option to view the camera feed for Block N. When the user taps on Block N, they are directed to Block N page that displays the latest image captured by the installed camera. This page offers a visual representation of the current state of the parking zone. Users have the option to tap the "Refresh Now" button to force reload and display the most recent image from the camera feed. If the user chooses not to refresh, the existing image remains on display, allowing them to observe without initiating any new requests.

After viewing the camera feed, the user may choose to return to the home page by tapping the "Back" button. If the user decides not to return immediately, they can continue viewing the camera feed. Eventually, when the user has completed their interaction with the system, they may choose to log out or exit the application, marking the end of the activity flow.

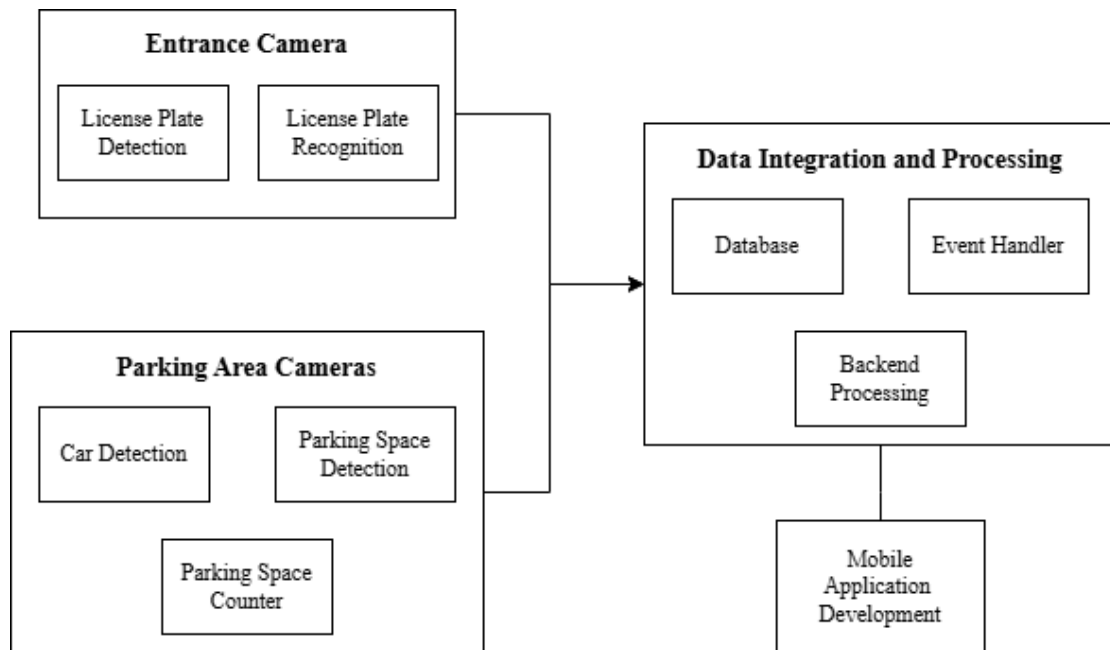
This activity sequence reflects a structured and user-centered interaction design. It ensures that users are guided clearly from the initial launch to authentication, real-time parking updates, live monitoring, and system exit. Each step is supported by automated backend processes that maintain data accuracy and deliver timely notifications, contributing to a responsive and functional smart parking management system.

## Chapter 4

### System Design

#### 4.1 System Block Diagram

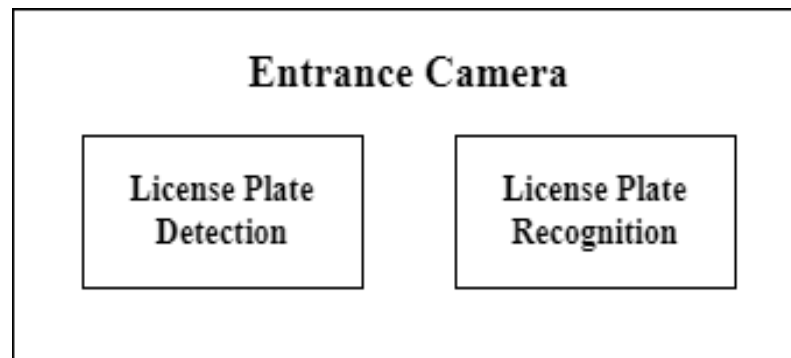
This section provides an overview of the smart parking system by presenting a block diagram that outlines the main components and their interactions. The diagram illustrates how various modules, including cameras, cloud services, and the mobile application, work together to deliver real-time parking updates and license plate recognition. Each part of the system is responsible for a specific function, and together they support accurate data processing, efficient information flow, and timely updates for users. This block diagram serves as a reference to better understand the role and connection of each system element, which will be further explained in the upcoming component specifications.



**Figure 4.1 Block Diagram of System Overview**

## 4.2 System Components Specifications

### 4.2.1 Entrance Camera Modules



**Figure 4.2 Component for Entrance Camera Modules**

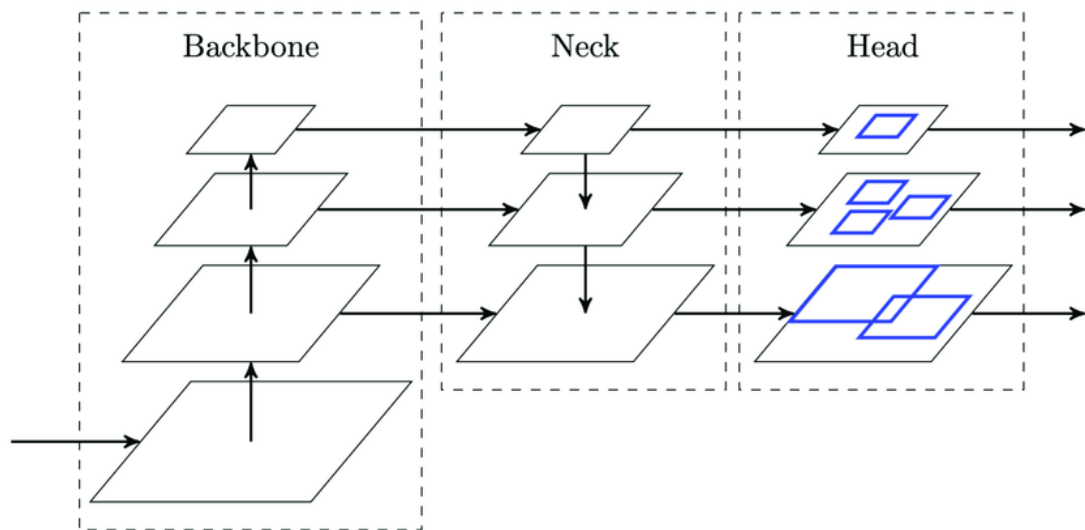
The entrance camera modules were the initial stages of the parking management process, focusing on car identification and tracking as they enter the parking facility. These modules ensure that the system begins monitoring car as soon as they arrive.

#### **License Plate Detection**

This module is responsible for detecting the presence of a car's license plate as it approaches the parking entrance. It employs image processing techniques to isolate the Region of Interest (ROI) containing the license plate from the rest of the vehicle. By using high-resolution cameras and advanced image processing algorithms, the system accurately detects license plates under various lighting conditions and angles. This detection module is fundamental for the subsequent recognition step, ensuring that only the relevant area is processed further. For implementing the license plate detection algorithms effectively, resources from YouTube were referenced for guidance [28]. The Malaysia License Plate dataset, consisting of 1,799 images sourced from Roboflow [29], was utilized for this project.

The deep learning framework employed in this system is based on the YOLOv8 object detection model. YOLOv8 is a fully convolutional neural network (CNN) that processes input images by dividing them into a grid and predicting bounding boxes for objects within each grid cell. The model utilizes convolutional layers to extract important visual features such as edges, textures, and patterns, which are essential for detecting license plates in diverse environments.

Figure 4.3 provides a simplified overview of the CNN architecture in YOLOv8, highlighting its core components: the backbone, neck, and head, which are used for feature extraction, multi-scale detection, and object localization [30].



**Figure 4.3 Simplified Overview of the CNN Architecture in YOLOv8**

The backbone is responsible for feature extraction, using a series of convolutional layers to process the input image and create feature maps. These feature maps are then enhanced by the neck, which fuses low-level (detailed) and high-level (abstract) features to handle objects at different scales. Finally, the head of the network predicts bounding boxes and confidence scores for detected objects, isolating the license plate from other areas of the image. By leveraging CNN architectures like YOLOv8, the system efficiently detects license plates with high accuracy, even in complex scenarios involving variations in lighting, angles, and image quality [31].

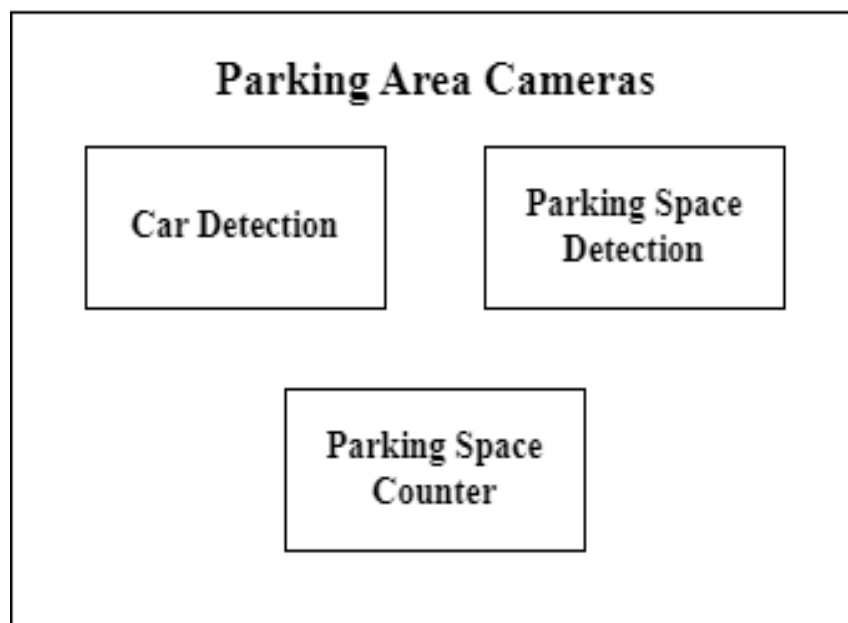
### License Plate Recognition

Once the license plate is successfully detected, the system employs OCR to extract the alphanumeric characters from the license plate [32]. This recognition process converts the visual data into a digital text format, which can then be stored and further processed. This step is important for maintaining a precise log of all cars entering the parking area, which facilitates efficient tracking, authorization checking, and enhanced security measures. The extracted license plate numbers are stored in a centralized database, associating each vehicle with a unique identifier. This approach streamlines tracking, monitoring, and management within the parking facility.

The architecture for license plate recognition typically involves several key components. Initially, the detected license plate image is pre-processed to enhance character visibility and improve recognition accuracy. This preprocessing might include techniques such as noise reduction, contrast adjustment, and image binarization. Following preprocessing, the OCR engine analyses the processed image to recognize and extract the alphanumeric characters. Modern OCR systems use deep learning models, such as CNNs or Transformer-based architectures, to decode text from images. These models are trained on large datasets of license plate images to accurately identify characters even in varying conditions. The OCR output is then post-processed to correct any recognition errors and ensure accurate transcription of the license plate number. This often involves techniques like character segmentation, error correction algorithms, and validation against known license plate formats.

Finally, the recognized license plate number is stored in the centralized database, where it can be linked to vehicle entry logs and used for various applications, such as access record, authorization checking and security monitoring.

### 4.2.2 Parking Area Camera Modules



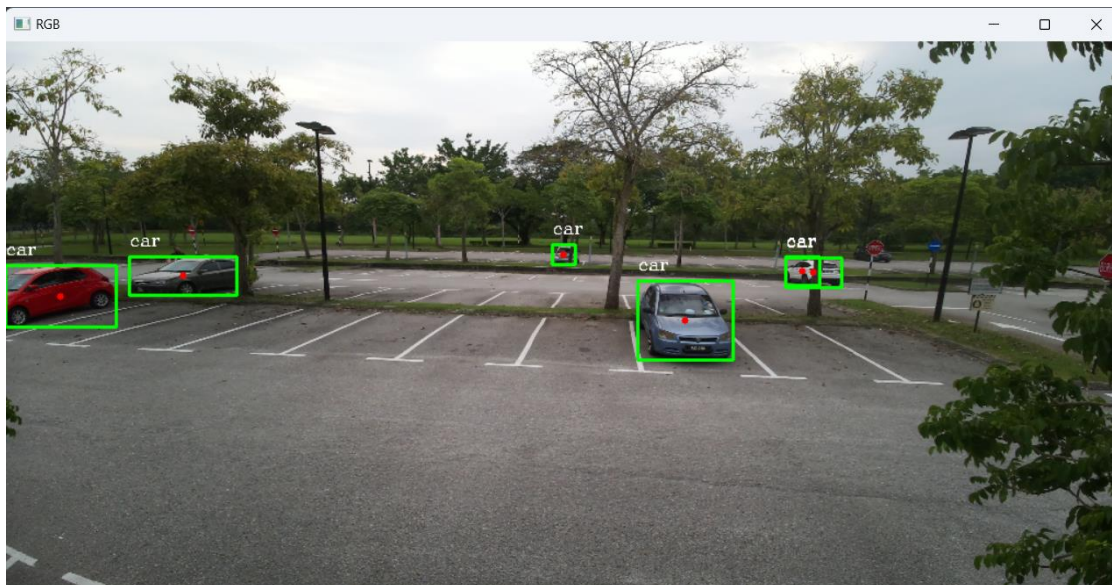
**Figure 4.4 Component for Parking Area Camera Modules**



The parking area camera modules are deployed within the parking lot to monitor the status of parking spaces and the movement of cars. These modules are essential for maintaining real-time visibility of parking occupancy.

### Car Detection

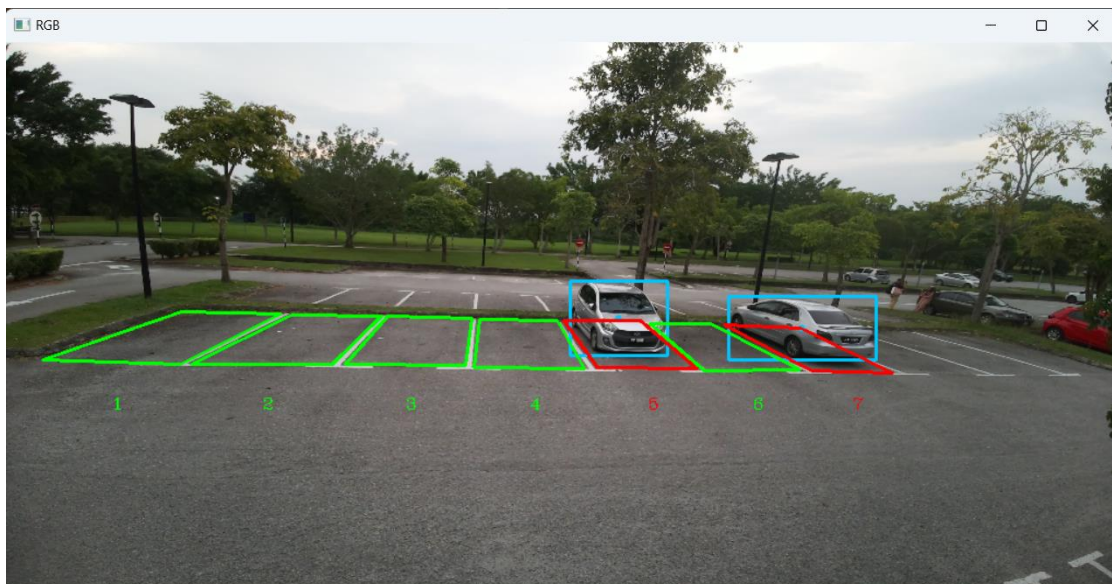
This module is designed to continuously monitor the parking area for the presence of cars. Utilizing YOLO v8, an object detection algorithm, the module can identify and track cars in real time. The high-speed and state-of-the-art accuracy of YOLO v8 allow for efficient monitoring even in busy parking lots with frequent car movements [33]. The system detects cars as they enter, exit, or move within the parking area, ensuring an up-to-date count of cars and their positions. To achieve an effectively functioning parking area module, the first step is to detect the object using YOLO v8. In this step, the primary goal is to detect cars visible within the camera's view. Figure 4.5 illustrates the cars detected in one of the camera views, with the detected cars marked by a green bounding box, a red center circle point, and a white label reading 'car'.



**Figure 4.5 Car Detection from One of the Camera Views**

### Parking Space Detection and Labelling

To prevent overlapping detection of cars in different camera views, parking space detection and labelling are necessary to assign responsibility to specific cameras for monitoring individual spaces and managing the counter for each. This module focuses on determining the occupancy status of individual parking spaces. By using predefined boundaries based on the coordinates of the four points of each parking lot, the system defines the specific area for each parking space. Labels are then assigned to each parking spot, allowing the system to analyse camera feeds and prevent overlapping detection or counting of the same space. By comparing the current state of each space against these boundaries, the module provides real-time status updates for every parking spot.

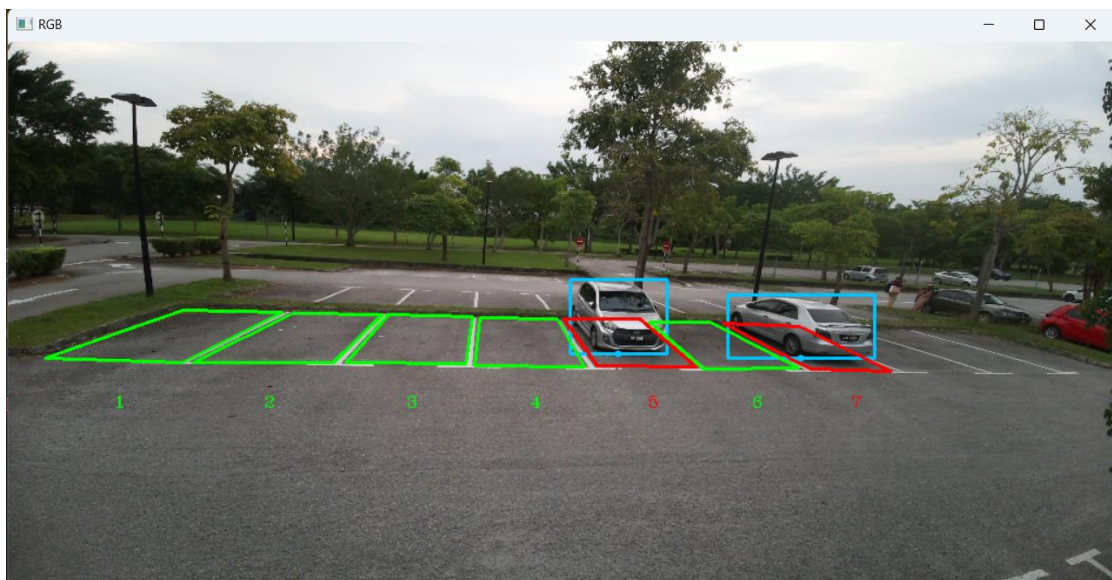


**Figure 4.6 Car Detection in Labelled Bounding Space**

As shown in Figure 4.6, if a car is detected inside a defined area, the boundary and label of the space will change from green to red, and only the car detected within the defined area will be shown with a bounding box. To have clear visualisation, the bounding box and center dot point was changed to be view in blue colour from the previous green bounding box and red circle point.

### Parking Space Counter

Before moving on to the counter module, it is essential to ensure that the car detection within the labelled bounding space module is functioning correctly. Initially, the method for detecting whether a car was inside the defined space relied on the center circle point of the car, as shown in Figure 4.6. However, it was observed that the center circle point of cars in spaces no. 5 and no. 7 was very close to the bounding box. This approach proved inadequate for accurate detection, as similar angles might lead to inaccurate results if the center circle point fell outside the defined area. In such cases, the space would be incorrectly considered unoccupied, leading to inaccuracies in the parking counter. To address this issue, the center circle point used for detection was adjusted to the bottom middle of the car's bounding box, as shown in Figure 4.7. This adjustment aims to ensure more reliable parking counter readings.



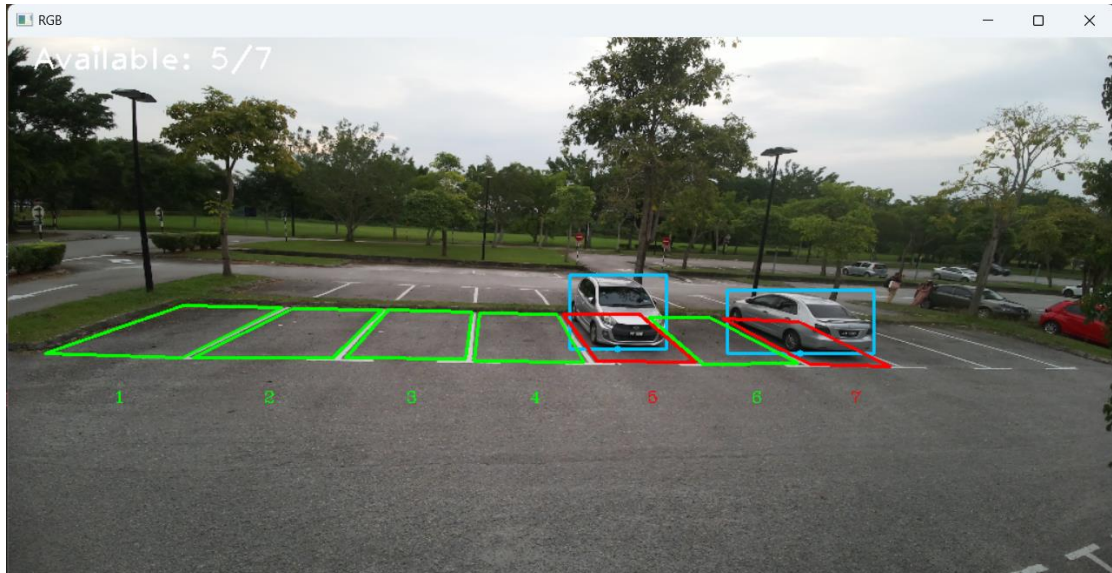
**Figure 4.7 Adjustment on Location of Circle Point**

Observations indicate that adjusting the detection to the bottom middle of the car's bounding box improves accuracy by ensuring the detection point of parked cars remains within the bounding box from this angle and reduces errors from similar viewing angles. Given the inability to install the camera at an ideal height and the limited resources available around the car park, such as a lack of walls or light posts, this is the best angle achievable at present. Consequently, this adjustment is particularly effective for enhancing detection precision from the current camera angle.



## CHAPTER 4

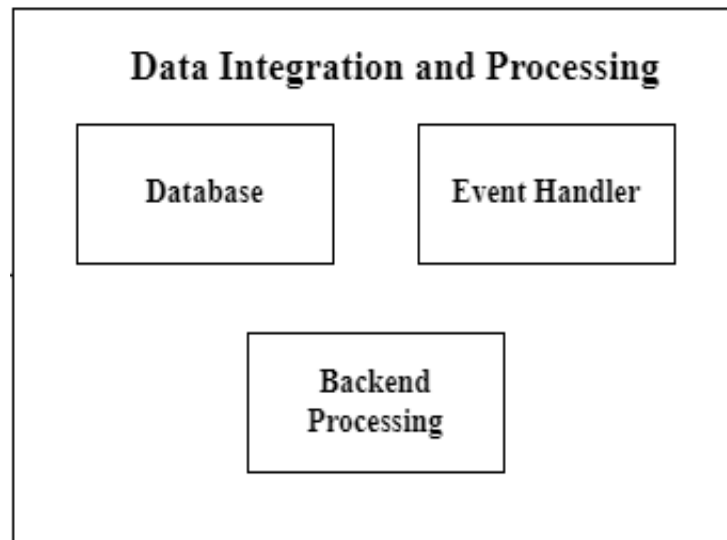
Moving on to the implementation of the available parking counter based on the labelled spaces in this view, as shown in Figure 4.8. The available spaces are displayed at the top left side of this view.



**Figure 4.8 Integrate All Previous Modules and Display Parking Counter**

Integrating data from the car detection and parking space detection and labelling modules, the available parking space counter maintains an accurate tally of the number of occupied and available parking spaces. This module dynamically updates the count as cars enter and leave the designated parking spaces monitored by each camera view. By continuously reflecting the current parking situation, the system ensures efficient management and enhances user satisfaction by providing up-to-date information on parking space availability. This data is crucial for informing drivers about available spaces and guiding them to the nearest open spot without wasting extra time.

### 4.2.3 Data Integration and Processing



**Figure 4.9 Component for Entrance Camera Modules**

Data integration and processing are central to ensuring that all the information collected by the entrance and parking area cameras is utilized effectively. This part of the system aggregates data from various modules, analyses it, and provides actionable insights.

#### **Database**

A centralized database serves as the repository for all data collected by the system. This includes license plate numbers, car detection logs, parking space statuses, and timestamps. The database enables efficient data storage, retrieval, and management, providing a historical record of all parking activities. It supports various functions such as auditing, reporting, and analysis, which are essential for operational efficiency and decision-making. Additionally, it assists security personnel by simplifying the identification of unauthorized vehicles. Rather than checking each car sticker manually, security staff can swiftly confirm parking authorization through the database records, easily locating any unauthorized vehicles.

#### **Event Handler**

The event handler module monitors the system for predefined events, such as a car entering the parking lot, a change in the status of a parking space, or a detected full carpark. Upon detecting an event, it triggers specific actions, such as updating the database, sending notifications to users, or alerting security personnel. The event

handler ensures that the system remains responsive and that all relevant parties are informed of significant events in real time.

### **Backend Processing**

Backend processing involves the analysis and correlation of data to provide meaningful insights. This module ensures that the data from different sources, such as license plate recognition, parking space detection and labelling, are accurately linked. It handles tasks like cross-referencing license plate numbers with parking space locations, checking for unauthorized vehicles, and optimizing parking space allocation. Backend processing is crucial for maintaining the integrity and accuracy of the system's data.

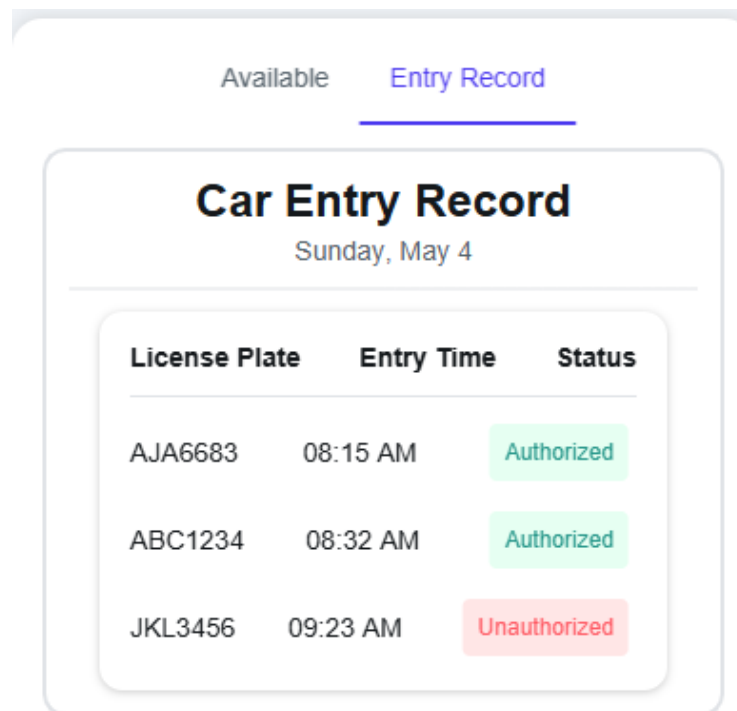
#### **4.2.4 Mobile Application Development**

The mobile application displays real-time data on parking space availability, vehicle locations, and license plate information. For drivers, the app provides a convenient way to monitor available empty spaces in specific block, zone, or even in certain camera view. the view the available parking lot number, locate their parked vehicles, and receive notifications about full parking status. For administrators, the dashboard offers tools for monitoring the entire parking facility, managing space allocation, and handling exceptions such as unauthorized vehicles or overstay. The application is designed to be user-friendly, ensuring that all users can easily access the information they need. It plays a vital role in enhancing the user experience, optimizing parking operations, and improving overall satisfaction.

### 4.3 Components Design in Application

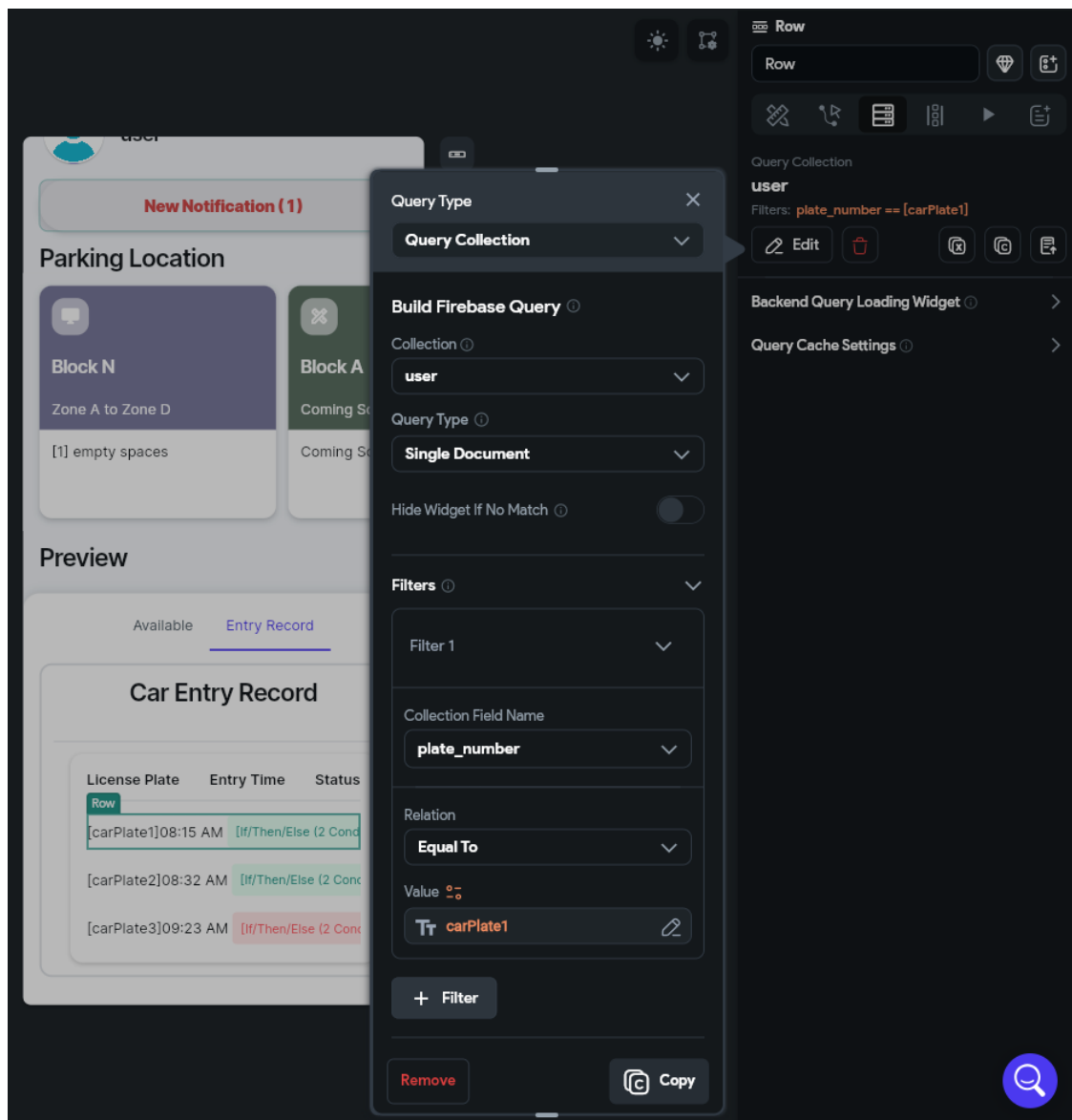
#### Entrance Camera Modules

Due to the lack of sufficient cameras, permission to record incoming car plate numbers, and access to the list of authorized vehicles, the OCR function is not currently integrated into the app. However, it has been developed and is ready for use once these issues are resolved. At the concept level, the purpose of the OCR is to verify car authorization. Although it remains a standalone function for now, a tab has been included in the app to check entry records, allowing the recognized plate number from the Pi camera to be compared with the database to determine authorization. This establishes a functional connection between the OCR and the app.



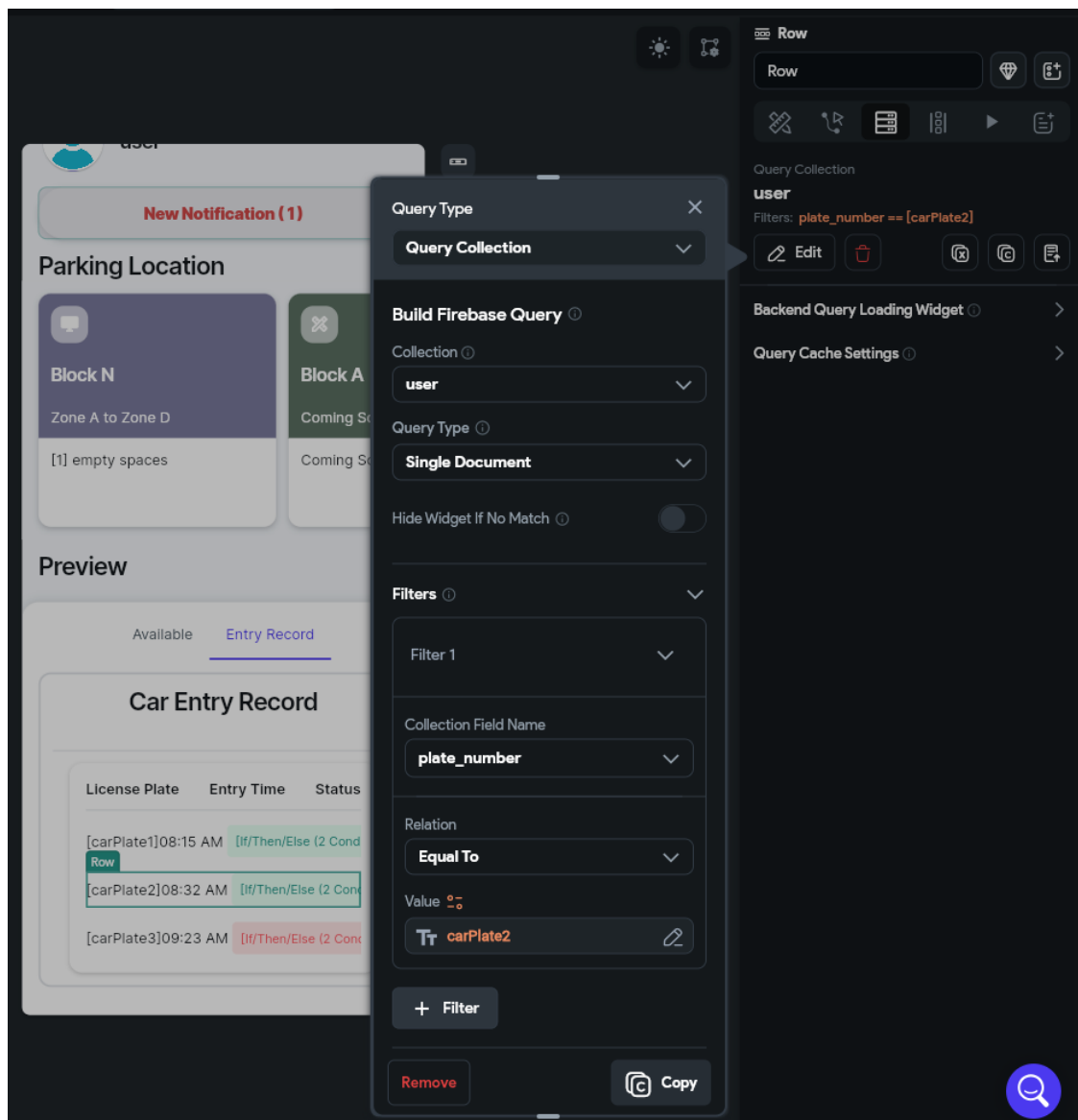
**Figure 4.10 Entry Record Tab**

Figure 4.10 illustrates the Entry Record tab in the application, which is conceptually designed to log and verify car plate numbers captured by the Pi camera using OCR. In the current development stage, a set of pre-listed plate numbers: carPlate1, carPlate2, and carPlate3, are used as input to simulate OCR results. These values are compared against the plate\_number field stored in the database for each user record, as shown in Figures 4.11 to 4.13.

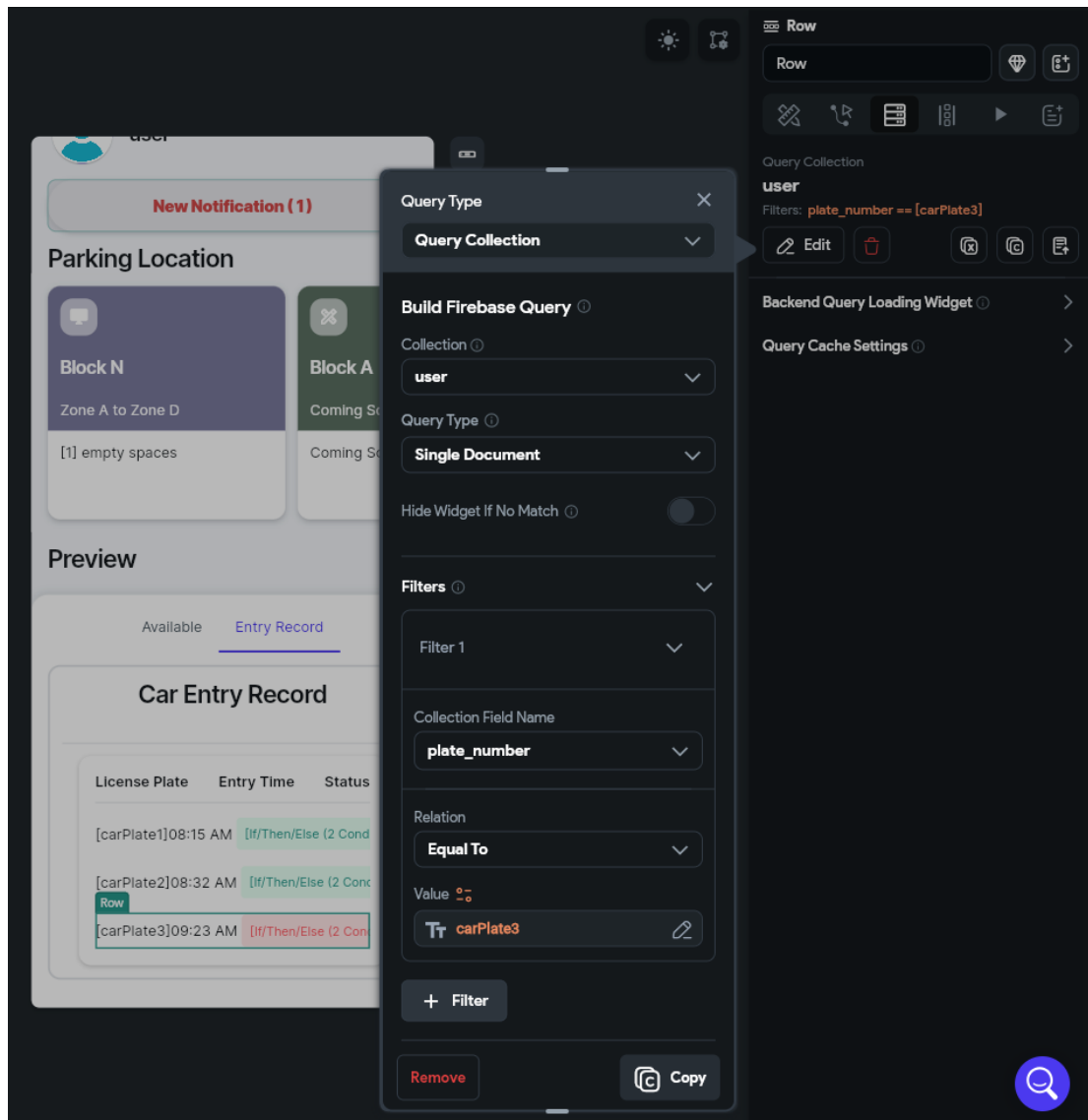


**Figure 4.11 Comparison of plate\_number with carPlate1**



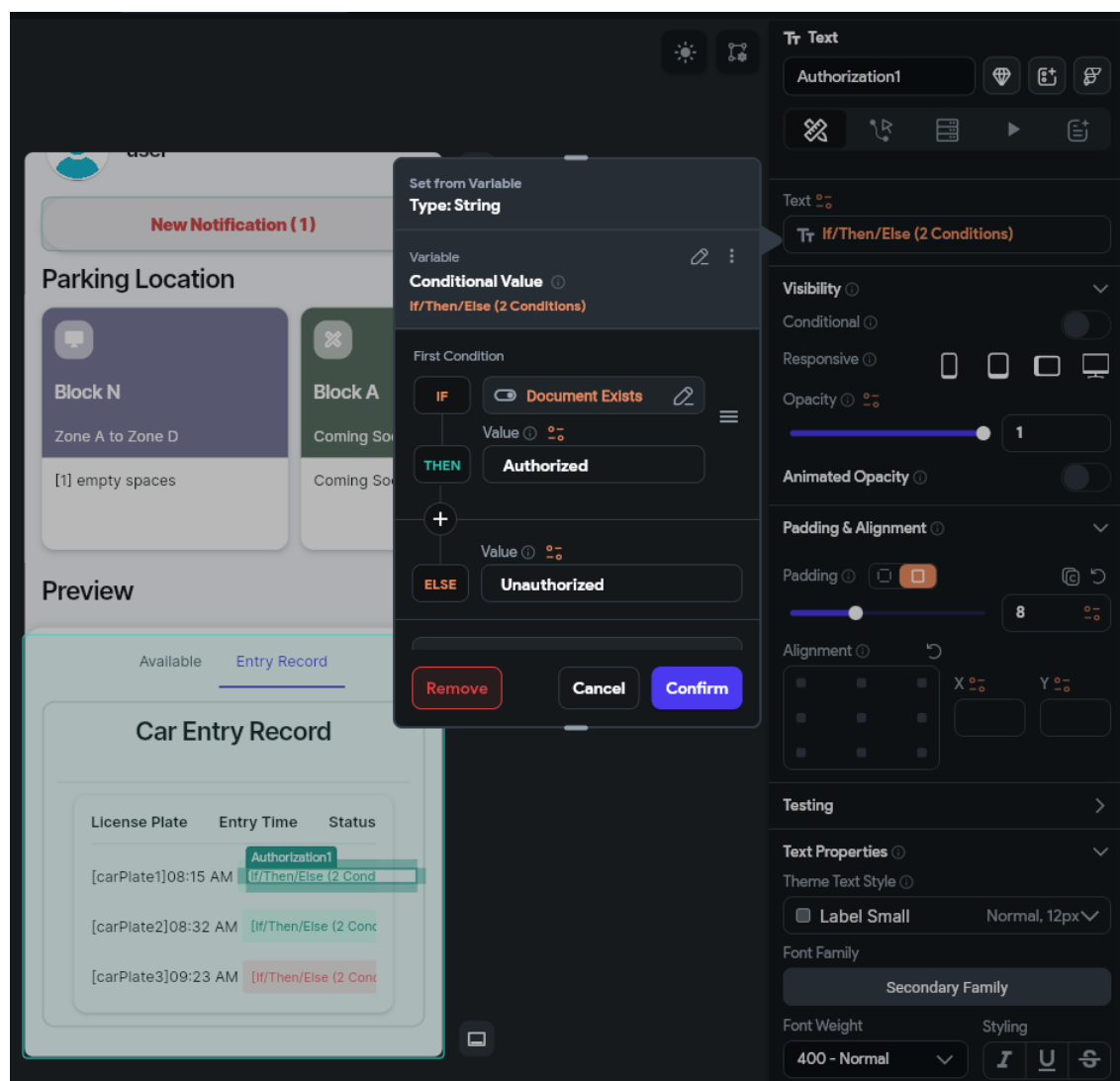


**Figure 4.12 Comparison of plate\_number with carPlate2**

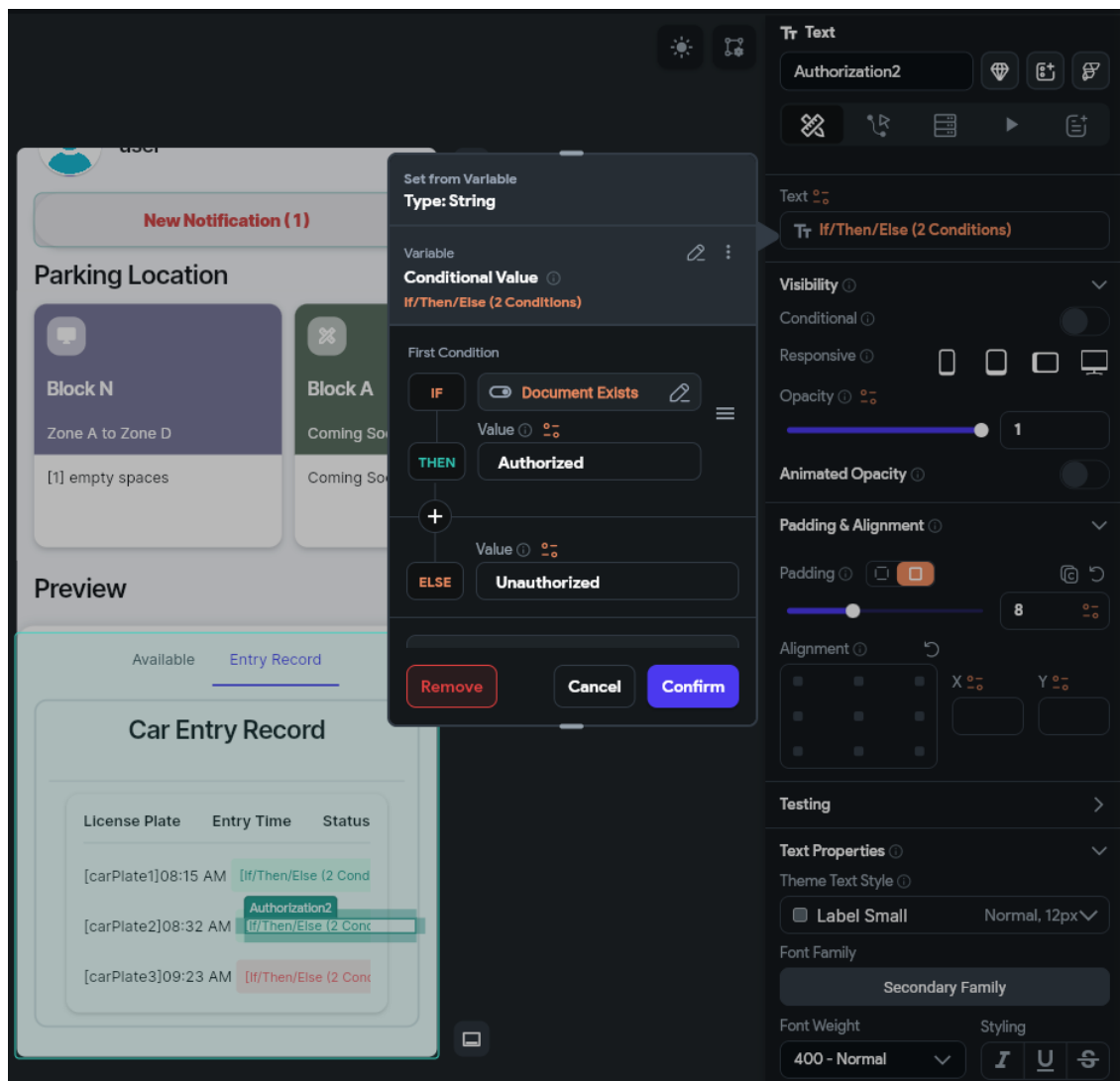


**Figure 4.13 Comparison of plate\_number with carPlate3**

If a matching plate\_number is found, the query returns a document indicating that the car is authorized. A conditional statement is then applied: if the document exists, the corresponding text field displays "Authorized"; otherwise, it displays "Unauthorized", as demonstrated in Figures 4.14 to 4.16. The test result for this conditional statement can refer to Figure 4.10. This logic helps determine whether a vehicle is permitted to enter the car park based on the stored plate number data, even though the OCR feature is not yet fully integrated into the app.



**Figure 4.14 Authorization Text field for carPlate1**



**Figure 4.15 Authorization Text field for carPlate2**

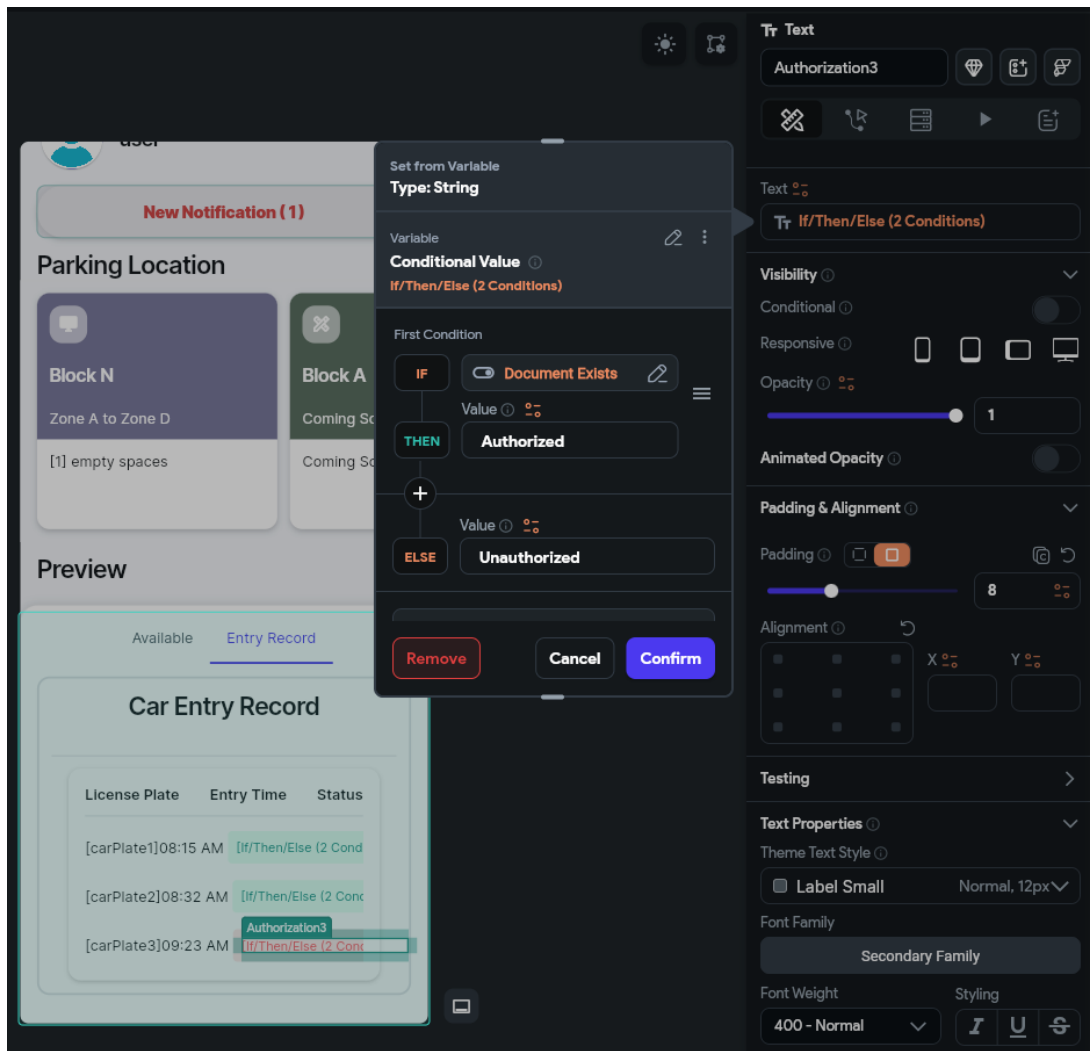


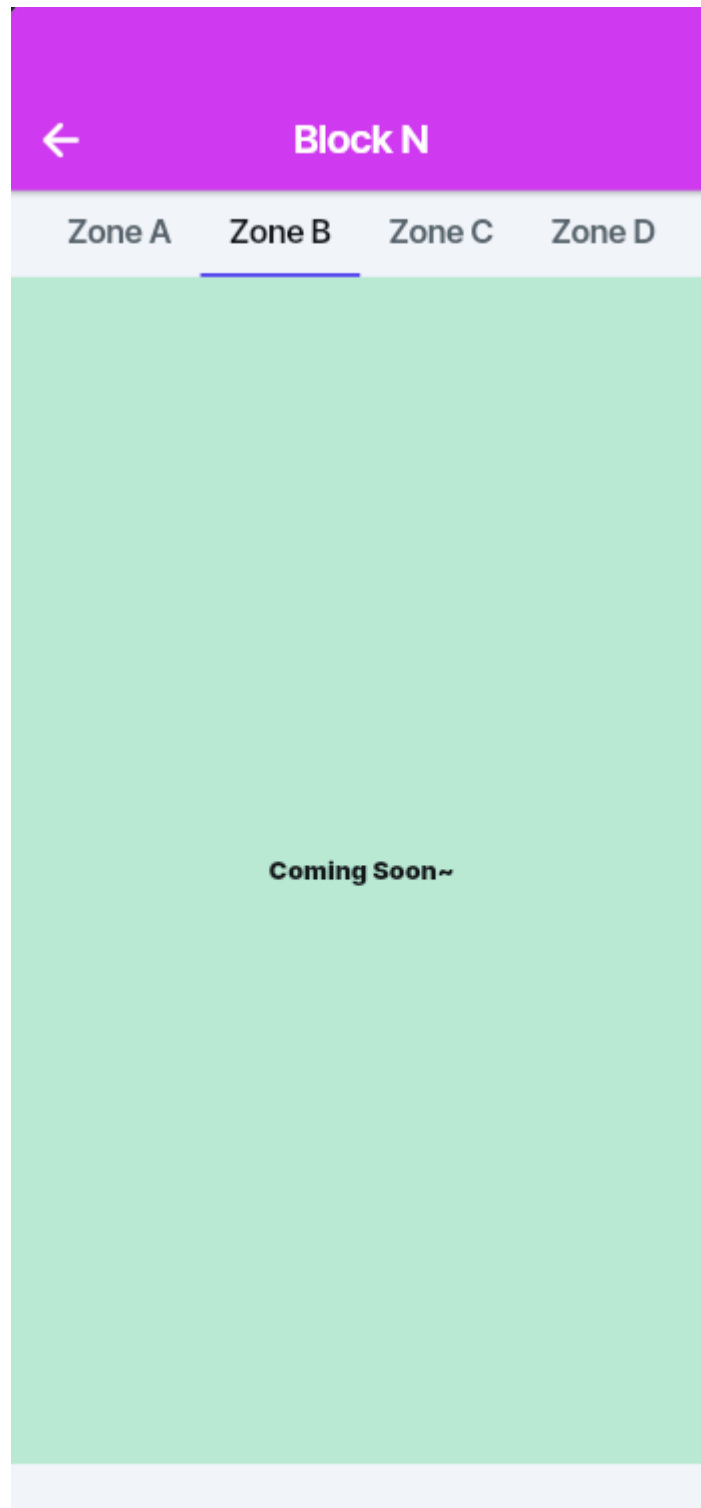
Figure 4.16 Authorization Text field for carPlate3

### Parking Area Camera Modules

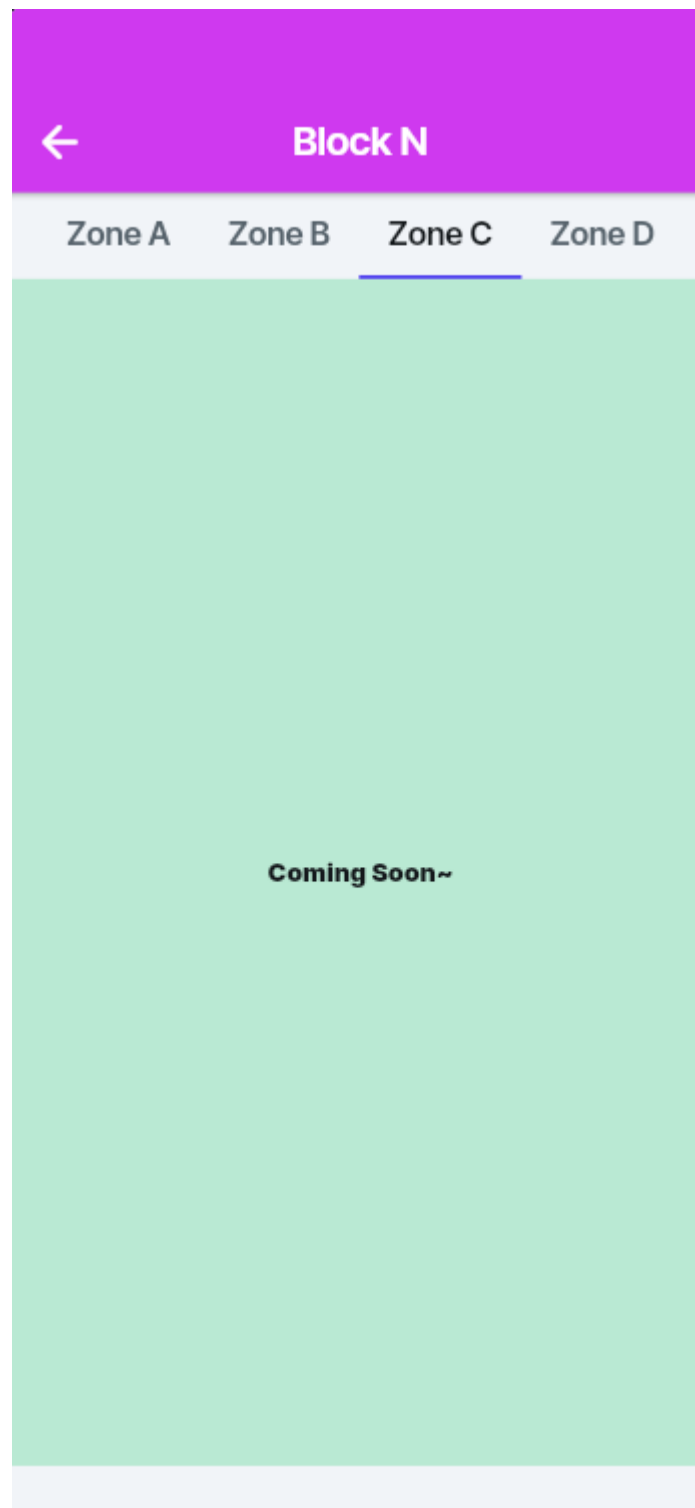
Figure 4.17 shows the zone definitions based on the BlockNCarparkMap. With clear differentiation of zones in Block N, the Block N page contains a tabbed interface for each zone, including Zone A, Zone B, Zone C, and Zone D. Due to the limited number of cameras available at this stage, only Zone A is actively covered. The output for tab pages of Zone B, Zone C, and Zone D are displayed in Figures 4.18 to 4.20, respectively.



**Figure 4.17 Zone Definitions**

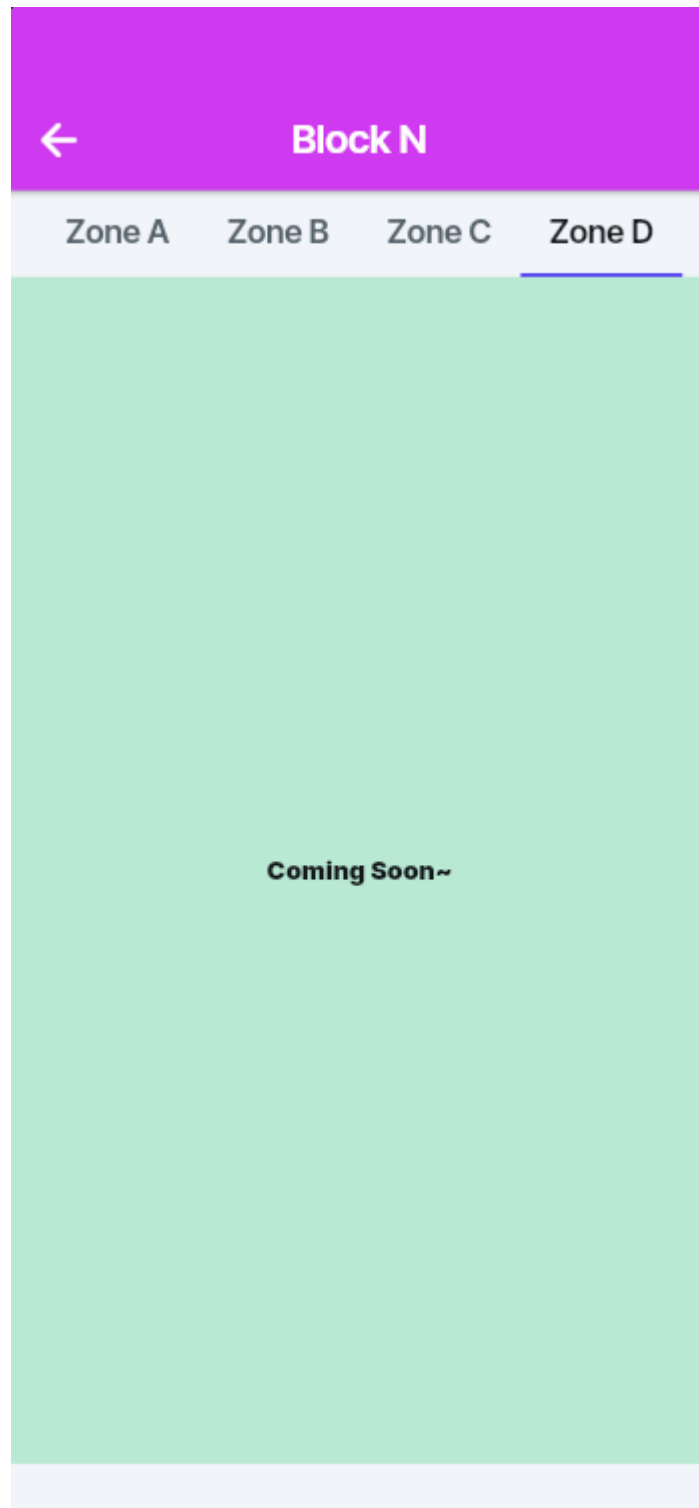


**Figure 4.18 Zone B Tab Page**



**Figure 4.19 Zone C Tab Page**



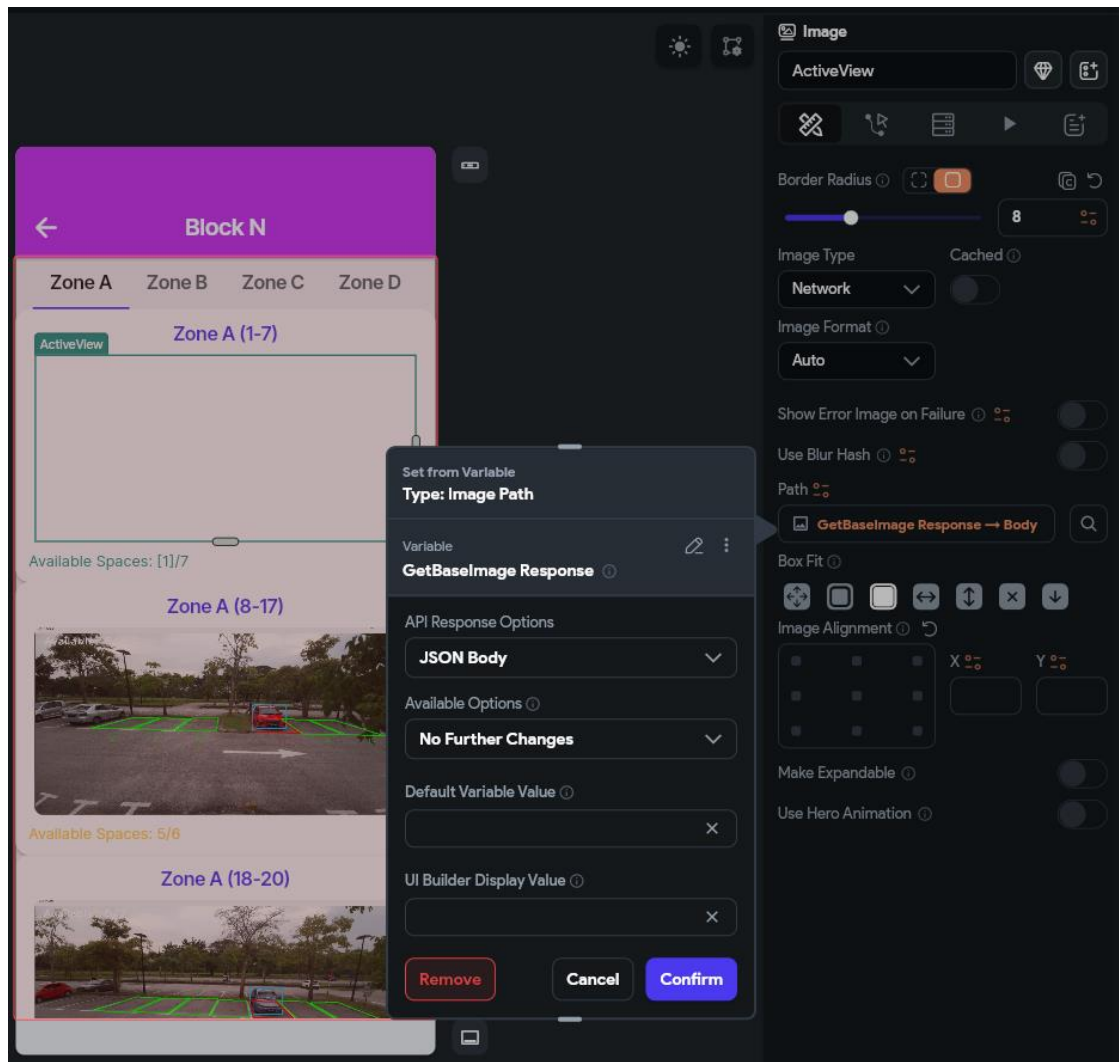


**Figure 4.20 Zone D Tab Page**

Figure 4.21 presents the design of the active camera view component for the Zone A tab. Each view is labeled with the zone alphabet and the corresponding parking lot numbers, followed by the camera feed. This camera view is retrieved using a GET API call that returns a JSON body, which includes an encoded base64 string representing the camera image, as shown in Figure 4.22. The overall implementation concept will be further explained in the next chapter.

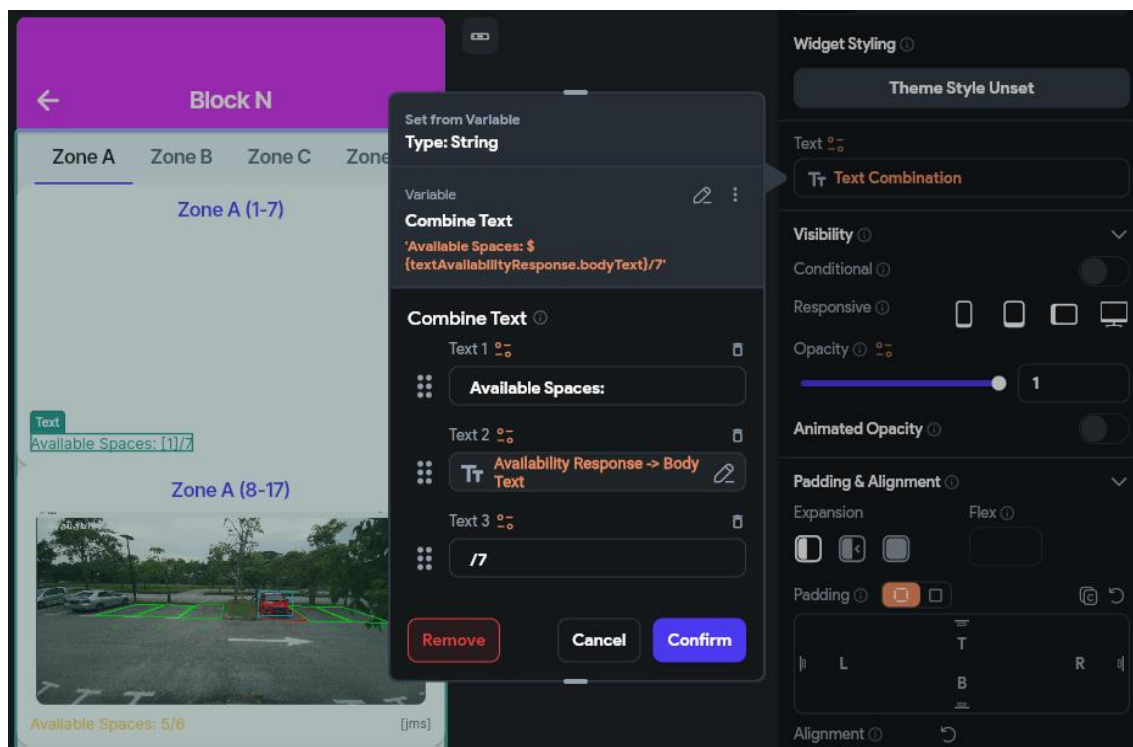


**Figure 4.21 Active Camera View**



**Figure 4.22** Setting on ActiveView Widget

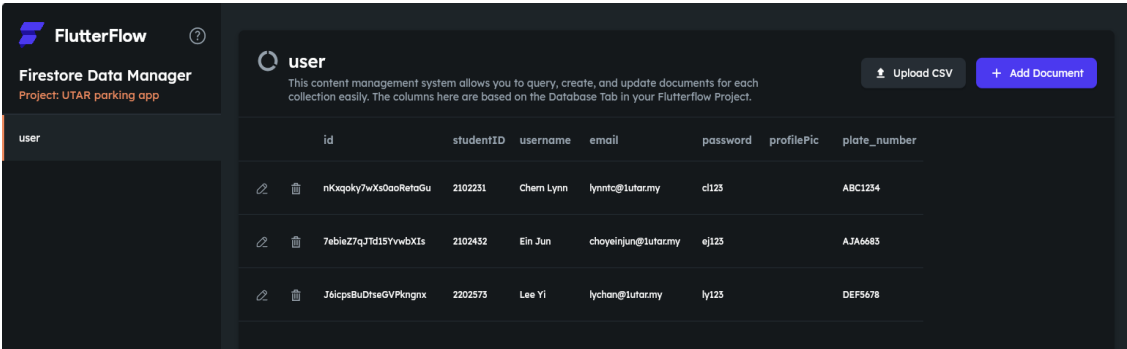
Below the camera image, there is an “available spaces” counter. This counter is also retrieved through an API call, as illustrated in Figure 4.23. The displayed text is generated by combining static text "Available Spaces: " with the dynamic string retrieved from the database, followed by another static text"/7". This formatting ensures the information is presented in a user-friendly and easily understandable way.



**Figure 4.23** Setting on Text Widget Below Image

Data Integration and Processing

Figure 4.24 displays the data stored in the *user* collection of the database, which plays a critical role in verifying license plate authorization and logging vehicle entries. Each document in this collection not only includes the *plate\_number* used for vehicle verification but also contains relevant user credentials such as email, password, and username. These fields are essential for managing user authentication, identifying users within the system, and supporting account-related features of the application. Thus, the *user* collection serves a dual function: managing user accounts and providing a reference for vehicle access control.









	id	studentID	username	email	password	profilePic	plate_number
 	nKxqky7wXs0aoRetaGu	2102231	Chern Lynn	lynntc@tutar.my	cl123		ABC1234
 	7ebieZ7qJ7d1S1YvwbX1s	2102432	Ein Jun	chayeinjun@tutar.my	ej123		AJA6683
 	J8lcp8BuDlseGVPKngnrx	2202573	Lee Yi	lychan@tutar.my	ly123		DEF5678

Figure 4.24 Sample Data Records in the User Collection

Within the FlutterFlow app, this component design outlines how the application interacts with the database to fetch, process, and display data that drives real-time decisions and user interactions. When a car is detected at the entrance, the system queries the *plate\_number* field to check whether the vehicle is registered and authorized to enter. This automated verification replaces manual security checks and enhances efficiency. Meanwhile, user credentials like email and password are integrated into the login system, ensuring that only authenticated users can access app features such as profile viewing or parking history.

Database Integration in UI

The application connects to a centralized Firestore database where each user document is structured to include key fields such as *plate\_number*, authorization status, and the timestamp of the last recorded entry. FlutterFlow’s built-in Firestore query actions are used to dynamically retrieve this data and bind it to UI components. For instance, when a plate number is detected, a query is triggered to compare it with the *plate\_number* field in the user collection for verification.

### **Logical Condition Handling**

To ensure accurate decisions based on query results, FlutterFlow's conditional logic builder is employed. It checks whether the query has returned a matching document. If a match exists, the interface updates to show "Authorized", signaling that the vehicle is permitted to enter. If no match is found, it shows "Unauthorized", alerting that the vehicle does not have access rights. This logic enables the system to act autonomously and immediately, without requiring user intervention or backend coding.

### **Real-time Updates**

In addition to immediate license plate verification, other components of the app such as the availability counters in the parking area tabs described in Section 4.3.2 also rely on real-time data. These counters show the number of available parking spaces by fetching values dynamically through Firestore queries or API calls. With the support of FlutterFlow's live update feature, users are always presented with the most recent data. This ensures greater accuracy and enhances user confidence in the information displayed by the system.

### **Data Binding and UI Presentation**

Fetches data is connected to text fields, labels, status indicators, and other user interface elements to ensure the app remains informative and easy to use. For example, the plate number, entry time, and access status are clearly shown on entry logs or dashboard screens. This smooth data binding allows users, including security personnel and app users, to monitor system activity in real time and make informed decisions based on the latest records stored in the user collection.

In summary, the integration between the application and its database backend supports automated decision-making, effective vehicle tracking, and secure user authentication. It ensures that every component, from login to license plate verification, functions together smoothly to uphold the performance and reliability of the smart car park management system.

### **4.4 System Components Interaction Operations**

The various components of the Smart Parking Management System work together through a series of interactions that ensure the system operates efficiently and effectively. Each component, from hardware to software, plays a unique role, and their integration ensures real-time monitoring, accurate vehicle identification, and a smooth user experience.

#### **Overview of Component Interactions**

The system is composed of several core components that work together to provide real-time parking management. These components include the entrance camera module, parking area camera modules, mobile application, data integration and processing systems, and the backend database. Each component has a specific task: the entrance and parking area cameras capture vehicle data, the mobile app offers users a way to interact with the system and view parking availability, and the backend system stores and processes data from the cameras. The components communicate effectively to ensure that parking availability is monitored continuously and updated in real time, and users are notified of parking space status.

#### **Camera and Raspberry Pi Interaction**

The entrance camera module and parking area camera modules are connected to the Raspberry Pi, which processes the image data captured by the cameras. When a vehicle enters the parking area, the entrance camera captures an image, which is then sent to the Raspberry Pi for processing. The Raspberry Pi analyses the image to detect the presence of a vehicle and processes the data accordingly. For the entrance camera, this involves using the optical character recognition module to extract license plate numbers, once fully implemented. The parking area cameras continuously monitor the parking bays and send data to the Raspberry Pi, which checks whether the parking bays are vacant or occupied and updates the backend database with the status. The Raspberry Pi communicates with the backend to ensure that the parking data is synchronized in real time, so users can view available spaces via the mobile application.

### **Optical Character Recognition and License Plate Recognition**

Although the entrance camera module is not yet prepared, it is designed to work with the OCR to automatically extract license plate numbers from images when vehicles enter the parking lot. Once the entrance camera captures an image, the Raspberry Pi processes it, and if the OCR module is integrated, it will use the technology to recognize the license plate number. The recognized plate number is then sent to the backend for verification against a list of authorized vehicles. Currently, since the OCR module is not yet in use in the mobile app, the system checks vehicle authorization by manually matching the plate number to a pre-registered list stored in the backend database. This manual method is a temporary solution, and once the OCR module is fully integrated, it will automatically recognize license plate numbers in real time, streamlining the authorization process.

### **Backend Interaction with Database**

The backend system acts as the central hub for storing, processing, and managing all data related to the parking system. It interacts with both the Raspberry Pi and the mobile application to synchronize parking space availability and vehicle authorization. When the Raspberry Pi processes data from the entrance or parking area cameras, it sends the information to the backend, where it is stored in the database. This includes updates on parking occupancy, availability and encoded base64 string of the latest image. The database also stores user information and vehicle records. The backend processes requests from the mobile application by retrieving the latest parking data and sending it to the app in real time. This ensures that users have access to the most accurate and up-to-date parking information. The backend plays a key role in ensuring that all components work together, and it ensures that data is securely stored and consistently updated.



### **Mobile Application and Backend Communication**

The mobile application provides the user interface for interacting with the system. Users can view real-time parking availability, check the status of individual parking spaces, and receive notifications when parking spaces are full. When a user opens the app and requests parking data, the app sends a query to the backend. The backend processes this request by retrieving the latest parking availability data from the database and sending it back to the app. The app then displays this information to the user. Additionally, the app sends updates to the backend when users make requests, such as reserving a parking space or checking parking availability in a specific area. Notifications are sent to users based on updates from the backend, such as when a parking area reaches full capacity, so users can make informed decisions. This smooth communication between the mobile app and backend ensures that users receive accurate and timely information.

The interaction between the components in the Smart Parking Management System is fundamental to its functionality and effectiveness. By ensuring that the cameras, Raspberry Pi, OCR modules, backend database, and mobile app communicate properly, the system delivers real-time parking management and vehicle authorization. Each component has a specific role, and their interactions ensure that the system operates smoothly. While certain features, such as automatic license plate recognition, are still being developed, the current system design supports future integration and scalability, allowing the system to adapt to new technologies and expand its capabilities as needed.

## Chapter 5

### System Implementation

#### 5.1 Hardware Setup

The hardware involved in this project includes a laptop, a mobile device, a Raspberry Pi 3, and a Raspberry Pi Camera Module 3. The laptop serves as the main development platform, where the detailed specifications of the laptop are provided in Table 5.1.

**Table 5.1 Specifications of Laptop**

Description	Specifications
Model	Huawei MateBook D16 2024
Processor	12th Gen Intel(R) Core(TM) i5-12450H
Operating System	Windows 11
Graphic	Intel(R) UHD Graphics
Memory	16GB RAM
Storage	512GB HDD

A mobile device is utilized for testing and deploying the dashboard within the mobile application, allowing real-time interaction and validation of the app's functionality. The Raspberry Pi 3, paired with the Raspberry Pi Camera Module 3, is critical for capturing video footage of cars and monitoring the parking area. This setup facilitates the implementation of real-time video processing and analysis, forming the core of the project's data collection and processing capabilities.

### 5.2 Software Setup

The project utilizes various software tools for development and viewing to enhance the workflow and efficiency of different tasks. Software used for development and viewing include:

- Thonny
- Google Colab
- FlutterFlow
- Raspberry Pi Imager
- PuTTY (64-bit)
- RealVNC Viewer
- Firebase

Thonny serves as the primary integrated development environment (IDE) for running the Python scripts that power the core functionality of the parking system. Specifically, Thonny is used on the Raspberry Pi to execute real-time image capture, process images to detect cars, define bounding boxes, and monitor the occupancy status of parking lots. It also manages the labelling of individual parking bays, updating carpark availability based on camera input.

Google Colab, on the other hand, is utilized for more resource-intensive tasks, such as car plate number recognition. It provides a cloud-based Python environment capable of leveraging GPU acceleration, which is ideal for training and testing the OCR model. The Colab environment processes images uploaded from test cases or collected during trials to detect and extract license plate text using a YOLO-based detection model and Tesseract OCR.

FlutterFlow is used to design and develop the mobile application interface in Dart, offering a visual drag-and-drop builder that significantly accelerates app creation. It supports smooth integration with Firebase, enabling real-time data display within the app, such as parking availability updates and camera image feeds. FlutterFlow's cross-platform capabilities ensure the app is accessible on both Android and iOS devices, making it convenient for users to monitor parking conditions on the go.

Raspberry Pi Imager is used to flash the Raspberry Pi OS onto the SD cards, preparing each Raspberry Pi unit for deployment in the parking system. After installation, PuTTY (64-bit) is used for secure SSH access to the Raspberry Pi terminals, allowing developers to remotely execute commands, update scripts, and configure settings. Meanwhile, RealVNC Viewer provides a graphical interface for remote desktop access, making it easier to visually monitor the Raspberry Pi environment, debug issues, and view image output directly from the connected cameras. These tools combined enable effective setup, control, and management of the edge devices without physical interaction.

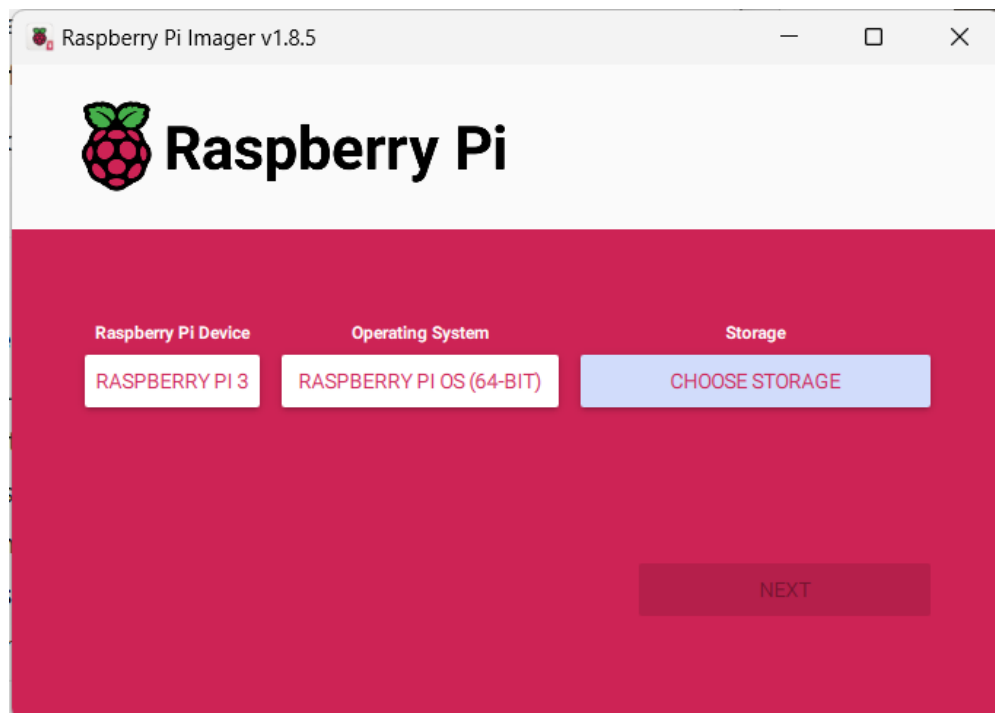
Firebase is used as the cloud-based backend of the system. The Realtime Database manages dynamic updates, such as tracking the occupancy status of parking bays, camera view statuses, and availability counts. These updates are delivered in real time to the mobile application, ensuring users receive timely and accurate information. Additionally, Firebase Cloud Firestore is used to store more structured project-related data, including user records, camera assignments, and configuration metadata. Together, the two databases offer a reliable and scalable solution for handling both frequent updates and organized data storage requirements.

### 5.3 Setting and Configuration

#### Step 1: Raspberry Pi Configuration using Raspberry Pi Imager

Before beginning any hardware connection, the Raspberry Pi was pre-configured using the Raspberry Pi Imager tool as shown in Figure 5.1 to enable efficient headless operation during testing. A USB SD card reader was used to insert the microSD card into the computer. Through Raspberry Pi Imager, the Raspberry Pi OS was selected and flashed onto the microSD card. During this process, advanced configuration settings were enabled to allow the Raspberry Pi to operate without the need for a monitor, keyboard, or mouse.

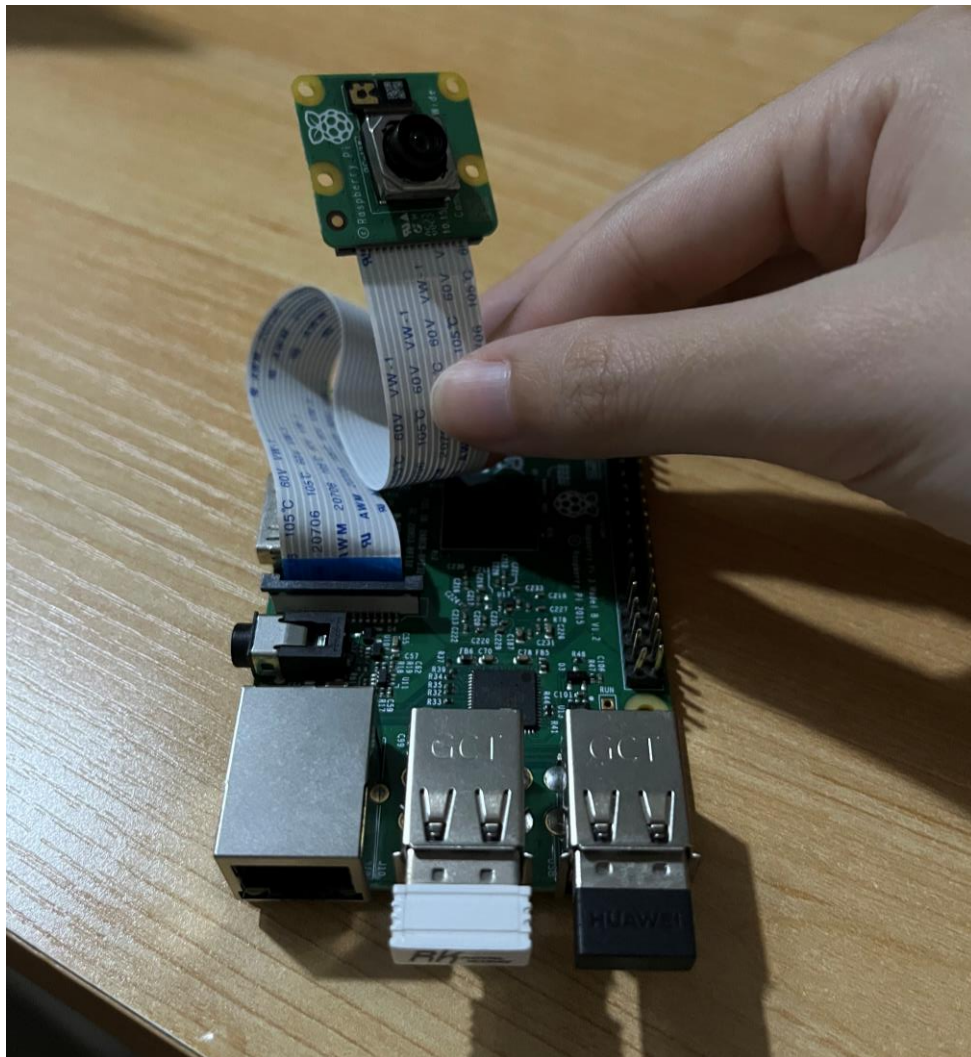
Key configurations included enabling Secure Shell (SSH) access for remote terminal control, setting a predefined username and password for secure login, and entering Wi-Fi credentials such as the SSID, password, and country code. These configurations ensured that upon powering up, the Raspberry Pi would automatically connect to the specified wireless network and be immediately accessible for remote operation. Once the flashing process was complete, the microSD card was safely ejected from the reader and inserted into the Raspberry Pi, completing the setup required for effective remote-controlled testing in the following steps.



**Figure 5.1 Raspberry Pi Imager**

### Step 2: Connect the Raspberry Pi Camera Module to the Raspberry Pi

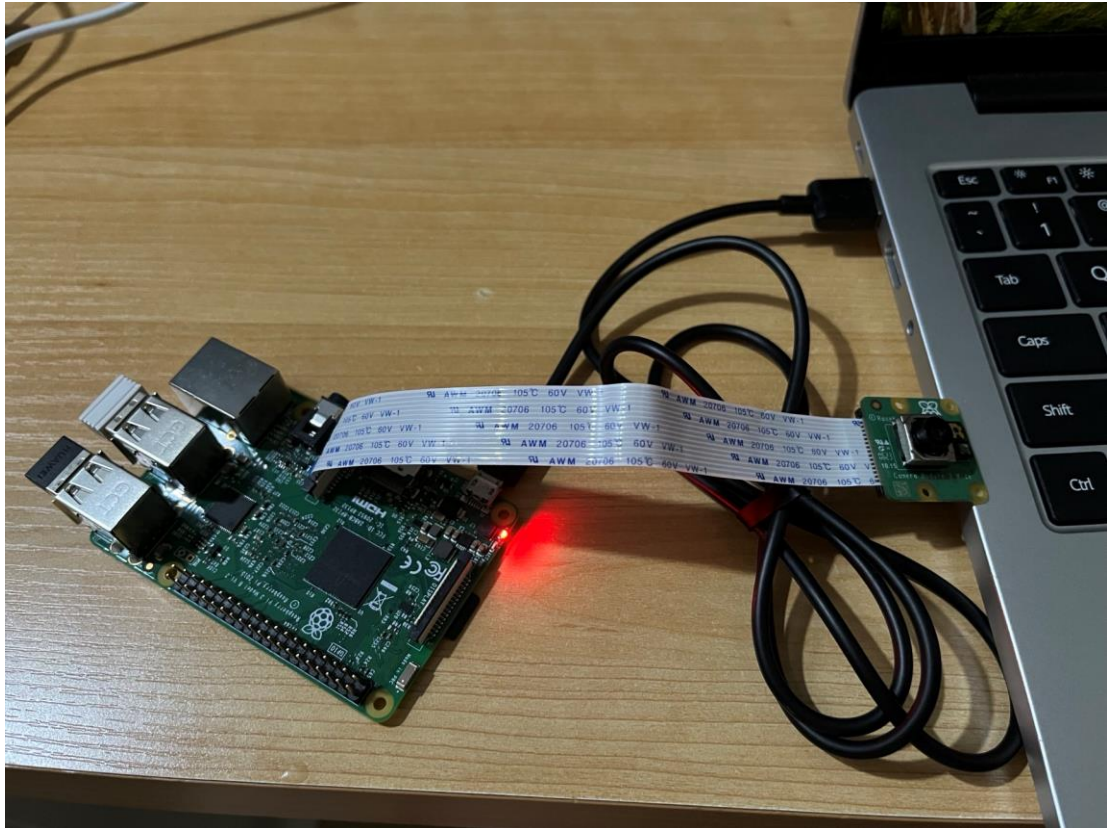
Once the Raspberry Pi's configuration was complete, the next step was to connect the Raspberry Pi Camera Module. The camera module was inserted into the Camera Serial Interface (CSI) port, located near the HDMI port on the Raspberry Pi board. The plastic clip on the CSI port was carefully lifted to make space for the ribbon cable from the camera. The cable was then inserted with the metal connectors facing the correct direction, ensuring a secure connection. The clip was then pressed back down to firmly hold the ribbon cable in place. This setup allowed the Raspberry Pi to capture images or video for real-time processing, which is crucial for testing the smart parking system. The complete setup of the Pi Camera connected to the Raspberry Pi is illustrated in Figure 5.2.



**Figure 5.2 Connection of Pi Camera to Raspberry Pi**

### Step 3: Power the Raspberry Pi

After successfully connecting the camera module, the Raspberry Pi was powered on using a suitable power supply. The power source can be the adapter, power bank or even from a laptop. This step was important to ensure that the Raspberry Pi was running and accessible for subsequent operations. Figure 5.3 illustrates the connection between the Raspberry Pi and a laptop using a standard USB Type-A to USB Micro-B cable.



**Figure 5.3 Connection of Raspberry Pi to a Laptop**

### Step 4: Accessing the Raspberry Pi Desktop Environment

After the Raspberry Pi was powered on and properly configured, accessing its desktop environment became a necessary step for further setup and Python script development. There were two main approaches used to access the Raspberry Pi interface, depending on the situation and purpose.

During the initial configuration phase, a physical connection was established by connecting the Raspberry Pi to a monitor using an HDMI cable. A keyboard and mouse were also plugged into the USB ports, allowing for full interaction with the graphical interface. This method as shown in Figure 5.4 offered a reliable way to visually verify the success of the operating system installation, test hardware components such as the camera module, and resolve any connectivity or configuration issues.

In the later stages of the project, particularly during portable testing and deployment, a more efficient method was preferred. The Raspberry Pi was accessed remotely using a laptop by employing PuTTY for terminal-based communication over SSH, and RealVNC Viewer for accessing the full desktop interface. This headless configuration eliminated the need to carry extra peripherals like a monitor or keyboard, making it highly convenient for use in different testing environments. This remote access approach proved especially beneficial during the experimental phase described in Chapter 6, where portability and ease of access were key considerations.



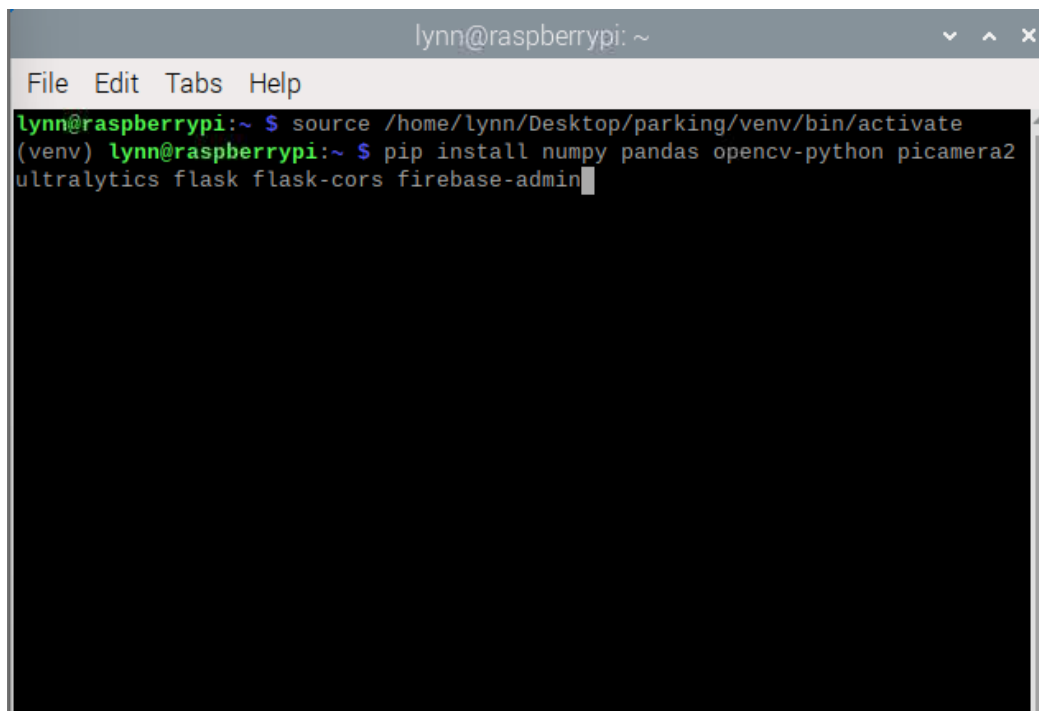


**Figure 5.4 Setup Raspberry Pi's Desktop on Monitor**

### Step 5: Developing and Running the Python Script in Thonny

Once access to the Raspberry Pi desktop was established, Python development for the smart parking system was conducted using Thonny, a lightweight integrated development environment (IDE) designed for ease of use on Raspberry Pi. The Python script developed in Thonny was responsible for several critical tasks. These included capturing images from the Pi Camera, processing the images to detect vehicles, drawing bounding boxes around detected cars, and tracking parking bay occupancy in real time. The script also labeled and numbered individual parking spots to monitor which bays were occupied or vacant.

To ensure a clean and organized Python environment, a virtual environment was created on the Raspberry Pi using the command `python3 -m venv /home/lynn/Desktop/parking/venv`. This practice helps isolate project dependencies and prevents conflicts with system-wide packages. The workflow included creating the virtual environment, activating it, installing necessary packages as illustrated in Figure 5.5, executing the Python script, and finally deactivating the environment after use. This process reflects good development practices for maintaining code portability and reproducibility across devices.



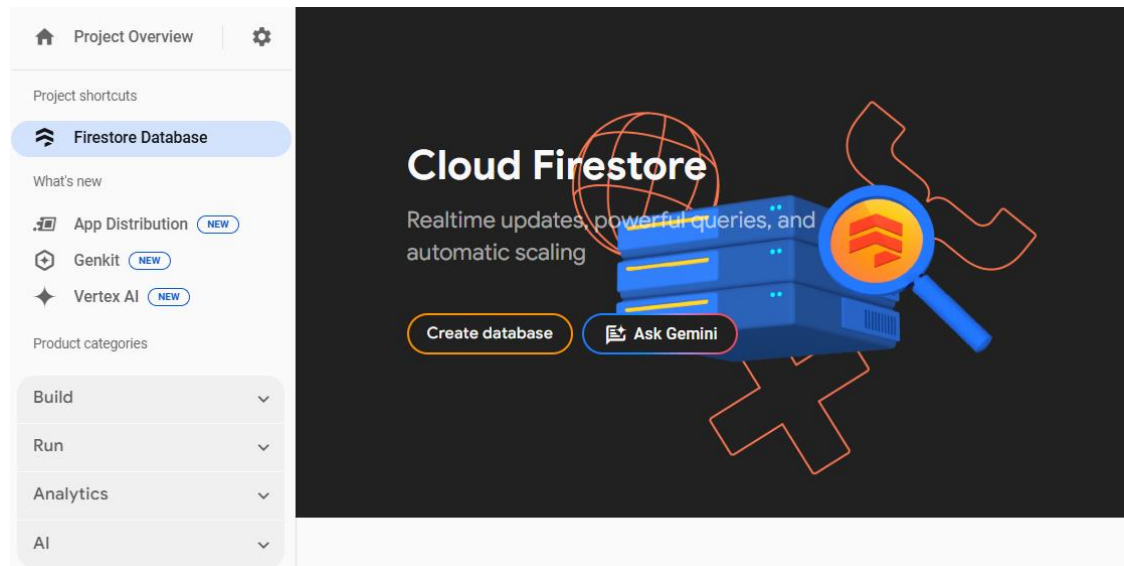
```
lynn@raspberrypi: ~  
File Edit Tabs Help  
lynn@raspberrypi:~ $ source /home/lynn/Desktop/parking/venv/bin/activate  
(venv) lynn@raspberrypi:~ $ pip install numpy pandas opencv-python picamera2  
ultralytics flask flask-cors firebase-admin
```

**Figure 5.5 Command in Terminal**

### Step 6: Integrating Firebase for Data Storage and Real-Time Communication

To enable smooth integration with the mobile application developed using FlutterFlow, the backend database system was constructed using Firebase, which offers both Cloud Firestore, also known as Firestore Database, and Realtime Database services. Each database played a distinct role in the system architecture, handling different types of data to ensure optimal performance and functionality.

The Firestore Database was designated for storing structured and relatively static data, such as user information, which could be retrieved to verify user authentication and manage authorization access within the system. The setup began by creating a new firestore database, as shown in Figure 5.6, followed by selecting Singapore as the preferred location to ensure low-latency access, as illustrated in Figure 5.7. Security rules were configured to control data access, depicted in Figure 5.8. Data collections were added either directly through the Firebase Console or via FlutterFlow's Firebase tab, as demonstrated in Figure 5.9. The structured layout of the Firestore database used in this project is presented in Figure 5.10.



**Figure 5.6 Create New Firestore Database**

**Create database** ✕

1 Set name and location — 2 Secure rules

★ Interested in Firestore with MongoDB compatibility? Create a Firestore Enterprise database in the Google Cloud Console [Learn more](#)

Database ID  
(default)

Location  
asia-southeast1 (Singapore) ▼

ⓘ Your location setting is where your Cloud Firestore data will be stored

! After you set this location, you cannot change it later. [Learn more](#)

Cancel **Next**

Figure 5.7 Select Cloud Firestore Data's Location

**Create database** ✕

1 Set name and location — 2 Secure rules

After you define your data structure, you will need to write rules to secure your data. [Learn more](#)

☐ Start in **production mode**  
Your data is private by default. Client read/write access will only be granted as specified by your security rules.

☒ Start in **test mode**  
Your data is open by default to enable quick setup. However, you must update your security rules within 30 days to enable long-term client read/write access.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if
        request.time < timestamp.date(2025, 6, 1);
    }
  }
}
```

! The default security rules for test mode allow anyone with your database reference to view, edit and delete all data in your database for the next 30 days

Cancel **Create**

Figure 5.8 Security Rules for Database

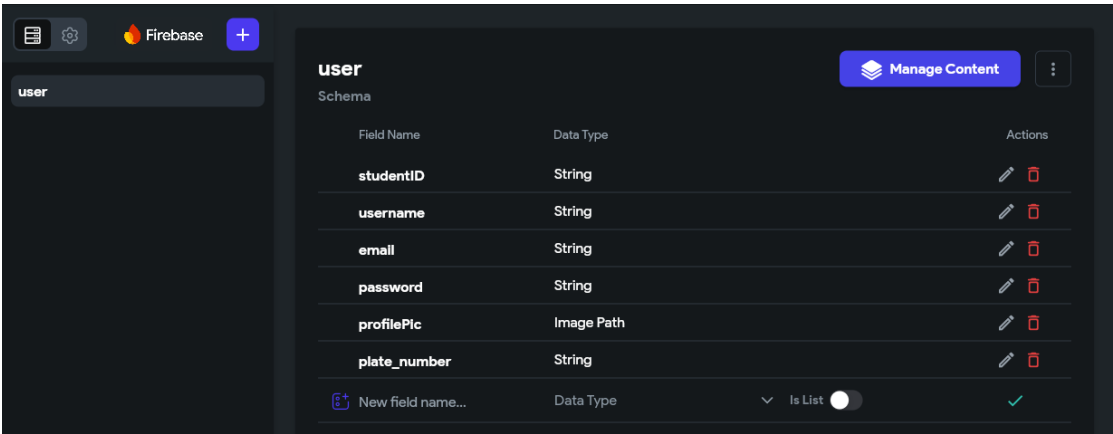


Figure 5.9 Firestore Database in FlutterFlow’s Firebase Tab

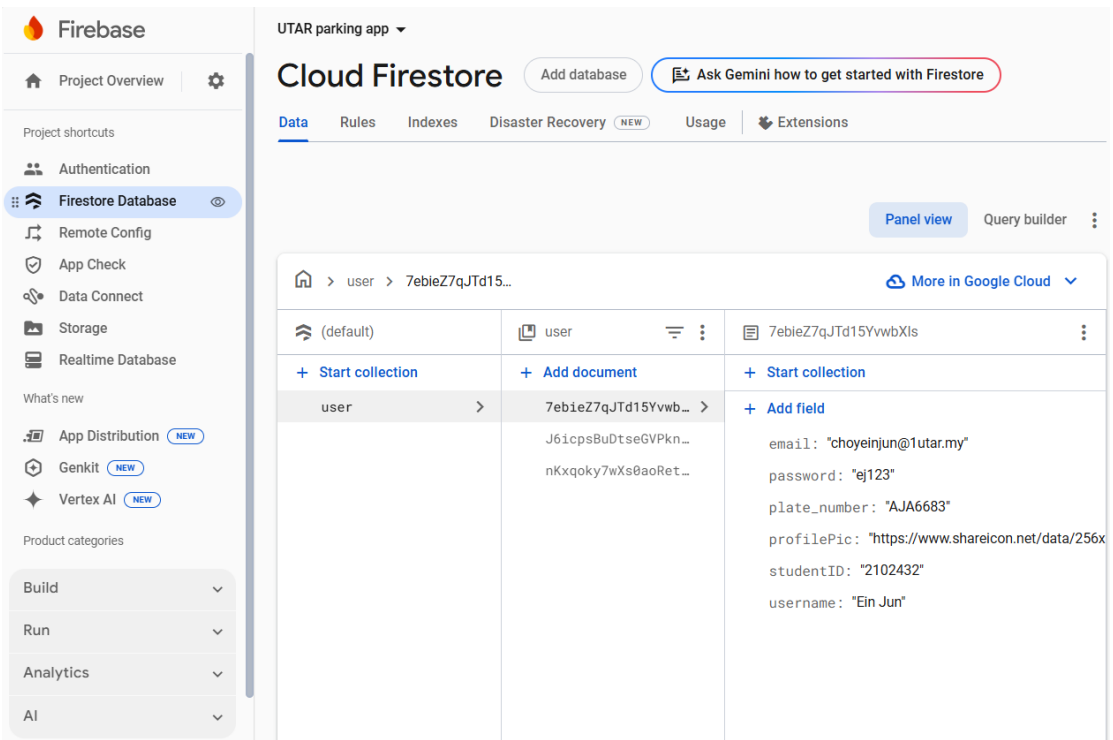
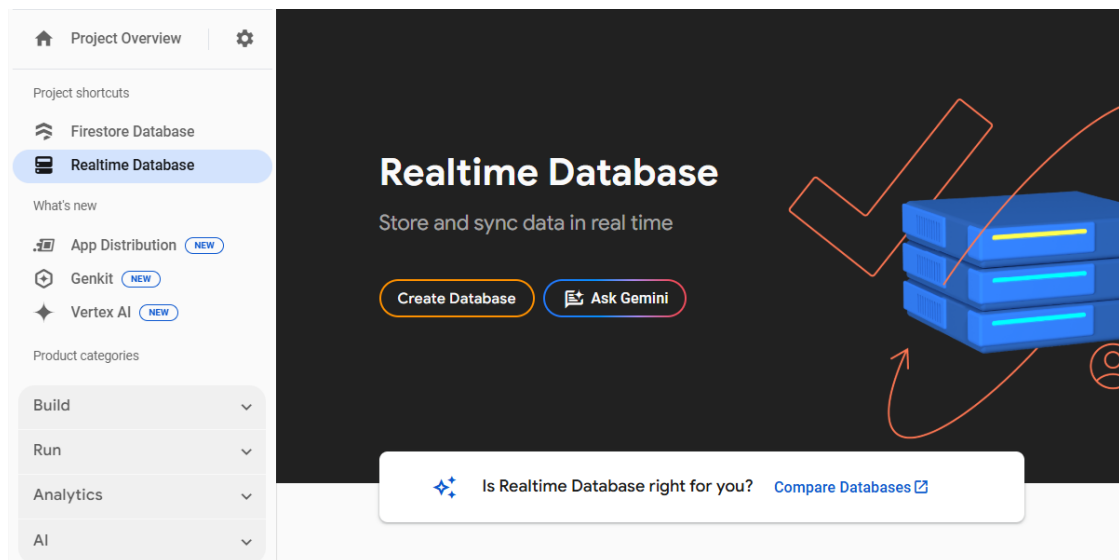


Figure 5.10 Firestore Database in Firebase Console

In contrast, the Firebase Realtime Database was configured to handle fast-changing and time-sensitive data such as parking lot occupancy status, live availability counts, and camera statuses. Its low-latency data delivery and real-time synchronization features made it ideal for this purpose. The setup process started by creating a new real time database as shown in Figure 5.11. Next, real time database location need to be select. In this project, Singapore region was selected, as shown in Figures 5.12. Lastly, select the application of security rules to manage read and write permissions. In currently, the option “Start in Test Mode” was selected as shown in Figure 5.13. Finally, the real time database is ready for used and the resulting Realtime Database structure used in the system is illustrated in Figure 5.14.



**Figure 5.11 Create New Realtime database**

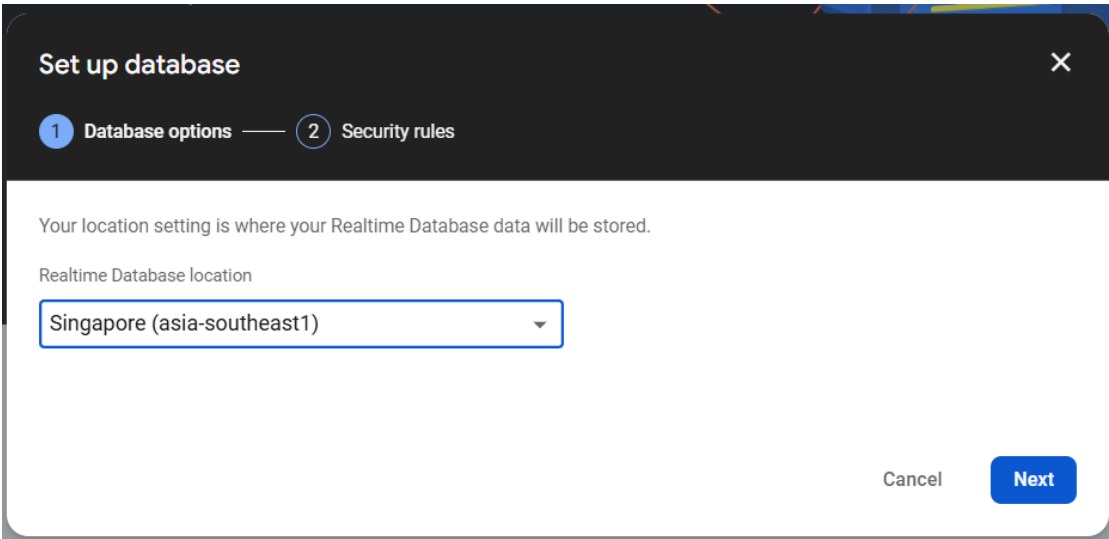


Figure 5.12 Select Realtime Database Location

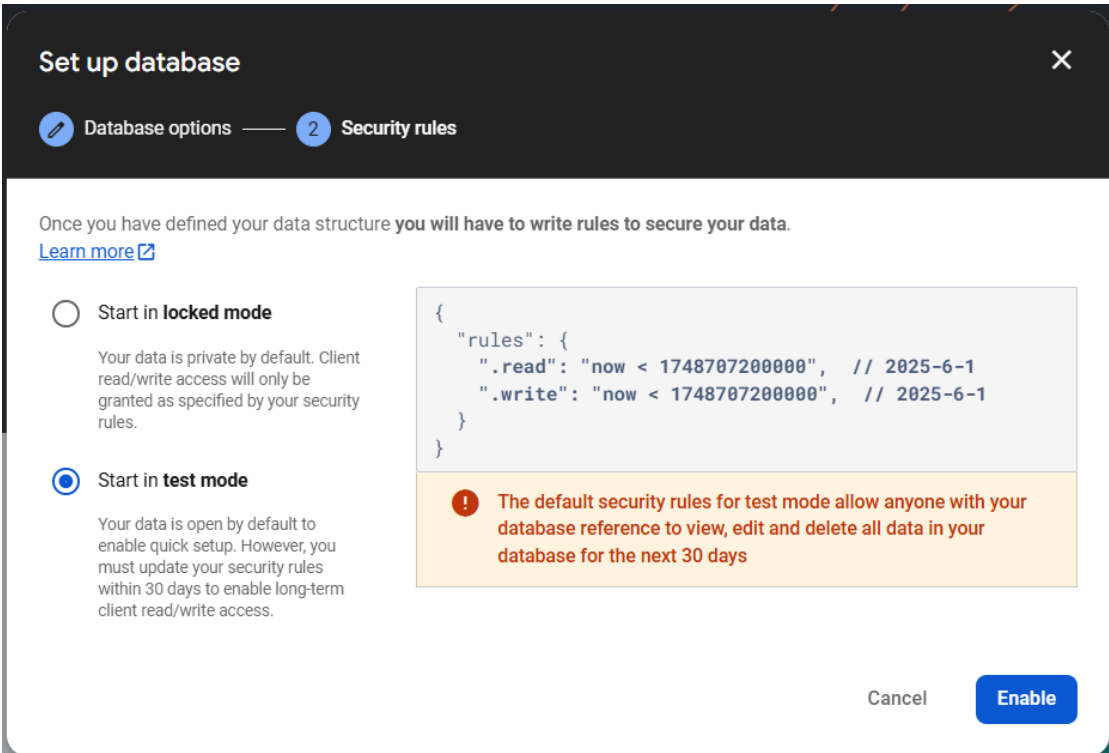
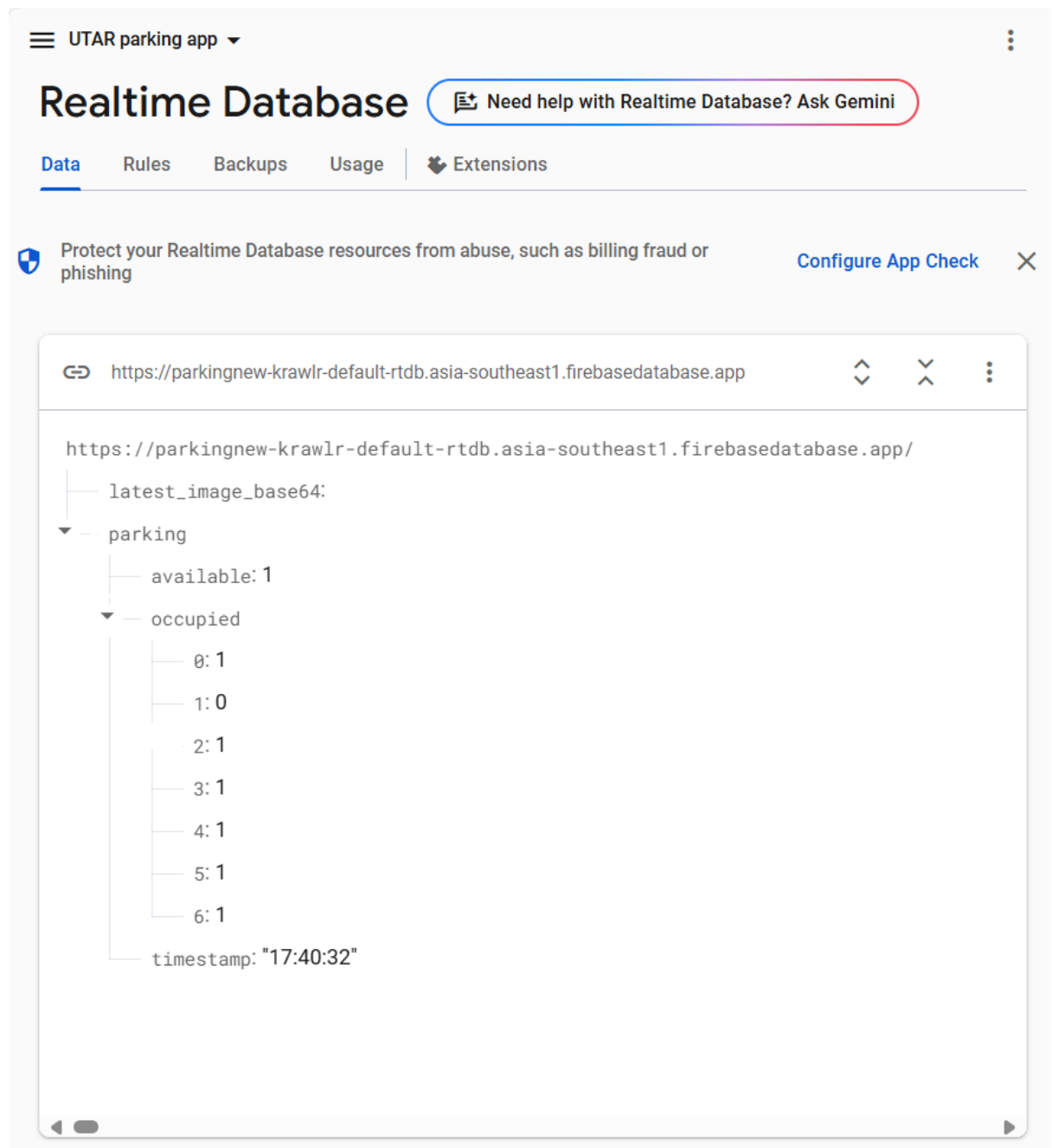


Figure 5.13 Security Rules for Realtime Database



**Figure 5.14 Realtime Database Structure**

By combining both Firestore and Realtime Database, the smart parking system achieved an efficient separation of responsibilities. Firestore was used to store configuration and user-related data, while Realtime Database ensured live updates were continuously reflected in the mobile application. This dual-database architecture provided strong data handling capabilities, enhanced responsiveness, and a reliable user experience across the system.



### Step 7: Mobile Application Interface Development Using FlutterFlow

To provide users with a visual and interactive way of monitoring the real-time status of the car park, a mobile application was developed using FlutterFlow, a low-code development platform. The app structure was designed to be intuitive and minimal, focusing on easy navigation and real-time data presentation from Firebase. The application consists of four main pages and one notification component:

#### **First Page**

The First Page of the mobile application serves as a welcoming landing screen designed to prepare users for the upcoming experience. This page contains a "Get Started" button, which acts as the entry point into the system. When the button is clicked, users are directed to the Login Page, where they can proceed with authentication.

#### **Login Page**

The login page is restricted to UTAR students and staff, the primary users of the smart parking system. Access to the system is restricted through this page to ensure only authorized users can sign in. The "Sign In" button performs multiple validation checks, particularly verifying the email address and matching the password against the registered user credentials stored in Firebase. These checks are implemented to maintain a secure and accountable login process. The detailed validation logic will be further explained in the next section.

#### **Home Page**

Once authenticated, users are directed to the Home Page, where their username is displayed as a personalized greeting. This page offers a comprehensive overview of the parking system, including block-wise carpark summaries, zone availability, and recent entry records.

The parking location section features clickable widgets for each block. By tapping on a specific block, users are navigated to another page that provides a more detailed view of that area. Additionally, the preview section displays both the number of available parking spaces in each zone and the latest entry records as two different tab page. The Available tab displays the number of empty parking spaces in each zone, helping users

quickly assess availability. While the Entry Record tab lists the latest vehicle entries, indicating whether each vehicle was authorized or unauthorized based on the license plate recognition system.

### **Block N Page (Detailed View)**

This page displays detailed car park status for a specific block involved in the project's current implementation phase. It shows real-time data retrieved from the Firebase Realtime Database, such as the number of vacant parking bays, latest camera view, timestamps of updates, and the status of individual parking slots. This page reflects the core functionality of the smart parking system's mobile interface and demonstrates direct integration with backend data.

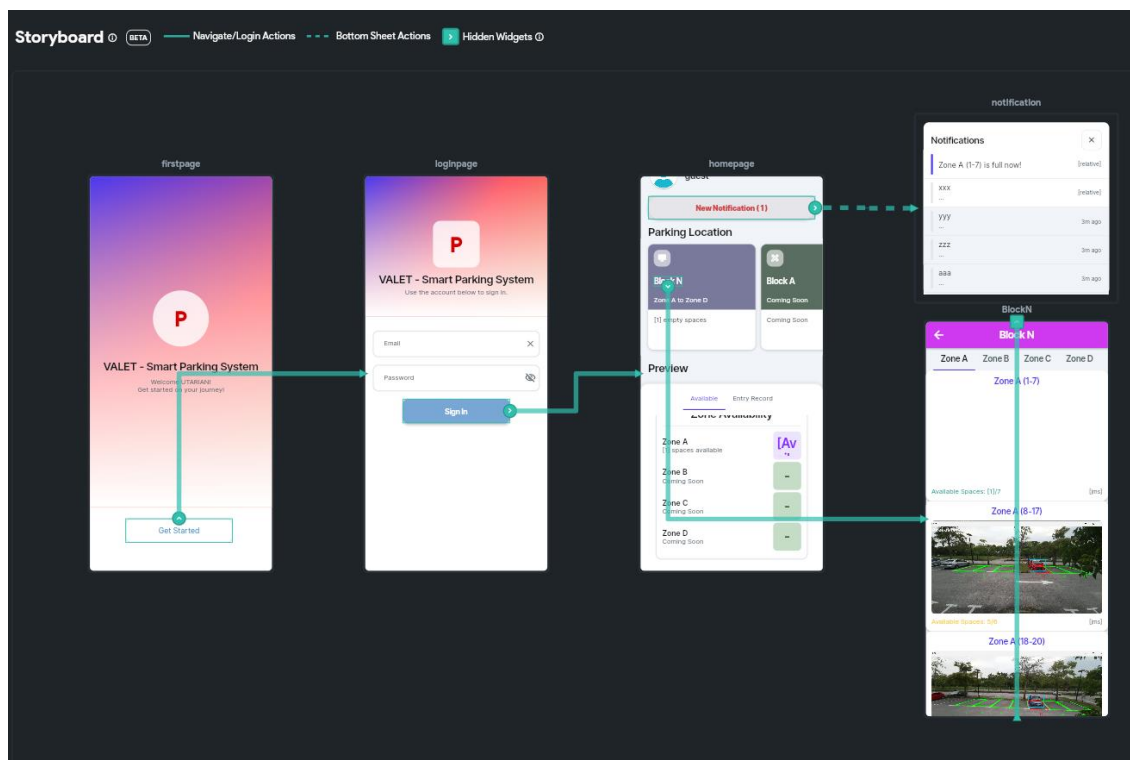
### **Notification Component**

A dynamic notification component was implemented and placed on the Home Page to alert users when the selected car park block is full. This feature is conditionally triggered based on live data from the Firebase Realtime Database. An if-else condition was used: when the number of available empty spaces equals zero, the notification wording becomes visible, and the button becomes clickable. This not only improves user experience but also helps avoid unnecessary searching for unavailable parking.

The interface logic and display conditions were constructed within FlutterFlow's UI logic builder and Firebase query integration, as will be further explained and visualized in the next section.

### 5.4 System Operation (with Screenshot)

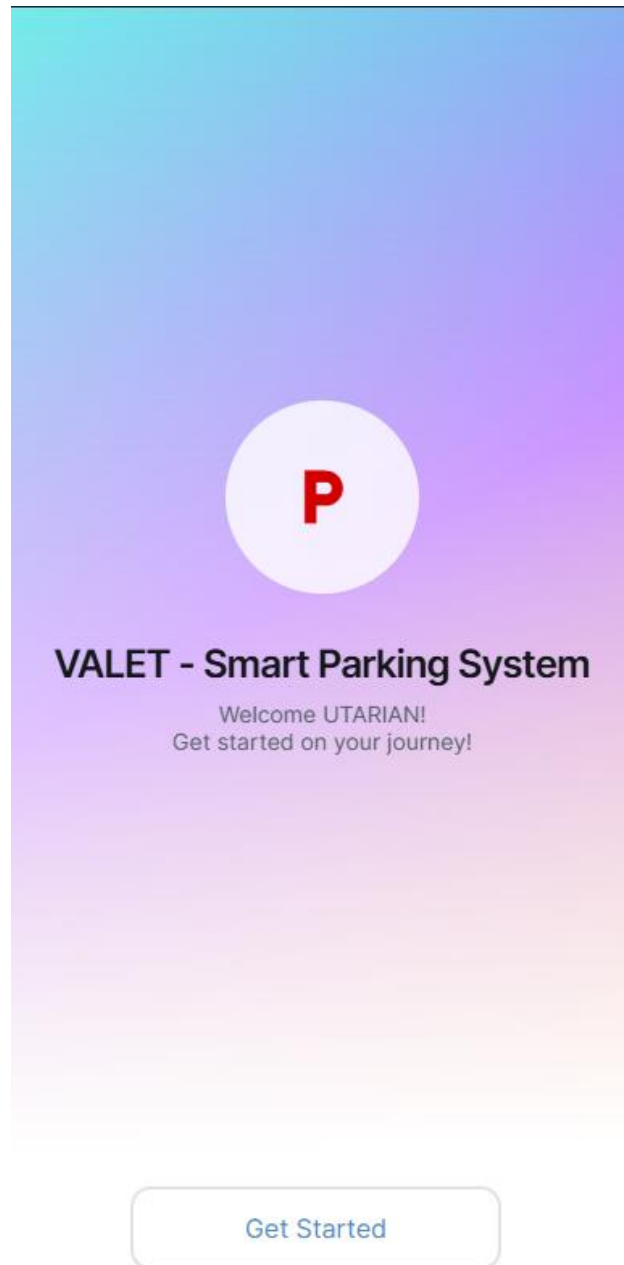
Figure 5.15 provides an overview of the navigational flow of the application. A series of structured tests were conducted systematically, following the sequence of actions that a typical user would perform. The testing process involved validating the functionality of all buttons, ensuring that required text fields are properly validated, verifying the accuracy and consistency of the data retrieved or submitted, and confirming that all system notifications and backend processes work as intended.



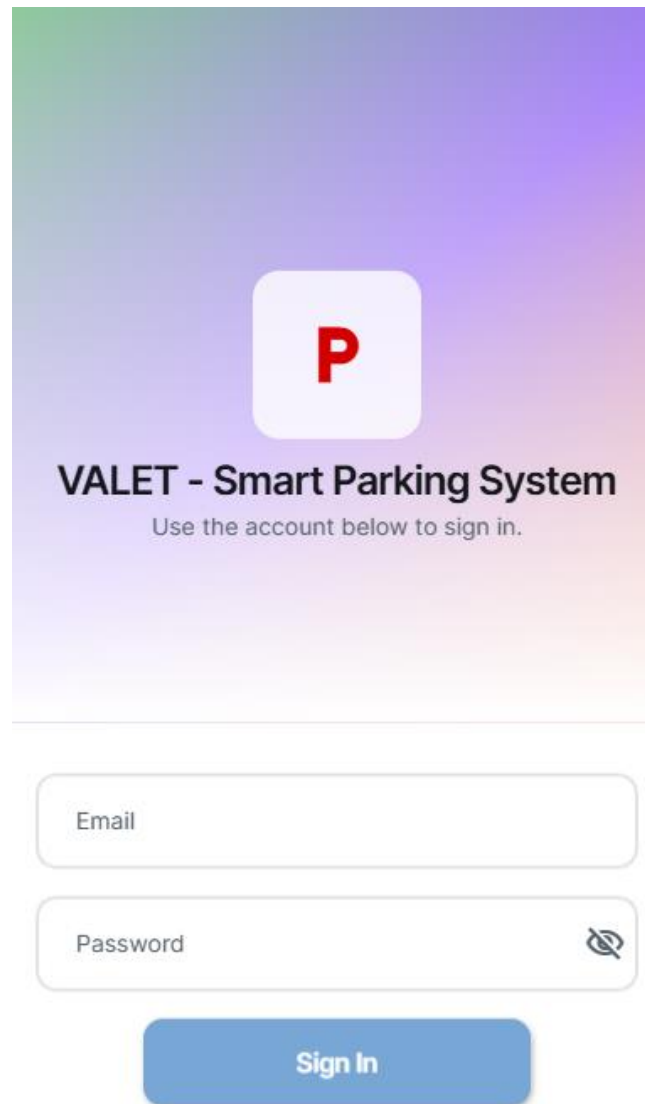
**Figure 5.15 Storyboard of the Smart Parking System Application**

### First Page

The first page of the application features a "Get Started" button as shown in Figure 5.16. When tapped, it directs the user to the login page. As illustrated in Figure 5.17, tapping the button triggers successful navigation.



**Figure 5.16 First Page**



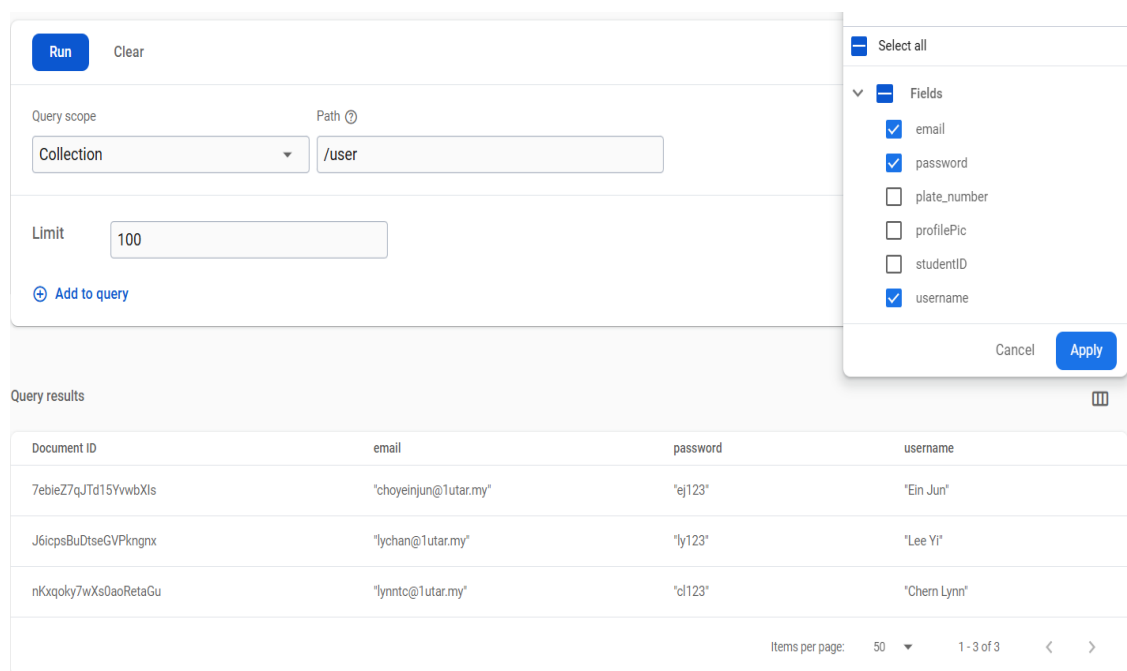
The image shows a login page for the 'VALET - Smart Parking System'. The background is a gradient of purple, blue, and green. At the top center is a white square with a red 'P' inside. Below this is the title 'VALET - Smart Parking System' in bold black text, followed by the subtitle 'Use the account below to sign in.' in a smaller font. Below the subtitle are two input fields: 'Email' and 'Password'. The 'Password' field has a toggle icon (an eye with a slash) on the right. Below the input fields is a blue 'Sign In' button.

**Figure 5.17 Login Page**

Testing confirmed that the "Get Started" button functions as intended, navigating users to the login screen without any issues.

## Login Page

On the login page, users are required to authenticate themselves by entering their UTAR email address and password. For testing purposes, three user accounts have been pre-created and stored in the database. The login credentials for these test accounts are displayed in Figure 5.18. By entering the provided email and password and pressing the "Sign In" button, users are able to successfully log in and navigate to the next page. Figures 5.19 to 5.21 demonstrate that the login information entered by users matches the records retrieved from the backend database.



The screenshot shows the Cloud Firestore console interface. At the top, there are buttons for 'Run' and 'Clear'. Below these, the 'Query scope' is set to 'Collection' and the 'Path' is '/user'. The 'Limit' is set to 100. A 'Select all' dialog is open on the right, showing the following fields: email (checked), password (checked), plate\_number (unchecked), profilePic (unchecked), studentID (unchecked), and username (checked). The 'Query results' section displays a table with the following data:

Document ID	email	password	username
7ebieZ7qJTd15YvwbXls	"choyeinjun@1utar.my"	"ej123"	"Ein Jun"
J6icpsBuDtseGVPkngnx	"lychan@1utar.my"	"ly123"	"Lee Yi"
nKxqoky7wXs0aoRetaGu	"lynntc@1utar.my"	"cl123"	"Chern Lynn"

At the bottom right, it shows 'Items per page: 50' and '1 - 3 of 3'.

**Figure 5.18 Query Data in Cloud Firestore**

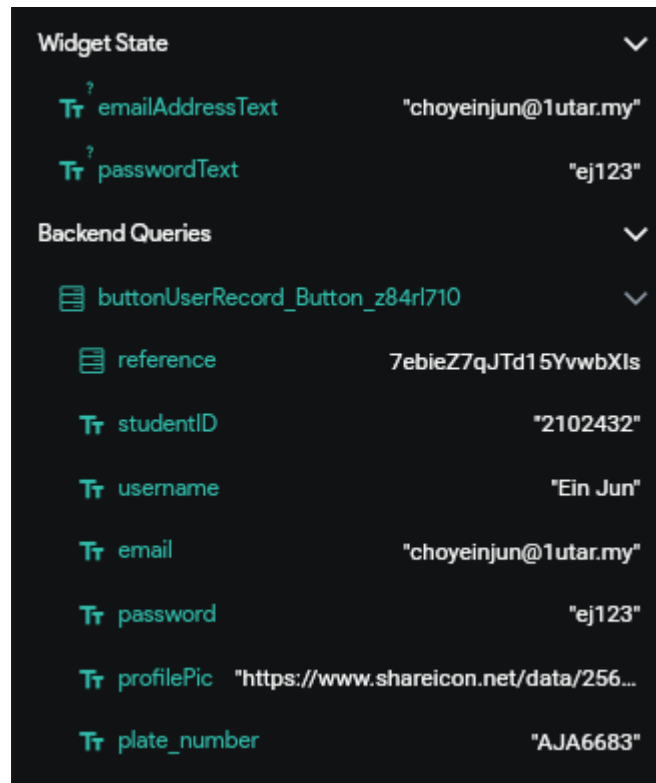


Figure 5.19 Successfully Sign Up for First User

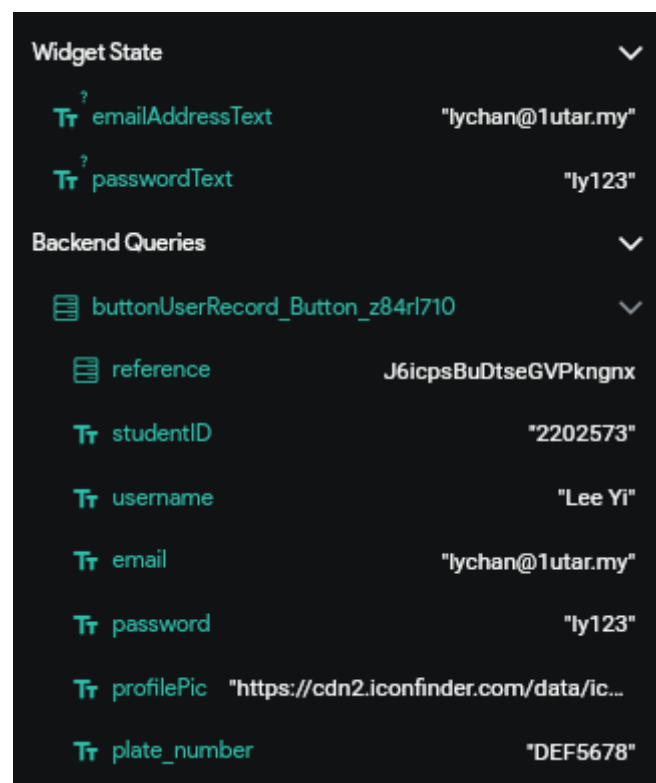
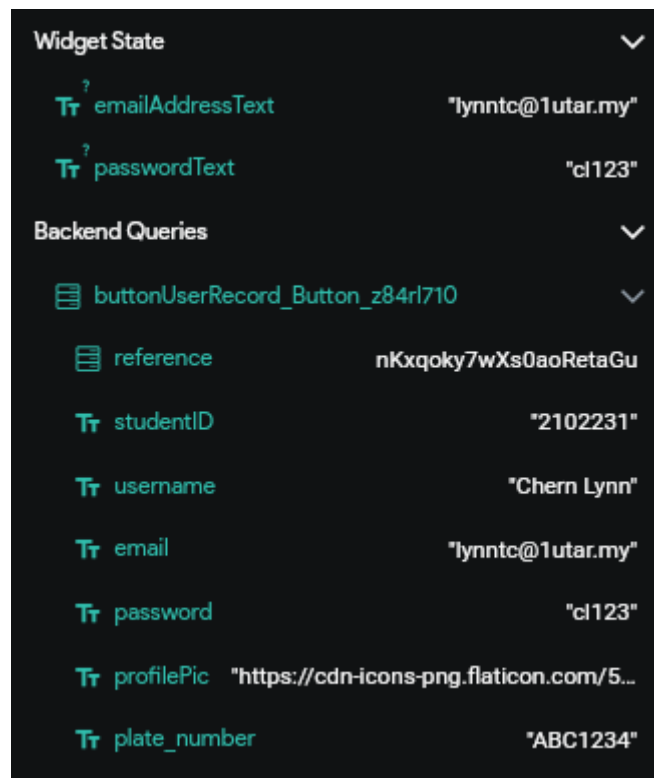


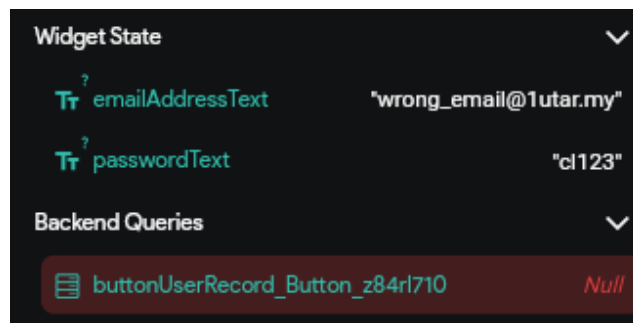
Figure 5.20 Successfully Sign Up for Second User



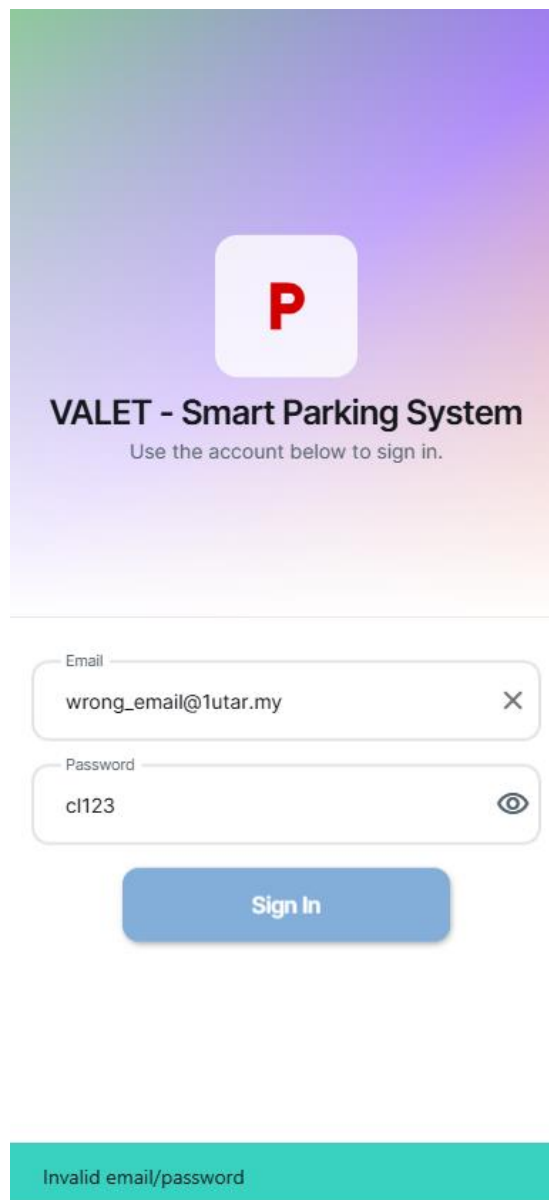
**Figure 5.21 Successfully Sign Up for Third User**

In cases where the login attempt is unsuccessful, an "Invalid email/password" error message will be displayed. This occurs under three conditions: when the user enters an incorrect email, an incorrect password, or both an incorrect email and password. Figures 5.22 to Figure 5.27 illustrate the three test cases corresponding to these conditions. Unless valid credentials are provided, users will not be able to access the next page of the application. This mechanism ensures that only authenticated users are allowed to proceed further into the system.

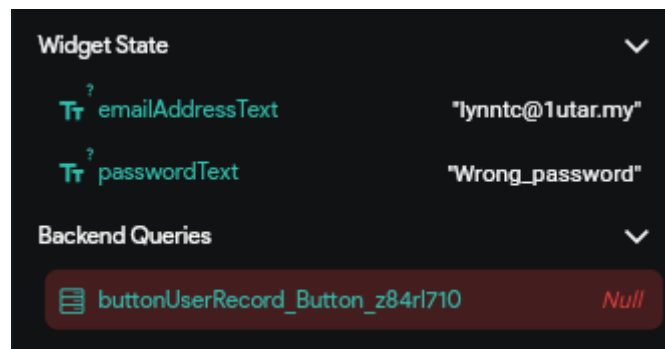




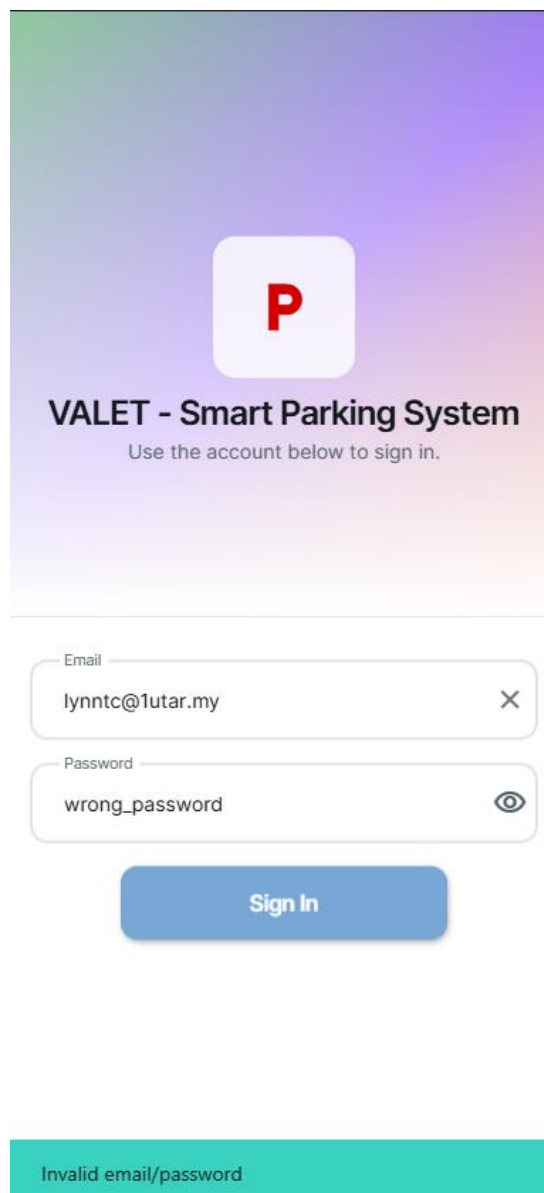
**Figure 5.22 Test Case for Wrong Email**



**Figure 5.23 Snack Bar Message for Wrong Email**



**Figure 5.24 Test Case for Wrong Password**



**Figure 5.25 Snack Bar Message for Wrong Password**

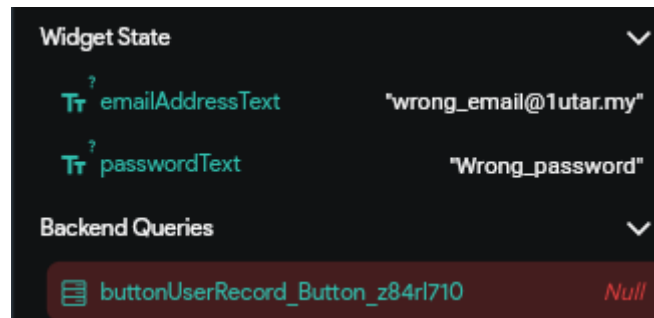


Figure 5.26 Test Case for Wrong Email and Password

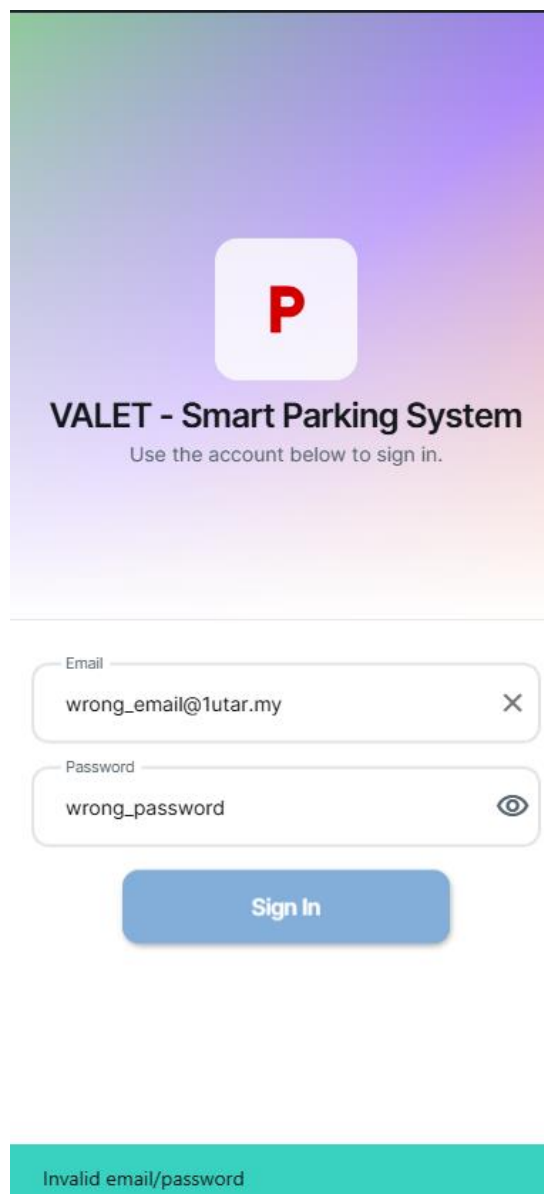
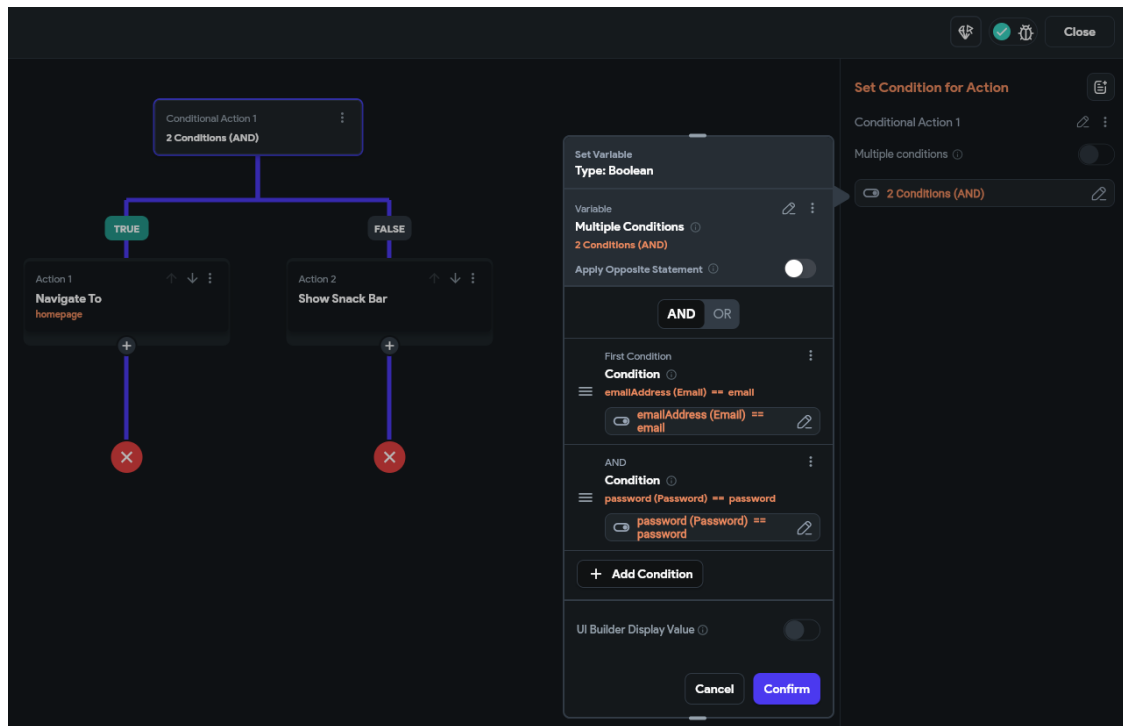


Figure 5.27 Snack Bar Message for Wrong Email and Password

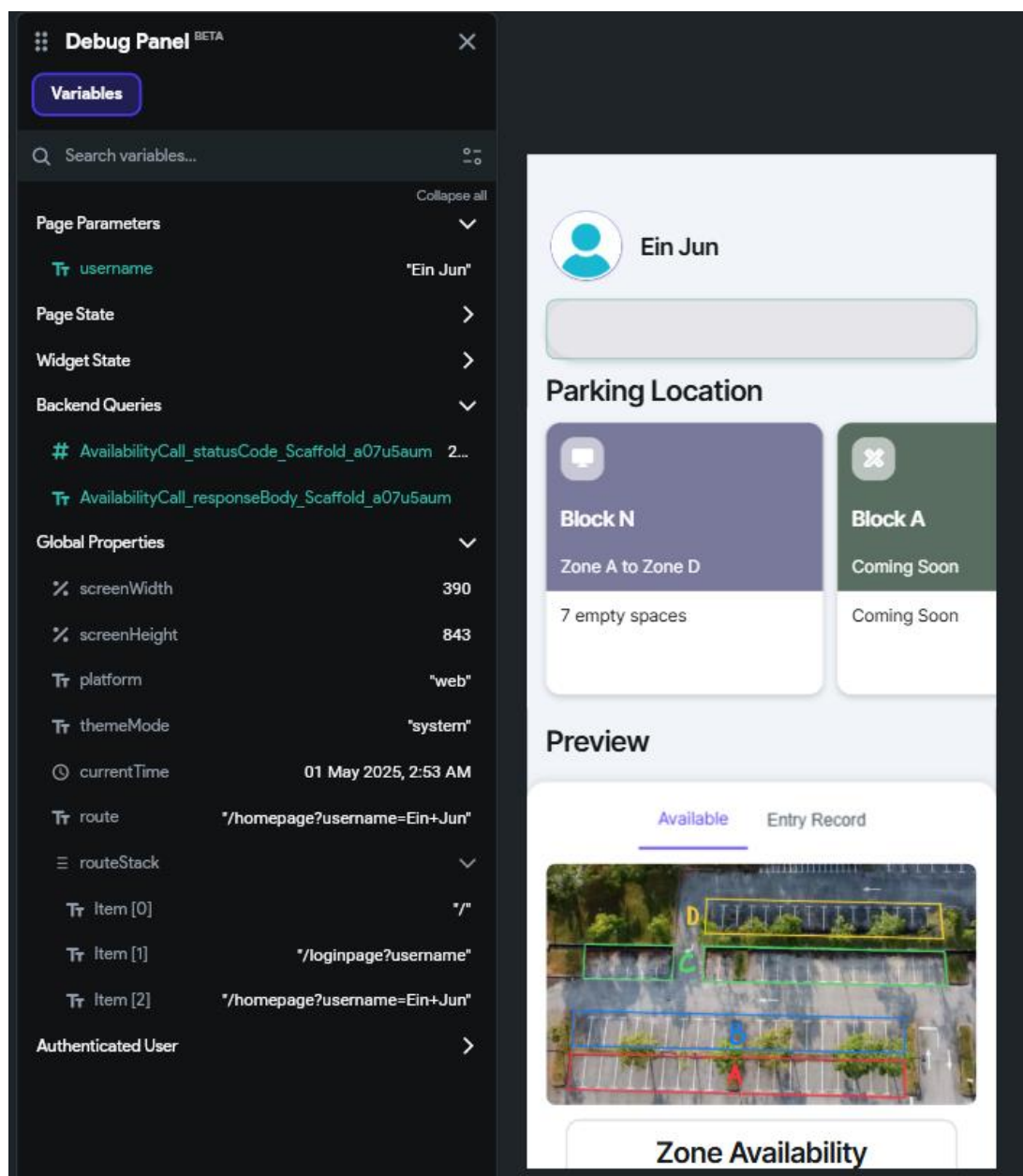
Figure 5.28 presents the multiple conditional checks configured for the "Sign In" button. The logic verifies whether the entered email matches the one stored in the database and whether the entered password is also correct. If both conditions are satisfied, the function returns true and navigates the user to the home page. Otherwise, a snack bar is displayed with the message: "Invalid email/password."



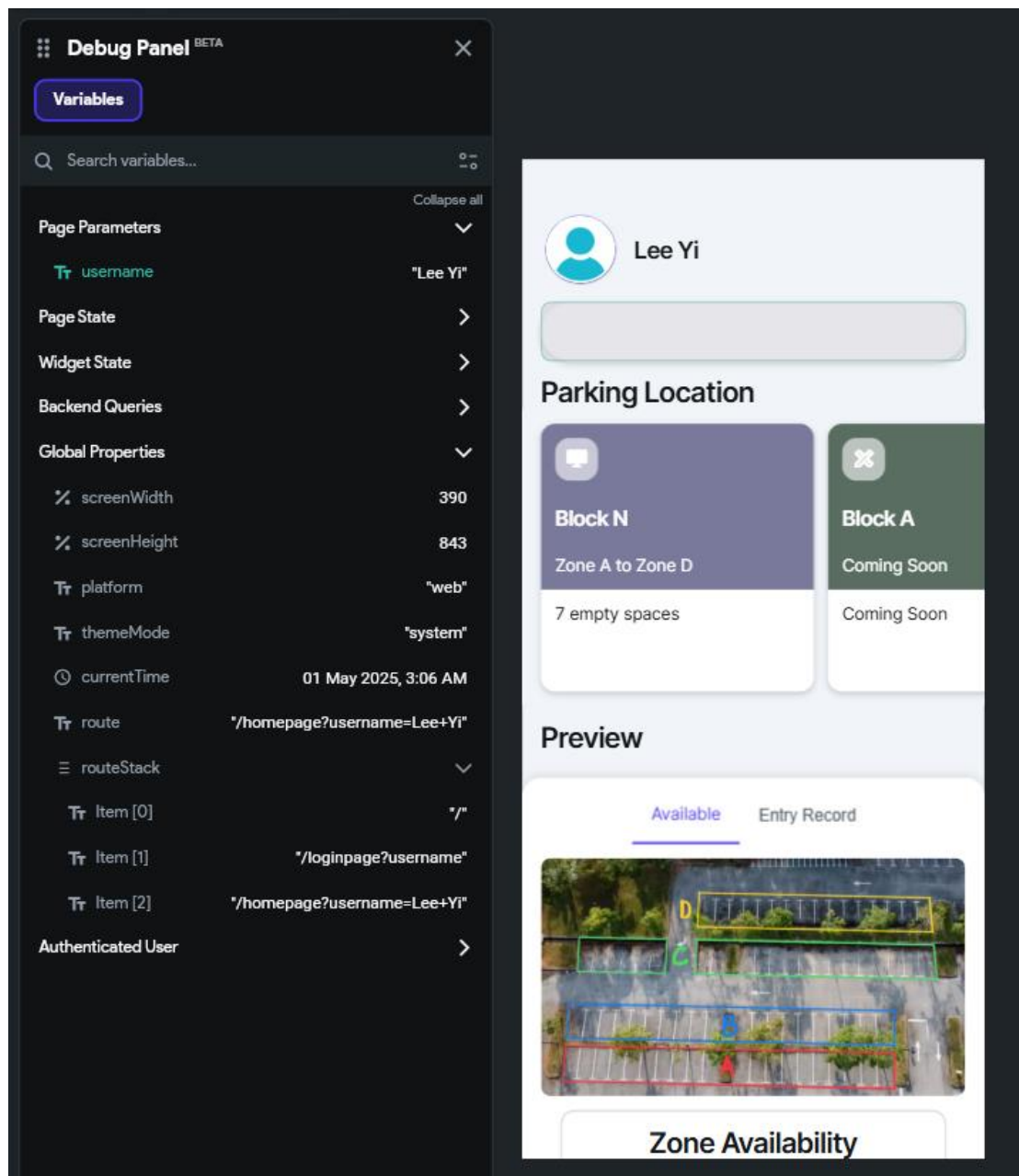
**Figure 5.28 Multiple Conditions in Sign In Button**

## Home Page

After a successful sign-in, the user is navigated to the home page, which serves as the central navigation hub. Figures 5.29 to Figure 5.31 show different views of the test mode home page, where the displayed username is passed as a parameter during navigation. Upon reaching the home page, the application retrieves the username from the passed parameter and displays it dynamically. This method ensures that each user sees personalized information without needing to query the database again, improving both efficiency and user experience.



**Figure 5.29 Home Page for First User in Test Mode**



**Figure 5.30 Home Page for Second User in Test Mode**

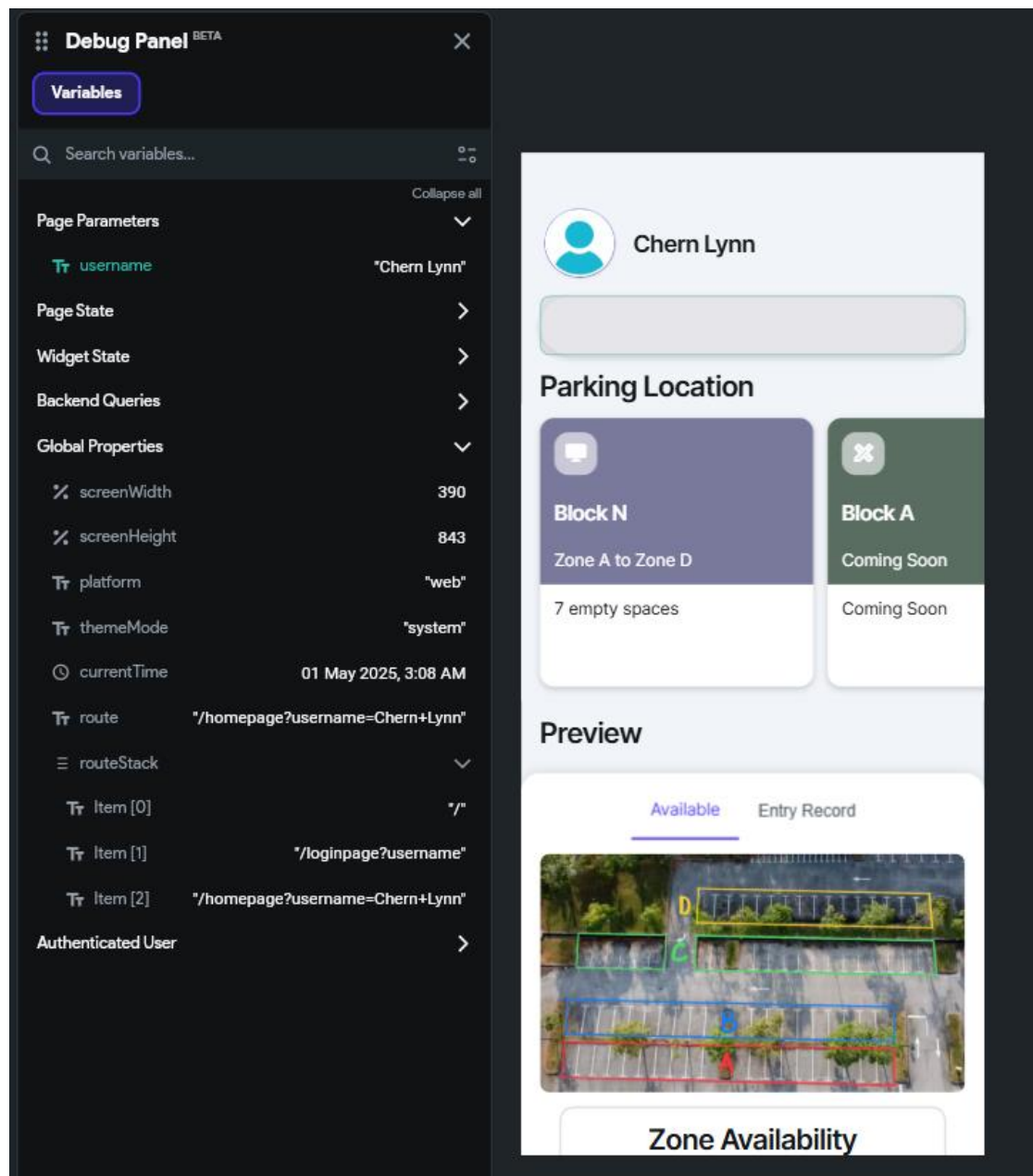
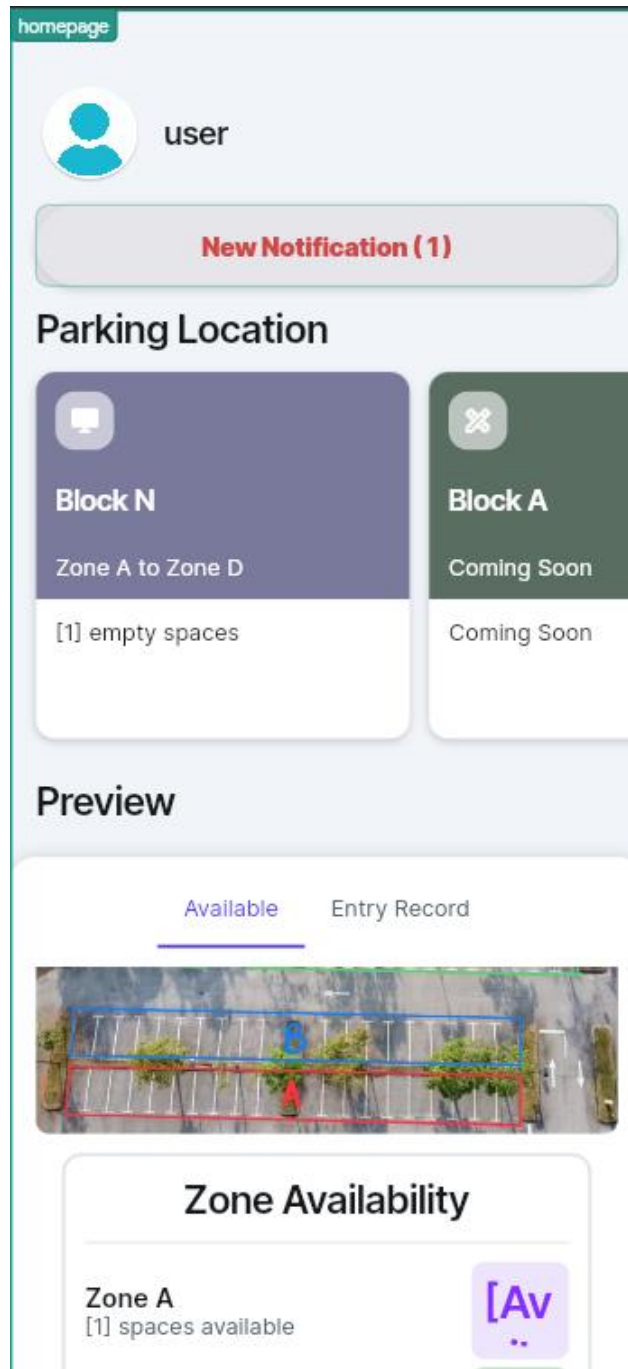


Figure 5.31 Home Page for Third User in Test Mode

Figure 5.32 illustrates the Home Page in preview mode, where real-time data is retrieved directly from Firebase. The interface is designed to dynamically display current parking availability by fetching live data for both zones and blocks. In this mode, elements enclosed in square brackets, such as [1], represent data placeholders that will be replaced with real-time values from the Firebase Realtime Database during actual use.

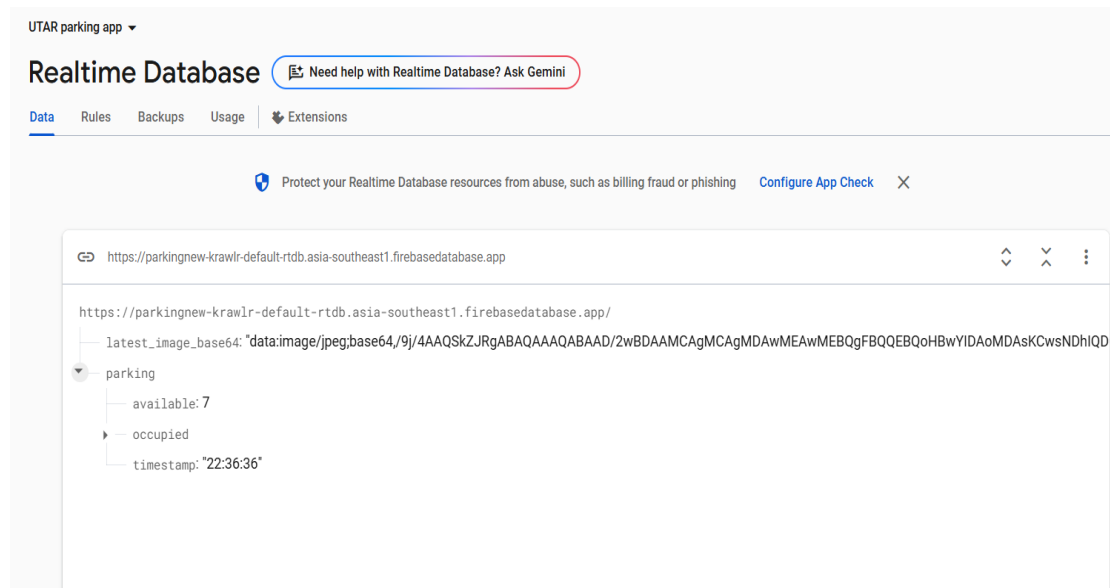


**Figure 5.32 Home Page in Preview Mode**



Specifically, the preview shows two main types of dynamic content: one inside block N box and another representing the availability of zone A within that block. The number displayed in the block's box provides a quick summary of total available parking bays in that block, helping users assess whether the block still has empty spaces. Below that, each block is further broken down into zones, with real-time indicators showing how many bays are available in each zone. This detailed breakdown allows users to identify which specific areas or rows within the parking lot still have free spaces, enabling more targeted and efficient navigation. The structure is designed to guide users from a broad overview of block availability down to the zone level, supporting informed decision-making and minimizing the time spent searching for a parking spot.

The real-time data can continuously be monitored through the Firebase Console, as shown in Figure 5.33. This console provides a live interface to view and track the contents of the Realtime Database, making it easier for developers to observe updates and ensure that the system is functioning correctly. It serves as a valuable tool for verifying whether the application is accurately storing and retrieving data in real time.



**Figure 5.33 Realtime Database in Firebase Console**

To enable data storage from the Raspberry Pi to Firebase using Python, the Firebase Admin SDK needs to be installed first. This can be done by executing the command `pip install firebase-admin`. After the SDK is successfully installed, Firebase must be initialized within the Python script, as demonstrated in Figure 5.34.

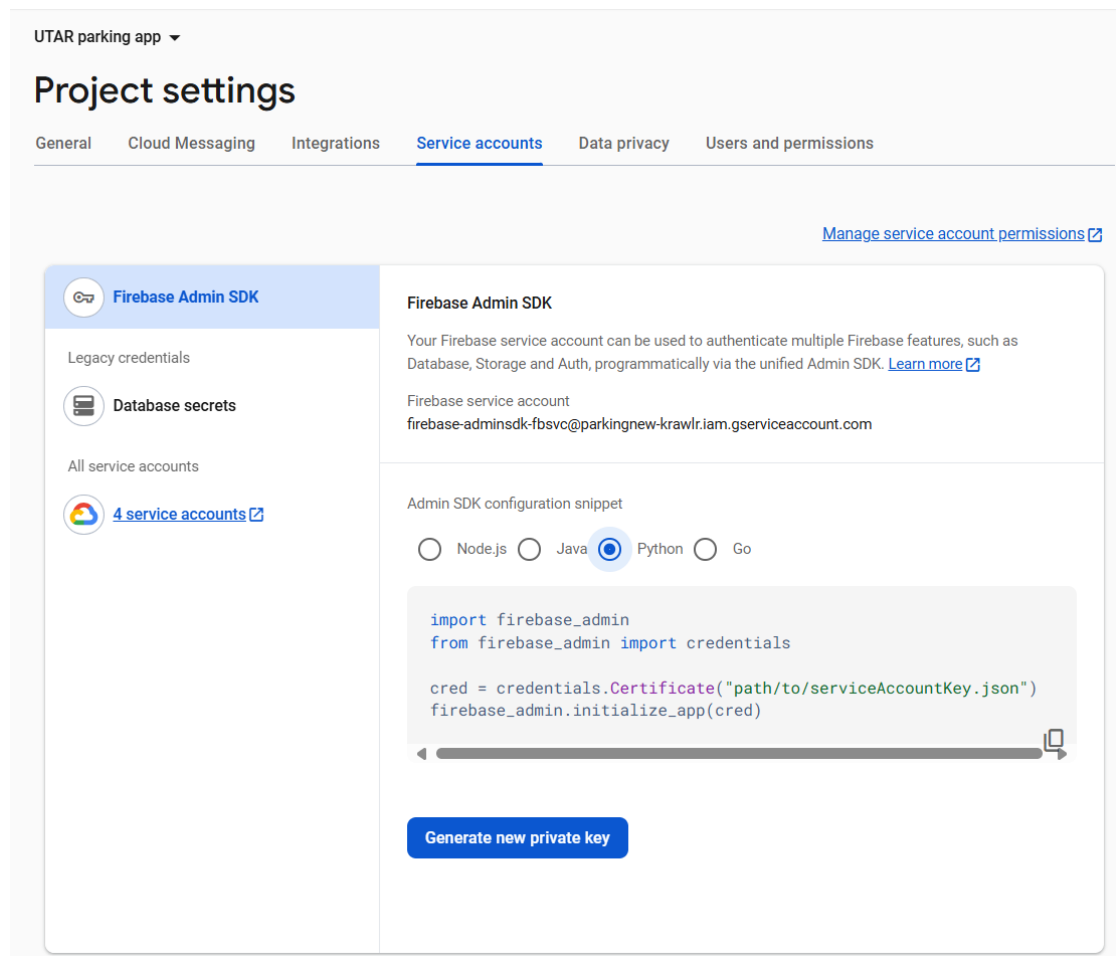
```

1 import numpy as np
2 import pandas as pd
3 import cv2
4 from picamera2 import Picamera2
5 from ultralytics import YOLO
6 import time
7 import threading
8 from flask import Flask, Response
9 from flask_cors import CORS
10 import signal
11 import sys
12 import firebase_admin
13 from firebase_admin import credentials, db
14 from datetime import datetime
15 import base64
16
17 # Initialize Flask app with CORS support
18 app = Flask(__name__)
19 CORS(app)
20
21 # Initialize Firebase
22 cred = credentials.Certificate("/home/lynn/Desktop/parking/serviceAccountKey.json")
23 firebase_admin.initialize_app(cred, {
24     'databaseURL': 'https://parkingnew-krawlr-default-rtdb.asia-southeast1.firebaseio.com'
25 })
26

```

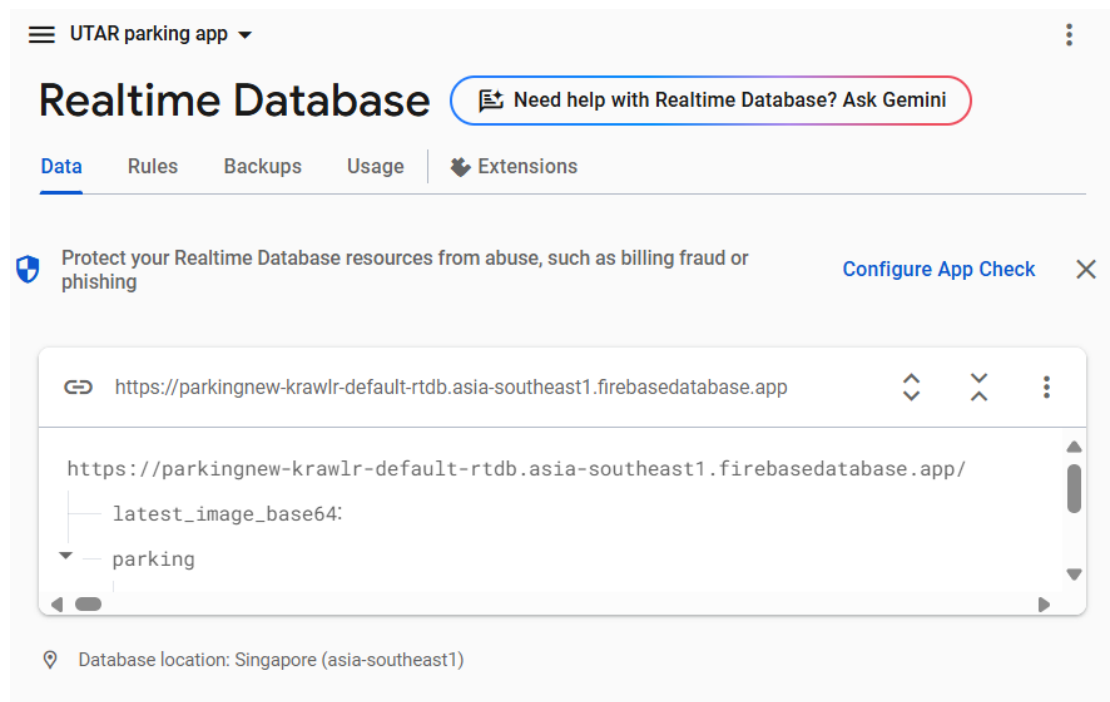
**Figure 5.34 Python Script Involve Setting of Firebase**

A crucial component of this setup is the `serviceAccountKey.json` file, which acts as a credential for secure access. This file must be downloaded from the Firebase Console by navigating to the "Service Accounts" tab and clicking on the "Generate new private key" button, as shown in Figure 5.35. Once downloaded, the file should be saved in an accessible directory for the script to reference. In this example, the file is placed inside the `/home/lynn/Desktop/parking` directory.



**Figure 5.35 Service Accounts Tab in Firebase Console**

Furthermore, the database URL required for connecting to Firebase is defined in line 24 of the parking.py script. This URL is essential for directing the script to the correct Firebase project. It can be found in the Realtime Database section of the Firebase Console, as highlighted in Figure 5.36. Including the correct URL ensures that all read and write operations are properly directed to the intended database instance.



**Figure 5.36 URL of Firebase**

### Notification

In reference to Figure 5.32, the preview mode clearly shows the label “New Notification (1)” embedded inside the notification button. However, this feature behaves differently during actual testing, as illustrated in Figures 5.29 to 5.31. In testing mode, the button container does not respond to user interactions, and the notification text is absent. This discrepancy is not a bug, but a deliberate feature controlled by the button settings and logic defined in Figures 5.37 and 5.38.

Specifically, as seen in Figure 5.37, the button is configured with a logic condition that disables it when certain criteria are not met. The conditional logic applied, detailed in Figure 5.38, checks the parking availability counter. If there are still available parking bays, which means the counter is not zero, the button remains disabled and hidden. Only when the number of available parking bays reaches zero which indicating the car park is fully occupied, the button becomes active, and the “New Notification (1)” text appears. This setup ensures that the notification is meaningful and contextually relevant by only appearing when user attention is truly needed, i.e., when the car park is full.

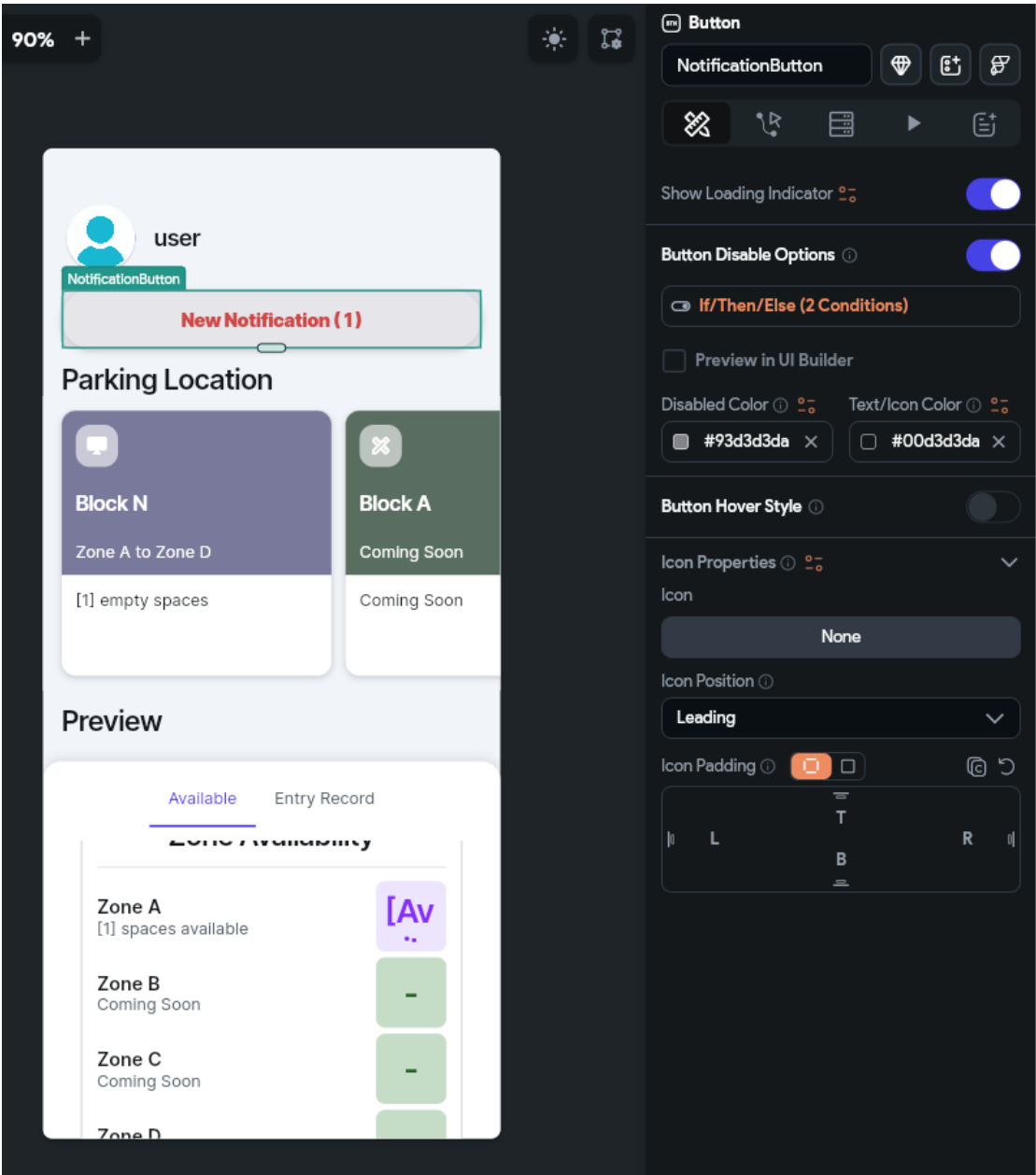
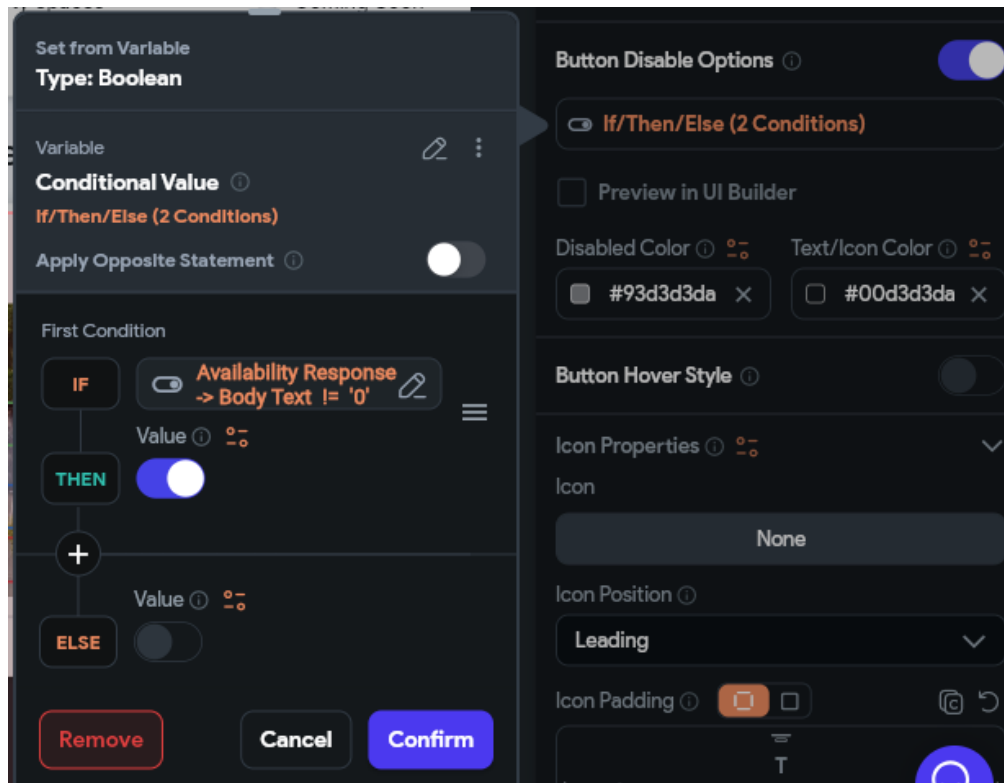
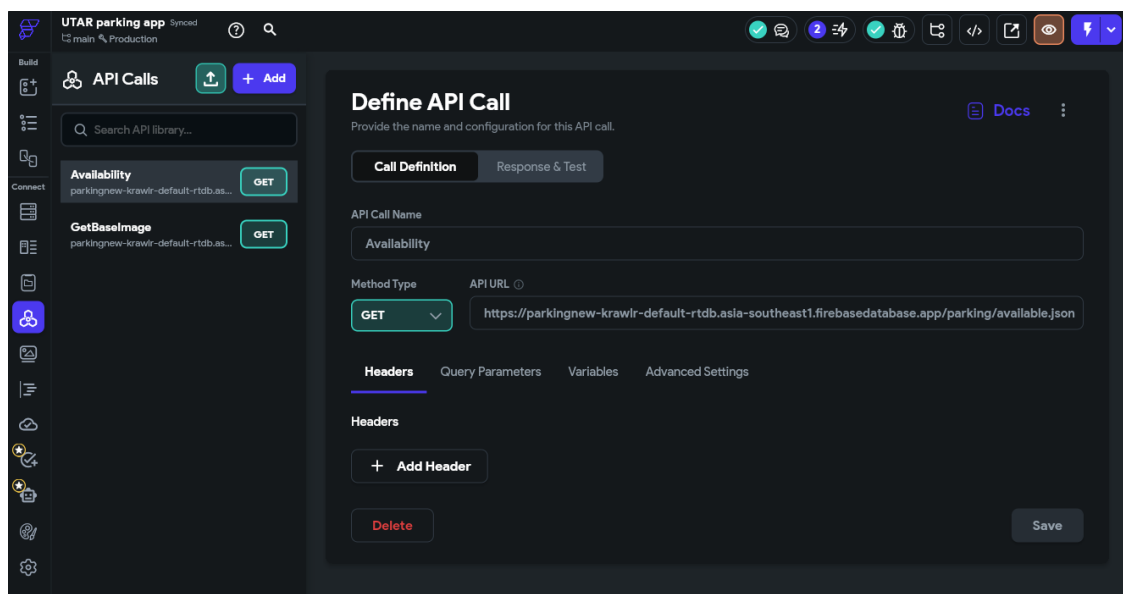


Figure 5.37 Button Disable Options



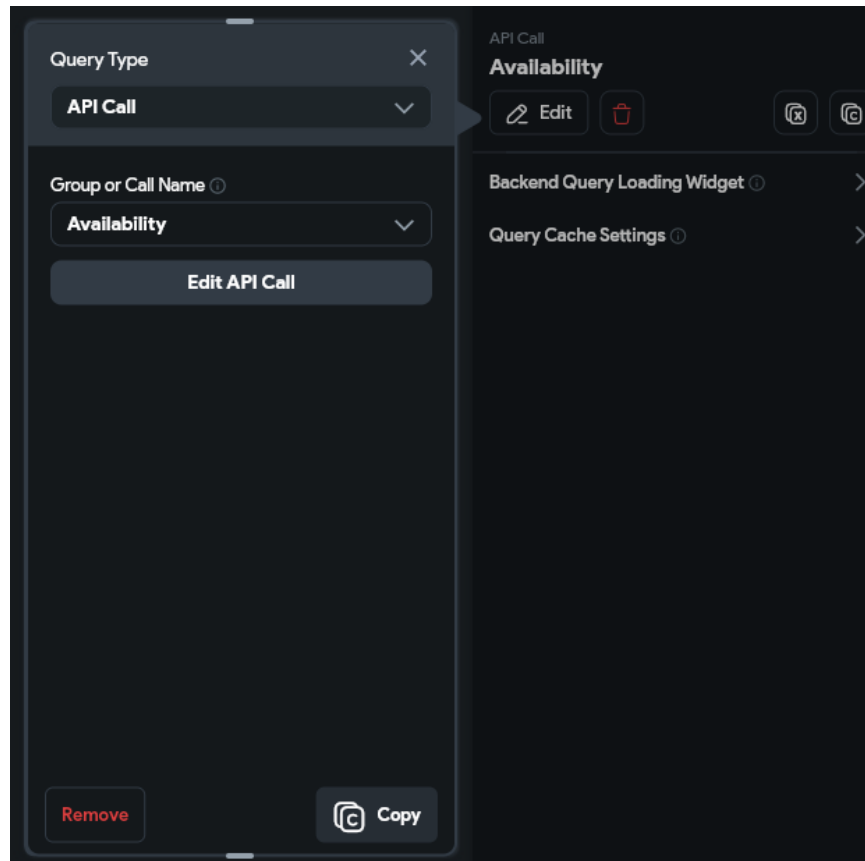
**Figure 5.38 If-else Condition Applied in Notification Button**

This behavior is driven by real-time data retrieved from the Firebase Realtime Database using an API call. Within the conditional logic, the “Availability Responses” refers to the response data returned by this API, which queries the path `/parking/available.json` in Firebase, as illustrated in Figure 5.36.



**Figure 5.39 Create API Call**

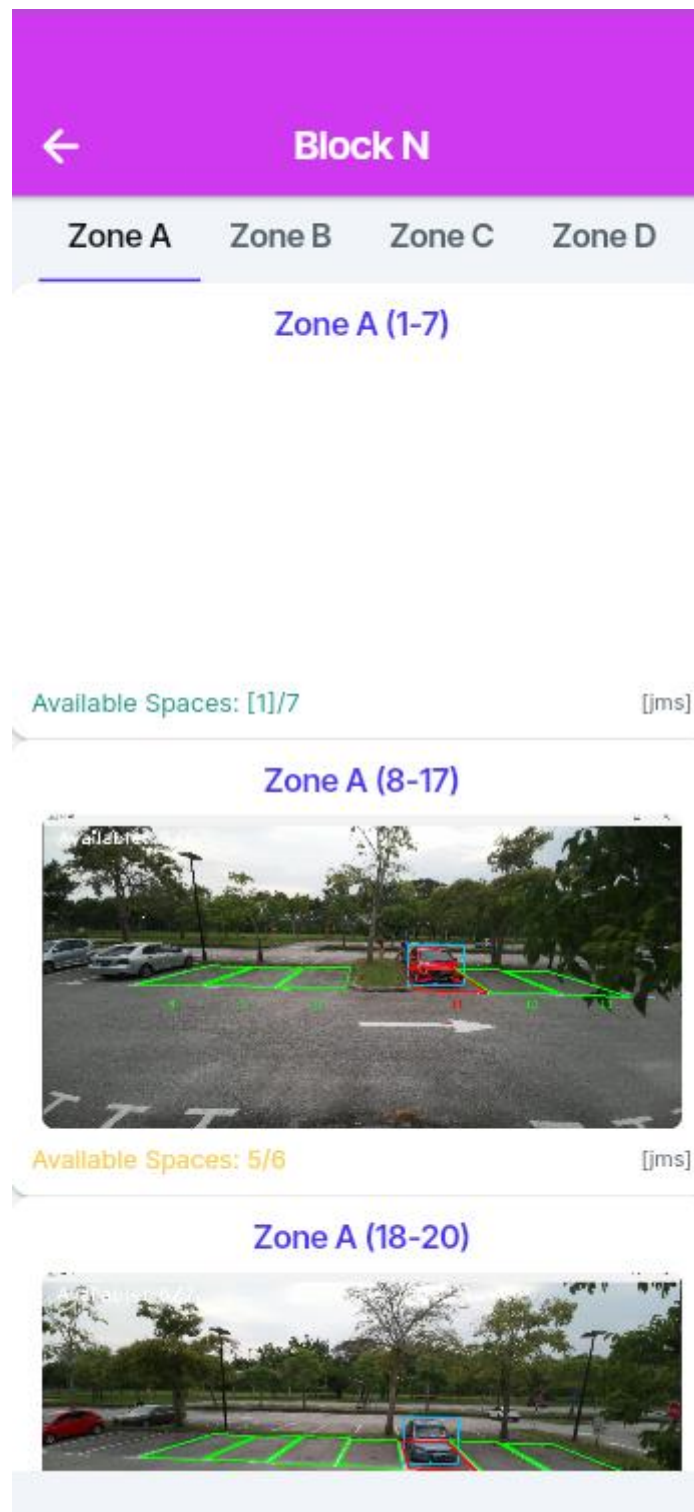
To use this data in widgets such as text displays or buttons, the API must first be created and integrated into the app's backend using FlutterFlow. As demonstrated in Figure 5.39, a custom API call named "Availability" is created. This API is then bound to the current page by navigating to the Backend Query tab in FlutterFlow, selecting "Add Query", and linking it to the API call. The configuration for how this API is connected and used within the app is shown in Figure 5.40.



**Figure 5.40 Setting to Apply API Call Named Availability**

By implementing this system, the app ensures a smarter and more user-friendly notification mechanism that reduces unnecessary alerts and focuses only on conditions that require the user's attention, specifically when the car park is fully occupied.

**Block N Page**



**Figure 5.41 Preview Mode**



In preview mode, the only functional component is the camera view that displays the number of available parking spaces. This data is retrieved from the Firebase Realtime Database and shown using a dynamic pattern enclosed in square brackets [ ], rather than as static text. Adjacent to the parking space counter, the [jms] placeholder represents the timestamp in the format hour:minute:second AM/PM. This timestamp indicates when the image was last updated, helping users verify whether the camera feed is refreshing correctly.

In Figure 5.41, the image is not shown because preview mode does not activate the live camera feed. This demonstrates that the image widget functions only when the camera is connected, and the Python script is actively running. It does not rely on a fixed or static image. The method used to display the live image in the app involves converting the captured image to a Base64 string.

To achieve this, the encoded image is prefixed with /latest\_image\_base64 and stored in the Firebase Realtime Database. The Python code responsible for this encoding and storage is shown in Figure 5.42.

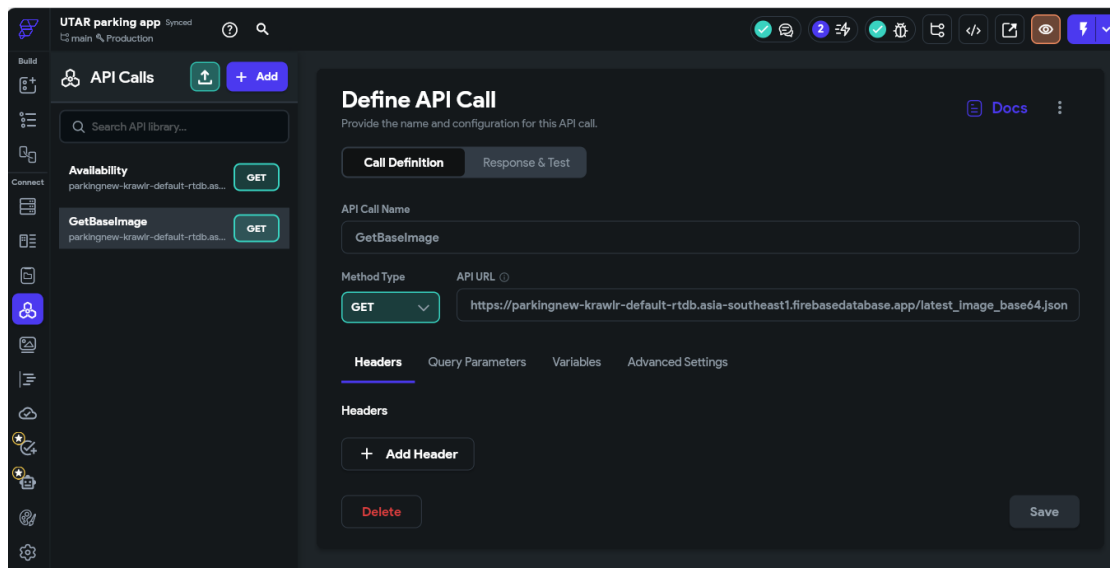
```

143 @app.route('/latest_image_base64')
144 def latest_image_base64():
145     try:
146         image_path = "/home/lynn/Desktop/parking/latest_output.jpg"
147         with open(image_path, "rb") as f:
148             encoded_string = base64.b64encode(f.read()).decode('utf-8')
149             data_uri = f"data:image/jpeg;base64,{encoded_string}"
150             db.reference("/latest_image_base64").set(data_uri)
151             return {"status": "success", "length": len(data_uri)}, 200
152     except Exception as e:
153         return {"error": str(e)}, 500

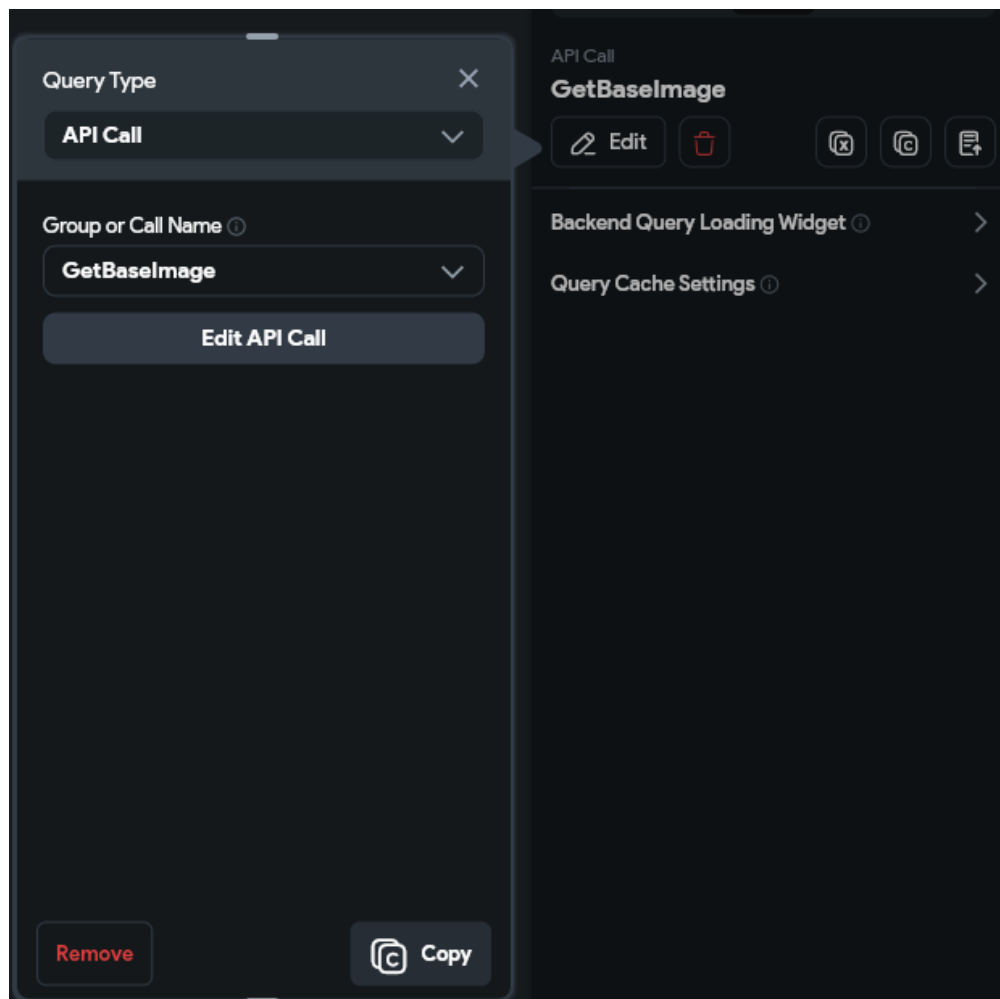
```

**Figure 5.42 Code Snippet for Storing Encoded String**

To retrieve this encoded string in the app, an API call is created as shown in Figure 5.43, and configured in FlutterFlow under the name GetBaseImage, as detailed in Figure 5.44. This ensures that the image is properly rendered on the mobile app without decoding errors.



**Figure 5.43 Create API Call to Get Encoded String**

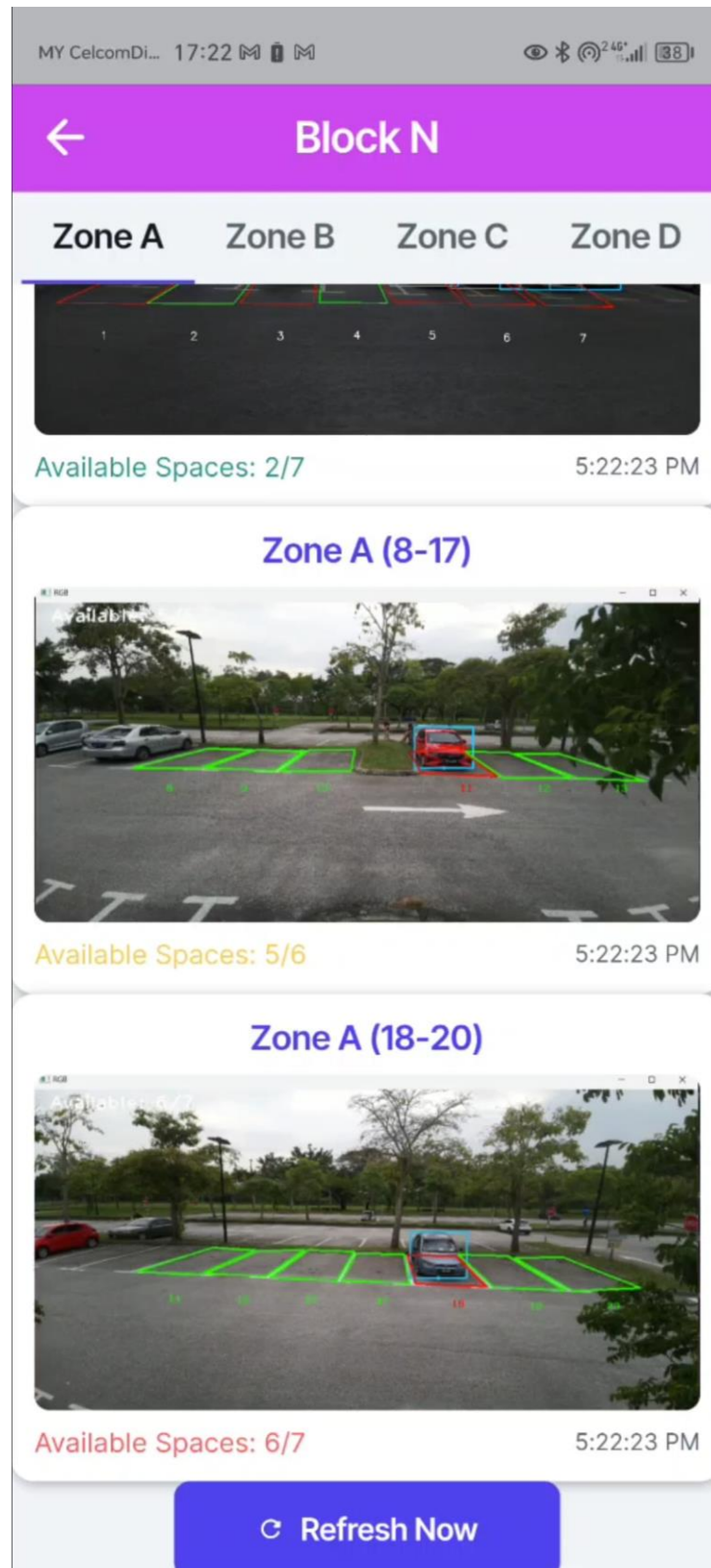


**Figure 5.44 Setting to Apply API Call Named GetBaseImage**

Although the camera is configured to capture and upload a new image every 30 seconds, the app does not automatically refresh the image if the user remains on the Block N Page continuously. This is due to limitations in the free plan of FlutterFlow, which restricts real-time automatic updates within a single page unless the user performs an action.

To overcome this limitation, a "Refresh Now" button is placed below the image on the page. This button allows users to manually reload the most recent image if they choose to stay on the same page and continuously monitor the parking situation. However, this limitation typically does not affect the user experience during standard or quick usage. In normal viewing situations, such as navigating to or returning from another page, the image will automatically update with the latest version without requiring the user to press the refresh button.

This design remains practical and user-friendly. For instance, if a user visits the page and sees that no parking spaces are currently available, they can remain on the page and press the "Refresh Now" button to check whether any vehicle has left. This allows them to stay informed without needing to leave or reload the app. The Refresh Now button, as shown in Figure 5.45, ensures that the user retains control over the image update process when continuous viewing is needed.



**Figure 5.45 Refresh Now Button in Block N Page**

### 5.5 Implementation Issues and Challenges

Implementing a real-time car parking location detection system presents several challenges, particularly concerning camera placement and environmental conditions. The location and angle of the cameras are critical for accurate detection; improperly placed cameras may fail to capture all vehicles, leading to undetected parked cars and incorrect assessments of available spaces. Given the limited resources around the car park area, such as light posts and walls, achieving a wide and high angle of view is challenging. This limitation necessitates installing more cameras to cover all parking spaces, increasing both the cost and the need for suitable installation locations, like light posts.

Coverage is another concern, as cameras must have a sufficient field of view to monitor all parking spaces effectively, requiring careful planning to avoid blind spots. Additionally, overlapping detection of parking spaces can lead to inaccurate counting. Given the large area to cover, precise labelling of parking spaces is essential to prevent overlapping detections. Peak hours also pose a significant challenge, as multiple cars entering simultaneously can overwhelm the system, making it difficult to track each vehicle accurately.

Environmental factors like weather and lighting conditions further complicate the implementation. Rain, and bright sunlight can obscure the camera's view, affecting the quality of detection. Nighttime or low-light conditions demand cameras with night vision or additional lighting, adding complexity to the system setup. Temperature extremes may also impact camera hardware, leading to potential malfunctions. Ensuring a reliable power supply for the cameras, especially in outdoor settings, is crucial to prevent disruptions in monitoring. Addressing these challenges requires a comprehensive approach, involving robust equipment, strategic placement, and advanced image processing capabilities to maintain accuracy and reliability in various conditions.

### 5.6 Concluding Remark

This chapter has provided a detailed overview of the implementation of the Smart Parking System, encompassing both hardware and software setups, configurations, and the operation of the system. The section also delved into the various challenges encountered during the development and testing phases. Specifically, the system's functionality was highlighted through screenshots that illustrate its operation, from the first page and login process to the dynamic display of parking availability on the home page and Block N Page. These aspects demonstrate the integration of both the front-end user interface and back-end real-time data processing to offer a seamless user experience.

The system operation section outlined the functionality of each key page in the application, showing how they interact with both the user and the backend. The first page provides a clean entry point to the app, featuring a “Get Started” button that directs users to the login interface. The login screen authenticates users based on predefined credentials stored in Firebase, ensuring only authorized individuals can access the system. Once authenticated, users are taken to the home page, which displays the real-time availability of parking spaces through a color-coded layout. This data is dynamically retrieved and updated using Firebase Realtime Database, allowing users to see the current status without needing to refresh the page.

The Block N Page builds upon this functionality by integrating a camera feed that visually confirms the parking status of a specific zone. Instead of streaming the video directly, the system captures an image from the camera every 30 seconds, converts it to a base64-encoded string using the Python backend, and uploads it to Firebase. The mobile app then fetches and displays the latest image, giving users a near real-time view of the parking condition. While the page does not automatically refresh due to limitations in the FlutterFlow free plan, a "Refresh Now" button allows users to manually reload the image if they remain on the page.

Additionally, the notification system is designed to alert users only when critical conditions, such as full parking availability, arise. This ensures that the notifications are contextually relevant, reducing unnecessary distractions and maintaining an

intuitive user experience. The Block N Page extends this functionality by displaying a live camera feed, providing users with a continuous overview of parking space status.

However, the implementation of such a system also presents challenges, particularly related to hardware setup, environmental factors, and real-time performance limitations. Issues such as camera placement, and the impact of weather conditions have been identified as critical factors to consider when deploying the system in a real-world scenario. Additionally, the constraints imposed by the free plan of FlutterFlow, such as the inability to refresh images in real time unless manually triggered, present limitations in continuous monitoring.

Despite these challenges, the application demonstrates a reliable solution for real-time parking monitoring, offering users an efficient way to navigate available parking spaces. Moving forward, addressing the identified issues—such as ensuring precise camera coverage, improving environmental resilience, and optimizing real-time data synchronization—will be crucial for enhancing the system's performance and scalability. The implementation of a stable power supply and advanced image processing techniques will be essential in maintaining consistent service, especially under varying weather or lighting conditions.

In conclusion, the Smart Parking System provides a promising approach to streamlining parking management. With ongoing improvements and enhancements, it has the potential to become a valuable tool for both urban and commercial parking systems, offering real-time insights that benefit both users and operators.

## Chapter 6

# System Evaluation and Discussion

### 6.1 System Testing and Performance Metrics

System testing is essential to ensure that the Smart Parking System application is reliable, functional, and easy to use. The main goal of testing is to check that every part of the app works according to the design requirements and to find any problems that could affect the system's performance or user experience.

Before testing the whole app, it is crucial to ensure that each individual function works accurately before integration. Each independent function is tested to ensure there are no issues when integrated and performing within the application. In this section, all individual functions are tested. The overall app functionality will be tested in the next section, which includes details such as the testing setup and results.

#### 6.1.1 License Plate Detection and Recognition

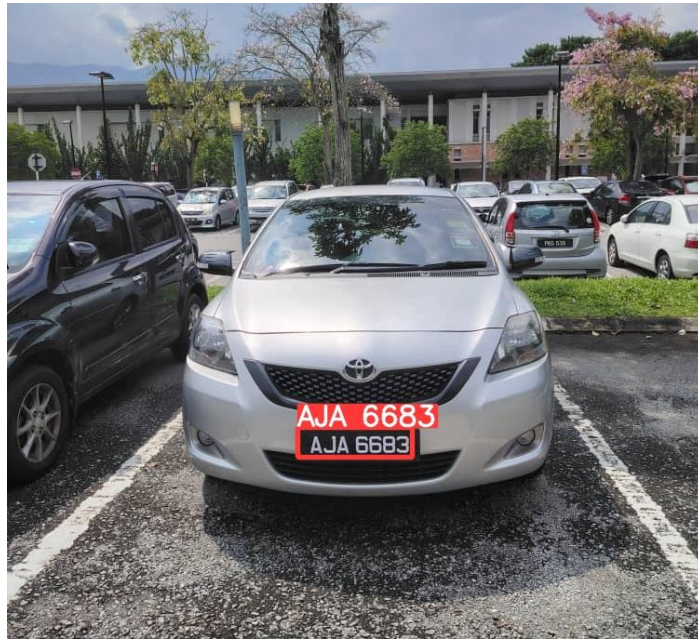
Figure 6.1 shows the results of the license plate detection using a dataset specifically trained for Malaysian license plates. This figure highlights the capability of the system to accurately detect license plates within the captured images, serving as a critical component of the overall vehicle identification process.



**Figure 6.1 License Plate Detection**



Further, Figure 6.2 demonstrates the application of OCR using the best-trained model (best.pt) from the license plate detection task. This step involves reading and recognizing the license plate numbers, which are essential for identifying and logging vehicle entries. Once the parking entrance camera is installed, it will detect moving cars, automatically recognize their plate numbers, and record them in our database.



**Figure 6.2 License Plate Number Recognition**

Although license plate detection and recognition are not the focus of the current stage of the project due to issues with camera access and permissions for recording all vehicles entering the car park, accuracy testing has been conducted to verify the reliability of the function for basic use. Once the permissions issue is resolved, this feature will serve as a foundational element for future work, making subsequent development easier. However, the system is not yet reliable for full integration, and further testing on moving vehicles and under different lighting and weather conditions is needed to improve stability and accuracy. As the system is not a priority for integration at this stage, it is sufficient for the concept phase.

The results of the accuracy testing for the license plate recognition system are detailed in Table 6.1, confirming its ability to recognize license plates accurately, with room for further improvement.

**Table 6.1 Accuracy Tests**

Image	Plate Number Recognized	Actual Plate Number
	MAM 45	MAM 45
	WA 688 M	WA 688 M
	WA 863 S	WA 863 \$
	DCW 7486	DCW 7486
	WYG 5361	WYG 5361
<b>Recognized correctly</b>		4/5
<b>Accuracy</b>		80%

These results indicate an 80% accuracy rate for license plate recognition, with 4 out of 5 plate numbers being correctly identified. These results demonstrate that the license plate recognition system is working as expected, although further adjustments are needed to enhance its accuracy.

### 6.1.2 Parking Area Module

To achieve comprehensive coverage of the entire car park, a systematic approach that integrates parking space labelling, detection, and counting functions using image stitching is employed to address the challenge of capturing a complete view due to the limitations of camera angles. Cameras are strategically positioned around the car park to capture different sections, with each camera focusing on a specific part of the parking area. This setup ensures that every parking space is covered, though overlapping sections are captured due to the limited field of view from each camera.

Labelling is crucial in this process as it assigns each parking space to a specific camera view. This labelling prevents overlapping detections by ensuring that no parking space is monitored by more than one camera. Each camera is responsible for monitoring its designated section, avoiding redundancy and ensuring that every parking space is accurately covered.

Captured images from different cameras are aligned based on similar angles to verify the effectiveness of the proposed detection method. Although the stitching process integrates the images to create a cohesive view, the primary focus remains on ensuring that each camera covers its designated section effectively.

The boundaries of each parking space are defined using a mouse function to obtain coordinates for each point of the parking space's bounding box. Initially, these bounding boxes are manually defined using four points to outline each space accurately.

Finally, the system utilizes the images from all cameras to monitor the entire car park in real-time. Despite the multiple views, this approach ensures that each parking space is effectively monitored by only one camera, providing accurate detection and counting. By carefully managing camera views and employing image stitching techniques, the system ensures complete and accurate monitoring of the car park, overcoming the constraints of camera placement and angles.

### **Parking Space Labelling, Detection and Counter**

Figures 6.3 to Figure 6.5 show images of the first 20 parking lots in the first row, captured using the Raspberry Pi Camera Module 3. These images give a visual overview of the parking area before the cameras were installed in a fixed position. They act as the initial data input for the parking detection system, helping to test and setup the system to accurately monitor and detect the parking space usage. By capturing these segments of the parking row, labelling each parking lot is necessary to prevent overlapping in the parking space detection across different camera views.



**Figure 6.3 View 1 Captured by Raspberry Pi Camera Module 3**



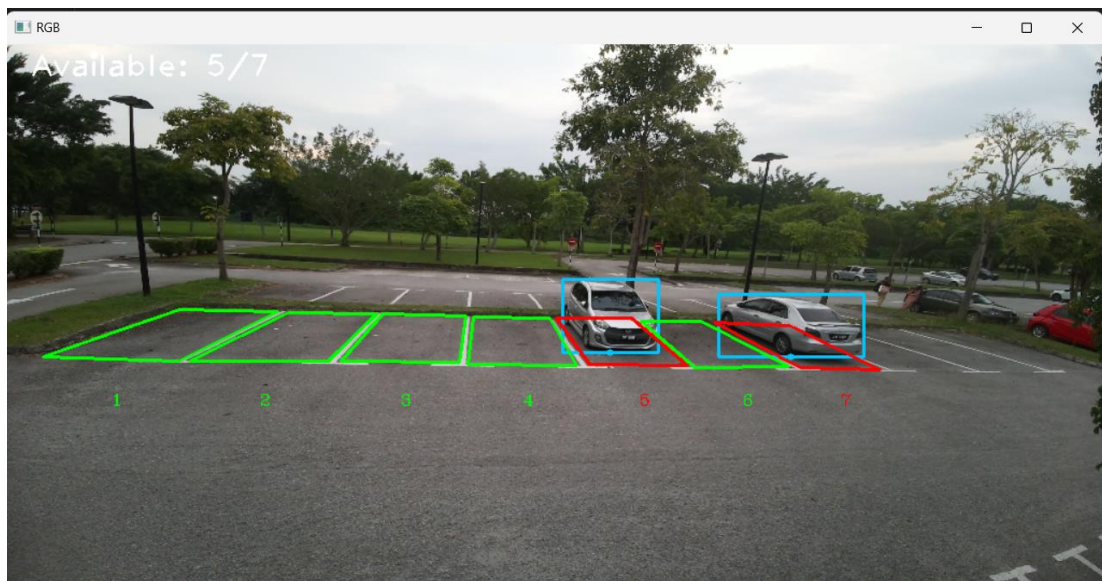


**Figure 6.4 View 2 Captured by Raspberry Pi Camera Module 3**

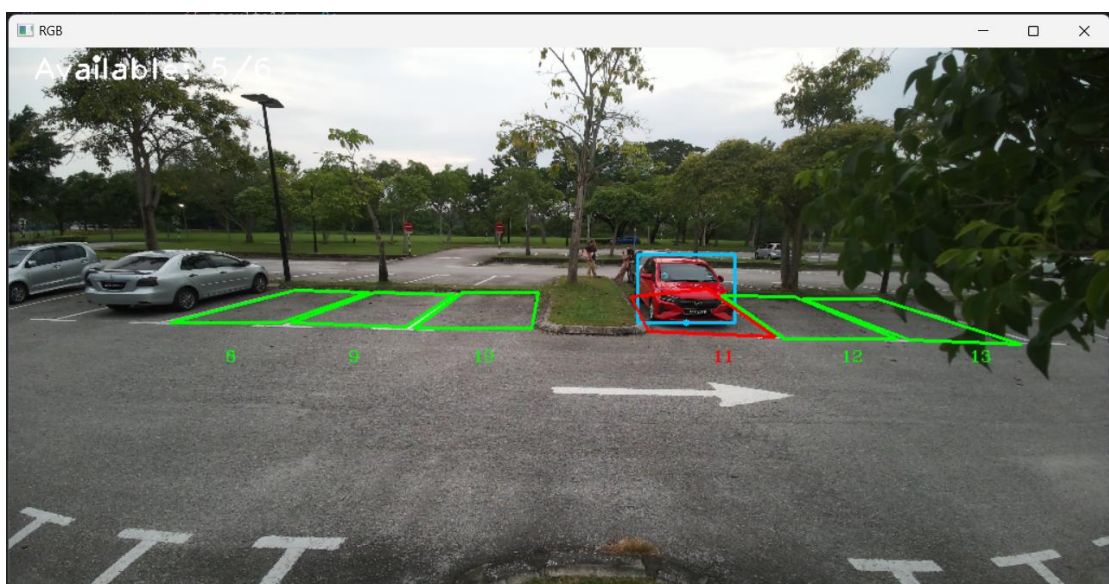


**Figure 6.5 View 3 Captured by Raspberry Pi Camera Module 3**

Figures 6.6 to Figure 6.8 present the same parking lot images, but with additional processing. These images include labeled numbers for each parking lot and boundary boxes drawn around each lot, based on coordinates detected through a mouse function. This step is important for defining the parking space boundaries, enabling the detection system to identify cars accurately. YOLO v8 is used for object detection especially in these cases, the system highlights cars within the defined boundary boxes. The detection results are annotated with boundary lines, and the number of available spaces is displayed in the top left corner of each figure.

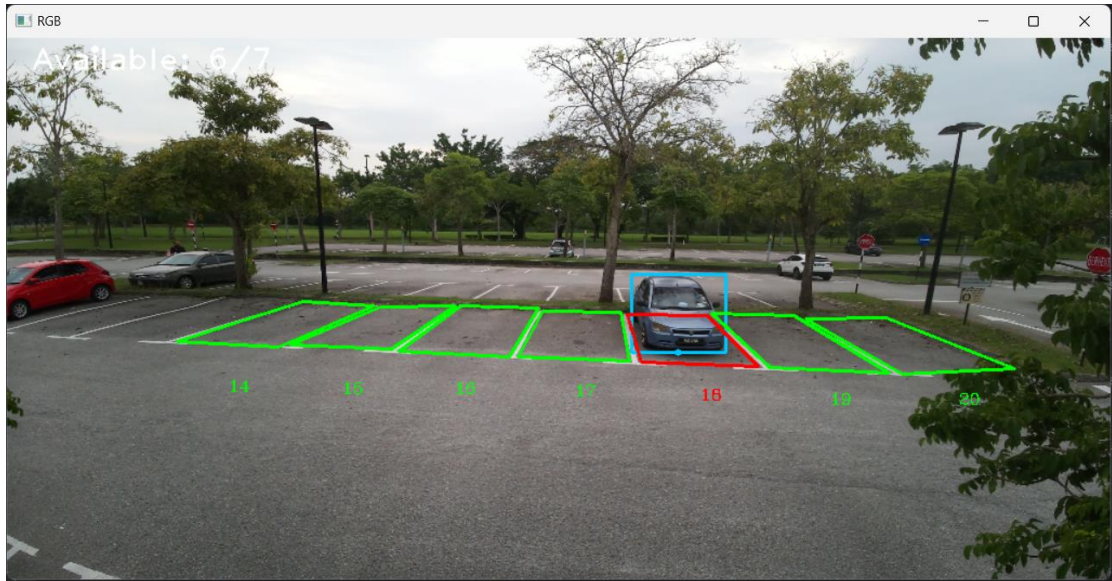


**Figure 6.6 Carpark Lots 1 - 7 with Boundaries and Labels**



**Figure 6.7 Carpark Lots 8 - 13 with Boundaries and Labels**





**Figure 6.8 Carpark Lots 14 - 20 with Boundaries and Labels**

In conclusion, the overall intuition behind was The YOLO model's approach to parking space detection integrates object detection with spatial mapping. The process begins as YOLO scans the image, divides it into a grid, and generates bounding boxes around detected objects (cars), outlined in blue. These boxes define the vehicle's position and size. YOLO then classifies the object as a car and maps it to specific parking spaces using predefined polygons (green and red) overlaid on the image. The system calculates the center of each car's bounding box and checks if it falls within the polygons using the `cv2.pointPolygonTest` function. Green boxes indicate vacant parking spots, while red boxes signify occupied ones. This method ensures real-time feedback on parking availability, providing users with an intuitive view of which spots are taken or available.

## 6.2 Testing Setup and Result

This section presents the testing process conducted to validate the functionality, integration, and overall performance of the smart parking system. Testing plays a crucial role in identifying potential issues, verifying that each component works as intended, and ensuring the system operates reliably under realistic conditions. It is important to confirm that the system performs consistently, especially when deployed in dynamic environments such as a parking lot.

The primary purpose of testing is to ensure that all components, including the Raspberry Pi, camera module, Python backend, Firebase database, and FlutterFlow mobile application, function correctly both independently and as a complete system. By simulating actual deployment scenarios, testing helps evaluate the accuracy of the camera setup in capturing images, the efficiency of data processing and transmission, and the responsiveness of the user interface in displaying real-time results.

### 6.2.1 Testing Setup

#### Step1: Fix the Position of the Pi Camera and Raspberry Pi with Power Supply

To begin the testing process, it was essential to create a compact and stable hardware setup that could be easily deployed on-site. The Raspberry Pi and its connected Pi Camera module needed to remain stationary throughout the test to ensure consistent camera angles and accurate image capture. To achieve this, both the Raspberry Pi and Pi Camera were physically attached to a power bank using rubber bands as shown in Figure 6.9 and Figure 6.10. The power bank supplied power to the Raspberry Pi through a USB Type-A to USB Micro-B cable.

This compact configuration served two main purposes. First, it allowed the Pi to operate as a standalone device during field testing without relying on external power sources such as wall sockets. Second, securing the Pi and camera onto the power bank helped maintain a fixed alignment, preventing unwanted movement or vibration during transportation and testing. Maintaining this fixed position was especially important to ensure the Pi Camera consistently pointed at the intended parking area, which was critical for capturing usable images for real-time analysis.





**Figure 6.9 Pi Camera Attached to a Power Bank**



**Figure 6.10 Raspberry Pi Attached to a Power Bank**

### Step2: Mount the Camera Setup Using a Phone Holder at a Fixed Location

Once the Raspberry Pi and camera module were assembled into a compact unit, the next step was to mount this setup at a stable and suitable location within the parking area. As shown in Figure 6.11, a standard mobile phone holder was used to grip and hold the assembled unit firmly in place. This holder provided the flexibility to adjust the angle and height of the camera while keeping it firmly in place during the test.

The purpose of this step was to simulate the actual deployment condition as closely as possible. Since permanent installation of the Pi Camera was not permitted at this stage due to access restrictions, the temporary use of a phone holder ensured that the camera remained stable and pointed at a consistent angle for image capturing. This temporary mounting approach allowed for thorough functional testing, such as capturing parking lot images, processing them, and verifying their output in the mobile app. It also ensured that the testing was done under realistic conditions with minimal deviation in the camera viewpoint. As shown in Figure 6.12, the setup was positioned in a fixed location in the carpark during testing to support accurate data collection.



**Figure 6.11 Entire Setup Held by Phone Holder**

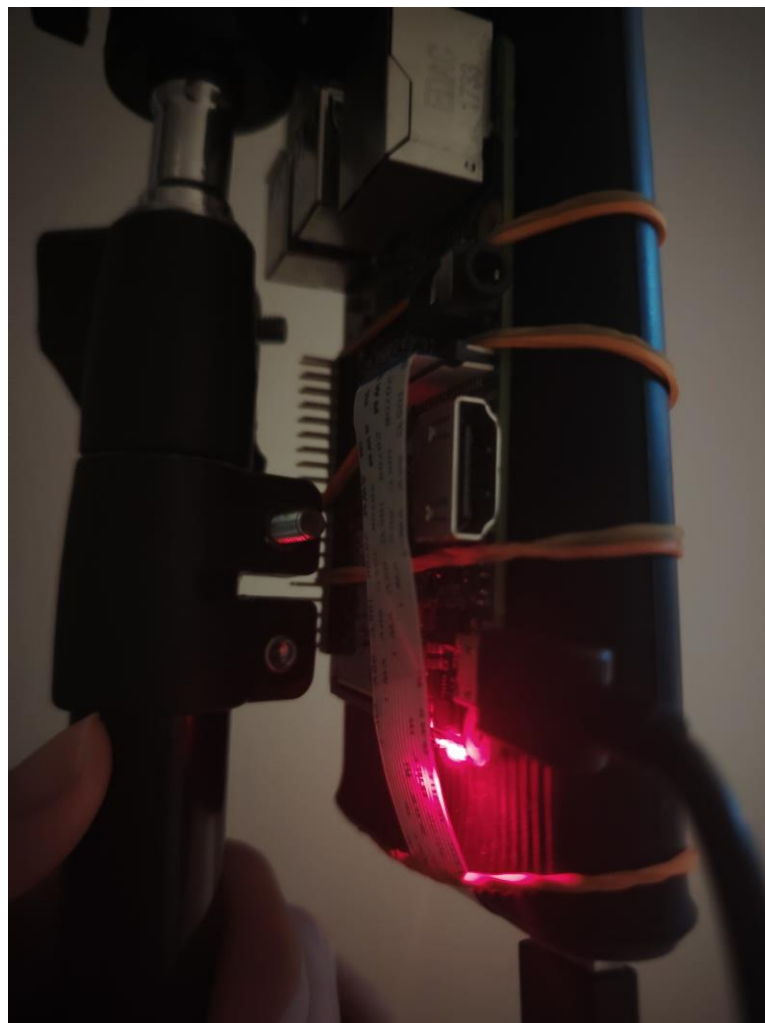


**Figure 6.12 Setup Positioned in the Carpark**

### Step 3: Power On the Raspberry Pi and Verify Status

With the hardware securely mounted and positioned, the next step was to power on the Raspberry Pi to prepare for testing. Once connected to the power bank, the Raspberry Pi was checked for signs of activity, such as the status LEDs lighting up as shown in Figure 6.13. These LEDs served as an initial indication that the system was receiving power and had successfully booted.

This step ensured that the Pi was operational and ready for remote access via the terminal. It also acted as a quick verification to rule out any power-related issues before proceeding to software-level interactions. Confirming power and readiness was a critical checkpoint before accessing the Pi's terminal and executing the Python script for capturing images and testing system functionality.

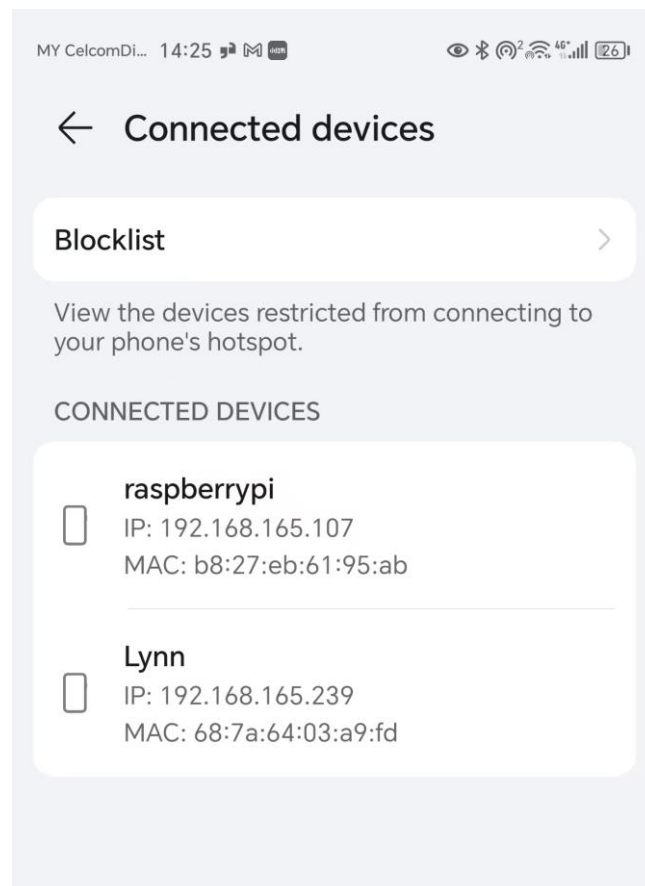


**Figure 6.13 Status LEDs Lighting Up**

#### Step 4: Identify Raspberry Pi's IP address

With the Raspberry Pi powered on and connected to the network, the next step was to identify its IP address. During the initial setup, Wi-Fi credentials were configured using the Raspberry Pi Imager, ensuring that the Raspberry Pi would automatically connect to the designated network once powered on. This configuration allowed the Pi to connect to the mobile hotspot or any specified network without manual intervention.

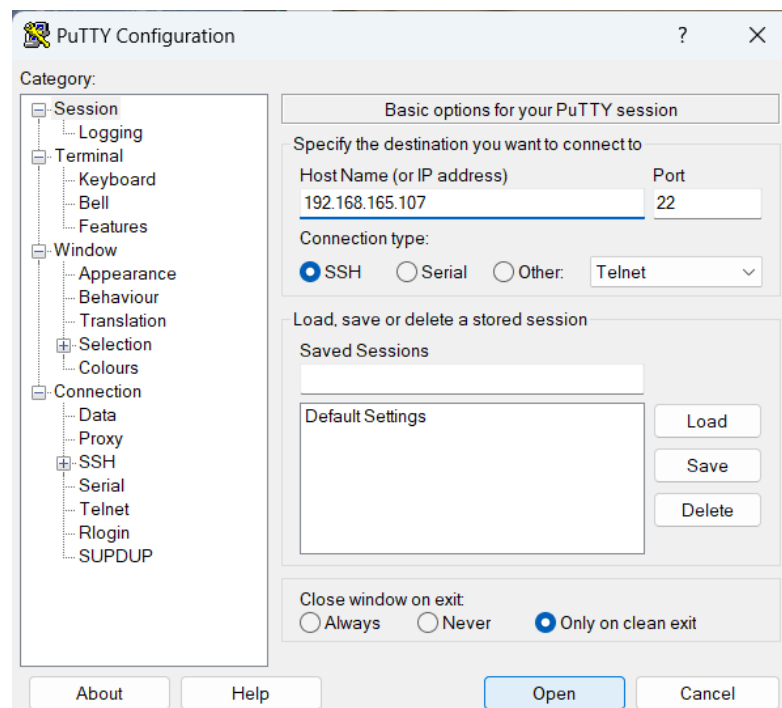
Once the Raspberry Pi was connected to the network, the next step was to identify its assigned IP address. This could be done using a tool like Advanced IP Scanner or by checking the list of connected devices through the mobile hotspot interface. The IP address assigned to the Raspberry Pi will be displayed in this list, as illustrated in Figure 6.14. This IP address was then used to connect to the Raspberry Pi over the network for the subsequent configuration steps. The goal of this step was to enable remote access to the Raspberry Pi without requiring a monitor, keyboard, or mouse, making the setup process faster and more efficient.



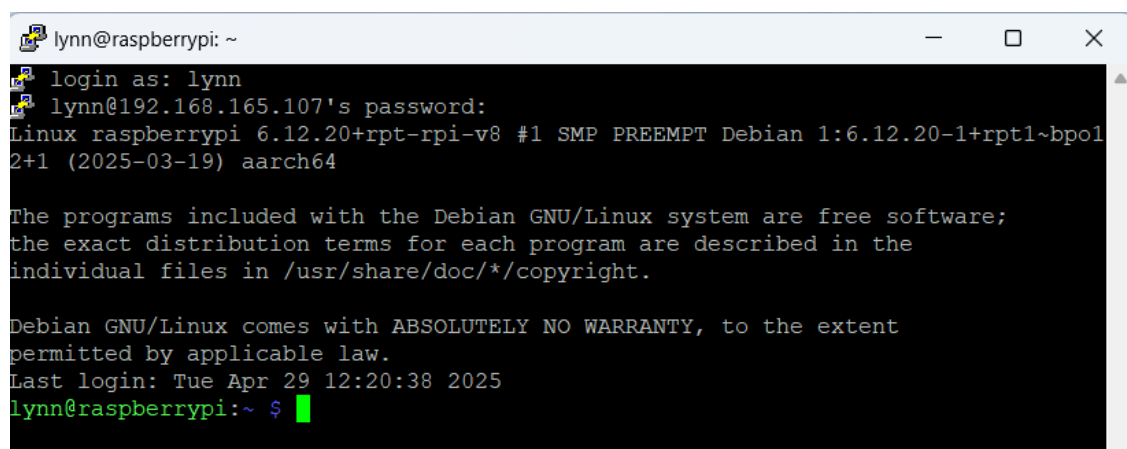
**Figure 6.14 Personal Hotspot's Connected Devices Interface**

### Step 5: Access the Raspberry Pi Using SSH

Once the IP address was identified, the Raspberry Pi was accessed remotely using an SSH client such as PuTTY. The IP address was entered into the PuTTY configuration window, as shown in Figure 6.15, enabling secure remote access. After entering the pre-configured username and password, the Raspberry Pi's terminal was accessed, as depicted in Figure 6.16. This setup allowed for further configuration and the execution of scripts without requiring direct physical interaction with the device. Remote access simplified the process, reducing the need for additional hardware and saving time.



**Figure 6.15 PuTTY Configuration Window**

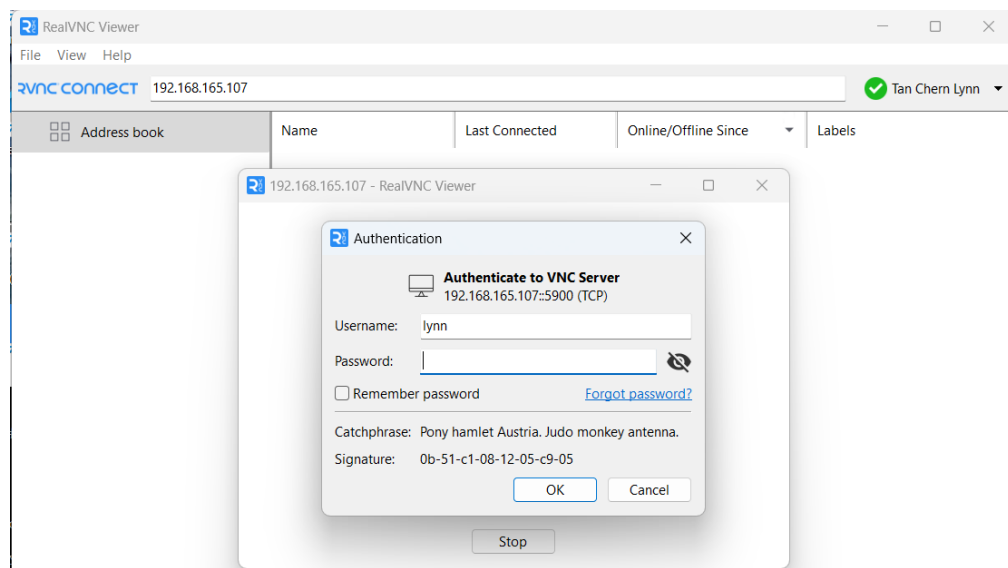


**Figure 6.16 Raspberry Pi Terminal**

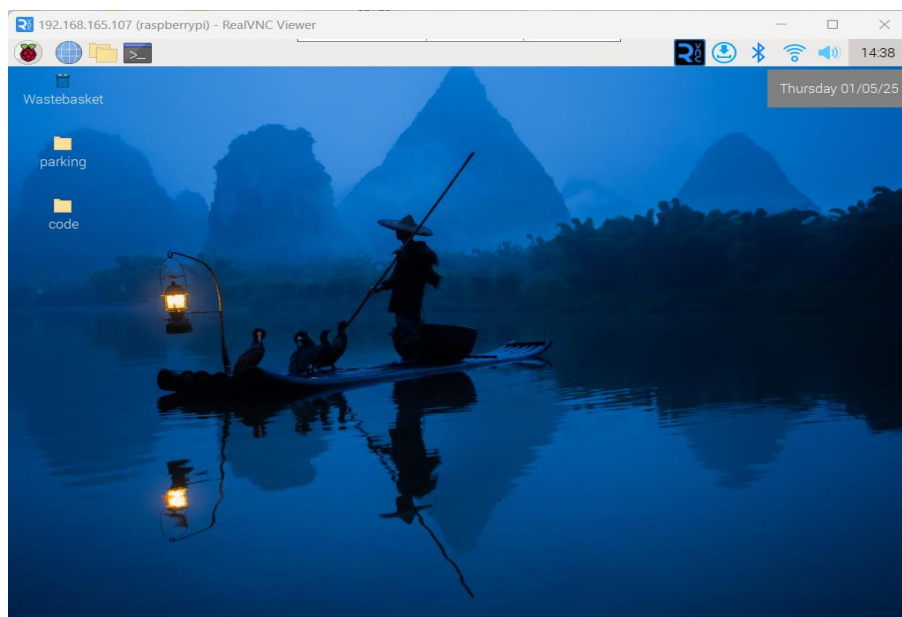


### Step 6: Access the Raspberry Pi Desktop Using RealVNC Viewer

To access the Raspberry Pi's graphical interface, RealVNC Viewer was utilized. The previously identified IP address was entered, and the configured username and password were used to log in, as shown in Figure 6.17. Once logged in, the Raspberry Pi's desktop environment was accessed remotely, as shown in Figure 6.18. This approach enabled easier navigation, file management, and configuration adjustments without the need for direct interaction with the hardware. The graphical interface facilitated tasks such as testing and modifying scripts, as well as monitoring system performance in real-time.



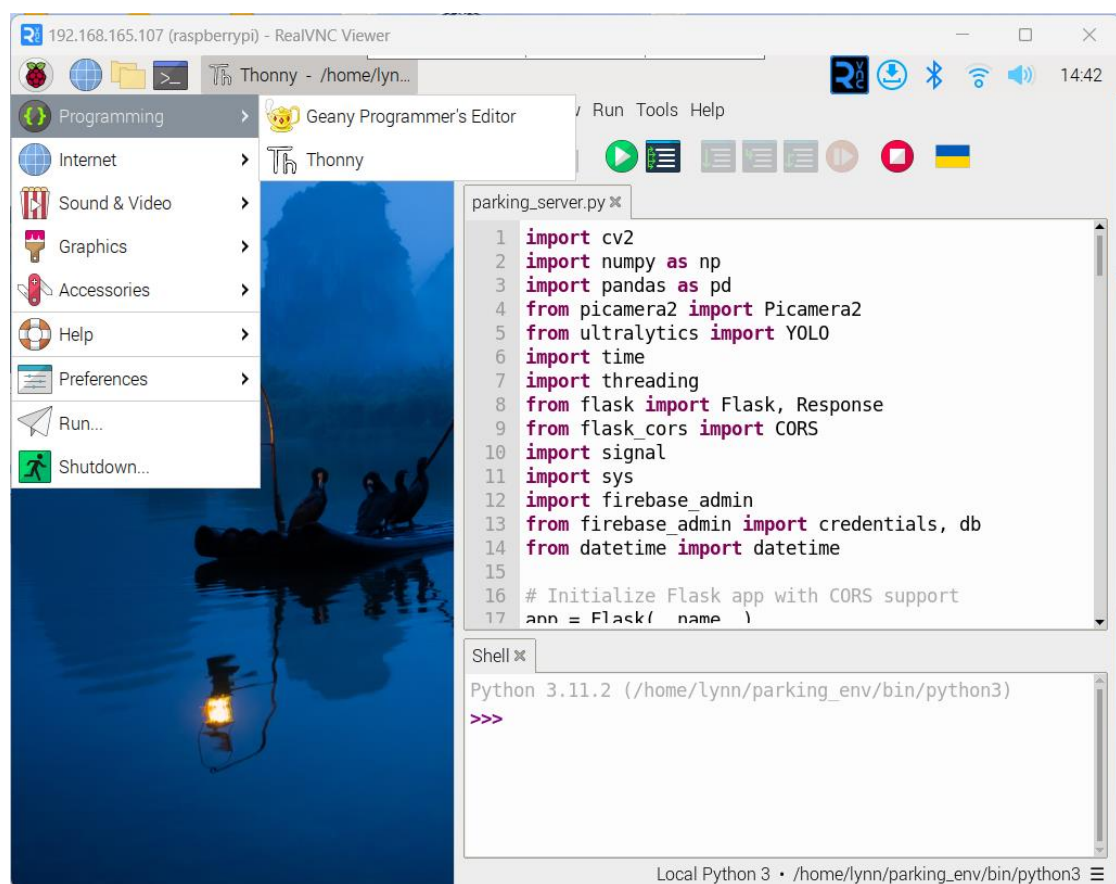
**Figure 6.17 RealVNC Viewer**



**Figure 6.18 Raspberry Pi's Desktop in RealVNC Viewer**

**Step 7: Prepare Python Environment and Ensure the Python Script is Ready to Run**

Once remote access to the Raspberry Pi was established, the next task was to ensure that the Python environment was properly prepared for execution. The Python script and all required dependencies had been organized in advance and placed into the appropriate directory. The necessary libraries such as OpenCV, Flask, and Ultralytics were installed and configured within a virtual environment. The script was then opened using the Thonny IDE on the Raspberry Pi's desktop to perform final checks and necessary adjustments before execution. Figure 6.19 illustrates the Python script opened in Thonny.

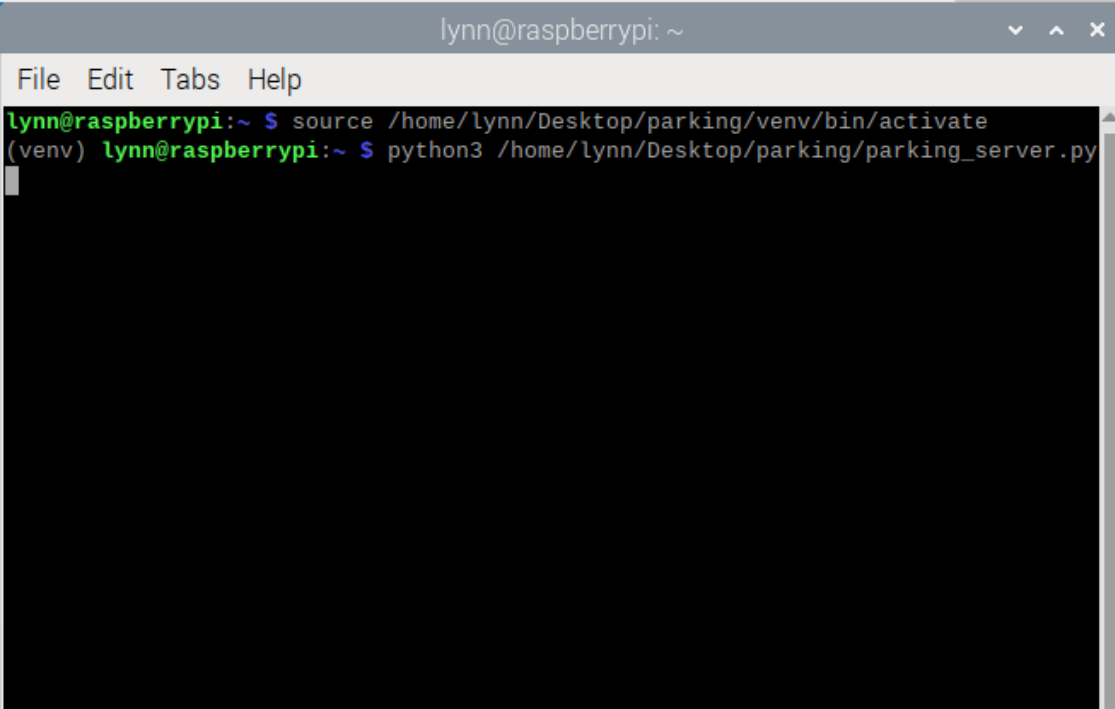


**Figure 6.19 Python Script in Thonny IDE**



### Step 8: Run the Python Script in the Virtual Environment

After confirming that all configurations were correctly set, the virtual environment was activated through the terminal. The Python script was then executed from within this environment. The script handled key tasks such as capturing camera input, performing object detection, and updating the database with real-time parking data. This step marked the beginning of the system's live monitoring functionality and was crucial in validating its performance. Figure 6.20 shows the command-line interface used to activate the virtual environment and run the python script.



```
lynn@raspberrypi: ~  
File Edit Tabs Help  
lynn@raspberrypi:~ $ source /home/lynn/Desktop/parking/venv/bin/activate  
(venv) lynn@raspberrypi:~ $ python3 /home/lynn/Desktop/parking/parking_server.py
```

**Figure 6.20 Script Execution in Terminal**

### Step 9: Verify Camera Operation and Test Run in Flutterflow

At this stage, it was essential to confirm that the Raspberry Pi camera module was operating as intended. The camera was tested by running the Python script, which initiated the image capturing process for object detection. The expected outputs were displayed in the terminal, as shown in Figure 6.21, confirming that the camera was correctly configured to monitor the parking area in real time with no errors related to the camera or other components. This step validated that the system was able to capture live data for parking detection.

```

lynn@raspberrypi: ~
File Edit Tabs Help
(venv) lynn@raspberrypi:~$ python3 /home/lynn/Desktop/parking/parking_server.py
/home/lynn/Desktop/parking/venv/lib/python3.11/site-packages/pandas/core/arrays/masked.py:60: UserWarning: Pandas requires version '1.3.6' or newer of 'bottleneck' (version '1.3.5' currently installed).
  from pandas.core import (
/home/lynn/Desktop/parking/venv/lib/python3.11/site-packages/matplotlib/projections/_init_.py:63: UserWarning: Unable to import Axes3D. This may be due to multiple versions of Matplotlib being installed (e.g. as a system package and as a pip package). As a result, the 3D projection is not available.
  warnings.warn("Unable to import Axes3D. This may be due to multiple versions of "
[8:09:06.013811938] [6863] INFO Camera camera_manager.cpp:327 libcamera v0.4.0+53-29156679
[8:09:06.062467006] [6876] WARN CameraSensorProperties camera_sensor_properties.cpp:473 No static properties available for 'imx708_wide'
[8:09:06.062831227] [6876] WARN CameraSensorProperties camera_sensor_properties.cpp:475 Please consider updating the camera sensor properties database
[8:09:06.090788774] [6876] WARN RPiSdn sdn.cpp:40 Using legacy SDN tuning - please consider moving SDN inside rpi.denoise
[8:09:06.096428013] [6876] WARN CameraSensor camera_sensor_legacy.cpp:501 'imx708_wide': No sensor delays found in static properties. Assuming unverified defaults.
[8:09:06.098503489] [6876] INFO RPi vc4.cpp:447 Registered camera /base/soc/i2c0mux/i2c@1/imx708@1a to Unicam device /dev/media3 and ISP device /dev/media0
[8:09:06.098768490] [6876] INFO RPi pipeline_base.cpp:1121 Using configuration file '/usr/share/libcamera/pipeline/rpi/vc4/rpi_apps.yaml'
[8:09:06.123546026] [6863] INFO Camera camera.cpp:1202 configuring streams: (0) 1020x500-RGB888 (1) 1536x864-SBGGR10_CS_I2P
[8:09:06.124517071] [6876] INFO RPi vc4.cpp:622 Sensor: /base/soc/i2c0mux/i2c@1/imx708@1a - Selected sensor format: 1536x864-SBGGR10_1X10 - Selected unicam format: 1536x864-pBAA
* Serving Flask app 'parking_server'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.165.107:5000
Press CTRL+C to quit

0: 320x640 1 person, 2 chairs, 10135.6ms
Speed: 68.5ms preprocess, 10135.6ms inference, 91.0ms postprocess per image at shape (1, 3, 320, 640)

0: 320x640 (no detections), 7962.8ms
Speed: 35.3ms preprocess, 7962.8ms inference, 12.4ms postprocess per image at shape (1, 3, 320, 640)

```

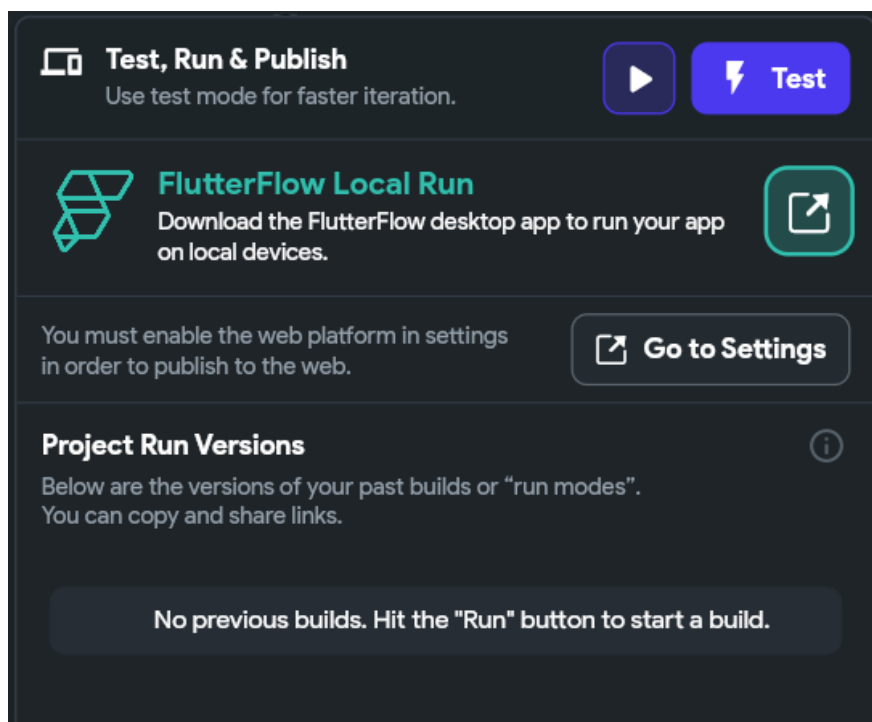
**Figure 6.21 Running Script**

In addition to the terminal output, the captured image could also be manually checked by inspecting the directory for the latest captured image, latest\_output.jpg, to ensure that the image was being saved correctly.

Once the camera functionality was confirmed, FlutterFlow was opened and the Test button was pressed, as shown in Figure 6.22, to begin testing the application. Testing was conducted not only through the FlutterFlow test mode on a laptop, but also using the FF Preview feature in the mobile app. Testing on both platforms was essential because certain functionalities accessible in the laptop test mode might not behave the same way in the mobile environment.

For instance, one issue encountered during mobile testing was an image loading error caused by encoding problems. While the processed image could be loaded successfully in the laptop test mode, the same image failed to load in the mobile app due to encoding issues. This was resolved by encoding the image into a base64 string within the Python script and storing that encoded string in the Firebase Realtime Database. The app then retrieved and decoded the image string successfully, allowing it to display properly on mobile.

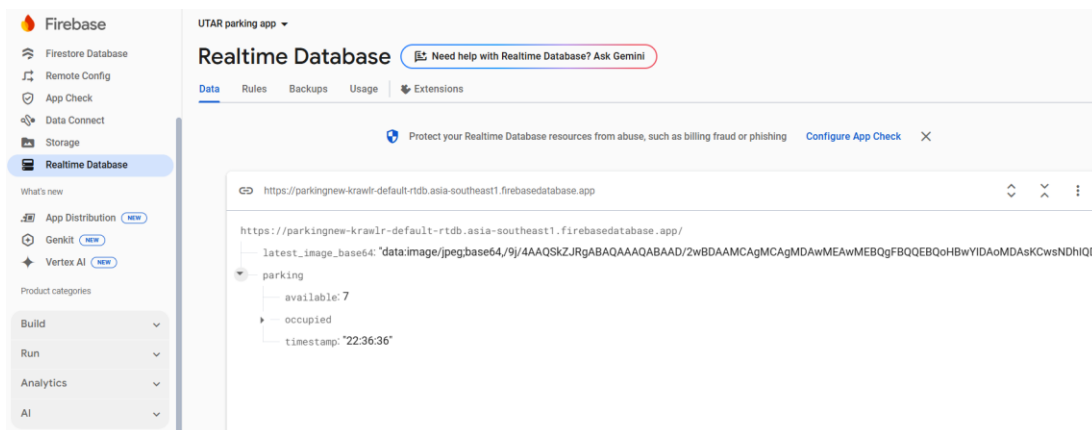
This step played a crucial role in integrating the real-time camera feed with the app's interface, enabling live interaction and comprehensive verification of the system across different platforms.



**Figure 6.22 Test Mode in Flutterflow**

### Step 10: Test the Real-Time Database Update

Once the camera was functioning and capturing the desired area, the next task was to ensure that the system was updating the real-time database correctly. The Firebase real-time database, as shown in Figure 6.23, was checked to confirm that the parking data, such as availability, occupancy status, and timestamp, along with the latest decoded image string, were being updated as intended. This verification ensured that the parking system was accurately logging the status of parking spaces in real time, as well as updating the decoded image string in real time. By saving the image in base64 format, the image data could easily be retrieved and displayed in the FlutterFlow app. The app's image widget was configured to retrieve and decode the base64 string from the Firebase database, allowing the real-time image to be displayed within the app interface. This ensured that users could view the most recent parking area image directly in the app, reflecting the live monitoring of parking spaces.



**Figure 6.23** Firebase Realtime Database

### Step 11: Test and Verify the Functionality of the Application

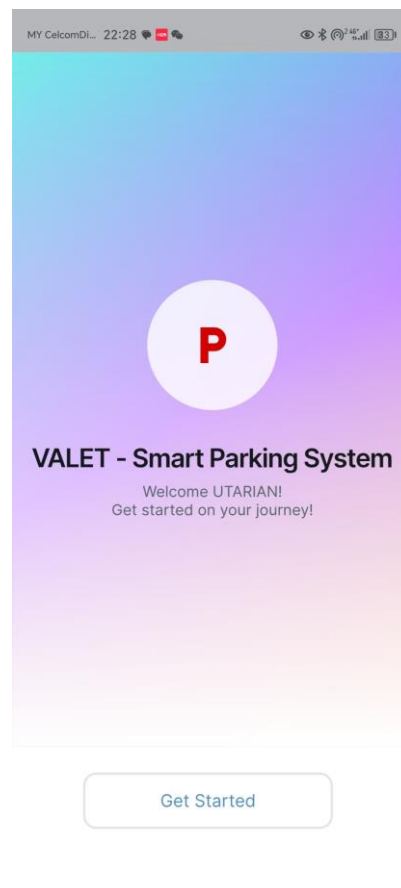
Once the real-time database was properly updating, the app was tested to ensure that the changes made in the parking system were accurately reflected in the app interface. The functionality of all features, including parking space status, navigation, user interactions, and authorization status, was thoroughly examined. A few test cases were conducted to evaluate different scenarios, such as checking parking space availability, updating the notification when the carpark is full, and ensuring the real-time image was displayed correctly. This final step confirmed that all components of the system were working together effectively, allowing users to monitor parking space availability efficiently through the app. The results of the test cases will be presented in the next section.

### 6.2.2 Testing Result

The testing results are presented in the sequence of the testing process to reflect a realistic user journey. All figures included in this section are actual screenshots taken directly from the mobile application interface during live testing sessions. The tests were conducted using the Honor X9B smartphone, which runs on the Android operating system. This ensures that the outcomes accurately reflect the app's performance and user experience on a typical Android device. Each screenshot captures the final visual output of the app at specific stages, validating the proper functioning and responsiveness of its features under real usage conditions.

#### First Page

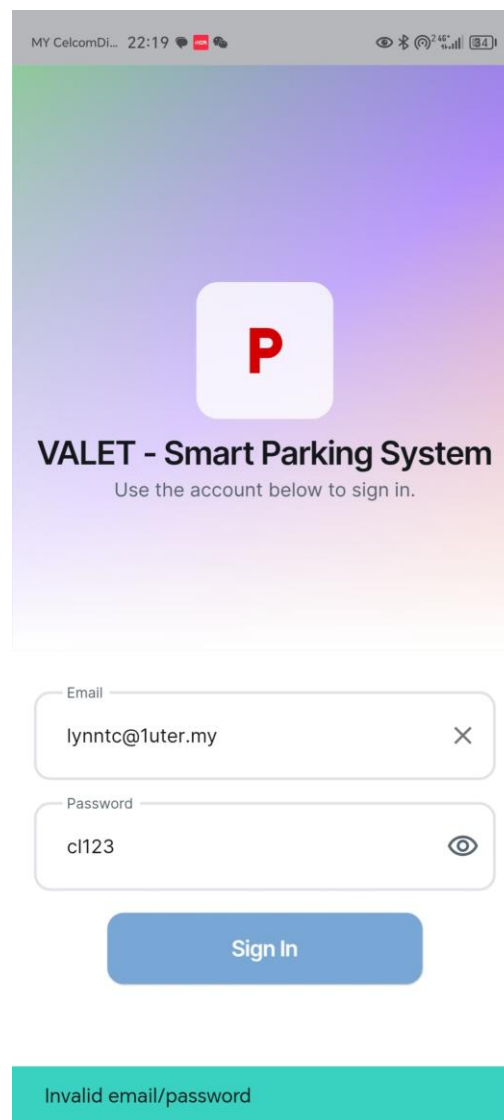
During the testing of the application, the first page was displayed successfully with proper formatting and a responsive layout. Figure 6.24 shows the actual view from mobile phone. The “Get Started” button functioned correctly, allowing smooth navigation to the login page when clicked. This confirmed that the initial transition between pages was working as designed.



**Figure 6.24 First Page’s View from Mobile Phone**

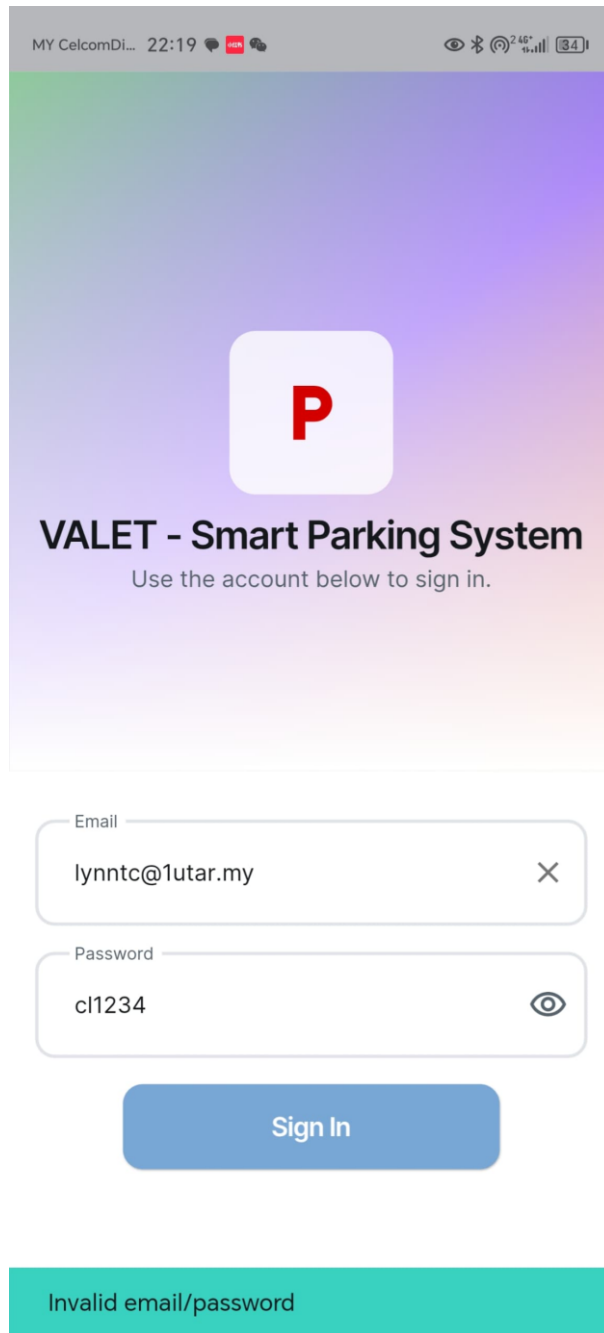
### Login Page

On the login page, a series of test cases were conducted to evaluate the system's ability to handle both valid and invalid input scenarios effectively. To ensure that the login page could handle incorrect input gracefully, several negative test cases were performed. In the first case, a typo was deliberately made in the email address by replacing “utar” with “uter”, which resulting in the input lynntc@1uter.my, while keeping the password correct as cl123. Upon submission, the system identified that the email did not exist in the database and responded with a snack bar message stating “Invalid email/password” as shown in Figure 6.25. This validated that the system correctly handles email validation and prevents unauthorized access due to typing errors.



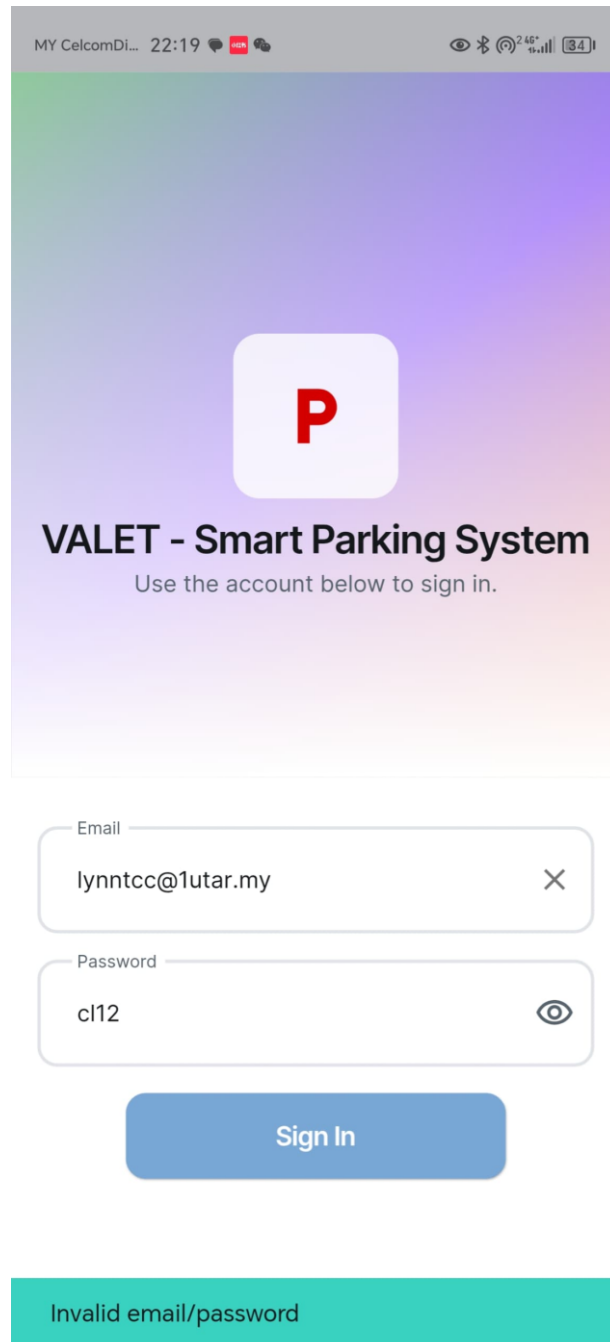
**Figure 6.25 First Negative Test Case for Login Page**

In the second case, the email was correctly entered as `lynntc@lutar.my`, but the password was typed with an extra digit — `cl1234` instead of the actual password `cl123`. The system again responded with an “Invalid email/password” message as shown in Figure 6.26, confirming that it does not allow login with incorrect passwords, even if the email is valid. This proved that the password-matching mechanism works with precision.



**Figure 6.26** Second Negative Test Case for Login Page

The third test case combined both incorrect email and password inputs. The email was entered as `lynntcc@utar.my`, which included an extra “c” character, and the password was shortened to `cl12`, omitting the last digit. Once submitted, the system detected that both credentials were invalid and displayed the same snackbar error message as shown in Figure 6.27. This indicated that the login validation logic works properly even when both fields contain errors, offering consistent error handling across all failed attempts.



**Figure 6.27 Third Negative Test Case for Login Page**



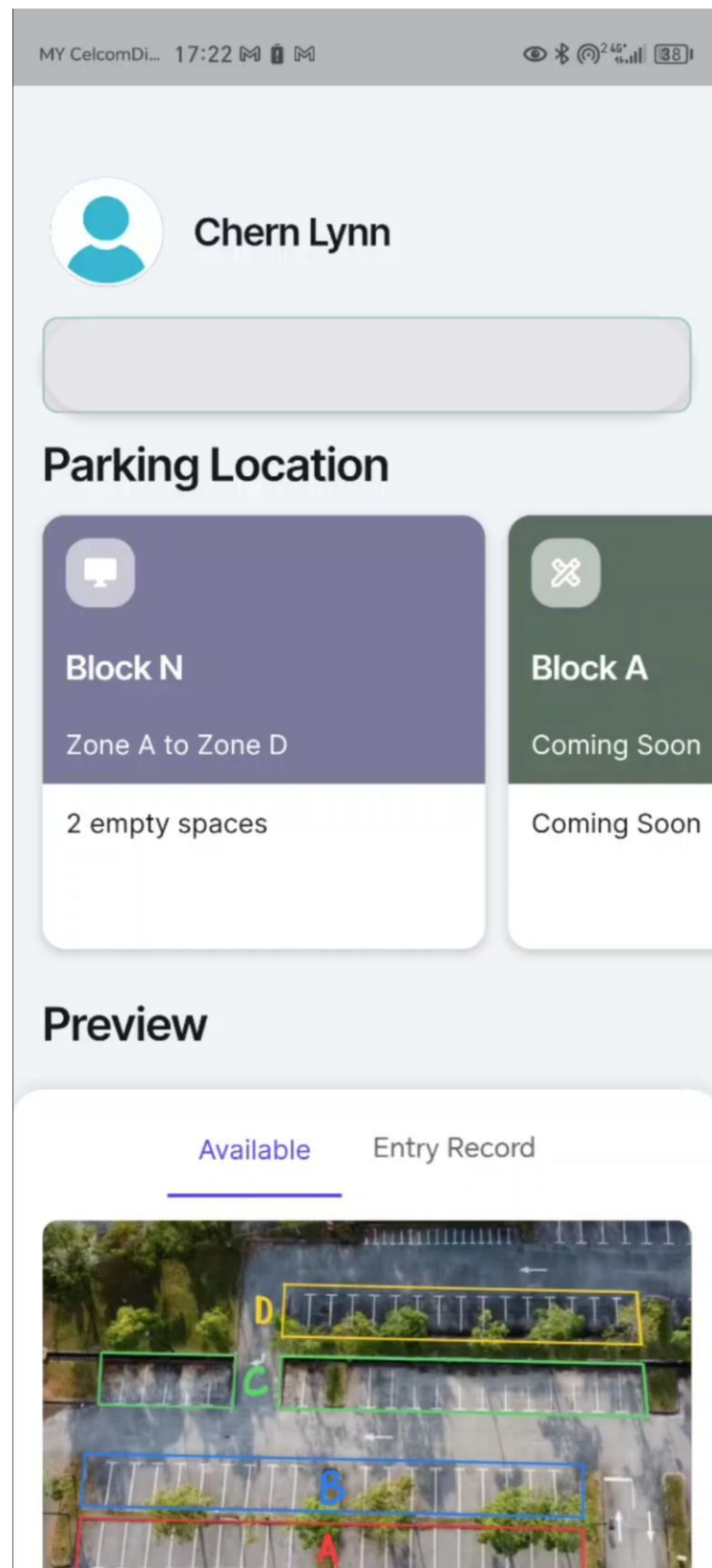
Lastly, the login system was tested using a registered user with the following correct credentials: email lynntc@lutar.my and password cl123. When these valid credentials were entered and the “Sign In” button was clicked, the system authenticated the user and successfully redirected them to the home page. This confirmed that the authentication system accurately verified valid users and granted them access to the system as intended.

Each of these test cases helped confirm that the login page reliably differentiates between valid and invalid entries. It provided clear feedback to users in the form of on-screen messages and restricted access unless the provided credentials matched a registered account. Overall, both the landing and login pages passed all relevant tests, supporting secure and smooth user interaction as the first step in accessing the system.

### **Home Page**

Once the home page was reached, several main features were tested to verify their accuracy and responsiveness to real-time conditions. The first feature tested was the automatic retrieval and display of the signed-in user’s name. Using a test user stored in the database, the system correctly showed “Chern Lynn” under the header after login using email lynntc@lutar.my and password cl123. The result shown in Figure 6.27 confirmed that the application could correctly identify the active user and fetch their personalized information.

The next feature tested was the notification update that alerts users when a parking zone becomes full. Initially, no notification appeared, which was expected since the monitored parking area still had available spaces. This confirmed that the notification system activates only under the appropriate conditions, avoiding unnecessary alerts and maintaining a clean user interface. The test case scenario in which the notification appears will be demonstrated later, once the car park becomes full. For now, let’s proceed with the scenario where there are still two empty spaces, as shown in Figures 6.28 and Figure 6.29.



**Figure 6.28 Home Page**

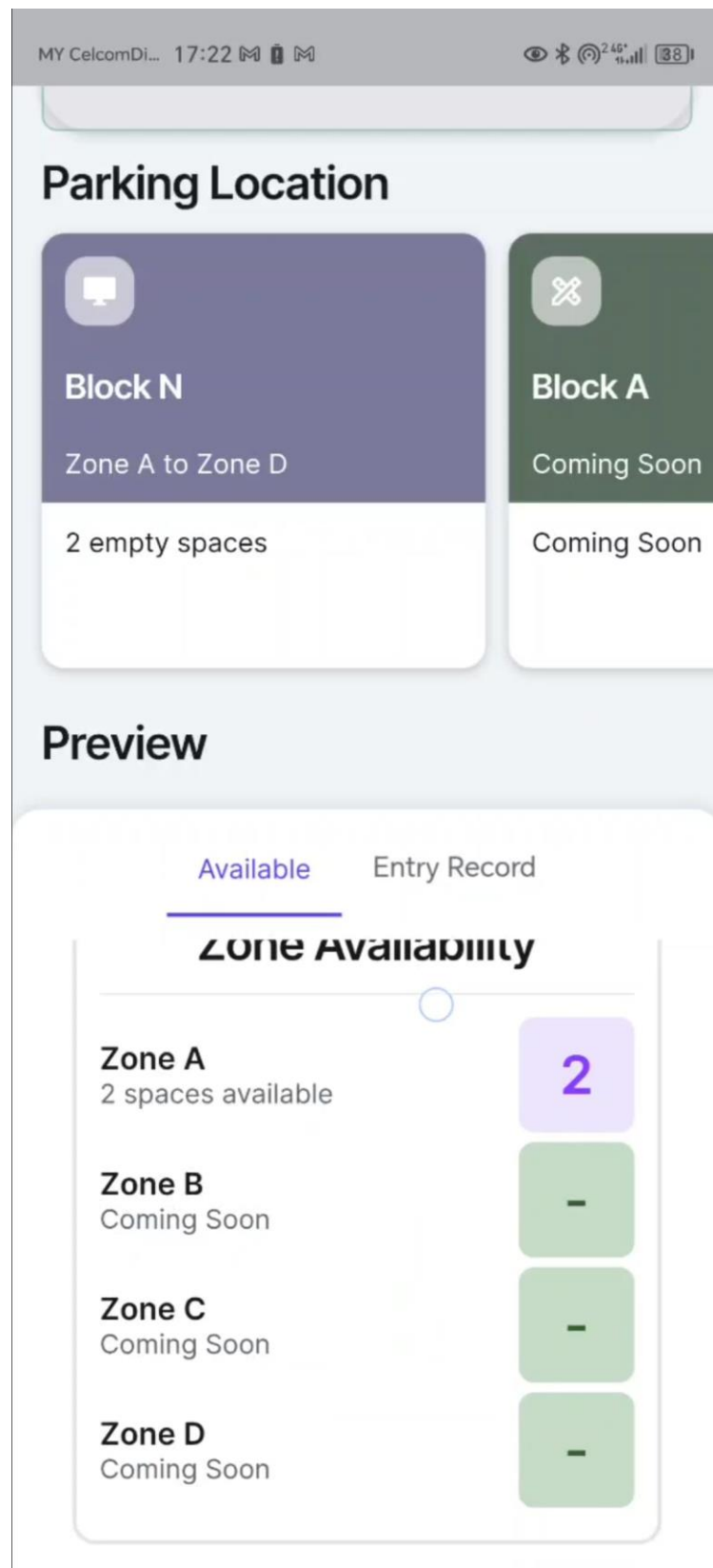
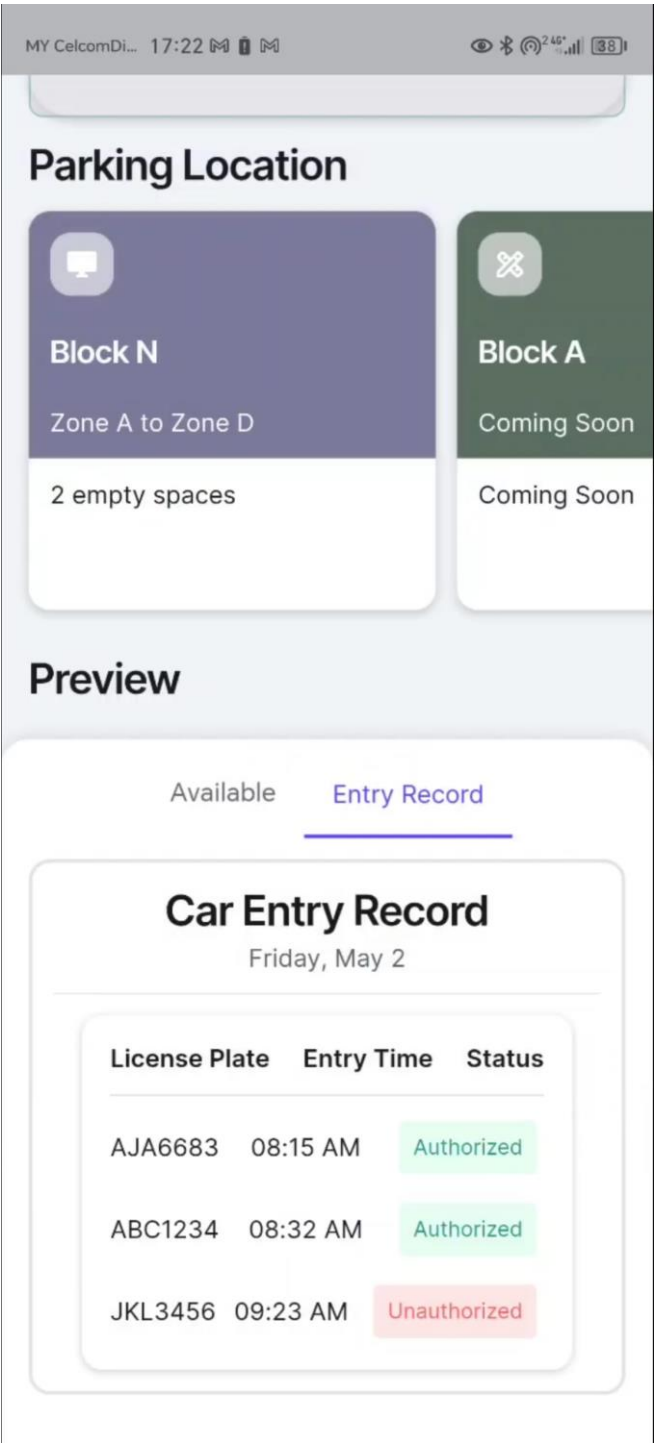


Figure 6.29 Empty Spaces for each Zone and Block

Moving on to the third feature where get to know there are 2 empty spaces in the entire block for this scenario. This feature examined was the total number of empty parking spaces available within Block N, shown in the Block N box on the home page. Because only one camera was active during this stage (monitoring Zone A, parking lot number 1 to 7), the available number displayed on the entire block matched the count for Zone A. This confirmed that the application was accurately reflecting available parking based on the camera input currently in use.

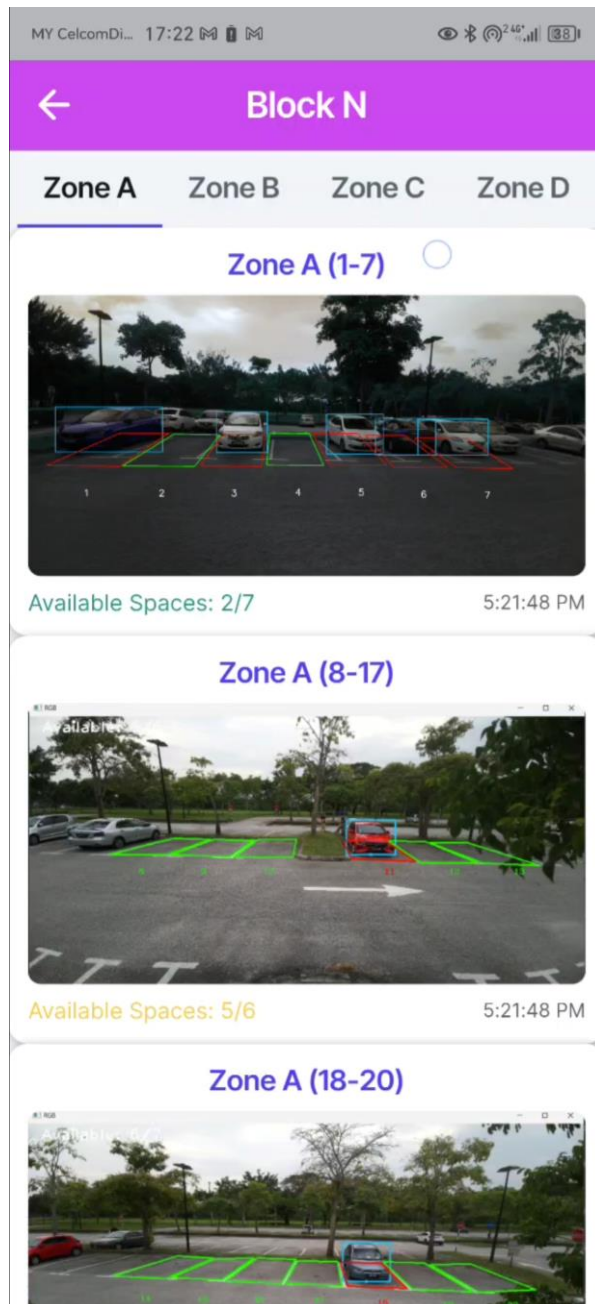
Similarly, the fourth feature was the display of empty spaces within individual zones of the block as shown on the preview section in Figure 6.28 and Figure 6.29. On the availability tab, the system correctly distinguished different zones and displayed the real-time count of available spaces for each. As only Zone A was being tracked with a working camera, its availability was shown correctly on the interface and updated as cars entered or exited the zone.

The fifth feature involved verifying whether detected vehicles were authorized, based on license plate data in the system database. Under the Entry Record tab, the system properly identified vehicles that were already registered and marked them as authorized. It also detected and flagged an unregistered vehicle as unauthorized. The testing result for this feature are shown in Figure 6.30. This demonstrated that the comparison system was functioning as intended to ensure controlled access to the parking area.



**Figure 6.30 Entry Record**

The sixth feature tested was the functionality of the Block N box on the home page. When the user tapped the Block N box, the application successfully navigated to the detailed Block N page, as illustrated in Figure 6.31. This page includes a tab bar interface that allows users to switch between different zones within Block N. Specifically, Tab 1 displays the camera feeds for Zone A, Tab 2 for Zone B, Tab 3 for Zone C, and Tab 4 for Zone D.



**Figure 6.31 Block N Page**

Up to this point, the six key features of the home page were verified to function correctly during testing. The final feature, which involves real-time and automatic updates of available parking spaces, will be demonstrated later as part of the continued testing flow. Similarly, the notification feature that alerts users when a parking zone becomes full will be showcased in a later scenario once the required conditions are met. Overall, the system successfully handled user authentication, parking space availability updates, vehicle authorization checks, and partial notification logic according to the intended design. These results indicate that the core features of the home page are functioning reliably.

### **Block N Page**

In the current stage of development, only the first camera view under Zone A is functional. This view displays parking lots labelled 1 to 7 and is connected to an active Raspberry Pi camera. The live video feed and real-time updates for this camera were functioning properly during testing. Below the feed, the availability counter and timestamp were also updating correctly in response to changes in the parking lot occupancy. While a slight delay was observed during refresh, it was attributed to image loading time rather than a flaw in the system itself.

Since only one camera in Zone A is operational at this stage, the number of available spaces shown on the home page earlier was identical to the availability count seen here. However, in future development phases, the home page's values will reflect the sum of all available spaces across multiple camera views within a specific zone or block.

In addition, the timestamp displayed next to the availability information accurately showed the time of the last image update. During testing, this time was mostly synchronized with the actual time shown on a mobile device, indicating the system was working as expected. In Figure 6.31, however, a few seconds of discrepancy were observed. This is normal, as the timestamp reflects the last update time of the image, not the current time. Thus, the time refresh function was confirmed to be operating correctly.

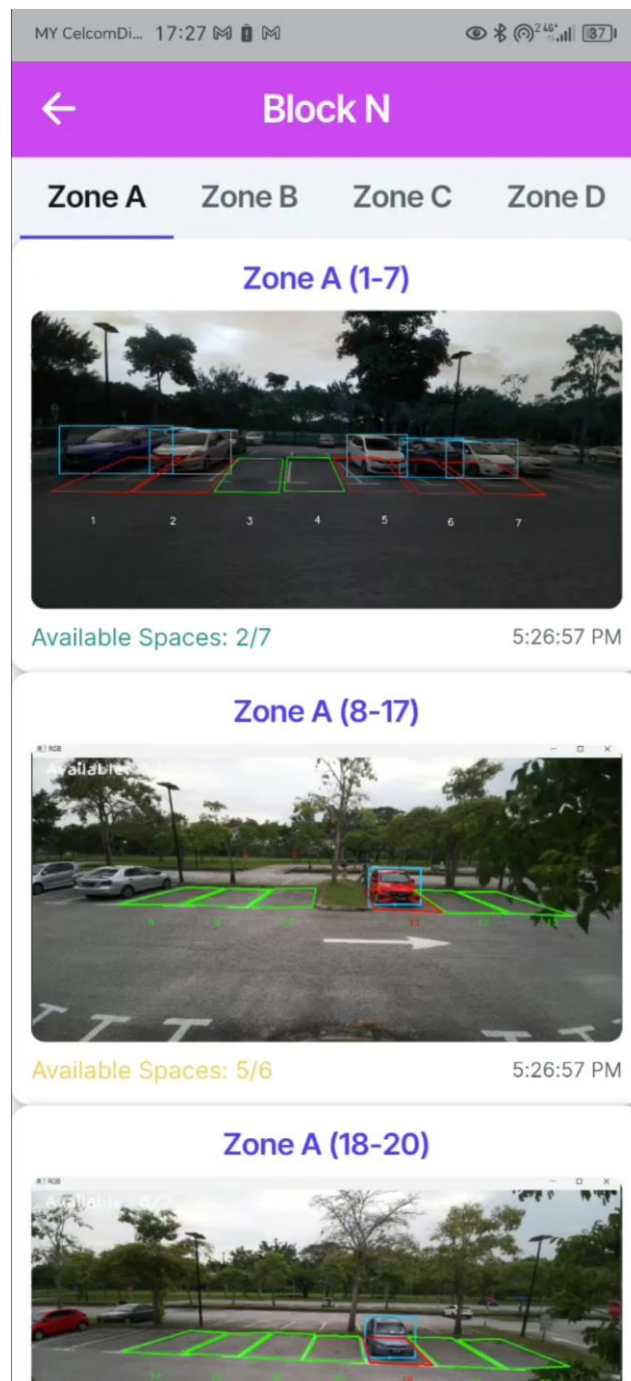
The main feature tested on this page was whether the system could automatically reflect real-time changes when a car arrived or left. At the start of testing as shown in Figure 6.30, parking lot number 2 and number 4 were vacant. When a car parked in lot number 2, the system changed that lot's box colour from green to red, and the available space counter was reduced by one. The test case result is shown in Figure 6.32.

After a few minutes, another car that was previously parked in lot number 3 left, as shown in Figure 6.33. The system updated the box colour from red back to green and increased the counter. These results demonstrated that the application could successfully track vehicle movement and update the display in real time.



**Figure 6.32** Changes when Car Parked on Empty Parking Lot





**Figure 6.33 Changes when Car leave Parking Lot**

As previously mentioned, this project currently uses a single camera to monitor Zone A, specifically covering parking lots one to seven. During testing, the system accurately showed the status of each lot, and the display responded correctly to changes. The reload timer worked well, matching the time of the latest camera update. These results supported the system's ability to reflect the real-time status of parking lots clearly and correctly.

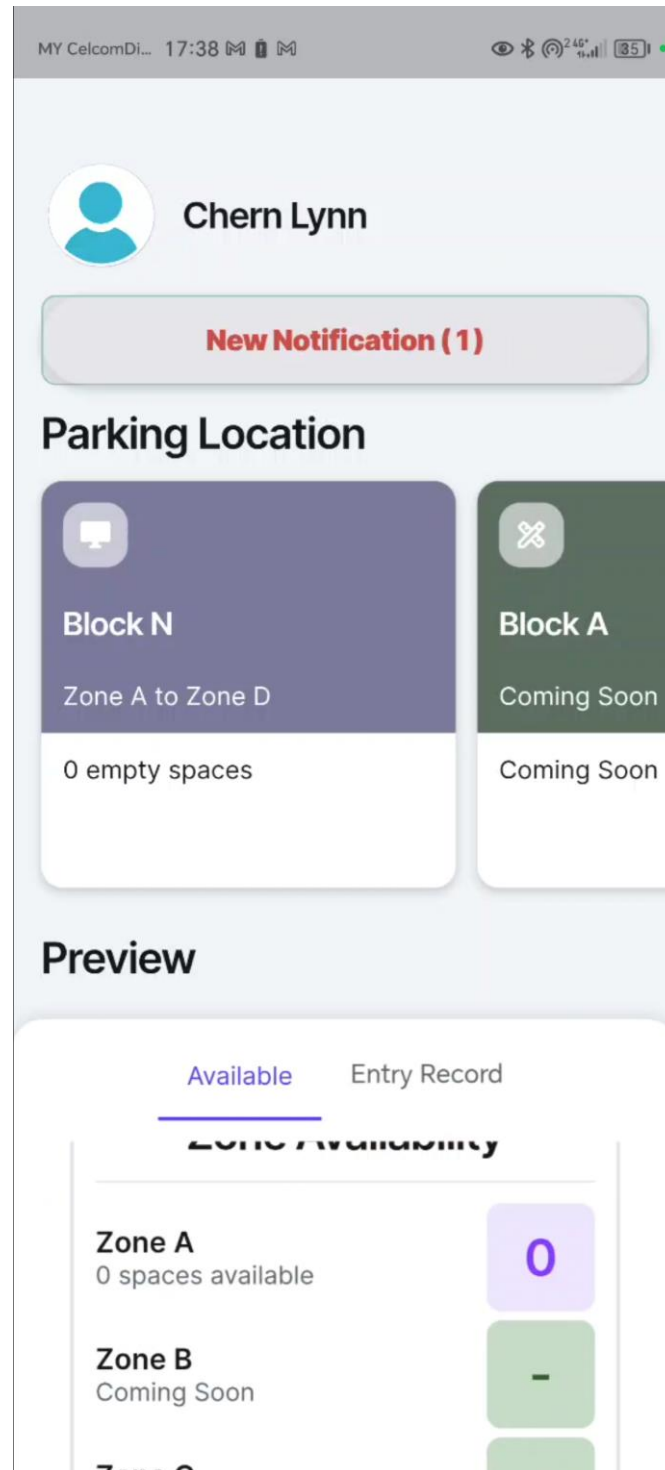
### Notifications

The next step involved testing the notification feature which involving changes in both Home page and Block N page. To test the notification feature under full-capacity conditions, the camera was temporarily adjusted to view a section where all lots were occupied. The system updated the available space count to zero, and every box turned red as shown in Figure 6.34.



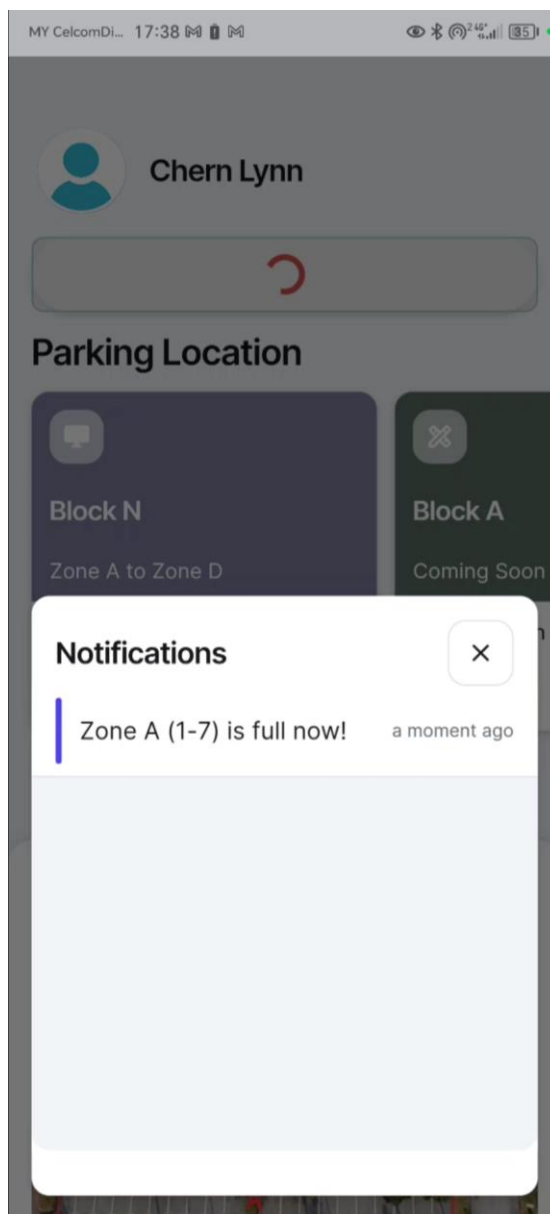
**Figure 6.34 Test Case for Full Parking View**

According to Figure 6.35, a red “New Notification” label appeared upon returning to the home page. Additionally, the real-time parking space counter was accurately updated, which verified the final feature of the home page mentioned earlier.



**Figure 6.35 New Notification when Parking Full**

When the notification button was clicked, a pop-out box appeared, overlapping the bottom part of the current screen. In this pop-out box, a message indicated that Zone A (lots 1–7) was currently full, as shown in Figure 6.36. This confirmed that the notification feature functioned as intended in scenarios where parking availability had reached its limit.



**Figure 6.36 Pop Up Content for the Notification**

With this final test case, all page transitions and system functionalities were successfully validated. The results demonstrated that the system operates smoothly and reliably.

### 6.3 Project Challenges

Developing and deploying a Smart Parking Management System for the entire UTAR Kampar campus presents several significant challenges that must be addressed for the system to function efficiently and sustainably. These challenges can be grouped into technical, logistical, and financial concerns, each of which impacts the overall success of the project. The scale of the campus, the need for accurate real-time data collection, and the financial costs of implementing and maintaining such a system require careful planning and coordination.

One major issue that must be addressed is the availability of a stable power supply throughout the parking areas. Many zones across the campus may not have easy access to consistent electricity, which is essential for operating the cameras and Raspberry Pi units. This limitation would require the installation of additional infrastructure, such as power lines or portable power sources, to ensure that the system remains operational across all parking zones. Without a reliable power source, the system would be unable to function effectively, making power availability a critical consideration in the project's setup phase.

The size of the campus itself further complicates the deployment. UTAR Kampar spans a large area with multiple blocks and parking zones, which means the system must cover a wide range of locations. As a result, a significant number of Raspberry Pi units and camera modules will be needed to ensure complete coverage. Determining the optimal placement for each camera is a crucial challenge. Cameras must be positioned strategically to cover all the carpark spaces or capture accurate license plate images, while avoiding blind spots and environmental interferences like trees, buildings, or weather conditions. The placement and coverage of these devices are crucial to the system's effectiveness, as improper positioning could reduce the accuracy of the parking spaces availability and license plate recognition system.

In addition to the technical challenges, the project poses considerable financial concerns. The need to procure and maintain a large number of devices, spread across the campus, will incur substantial upfront and ongoing costs. This includes the installation of hardware, as well as the ongoing maintenance and troubleshooting

required to ensure the system continues to operate smoothly. Moreover, as the system expands, the free-tier FlutterFlow plan, used to develop the app, may not offer the necessary features for a large-scale deployment. Upgrading to a higher-tier plan will increase costs, which must be accounted for in the project budget.

Testing and ensuring the reliability of the system adds another layer of complexity. With a large campus area, comprehensive testing is required to assess the system's performance in different blocks and parking zones. Thorough testing helps identify any issues related to coverage, accuracy, or functionality. Additionally, testing must account for varying conditions such as traffic density, weather fluctuations, and power inconsistencies. Given the size and scope of the system, this testing phase could be both time-consuming and resource intensive.

Finally, the manpower required for installation, configuration, testing, and ongoing maintenance cannot be overlooked. The project will require a team of skilled personnel to install the hardware, conduct detailed testing, and address any issues that arise during deployment. This high demand for labour, coupled with the extensive number of components involved, makes the project labour-intensive and challenging to manage.

In summary, the deployment of a Smart Parking Management System at entire UTAR Kampar campus faces significant technical, logistical, and financial challenges. Addressing these challenges systematically is crucial to ensuring the system operates efficiently and remains sustainable in the long term.

### 6.4 Objectives Evaluation

#### **Building a UTAR Carpark Management App Using FlutterFlow**

The mobile application was successfully developed using FlutterFlow, offering a clean and intuitive user interface for car park monitoring. Users were able to navigate through various features such as checking parking space availability, receiving notifications when the carpark is full, and viewing images of the parking area to identify position of empty spaces. The use of FlutterFlow's cross-platform capabilities enabled rapid UI design and real-time data binding with Firebase. During testing, the app responded efficiently to changes in the database, demonstrating strong integration and reliable synchronization between the frontend and backend components.

#### **Developing a Robust OCR for Car Plate Recognition**

A reliable OCR component was developed using a YOLO-based model in combination with EasyOCR for license plate text extraction. The system was tested under various lighting conditions, such as glare during the day and overcast skies, and it demonstrated strong accuracy in recognizing license plate characters. However, this feature was only partially integrated into the final app. Since the primary focus of the project was on user-facing functionalities for students and staff, rather than administrative monitoring, full implementation of real-time license plate recognition was limited. The lack of a comprehensive database of authorized vehicle numbers and a limited number of Pi cameras also constrained live detection capabilities. Nonetheless, the OCR module was successfully linked with basic authorization logic in testing scenarios, and it remains a promising area for future development, particularly if the system is expanded to support administrative tasks or the detection of unauthorized vehicles.

#### **Designing a Labelling System for Parking Bays in UTAR**

A systematic labelling mechanism was introduced, with each parking bay assigned a unique identifier based on its location. Each camera was programmed to monitor specific zones, avoiding overlap and ensuring accurate vehicle detection. This strategy greatly improved the reliability of the occupancy counter and reduced false detections caused by overlapping views. The implementation proved effective during system trials, allowing the app to accurately identify which bays were vacant or occupied.

### **Providing Real-Time Parking Zone Monitoring with Live Availability and Visual Updates**

The system was developed to enable real-time monitoring of parking zones by displaying both the number of available parking bays and the latest visual feed from each camera view. This feature helps users make quick and informed decisions on where to park based on current occupancy across various blocks or zones within the campus. To maintain stable performance on the Raspberry Pi devices, the system updates data every 30 seconds. This interval has been found to be more than sufficient for capturing noticeable changes in parking availability while keeping the system efficient and responsive. Additionally, users can manually refresh the camera view images within the app to access the most recent visuals of each parking area. By combining live availability data with updated images, the feature improves user experience and reduces the stress of finding a parking spot.

### **Alerting Users on Full Parking Areas Through In-App Notifications**

The system implemented a real-time alert feature that notifies users when specific parking areas are full. Instead of highlighting individual vacant bays, the app monitors camera input linked to Firebase and updates the occupancy status of each camera's assigned zone. When all monitored spaces within a camera's view are occupied, the app displays a clear notification indicating which area is currently full. This allows users to make informed decisions without wasting time entering fully occupied zones. The feature was successfully tested and shown to be responsive, offering timely and accurate updates that enhanced the overall user experience by reducing unnecessary searching during peak hours.



### 6.5 Concluding Remark

This chapter has provided a comprehensive evaluation of the Smart Parking Management System by reviewing its performance during testing, assessing how effectively it met the initial objectives, and identifying real-world challenges encountered throughout the process. Building upon the work presented in earlier chapters, which included problem identification, literature review, system design, and implementation, this section served as a vital point for reflecting on the overall success and feasibility of the project.

The system testing and performance analysis confirmed that the core features functioned reliably under the conditions defined in the testing setup. The mobile application, developed using FlutterFlow, delivered a user-friendly experience, enabling students and staff to monitor parking availability in real time. The inclusion of dynamic updates through Firebase Realtime Database, image feeds from Pi camera modules, and in-app alerts during full-capacity conditions significantly enhanced user interaction with the system. The backend, supported by Firebase, maintained data synchronization with minimal latency, even during frequent updates, which contributed to a consistently responsive user experience.

Testing further validated the potential to integrate the OCR module for license plate recognition into the app. However, the OCR was not used in the authorization testing due to the lack of extra cameras and permission to record vehicle plate numbers. Instead, a list of authorized plates was created, and the system identified authorization based on whether the plate appeared in the database as an authorized vehicle. While the OCR functionality was prepared, it was not implemented in the app at this stage. Once additional cameras are available and permission to record plate numbers is granted, the OCR module can be fully integrated to automatically detect plate numbers and determine authorization in real-time.

Throughout the development and deployment phases, several key challenges emerged. From a technical standpoint, issues related to camera placement and environmental factors such as lighting and weather had a noticeable impact on image quality and detection accuracy. Logistically, the large coverage area of the UTAR Kampar campus

and the limitations in available power sources presented further difficulties in maintaining stable camera performance. Additionally, the constraints imposed by the FlutterFlow free plan, such as the inability to refresh images automatically, affected some aspects of real-time monitoring. Despite these constraints, the development team was able to implement a zone-based monitoring strategy, optimize camera positions, and introduce a bay labeling system to improve accuracy and reduce potential detection errors.

An evaluation of the system's objectives confirmed that most of the intended goals were achieved within the boundaries of available resources and tools. Functional features such as real-time parking updates, image-based bay recognition, and user alerts were implemented and tested successfully. Some advanced features, including administrative controls, automated vehicle authorization based on license plates, and enhanced scalability beyond FlutterFlow's free tier, remain areas for future improvement. Nonetheless, the existing system has already proven its value in terms of user functionality and operational feasibility.

In conclusion, this chapter has demonstrated that the Smart Parking Management System is not only technically viable but also capable of addressing real-world parking challenges in a campus environment. The insights gained through testing, combined with the experience of resolving implementation difficulties, provide a strong basis for further development. With additional support in terms of funding, infrastructure, and software capabilities, the system holds strong potential to grow into a fully automated parking management platform for UTAR Kampar and similar institutions.

## Chapter 7

### Conclusion and Recommendation

#### 7.1 Conclusion

The Smart Parking Management System for UTAR Kampar presents a comprehensive and structured solution to the university's longstanding parking challenges. This project was developed in response to key issues such as unauthorized parking, overcrowded parking spaces, and the inefficiencies of manual monitoring. These problems have led to daily frustration for students, staff, and visitors, highlighting the need for a more systematic and intelligent parking management approach. The existing method lacked the capability to monitor, control, and provide up-to-date information on parking bay availability, often resulting in miscommunication and space underutilization.

To overcome these limitations, the proposed system integrates several core technologies including optical character recognition for license plate detection, periodic camera monitoring, Firebase cloud storage, and a mobile application interface. These components work in tandem to deliver a more organized, secure, and user-friendly parking experience across campus.

The foundation of this system was laid through a comprehensive literature review that explored previous studies in OCR, automatic license plate recognition, CNN, and AIoT-powered surveillance. These studies guided the selection of tools and methods, while also highlighting key weaknesses in existing systems, such as the lack of seamless integration and field-tested practicality. The focus of this project was therefore placed on combining these technologies into a real-world, deployable solution.

The system was developed around two primary modules: the entrance camera module and the parking area camera module. To implement the entrance module within the mobile application, an automated vehicle verification feature was created under the Entry Record tab. This feature captures the license plate of an incoming vehicle and cross-checks it against a database of authorized plate numbers stored in Firebase. If the plate is verified, the vehicle is marked as authorized without requiring any manual inspection, allowing for fast entry validation.

Simultaneously, the parking area camera module captures images of parking bays at fixed intervals of thirty seconds, analyzes them for occupancy using image processing techniques, and updates the status to Firebase. This periodic image capture ensures the system consistently tracks and updates parking bay availability in near real-time. These frequent updates are sufficient to support timely decision-making for drivers and security personnel, bridging the gap between real-time monitoring and resource constraints.

Additionally, a labeling system was introduced to uniquely identify each parking bay, allowing users to not only check space availability but also pinpoint the exact location of an empty bay within a specific block. This makes the parking process more intuitive and less time-consuming, especially during peak hours. The labeling system further enhances usability by reducing confusion and misdirection, supporting both first-time visitors and regular users in finding available spaces more quickly and confidently.

The mobile application, built using FlutterFlow, serves as the user interface for both users and administrators. It allows users to log in securely, view the most recent parking availability by bay, check selected live camera views from designated blocks, and receive automatic alerts when areas are full. Security guards also benefit from the centralized dashboard, which enables them to direct vehicles more efficiently and reduce confusion during peak hours.

In conclusion, the Smart Parking Management System developed in this project not only addresses the existing parking challenges at UTAR Kampar but also sets a new standard for campus parking solutions. Implementing this system will lead to a more organized, secure, and user-friendly parking environment, significantly improving the overall parking experience for the university community. By aligning with modern technological trends, this project underscores UTAR's commitment to innovation and efficiency, ultimately contributing to a safer and more resource-optimized campus. The continued development and integration of the system's components will ensure that it remains a vital tool for managing parking on campus, benefiting students, staff, and visitors alike.

### 7.2 Recommendation

To further improve the system's reliability and performance, several recommendations are proposed for future work. First, it is important to test the license plate recognition module with live vehicle movement instead of static images. This will help determine whether the system can handle motion blur, different lighting conditions, and real-time processing demands effectively. Testing should be carried out under various weather conditions and at different times of the day to simulate real-world usage.

Second, more camera units should be installed to cover additional blocks or entrances on campus. This would allow the system to track multiple entry points and improve the overall accuracy of parking availability updates. Higher-resolution cameras or better positioning could also help in reducing recognition errors caused by unclear images.

From a software perspective, shifting from FlutterFlow to premium plan or a full custom development approach would allow greater flexibility in adding advanced features such as admin role assignment, integration with the university's vehicle database, and enhanced notification options. Offline access and faster performance could also be achieved through native mobile development.

Additional features such as vehicle exit detection, record history tracking, and parking analytics can be included in the future to provide more insights into parking usage patterns. Implementing a centralized dashboard for admin monitoring can also be considered, which would show real-time updates across all blocks and allow manual overrides if needed.

Lastly, efforts should be made to ensure the system remains stable during power or network outages. Backup systems or low-energy hardware such as solar-powered camera modules can be explored to improve system availability.

Overall, the current system serves as a practical prototype with real-world value and can be further developed into a complete campus-wide solution with continued testing and optimization.

## REFERENCES

- [1] A. Rosebrock, Ed., “OpenCV: Automatic License/Number Plate Recognition (ANPR) with python,” in PyImageSearch, Sep. 2020. [Online]. Available: <https://pyimagesearch.com/2020/09/21/opencv-automatic-license-number-plate-recognition-anpr-with-python/>
- [2] C. Patel, D. Shah, and A. Patel, Eds., “Automatic Number Plate Recognition System (ANPR): A Survey,” in International Journal of Computer Applications, May. 2013. [Online]. Available: <https://research.ijcaonline.org/volume69/number9/pxc3887665.pdf>
- [3] N. do V. Dalarmelina, M. A. Teixeira, and R. I. Meneguette, Eds., “A Real-Time Automatic Plate Recognition System Based on Optical Character Recognition and Wireless Sensor Networks for ITS,” in National Library of Medicine, Dec. 2019. [Online]. Available: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6982939/>
- [4] C. Tomasi and R. Manduchi, Eds., “Bilateral filtering for gray and color images,” in IEEE Xplore, July. 1998. [Online]. Available: <https://ieeexplore.ieee.org/document/710815>
- [5] Tesseract-Ocr. “Tesseract-OCR/tesseract: Tesseract open source OCR engine (main repository).” GitHub. <https://github.com/tesseract-ocr/tesseract> (accessed Dec. 10, 2023).
- [6] A. A. Elsonbaty and M. Shams, “The Smart Parking Management System,” in International Journal of Computer Science and Information Technology, Aug. 2020. [Online]. Available: <https://arxiv.org/pdf/2009.13443>
- [7] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, Eds., “Object detection with discriminatively trained part-based models,” in IEEE Xplore, Sep. 2010. [Online]. Available: <https://ieeexplore.ieee.org/document/5255236>

## REFERENCES

- [8] S. Andre & S. Mahmud & P. Bedy, Eds., “Indonesian Vehicles Number Plates Recognition System Using Multi Layer Perceptron Neural Network and Connected Component Labelling,” in *International Journal on Information and Communication Technology (IJoICT)*, Dec. 2015. [Online]. Available: [https://www.researchgate.net/publication/299404428\\_Indonesian\\_Vehicles\\_Number\\_Plates\\_Recognition\\_System\\_Using\\_Multi\\_Layer\\_Perceptron\\_Neural\\_Network\\_and\\_Connected\\_Component\\_Labelling](https://www.researchgate.net/publication/299404428_Indonesian_Vehicles_Number_Plates_Recognition_System_Using_Multi_Layer_Perceptron_Neural_Network_and_Connected_Component_Labelling)
- [9] P. Kaur, Y. Kumar, and S. Gupta, Eds., “Artificial intelligence techniques for the recognition of multi-plate multi-vehicle tracking systems: A systematic review- archives of computational methods in engineering,” in *SpringerLink*, May. 2022. [Online]. Available: <https://link.springer.com/article/10.1007/s11831-022-09753-4>
- [10] V. U. Tamse. “Understanding convolutional neural networks (CNN or ConvNet).” *Medium*. <https://medium.com/analytics-vidhya/understanding-convolutional-neural-networks-cnn-or-convnet-128dc51811f3> (accessed Dec. 10, 2023).
- [11] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, Eds., “Gradient-based learning applied to document recognition,” in *IEEE Xplore*, Nov. 1998. [Online]. Available: <https://ieeexplore.ieee.org/document/726791>
- [12] A. K. G. Inc et al., “ImageNet classification with deep convolutional Neural Networks,” in *Communications of the ACM*, May. 2017. [Online]. Available: <https://dl.acm.org/doi/10.1145/3065386>
- [13] K. Simonyan, and A. Zisserman, Eds., “Very deep convolutional networks for large-scale image recognition,” in *arXiv.org*, April. 2015. [Online]. Available: <https://arxiv.org/abs/1409.1556>
- [14] C. Szegedy et al., “Going deeper with convolutions,” in *arXiv.org*, Sep. 2014. [Online]. Available: <https://arxiv.org/abs/1409.4842>

## REFERENCES

- [15] K. He, X. Zhang, S. Ren, and J. Sun, Eds., “Deep residual learning for image recognition,” in IEEE Xplore, June. 2016. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/7780459/>
- [16] S. Saha. “A comprehensive guide to Convolutional Neural Networks - the eli5 way.” Medium. <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53> (accessed Dec. 10, 2023).
- [17] T. Wood. “Convolutional Neural Network.” DeepAI. <https://deepai.org/machine-learning-glossary-and-terms/convolutional-neural-network> (accessed Dec. 10, 2023).
- [18] A. J. Mohammed, J. A. Mohammed, and A. I. Melhum, Eds., “Comparative Study of Font Recognition Using Convolutional Neural Networks And Two Feature Extraction Methods With Support Vector Machine,” in Iraqi Journal for Computers and Informatics, Sep. 2023. [Online]. Available: <https://doi.org/10.25195/ijci.v49i2.434>
- [19] M. Mishra. “Convolutional Neural Networks, Explained.” Medium. <https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939> (accessed Dec. 10, 2023).
- [20] S. Saha, “A Guide to Convolutional Neural Networks — the ELI5 way,” Saturn Cloud Blog, Dec. 15, 2018. <https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/>
- [21] W. C. YAN Chunman, Ed., “Development and application of Convolutional Neural Network Model,” in Doaj, Jan. 2021. [Online]. Available: <https://doaj.org/article/7ffe18cd35d642ddb12c4037296506cf>
- [22] “What is Optical Character Recognition (OCR): Overview and use cases | SuperAnnotate,” [www.superannotate.com](http://www.superannotate.com). <https://www.superannotate.com/blog/ocr-overview-and-use-cases>



## REFERENCES

- [23] D. V. Everen, "OCR Technology: Its History, How it Works, and Use Cases," Veryfi, May 31, 2023. <https://www.veryfi.com/technology/ocr/>
- [24] "A Journey Through History: The Evolution of OCR Technology," [www.docsumo.com](https://www.docsumo.com/blog/optical-character-recognition-history). <https://www.docsumo.com/blog/optical-character-recognition-history>
- [25] M. A. Aweil, A. I. Abidi, "Review on Optical Character Recognition." [https://www.researchgate.net/publication/334162853\\_REVIEW\\_ON\\_OPTICAL\\_CHARACTER\\_RECOGNITION](https://www.researchgate.net/publication/334162853_REVIEW_ON_OPTICAL_CHARACTER_RECOGNITION)
- [26] A. Fantini, "ALPR System: how work, cameras and components," Tattile - Custom Vision Solutions, Aug. 04, 2021. [https://www.tattile.com/alpr-system/?doing\\_wp\\_cron=1701963919.0260601043701171875000](https://www.tattile.com/alpr-system/?doing_wp_cron=1701963919.0260601043701171875000) (accessed Dec. 10, 2023).
- [27] "Complete Guide to Learning LPR/ALPR Camera Systems." <https://www.sintrones.com/application/lpr-alpr-camera-systems/> (accessed Dec. 10, 2023).
- [28] SPARKLERS. How to Set Up and Use YOLOv8 for Object Detection. (Mar. 7, 2023). Accessed: Sep. 3, 2024. [Online Video]. Available: <https://www.youtube.com/watch?v=zh1CZK7chbo&t=6s>
- [29] Haziq Kamarozaman. Test YOLOv8 Model with Roboflow. (Aug. 2024). Accessed: Sep. 3, 2024. [Online]. Available: <https://universe.roboflow.com/haziq-kamarozaman-klg3n/test-nm39c>
- [30] M. Shroff, "Know Your Neural Network Architecture: More by Understanding These Terms," Medium, May 16, 2023. [Online]. Available: <https://medium.com/@shroffmegha6695/know-your-neural-network-architecture-more-by-understanding-these-terms-67faf4ea0efb>. [Accessed: Sep. 6, 2024].
- [31] G. Boesch, "YOLOv8 Guide," Viso.ai, Dec. 18, 2023. [Online]. Available: <https://viso.ai/deep-learning/yolov8-guide/>. [Accessed: Sep. 6, 2024].

## REFERENCES

- [32] SPARKLERS. Deep Learning Object Detection Tutorial. (Jun. 15, 2023).  
Accessed: Sep. 3, 2024. [Online Video]. Available:  
<https://www.youtube.com/watch?v=k4OIaCOH9NY>
- [33] J. Torres, "COCO Dataset for Object Detection," Ultralytics, Jan. 15, 2024.  
[Online]. Available: <https://docs.ultralytics.com/datasets/detect/coco/>. [Accessed:  
Sep. 3, 2024].



# VALET - SMART PARKING MANAGEMENT SYSTEM FOR UTAR KAMPAR

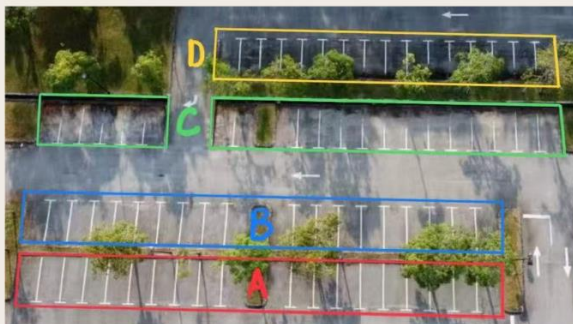
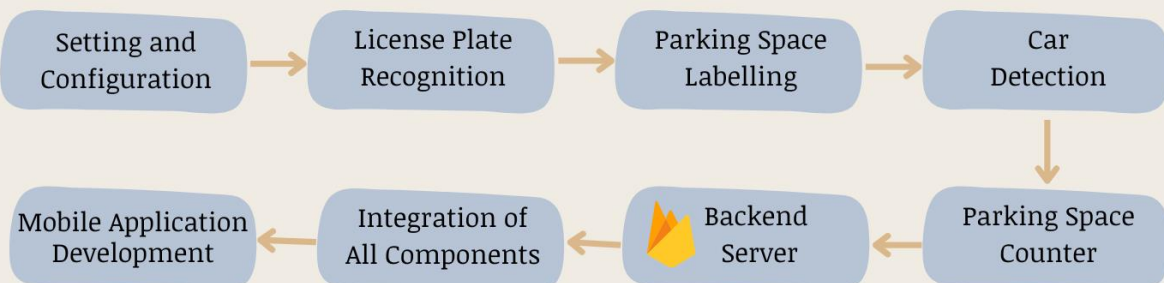
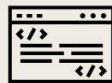
## INTRODUCTION

This project develops a smart parking system using AIoT and OCR technologies. It enables authorization checking, real-time tracking of parked vehicles, and checking parking space availability through a user-friendly app. The system enhances parking convenience and campus security.

## OBJECTIVES

- Building a UTAR Carpark Management App Using FlutterFlow
- Developing a Robust OCR for Car Plate Recognition
- Designing a Labelling System for Parking Bays in UTAR
- Providing Real-Time Parking Zone Monitoring with Live Availability and Visual Updates
- Alerting Users on Full Parking Areas Through In-App Notifications

## METHODOLOGY



## CONCLUSION

The Smart Parking Management System for UTAR Kampar enhances parking efficiency through real-time vehicle detection, automated space monitoring, and a user-friendly mobile app, leading to a more organized, secure, and efficient parking experience for the university community.

**Project Developer: Tan Chern Lynn**  
**Project Supervisor: Dr Aun Yichiet**

## APPENDIX

**Coding Work:****Thonny (parking.py):**

```

import numpy as np
import pandas as pd
import cv2
from picamera2 import Picamera2
from ultralytics import YOLO
import time
import threading
from flask import Flask, Response
from flask_cors import CORS
import signal
import sys
import firebase_admin
from firebase_admin import credentials, db
from datetime import datetime
import base64

# Initialize Flask app with CORS support
app = Flask(__name__)
CORS(app)

# Initialize Firebase
cred = credentials.Certificate("/home/lynn/Desktop/parking/serviceAccountKey.json")
firebase_admin.initialize_app(cred, {
    'databaseURL': 'https://parkingnew-krawlr-default-rtdb.asia-southeast1.firebaseio.com'
})

# Initialize PiCamera
picam2 = Picamera2()

```

## APPENDIX

```
picam2.configure(picam2.create_still_configuration(main={"format": "RGB888"}))
```

```
time.sleep(2)
```

```
picam2.start()
```

```
# Initialize YOLO model
```

```
model = YOLO('yolov8s.pt')
```

```
# Load class list
```

```
with open("/home/lynn/Desktop/parking/coco.txt", "r") as my_file:
```

```
    class_list = my_file.read().split("\n")
```

```
# Define rescaled parking areas (coordinates rescaled manually outside Python)
```

```
parking_areas = [
```

```
    [(761, 1275), (1161, 1278), (761, 1550), (156, 1542)],
```

```
    [(1206, 1284), (1586, 1291), (1378, 1550), (782, 1547)],
```

```
    [(1613, 1284), (1963, 1291), (1931, 1550), (1425, 1547)],
```

```
    [(1997, 1275), (2325, 1280), (2444, 1544), (1972, 1540)],
```

```
    [(2350, 1276), (2723, 1280), (2966, 1544), (2495, 1540)],
```

```
    [(2790, 1293), (3110, 1293), (3465, 1557), (3025, 1560)],
```

```
    [(3200, 1315), (3626, 1320), (4006, 1570), (3540, 1560)]
```

```
]
```

```
def update_firebase(available, occupied):
```

```
    ref = db.reference('/parking')
```

```
    ref.set({
```

```
        'available': int(available),
```

```
        'occupied': [int(x) for x in occupied],
```

```
        'timestamp': datetime.now().strftime("%H:%M:%S")
```

```
    })
```

```
def capture_and_process():
```

```
    while True:
```

```
        try:
```

## APPENDIX

```
frame = picam2.capture_array()
frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
#frame = cv2.resize(frame, (1020, 500)) # Rescale to match predefined boundary
positions
```

```
results = model.predict(frame)
detections = pd.DataFrame(results[0].boxes.data).astype("float")

occupied_spaces = [0] * len(parking_areas)

for _, row in detections.iterrows():
    x1, y1, x2, y2, _, class_id = map(int, row)
    if 'car' in class_list[class_id]:
        cx, cy = (x1 + x2) // 2, y2
        for i, area in enumerate(parking_areas):
            if cv2.pointPolygonTest(np.array(area, np.int32), (cx, cy), False) >= 0:
                cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 197, 0), 5)
                cv2.circle(frame, (cx, cy), 3, (255, 197, 0), -1)
                occupied_spaces[i] = 1
```

```
available_spaces = len(parking_areas) - sum(occupied_spaces)
update_firebase(available_spaces, occupied_spaces)
```

```
# Draw parking areas with more accurate alignment
for i, area in enumerate(parking_areas):
    color = (0, 0, 255) if occupied_spaces[i] else (0, 255, 0)
    cv2.polylines(frame, [np.array(area, np.int32)], True, color, 6)
```

```
# Get mid-point of bottom line (between point 3 and 4)
x3, y3 = area[2]
x4, y4 = area[3]
mid_x = (x3 + x4) // 2
```

## APPENDIX

```
mid_y = (y3 + y4) // 2 + 250 # Slight upward shift

# Draw spot number at bottom middle
cv2.putText(frame, str(i + 1), (mid_x, mid_y),
            cv2.FONT_HERSHEY_SIMPLEX, 2.5, (255, 255, 255), 6)

cv2.putText(frame, f"Available: {available_spaces}/{len(parking_areas)}", (50, 30),
            cv2.FONT_HERSHEY_PLAIN, 3, (255, 255, 255), 4)

output_path = "/home/lynn/Desktop/parking/latest_output.jpg"
success = cv2.imwrite(output_path, frame, [cv2.IMWRITE_JPEG_QUALITY, 90])
if success:
    try:
        with open(output_path, "rb") as f:
            encoded_string = base64.b64encode(f.read()).decode('utf-8')
            data_uri = f"data:image/jpeg;base64,{encoded_string}"
            db.reference("/latest_image_base64").set(data_uri)
    except Exception as e:
        print(f"Error saving base64 to Firebase: {str(e)}")
    else:
        print("Error: Failed to save image!")

time.sleep(10)

except Exception as e:
    print(f"Error in capture_and_process: {str(e)}")
    time.sleep(5)

@app.route('/latest_image.jpg')
def latest_image():
    try:
```

## APPENDIX

```
image_path = "/home/lynn/Desktop/parking/latest_output.jpg"
with open(image_path, "rb") as f:
    return Response(
        f.read(),
        mimetype='image/jpeg',
        headers={
            'Content-Type': 'image/jpeg',
            'Cache-Control': 'no-cache, no-store, must-revalidate',
            'Pragma': 'no-cache',
            'Expires': '0'
        }
    )
except Exception as e:
    return str(e), 500

@app.route('/latest_image_base64')
def latest_image_base64():
    try:
        image_path = "/home/lynn/Desktop/parking/latest_output.jpg"
        with open(image_path, "rb") as f:
            encoded_string = base64.b64encode(f.read()).decode('utf-8')
        data_uri = f"data:image/jpeg;base64,{encoded_string}"
        db.reference("/latest_image_base64").set(data_uri)
        return {"status": "success", "length": len(data_uri)}, 200
    except Exception as e:
        return {"error": str(e)}, 500

@app.route('/parking_data')
def parking_data():
    ref = db.reference('/parking')
    return ref.get()

def signal_handler(sig, frame):
```



## APPENDIX

```
print("\nShutting down gracefully...")
picam2.stop()
sys.exit(0)

if __name__ == "__main__":
    signal.signal(signal.SIGINT, signal_handler)
    threading.Thread(target=capture_and_process, daemon=True).start()
    app.run(host="0.0.0.0", port=5000, debug=False)
```

**Thonny (To obtain Figure 6.6):**

```

import cv2
import pandas as pd
import numpy as np
from ultralytics import YOLO
import time

model=YOLO('yolov8s.pt')

def RGB(event, x, y, flags, param):
    if event == cv2.EVENT_MOUSEMOVE :
        colorsBGR = [x, y]
        print(colorsBGR)

cv2.namedWindow('RGB')
cv2.setMouseCallback('RGB', RGB)

cap=cv2.VideoCapture('pic1.jpg')

my_file = open("coco.txt", "r")
data = my_file.read()
class_list = data.split("\n")

area1=[(161,246),(252,248),(160,294),(34,290)]
area2=[(262,249),(344,250),(299,294),(170,294)]
area3=[(350,250),(427,252),(421,295),(311,294)]
area4=[(434,253),(505,254),(530,298),(429,297)]
area5=[(511,254),(581,254),(633,298),(542,297)]
area6=[(590,257),(646,257),(727,299),(640,300)]
area7=[(658,258),(728,261),(812,302),(738,300)]

```

## APPENDIX

```
while True:
    ret,frame = cap.read()
    if not ret:
        break
    #time.sleep(1)
    frame=cv2.resize(frame,(1020,500))

    results=model.predict(frame)
#   print(results)
    a=results[0].boxes.data
    px=pd.DataFrame(a).astype("float")
#   print(px)

    list1=[]
    list2=[]
    list3=[]
    list4=[]
    list5=[]
    list6=[]
    list7=[]

    for index,row in px.iterrows():
#       print(row)

        x1=int(row[0])
        y1=int(row[1])
        x2=int(row[2])
        y2=int(row[3])
        d=int(row[5])
        c=class_list[d]
        if 'car' in c:
            cx=int(x1+x2)//2
            cy=int(y1+y2)//2
```

```
results1 = cv2.pointPolygonTest(np.array(area1,np.int32),((cx,y2)),False)
```

```
if results1 >= 0:
```

```
    cv2.rectangle(frame,(x1,y1),(x2,y2),(255,197,0),2)
```

```
    cv2.circle(frame,(cx,y2),3,(255,197,0),-1)
```

```
    list1.append(c)
```

```
results2 = cv2.pointPolygonTest(np.array(area2, np.int32), ((cx, y2)), False)
```

```
if results2 >= 0:
```

```
    cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 197, 0), 2)
```

```
    cv2.circle(frame, (cx, y2), 3, (255,197,0), -1)
```

```
    list2.append(c)
```

```
results3 = cv2.pointPolygonTest(np.array(area3, np.int32), ((cx, y2)), False)
```

```
if results3 >= 0:
```

```
    cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 197, 0), 2)
```

```
    cv2.circle(frame, (cx, y2), 3, (255, 197, 0), -1)
```

```
    list3.append(c)
```

```
results4 = cv2.pointPolygonTest(np.array(area4, np.int32), ((cx, y2)), False)
```

```
if results4 >= 0:
```

```
    cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 197, 0), 2)
```

```
    cv2.circle(frame, (cx, y2), 3, (255, 197, 0), -1)
```

```
    list4.append(c)
```

```
results5 = cv2.pointPolygonTest(np.array(area5, np.int32), ((cx, y2)), False)
```

```
if results5 >= 0:
```

```
    cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 197, 0), 2)
```

```
    cv2.circle(frame, (cx, y2), 3, (255, 197, 0), -1)
```

```
    list5.append(c)
```

```
results6 = cv2.pointPolygonTest(np.array(area6, np.int32), ((cx, y2)), False)
```

```
if results6 >= 0:
```

## APPENDIX

```
cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 197, 0), 2)
cv2.circle(frame, (cx, y2), 3, (255, 197, 0), -1)
list6.append(c)

results7 = cv2.pointPolygonTest(np.array(area7, np.int32), ((cx, y2)), False)
if results7 >= 0:
    cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 197, 0), 2)
    cv2.circle(frame, (cx, y2), 3, (255, 197, 0), -1)
    list7.append(c)

a1 = (len(list1))
a2 = (len(list2))
a3 = (len(list3))
a4 = (len(list4))
a5 = (len(list5))
a6 = (len(list6))
a7 = (len(list7))

o = (a1 + a2 + a3 + a4 + a5 + a6 + a7)
space = (7 - o)
print(space)

if a1==1:
    cv2.polylines(frame,[np.array(area1,np.int32)],True,(0,0,255),2)
    cv2.putText(frame,str('1'),(97,335),cv2.FONT_HERSHEY_COMPLEX,0.5,(0,0,255),1)
else:
    cv2.polylines(frame,[np.array(area1,np.int32)],True,(0,255,0),2)
    cv2.putText(frame,str('1'),(97,335),cv2.FONT_HERSHEY_COMPLEX,0.5,(0,255,0),1)

if a2==1:
    cv2.polylines(frame,[np.array(area2,np.int32)],True,(0,0,255),2)
```

## APPENDIX

```
cv2.putText(frame,str('2'),(235,335),cv2.FONT_HERSHEY_COMPLEX,0.5,(0,0,255),1)
```

```
else:
```

```
    cv2.polylines(frame,[np.array(area2,np.int32)],True,(0,255,0),2)
```

```
cv2.putText(frame,str('2'),(235,335),cv2.FONT_HERSHEY_COMPLEX,0.5,(0,255,0),1)
```

```
if a3==1:
```

```
    cv2.polylines(frame,[np.array(area3,np.int32)],True,(0,0,255),2)
```

```
cv2.putText(frame,str('3'),(366,335),cv2.FONT_HERSHEY_COMPLEX,0.5,(0,0,255),1)
```

```
else:
```

```
    cv2.polylines(frame,[np.array(area3,np.int32)],True,(0,255,0),2)
```

```
cv2.putText(frame,str('3'),(366,335),cv2.FONT_HERSHEY_COMPLEX,0.5,(0,255,0),1)
```

```
if a4==1:
```

```
    cv2.polylines(frame,[np.array(area4,np.int32)],True,(0,0,255),2)
```

```
cv2.putText(frame,str('4'),(480,335),cv2.FONT_HERSHEY_COMPLEX,0.5,(0,0,255),1)
```

```
else:
```

```
    cv2.polylines(frame,[np.array(area4,np.int32)],True,(0,255,0),2)
```

```
cv2.putText(frame,str('4'),(480,335),cv2.FONT_HERSHEY_COMPLEX,0.5,(0,255,0),1)
```

```
if a5==1:
```

```
    cv2.polylines(frame,[np.array(area5,np.int32)],True,(0,0,255),2)
```

```
cv2.putText(frame,str('5'),(588,335),cv2.FONT_HERSHEY_COMPLEX,0.5,(0,0,255),1)
```

```
else:
```

```
    cv2.polylines(frame,[np.array(area5,np.int32)],True,(0,255,0),2)
```

```
cv2.putText(frame,str('5'),(588,335),cv2.FONT_HERSHEY_COMPLEX,0.5,(0,255,0),1)
```

## APPENDIX

```
if a6==1:
    cv2.polylines(frame,[np.array(area6,np.int32)],True,(0,0,255),2)

cv2.putText(frame,str('6'),(684,335),cv2.FONT_HERSHEY_COMPLEX,0.5,(0,0,255),1)
else:
    cv2.polylines(frame,[np.array(area6,np.int32)],True,(0,255,0),2)

cv2.putText(frame,str('6'),(684,335),cv2.FONT_HERSHEY_COMPLEX,0.5,(0,255,0),1)

if a7==1:
    cv2.polylines(frame,[np.array(area7,np.int32)],True,(0,0,255),2)

cv2.putText(frame,str('7'),(775,335),cv2.FONT_HERSHEY_COMPLEX,0.5,(0,0,255),1)
else:
    cv2.polylines(frame,[np.array(area7,np.int32)],True,(0,255,0),2)

cv2.putText(frame,str('7'),(775,335),cv2.FONT_HERSHEY_COMPLEX,0.5,(0,255,0),1)

    cv2.putText(frame, "Available: " + str(space) + "/" + str(7), (23, 30),
cv2.FONT_HERSHEY_PLAIN, 2, (255, 255, 255),
    2)

cv2.imshow("RGB", frame)

if cv2.waitKey(0)&0xFF==27:
    break
cap.release()
cv2.destroyAllWindows()
#stream.stop()
```

**Thonny (To obtain Figure 6.7):**

```

import cv2
import pandas as pd
import numpy as np
from ultralytics import YOLO
import time

model = YOLO('yolov8s.pt')

def RGB(event, x, y, flags, param):
    if event == cv2.EVENT_MOUSEMOVE:
        colorsBGR = [x, y]
        print(colorsBGR)

cv2.namedWindow('RGB')
cv2.setMouseCallback('RGB', RGB)

cap = cv2.VideoCapture('pic2.jpg')

my_file = open("coco.txt", "r")
data = my_file.read()
class_list = data.split("\n")

area8 = [(265,224),(333,226),(251,254),(149,254)]
area9 = [(339,226),(412,228),(367,258),(261,254)]
area10 = [(420, 226), (490, 226), (484,259), (375,259)]
area11 = [(576,230), (647,223), (709,267), (592, 262)]
area12 = [(660, 229), (727, 230), (822, 269), (718, 269)]
area13 = [(735, 232), (804, 232), (937, 274), (834, 270)]

while True:

```



## APPENDIX

```
ret, frame = cap.read()
if not ret:
    break
# time.sleep(1)
frame = cv2.resize(frame, (1020, 500))

results = model.predict(frame)
# print(results)
a = results[0].boxes.data
px = pd.DataFrame(a).astype("float")
# print(px)

list8 = []
list9 = []
list10 = []
list11 = []
list12 = []
list13 = []
for index, row in px.iterrows():
    # print(row)

    x1 = int(row[0])
    y1 = int(row[1])
    x2 = int(row[2])
    y2 = int(row[3])
    d = int(row[5])
    c = class_list[d]
    if 'car' in c:
        cx = int(x1 + x2) // 2
        cy = int(y1 + y2) // 2

    results8 = cv2.pointPolygonTest(np.array(area8, np.int32), ((cx, y2)), False)
    if results8 >= 0:
```

```

cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 197, 0), 2)
cv2.circle(frame, (cx, y2), 3, (255, 197, 0), -1)
list8.append(c)

```

```

results9 = cv2.pointPolygonTest(np.array(area9, np.int32), ((cx, y2)), False)
if results9 >= 0:

```

```

    cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 197, 0), 2)
    cv2.circle(frame, (cx, y2), 3, (255, 197, 0), -1)
    list9.append(c)

```

```

results10 = cv2.pointPolygonTest(np.array(area10, np.int32), ((cx, y2)), False)
if results10 >= 0:

```

```

    cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 197, 0), 2)
    cv2.circle(frame, (cx, y2), 3, (255, 197, 0), -1)
    list10.append(c)

```

```

results11 = cv2.pointPolygonTest(np.array(area11, np.int32), ((cx, y2)), False)
if results11 >= 0:

```

```

    cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 197, 0), 2)
    cv2.circle(frame, (cx, y2), 3, (255, 197, 0), -1)
    list11.append(c)

```

```

results12 = cv2.pointPolygonTest(np.array(area12, np.int32), ((cx, y2)), False)
if results12 >= 0:

```

```

    cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 197, 0), 2)
    cv2.circle(frame, (cx, y2), 3, (255, 197, 0), -1)
    list12.append(c)

```

```

results13 = cv2.pointPolygonTest(np.array(area13, np.int32), ((cx, y2)), False)
if results13 >= 0:

```

```

    cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 197, 0), 2)
    cv2.circle(frame, (cx, y2), 3, (255, 197, 0), -1)
    list12.append(c)

```

## APPENDIX

```
a8 = (len(list8))
a9 = (len(list9))
a10 = (len(list10))
a11 = (len(list11))
a12 = (len(list12))
a13 = (len(list13))

o = (a8 + a9 + a10 + a11 + a12 + a13)
space = (6 - o)
print(space)

if a8 == 1:
    cv2.polylines(frame, [np.array(area8, np.int32)], True, (0, 0, 255), 2)
    cv2.putText(frame, str('8'), (200, 290), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0,
255), 1)
else:
    cv2.polylines(frame, [np.array(area8, np.int32)], True, (0, 255, 0), 2)
    cv2.putText(frame, str('8'), (200, 290), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0,
255, 0), 1)

if a9 == 1:
    cv2.polylines(frame, [np.array(area9, np.int32)], True, (0, 0, 255), 2)
    cv2.putText(frame, str('9'), (314, 290), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0,
255), 1)
else:
    cv2.polylines(frame, [np.array(area9, np.int32)], True, (0, 255, 0), 2)
    cv2.putText(frame, str('9'), (314, 290), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0,
255, 0), 1)

if a10 == 1:
    cv2.polylines(frame, [np.array(area10, np.int32)], True, (0, 0, 255), 2)
```

## APPENDIX

```
cv2.putText(frame, str('10'), (429, 290), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 255), 1)
```

```
else:
```

```
cv2.polylines(frame, [np.array(area10, np.int32)], True, (0, 255, 0), 2)
```

```
cv2.putText(frame, str('10'), (429, 290), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0), 1)
```

```
if a11 == 1:
```

```
cv2.polylines(frame, [np.array(area11, np.int32)], True, (0, 0, 255), 2)
```

```
cv2.putText(frame, str('11'), (651, 290), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 255), 1)
```

```
else:
```

```
cv2.polylines(frame, [np.array(area11, np.int32)], True, (0, 255, 0), 2)
```

```
cv2.putText(frame, str('11'), (651, 290), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0), 1)
```

```
if a12 == 1:
```

```
cv2.polylines(frame, [np.array(area12, np.int32)], True, (0, 0, 255), 2)
```

```
cv2.putText(frame, str('12'), (770, 290), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 255), 1)
```

```
else:
```

```
cv2.polylines(frame, [np.array(area12, np.int32)], True, (0, 255, 0), 2)
```

```
cv2.putText(frame, str('12'), (770, 290), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0), 1)
```

```
if a13 == 1:
```

```
cv2.polylines(frame, [np.array(area13, np.int32)], True, (0, 0, 255), 2)
```

```
cv2.putText(frame, str('13'), (889, 290), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 255), 1)
```

```
else:
```

```
cv2.polylines(frame, [np.array(area13, np.int32)], True, (0, 255, 0), 2)
```

```
cv2.putText(frame, str('13'), (889, 290), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0), 1)
```

## APPENDIX

```
    cv2.putText(frame, "Available: " + str(space)+"/6", (23, 30),  
cv2.FONT_HERSHEY_PLAIN, 2, (255, 255, 255),  
    2)
```

```
cv2.imshow("RGB", frame)
```

```
if cv2.waitKey(0) & 0xFF == 27:
```

```
    break
```

```
cap.release()
```

```
cv2.destroyAllWindows()
```

```
# stream.stop()
```

**Thonny (To obtain Figure 6.8):**

```

import cv2
import pandas as pd
import numpy as np
from ultralytics import YOLO
import time

model = YOLO('yolov8s.pt')

def RGB(event, x, y, flags, param):
    if event == cv2.EVENT_MOUSEMOVE:
        colorsBGR = [x, y]
        print(colorsBGR)

cv2.namedWindow('RGB')
cv2.setMouseCallback('RGB', RGB)

cap = cv2.VideoCapture('pic3.jpg')

my_file = open("coco.txt", "r")
data = my_file.read()
class_list = data.split("\n")

area14 = [(273,243),(339,247),(251,285),(157,280)]
area15 = [(345,248),(414,247),(355,287),(262,283)]
area16 = [(419, 247), (491, 249), (464,294), (362,289)]
area17 = [(495,252), (567,253), (575,298), (472, 294)]
area18 = [(573, 255), (647, 256), (694, 303), (586, 299)]
area19 = [(651,254),(727,256),(815,309),(705,304)]
area20 = [(738,259),(812,258),(931,305),(827,310)]

```

## APPENDIX

```
while True:
    ret, frame = cap.read()
    if not ret:
        break
    # time.sleep(1)
    frame = cv2.resize(frame, (1020, 500))

    results = model.predict(frame)
    # print(results)
    a = results[0].boxes.data
    px = pd.DataFrame(a).astype("float")
    # print(px)

    list14 = []
    list15 = []
    list16 = []
    list17 = []
    list18 = []
    list19 = []
    list20 = []

    for index, row in px.iterrows():
        # print(row)

        x1 = int(row[0])
        y1 = int(row[1])
        x2 = int(row[2])
        y2 = int(row[3])
        d = int(row[5])
        c = class_list[d]
        if 'car' in c:
            cx = int(x1 + x2) // 2
            cy = int(y1 + y2) // 2
```

```
results14 = cv2.pointPolygonTest(np.array(area14, np.int32), ((cx, y2)), False)
```

```
if results14 >= 0:
```

```
    cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 197, 0), 2)
```

```
    cv2.circle(frame, (cx, y2), 3, (255, 197, 0), -1)
```

```
    list14.append(c)
```

```
results15 = cv2.pointPolygonTest(np.array(area15, np.int32), ((cx, y2)), False)
```

```
if results15 >= 0:
```

```
    cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 197, 0), 2)
```

```
    cv2.circle(frame, (cx, y2), 3, (255, 197, 0), -1)
```

```
    list15.append(c)
```

```
results16 = cv2.pointPolygonTest(np.array(area16, np.int32), ((cx, y2)), False)
```

```
if results16 >= 0:
```

```
    cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 197, 0), 2)
```

```
    cv2.circle(frame, (cx, y2), 3, (255, 197, 0), -1)
```

```
    list16.append(c)
```

```
results17 = cv2.pointPolygonTest(np.array(area17, np.int32), ((cx, y2)), False)
```

```
if results17 >= 0:
```

```
    cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 197, 0), 2)
```

```
    cv2.circle(frame, (cx, y2), 3, (255, 197, 0), -1)
```

```
    list17.append(c)
```

```
results18 = cv2.pointPolygonTest(np.array(area18, np.int32), ((cx, y2)), False)
```

```
if results18 >= 0:
```

```
    cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 197, 0), 2)
```

```
    cv2.circle(frame, (cx, y2), 3, (255, 197, 0), -1)
```

```
    list18.append(c)
```

```
results19 = cv2.pointPolygonTest(np.array(area19, np.int32), ((cx, y2)), False)
```

```
if results19 >= 0:
```



## APPENDIX

```
cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 197, 0), 2)
cv2.circle(frame, (cx, y2), 3, (255, 197, 0), -1)
list19.append(c)
```

```
results20 = cv2.pointPolygonTest(np.array(area20, np.int32), ((cx, y2)), False)
if results20 >= 0:
    cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 197, 0), 2)
    cv2.circle(frame, (cx, y2), 3, (255, 197, 0), -1)
    list20.append(c)
```

```
a14 = (len(list14))
a15 = (len(list15))
a16 = (len(list16))
a17 = (len(list17))
a18 = (len(list18))
a19 = (len(list19))
a20 = (len(list20))
```

```
o = (a14 + a15 + a16 + a17 + a18 + a19 + a20)
space = (7 - o)
print(space)
```

```
if a14 == 1:
    cv2.polylines(frame, [np.array(area14, np.int32)], True, (0, 0, 255), 2)
    cv2.putText(frame, str('14'), (204, 326), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0,
255), 1)
else:
    cv2.polylines(frame, [np.array(area14, np.int32)], True, (0, 255, 0), 2)
    cv2.putText(frame, str('14'), (204, 326), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0,
255, 0), 1)
```

```
if a15 == 1:
    cv2.polylines(frame, [np.array(area15, np.int32)], True, (0, 0, 255), 2)
```

## APPENDIX

```
cv2.putText(frame, str('15'), (309, 328), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 255), 1)
```

```
else:
```

```
cv2.polylines(frame, [np.array(area15, np.int32)], True, (0, 255, 0), 2)
```

```
cv2.putText(frame, str('15'), (309, 328), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0), 1)
```

```
if a16 == 1:
```

```
cv2.polylines(frame, [np.array(area16, np.int32)], True, (0, 0, 255), 2)
```

```
cv2.putText(frame, str('16'), (413, 328), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 255), 1)
```

```
else:
```

```
cv2.polylines(frame, [np.array(area16, np.int32)], True, (0, 255, 0), 2)
```

```
cv2.putText(frame, str('16'), (413, 328), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0), 1)
```

```
if a17 == 1:
```

```
cv2.polylines(frame, [np.array(area17, np.int32)], True, (0, 0, 255), 2)
```

```
cv2.putText(frame, str('17'), (524, 330), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 255), 1)
```

```
else:
```

```
cv2.polylines(frame, [np.array(area17, np.int32)], True, (0, 255, 0), 2)
```

```
cv2.putText(frame, str('17'), (524, 330), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0), 1)
```

```
if a18 == 1:
```

```
cv2.polylines(frame, [np.array(area18, np.int32)], True, (0, 0, 255), 2)
```

```
cv2.putText(frame, str('18'), (640, 334), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 255), 1)
```

```
else:
```

```
cv2.polylines(frame, [np.array(area18, np.int32)], True, (0, 255, 0), 2)
```

```
cv2.putText(frame, str('18'), (640, 334), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0), 1)
```

## APPENDIX

```
if a19 == 1:
    cv2.polylines(frame, [np.array(area19, np.int32)], True, (0, 0, 255), 2)
    cv2.putText(frame, str('19'), (760, 338), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0,
255), 1)
else:
    cv2.polylines(frame, [np.array(area19, np.int32)], True, (0, 255, 0), 2)
    cv2.putText(frame, str('19'), (760, 338), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0,
255, 0), 1)

if a20 == 1:
    cv2.polylines(frame, [np.array(area20, np.int32)], True, (0, 0, 255), 2)
    cv2.putText(frame, str('20'), (879, 338), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0,
255), 1)
else:
    cv2.polylines(frame, [np.array(area20, np.int32)], True, (0, 255, 0), 2)
    cv2.putText(frame, str('20'), (879, 338), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0,
255, 0), 1)

cv2.putText(frame, "Available: " + str(space)+"/7", (23, 30),
cv2.FONT_HERSHEY_PLAIN, 2, (255, 255, 255),
2)

cv2.imshow("RGB", frame)

if cv2.waitKey(0) & 0xFF == 27:
    break
cap.release()
cv2.destroyAllWindows()
# stream.stop()
```

**Google Colab (License Plate Detection):**

```
!git clone https://github.com/Arijit1080/Licence-Plate-Detection-using-YOLO-V8.git
```

```
cd /content/Licence-Plate-Detection-using-YOLO-V8
```

```
!pip install roboflow
```

```
from roboflow import Roboflow
rf = Roboflow(api_key="KQ1Y7No9mw3gdDcV9Oms")
project = rf.workspace("haziq-kamarozaman-klg3n").project("test-nm39c") version =
project.version(1)
dataset = version.download("yolov8")
```

```
!pip install -r requirements.txt
```

```
!python /content/Licence-Plate-Detection-using-YOLO-
V8/ultralytics/yolo/v8/detect/train.py model=/content/yolov8n.pt data=/content/Licence-
Plate-Detection-using-YOLO-V8/test-1/data.yaml epochs=100
!python /content/Licence-Plate-Detection-using-YOLO-
V8/ultralytics/yolo/v8/detect/predict.py model='/content/Licence-Plate-Detection- using-
YOLO-V8/runs/detect/train11/weights/best.pt' source='/content/Licence- Plate-Detection-
using-YOLO-V8/car.jpg'
```

**Google Colab (License Plate Number Recognition – OCR):**

```
!git clone https://github.com/Arijit1080/Licence-Plate-Detection-and-Recognition-using-  
YOLO-V8-EasyOCR.git
```

```
cd /content/Licence-Plate-Detection-and-Recognition-using-YOLO-V8-EasyOCR
```

```
pip install -r requirements.txt
```

```
!python /content/Licence-Plate-Detection-and-Recognition-using-YOLO-V8-  
EasyOCR/predictWithOCR.py model='/content/best.pt' source='/content/car.jpg'
```

**Flutterflow (firstpage):**

```
import '/flutter_flow/flutter_flow_animations.dart';
import '/flutter_flow/flutter_flow_theme.dart';
import '/flutter_flow/flutter_flow_util.dart';
import '/flutter_flow/flutter_flow_widgets.dart';
import 'dart:math';
import 'dart:ui';
import '/index.dart';
import 'package:flutter/material.dart';
import 'package:flutter/scheduler.dart';
import 'package:flutter_animate/flutter_animate.dart';
import 'package:google_fonts/google_fonts.dart';
import 'package:provider/provider.dart';

import 'firstpage_model.dart';
export 'firstpage_model.dart';

class FirstpageWidget extends StatefulWidget {
  const FirstpageWidget({ super.key });

  static String routeName = 'firstpage';
  static String routePath = '/firstpage';

  @override
  State<FirstpageWidget> createState() => _FirstpageWidgetState();
}

class _FirstpageWidgetState extends State<FirstpageWidget>
  with TickerProviderStateMixin {
  late FirstpageModel _model;

  final scaffoldKey = GlobalKey<ScaffoldState>();
```

## APPENDIX

```
final animationsMap = <String, AnimationInfo>{ };

@override
void initState() {
  super.initState();
  _model = createModel(context, () => FirstpageModel());

  animationsMap.addAll({
    'containerOnPageLoadAnimation1': AnimationInfo(
      trigger: AnimationTrigger.onPageLoad,
      effectsBuilder: () => [
        VisibilityEffect(duration: 1.ms),
        FadeEffect(
          curve: Curves.easeInOut,
          delay: 0.0.ms,
          duration: 400.0.ms,
          begin: 0.0,
          end: 1.0,
        ),
        ScaleEffect(
          curve: Curves.easeInOut,
          delay: 0.0.ms,
          duration: 400.0.ms,
          begin: Offset(3.0, 3.0),
          end: Offset(1.0, 1.0),
        ),
      ],
    ),
    'containerOnPageLoadAnimation2': AnimationInfo(
      trigger: AnimationTrigger.onPageLoad,
      effectsBuilder: () => [
        VisibilityEffect(duration: 300.ms),
        FadeEffect(
```

```

        curve: Curves.easeInOut,
        delay: 300.0.ms,
        duration: 300.0.ms,
        begin: 0.0,
        end: 1.0,
    ),
    ScaleEffect(
        curve: Curves.bounceOut,
        delay: 300.0.ms,
        duration: 300.0.ms,
        begin: Offset(0.6, 0.6),
        end: Offset(1.0, 1.0),
    ),
],
),
'textOnPageLoadAnimation1': AnimationInfo(
    trigger: AnimationTrigger.onPageLoad,
    effectsBuilder: () => [
        VisibilityEffect(duration: 350.ms),
        FadeEffect(
            curve: Curves.easeInOut,
            delay: 350.0.ms,
            duration: 400.0.ms,
            begin: 0.0,
            end: 1.0,
        ),
        MoveEffect(
            curve: Curves.easeInOut,
            delay: 350.0.ms,
            duration: 400.0.ms,
            begin: Offset(0.0, 30.0),
            end: Offset(0.0, 0.0),
        ),
    ],
),

```



```

    ],
  ),
  'textOnPageLoadAnimation2': AnimationInfo(
    trigger: AnimationTrigger.onPageLoad,
    effectsBuilder: () => [
      VisibilityEffect(duration: 400.ms),
      FadeEffect(
        curve: Curves.easeInOut,
        delay: 400.0.ms,
        duration: 400.0.ms,
        begin: 0.0,
        end: 1.0,
      ),
      MoveEffect(
        curve: Curves.easeInOut,
        delay: 400.0.ms,
        duration: 400.0.ms,
        begin: Offset(0.0, 30.0),
        end: Offset(0.0, 0.0),
      ),
    ],
  ),
  'rowOnPageLoadAnimation': AnimationInfo(
    trigger: AnimationTrigger.onPageLoad,
    effectsBuilder: () => [
      VisibilityEffect(duration: 300.ms),
      FadeEffect(
        curve: Curves.easeInOut,
        delay: 300.0.ms,
        duration: 600.0.ms,
        begin: 0.0,
        end: 1.0,
      ),
    ],
  ),

```

## APPENDIX

```
ScaleEffect(  
  curve: Curves.bounceOut,  
  delay: 300.0.ms,  
  duration: 600.0.ms,  
  begin: Offset(0.6, 0.6),  
  end: Offset(1.0, 1.0),  
),  
],  
),  
});  
}
```

```
@override  
void dispose() {  
  _model.dispose();  
  
  super.dispose();  
}
```

```
@override  
Widget build(BuildContext context) {  
  return GestureDetector(  
    onTap: () {  
      FocusScope.of(context).unfocus();  
      FocusManager.instance.primaryFocus?.unfocus();  
    },  
    child: Scaffold(  
      key: scaffoldKey,  
      backgroundColor: FlutterFlowTheme.of(context).secondaryBackground,  
      body: Column(  
        mainAxisAlignment: MainAxisAlignment.max,  
        mainAxisAlignment: MainAxisAlignment.spaceBetween,  
        children: [  

```

```

Expanded(
  child: Container(
    width: double.infinity,
    height: 500,
    decoration: BoxDecoration(
      gradient: LinearGradient(
        colors: [
          Color(0xCA4EE9E4),
          Color(0xFFAC59FF),
          FlutterFlowTheme.of(context).tertiary
        ],
        stops: [0, 0.5, 1],
        begin: AlignmentDirectional(-1, -1),
        end: AlignmentDirectional(1, 1),
      ),
    ),
  child: Container(
    width: 100,
    height: 100,
    decoration: BoxDecoration(
      gradient: LinearGradient(
        colors: [
          Color(0x00FFFFFF),
          FlutterFlowTheme.of(context).secondaryBackground
        ],
        stops: [0, 1],
        begin: AlignmentDirectional(0, -1),
        end: AlignmentDirectional(0, 1),
      ),
    ),
  child: Column(
    mainAxisAlignment: MainAxisAlignment.max,
    mainAxisAlignment: MainAxisAlignment.center,

```

```

children: [
  Container(
    width: 120,
    height: 120,
    decoration: BoxDecoration(
      color: FlutterFlowTheme.of(context).accent4,
      shape: BoxShape.circle,
    ),
    child: Padding(
      padding: EdgeInsets.all(8),
      child: Icon(
        Icons.local_parking,
        color: Color(0xFFD40000),
        size: 44,
      ),
    ),
  ).animateOnPageLoad(
    animationsMap['containerOnPageLoadAnimation2']!),
  Text(
    '\nVALET - Smart Parking System',
    style: FlutterFlowTheme.of(context)
      .displaySmall
      .override(
        font: GoogleFonts.interTight(
          fontWeight: FontWeight.w600,
          fontStyle: FlutterFlowTheme.of(context)
            .displaySmall
            .fontStyle,
        ),
        color: FlutterFlowTheme.of(context).primaryText,
        fontSize: 24,
        letterSpacing: 0.0,
        fontWeight: FontWeight.w600,

```

```

        fontStyle: FlutterFlowTheme.of(context)
          .displaySmall
          .fontStyle,
      ),
    ).animateOnPageLoad(
      animationsMap['textOnPageLoadAnimation1']!),
    Padding(
      padding: EdgeInsetsDirectional.fromSTEB(44, 8, 44, 0),
      child: Text(
        'Welcome UTARIAN! \nGet started on your journey!',
        textAlign: TextAlign.center,
        style:
          FlutterFlowTheme.of(context).labelMedium.override(
            font: GoogleFonts.inter(
              fontWeight: FlutterFlowTheme.of(context)
                .labelMedium
                .fontWeight,
              fontStyle: FlutterFlowTheme.of(context)
                .labelMedium
                .fontStyle,
            ),
            letterSpacing: 0.0,
            fontWeight: FlutterFlowTheme.of(context)
              .labelMedium
              .fontWeight,
            fontStyle: FlutterFlowTheme.of(context)
              .labelMedium
              .fontStyle,
          ),
    ).animateOnPageLoad(
      animationsMap['textOnPageLoadAnimation2']!),
  ),
],

```

```

    ),
    ),
    ).animateOnPageLoad(
        animationsMap['containerOnPageLoadAnimation1']!),
    ),
    Padding(
        padding: EdgeInsetsDirectional.fromSTEB(16, 24, 16, 44),
        child: Row(
            mainAxisAlignment: MainAxisAlignment.max,
            children: [
                Expanded(
                    child: Align(
                        alignment: AlignmentDirectional(0, 0),
                        child: Padding(
                            padding: EdgeInsetsDirectional.fromSTEB(0, 0, 8, 16),
                            child: FFButtonWidget(
                                onPressed: () async {
                                    context.pushNamed(
                                        LoginpageWidget.routeName,
                                        queryParameters: {
                                            'username': serializeParam(
                                                "",
                                                ParamType.String,
                                            ),
                                        }.withoutNulls,
                                    );
                                },
                                text: 'Get Started',
                                options: FFButtonOptions(
                                    width: 230,
                                    height: 52,
                                    padding: EdgeInsetsDirectional.fromSTEB(0, 0, 0, 0),
                                    iconPadding:

```

```
EdgeInsetsDirectional.fromSTEB(0, 0, 0, 0),
color: FlutterFlowTheme.of(context)
    .secondaryBackground,
textStyle:
    FlutterFlowTheme.of(context).bodyLarge.override(
        font: GoogleFonts.inter(
            fontWeight: FlutterFlowTheme.of(context)
                .bodyLarge
                .fontWeight,
            fontStyle: FlutterFlowTheme.of(context)
                .bodyLarge
                .fontStyle,
        ),
        color: Color(0xFF4E8FC3),
        letterSpacing: 0.0,
        fontWeight: FlutterFlowTheme.of(context)
            .bodyLarge
            .fontWeight,
        fontStyle: FlutterFlowTheme.of(context)
            .bodyLarge
            .fontStyle,
    ),
elevation: 0,
borderSide: BorderSide(
    color: FlutterFlowTheme.of(context).alternate,
    width: 2,
),
borderRadius: BorderRadius.circular(12),
),
),
),
),
),
```

## APPENDIX

```
        ],  
        ).animateOnPageLoad(animationsMap['rowOnPageLoadAnimation']!),  
    ),  
    ],  
    ),  
    ),  
);  
}  
}
```



**Flutterflow (loginpage):**

```

import '/backend/backend.dart';
import '/flutter_flow/flutter_flow_animations.dart';
import '/flutter_flow/flutter_flow_theme.dart';
import '/flutter_flow/flutter_flow_util.dart';
import '/flutter_flow/flutter_flow_widgets.dart';
import 'dart:math';
import 'dart:ui';
import '/index.dart';
import 'package:easy_debounce/easy_debounce.dart';
import 'package:flutter/material.dart';
import 'package:flutter/scheduler.dart';
import 'package:flutter_animate/flutter_animate.dart';
import 'package:google_fonts/google_fonts.dart';
import 'package:provider/provider.dart';

import 'loginpage_model.dart';
export 'loginpage_model.dart';

class LoginpageWidget extends StatefulWidget {
  const LoginpageWidget({
    super.key,
    required this.username,
  });

  final String? username;

  static String routeName = 'loginpage';
  static String routePath = '/loginpage';

  @override
  State<LoginpageWidget> createState() => _LoginpageWidgetState();
}

```

```

class _LoginPageWidgetState extends State<LoginPageWidget>
  with TickerProviderStateMixin {
  late LoginpageModel _model;

  final scaffoldKey = GlobalKey<ScaffoldState>();

  final animationsMap = <String, AnimationInfo>{ };

  @override
  void initState() {
    super.initState();
    _model = createModel(context, () => LoginpageModel());

    _model.emailAddressTextController ??= TextEditingController();
    _model.emailAddressFocusNode ??= FocusNode();

    _model.passwordTextController ??= TextEditingController();
    _model.passwordFocusNode ??= FocusNode();

    animationsMap.addAll({
      'containerOnPageLoadAnimation': AnimationInfo(
        trigger: AnimationTrigger.onPageLoad,
        effectsBuilder: () => [
          VisibilityEffect(duration: 1.ms),
          FadeEffect(
            curve: Curves.easeInOut,
            delay: 0.0.ms,
            duration: 300.0.ms,
            begin: 0.0,
            end: 1.0,
          ),
          ScaleEffect(

```

```

        curve: Curves.bounceOut,
        delay: 0.0.ms,
        duration: 300.0.ms,
        begin: Offset(0.6, 1.0),
        end: Offset(1.0, 1.0),
    ),
],
),
'textOnPageLoadAnimation1': AnimationInfo(
    trigger: AnimationTrigger.onPageLoad,
    effectsBuilder: () => [
        VisibilityEffect(duration: 100.ms),
        FadeEffect(
            curve: Curves.easeInOut,
            delay: 100.0.ms,
            duration: 400.0.ms,
            begin: 0.0,
            end: 1.0,
        ),
        MoveEffect(
            curve: Curves.easeInOut,
            delay: 100.0.ms,
            duration: 400.0.ms,
            begin: Offset(0.0, 30.0),
            end: Offset(0.0, 0.0),
        ),
    ],
),
'textOnPageLoadAnimation2': AnimationInfo(
    trigger: AnimationTrigger.onPageLoad,
    effectsBuilder: () => [
        VisibilityEffect(duration: 150.ms),
        FadeEffect(

```

```

        curve: Curves.easeInOut,
        delay: 150.0.ms,
        duration: 400.0.ms,
        begin: 0.0,
        end: 1.0,
    ),
    MoveEffect(
        curve: Curves.easeInOut,
        delay: 150.0.ms,
        duration: 400.0.ms,
        begin: Offset(0.0, 30.0),
        end: Offset(0.0, 0.0),
    ),
],
),
'columnOnPageLoadAnimation': AnimationInfo(
    trigger: AnimationTrigger.onPageLoad,
    effectsBuilder: () => [
        FadeEffect(
            curve: Curves.easeInOut,
            delay: 200.0.ms,
            duration: 400.0.ms,
            begin: 0.0,
            end: 1.0,
        ),
        MoveEffect(
            curve: Curves.easeInOut,
            delay: 200.0.ms,
            duration: 400.0.ms,
            begin: Offset(0.0, 60.0),
            end: Offset(0.0, 0.0),
        ),
        TiltEffect(

```

## APPENDIX

```
        curve: Curves.easeInOut,  
        delay: 200.0.ms,  
        duration: 400.0.ms,  
        begin: Offset(-0.349, 0),  
        end: Offset(0, 0),  
      ),  
    ],  
  ),  
});  
}
```

```
@override  
void dispose() {  
  _model.dispose();  
  
  super.dispose();  
}
```

```
@override  
Widget build(BuildContext context) {  
  return GestureDetector(  
    onTap: () {  
      FocusScope.of(context).unfocus();  
      FocusManager.instance.primaryFocus?.unfocus();  
    },  
    child: Scaffold(  
      key: scaffoldKey,  
      backgroundColor: FlutterFlowTheme.of(context).secondaryBackground,  
      body: SingleChildScrollView(  
        child: Column(  
          mainAxisAlignment: MainAxisAlignment.max,  
          children: [  
            Container(  

```

```

width: double.infinity,
height: 427.66,
decoration: BoxDecoration(
  gradient: LinearGradient(
    colors: [
      Color(0xFF8FCA9B),
      Color(0xFFA075F8),
      FlutterFlowTheme.of(context).tertiary
    ],
    stops: [0, 0.5, 1],
    begin: AlignmentDirectional(-1, -1),
    end: AlignmentDirectional(1, 1),
  ),
),
child: Container(
  width: 100,
  height: 100,
  decoration: BoxDecoration(
    gradient: LinearGradient(
      colors: [
        Color(0x00FFFFFF),
        FlutterFlowTheme.of(context).secondaryBackground
      ],
      stops: [0, 1],
      begin: AlignmentDirectional(0, -1),
      end: AlignmentDirectional(0, 1),
    ),
  ),
  child: Padding(
    padding: EdgeInsetsDirectional.fromSTEB(0, 50, 0, 0),
    child: Column(
      mainAxisAlignment: MainAxisAlignment.max,
      mainAxisSize: MainAxisSize.max,
      crossAxisAlignment: CrossAxisAlignment.center,

```

```

children: [
  Container(
    width: 100,
    height: 100,
    decoration: BoxDecoration(
      color: FlutterFlowTheme.of(context).accent4,
      borderRadius: BorderRadius.circular(16),
    ),
    child: Padding(
      padding: EdgeInsets.all(8),
      child: Icon(
        Icons.local_parking,
        color: Color(0xFFD40000),
        size: 44,
      ),
    ),
  ).animateOnPageLoad(
    animationsMap['containerOnPageLoadAnimation']!),
  Padding(
    padding: EdgeInsetsDirectional.fromSTEB(0, 10, 0, 0),
    child: Text(
      'VALET - Smart Parking System',
      style: FlutterFlowTheme.of(context)
        .headlineSmall
        .override(
          font: GoogleFonts.interTight(
            fontWeight: FlutterFlowTheme.of(context)
              .headlineSmall
              .fontWeight,
            fontStyle: FlutterFlowTheme.of(context)
              .headlineSmall
              .fontStyle,
          ),
        ),
    ),
  ),

```





```

    ),
    ).animateOnPageLoad(
      animationsMap['textOnPageLoadAnimation2']!),
    ),
  ],
),
),
),
),
),
Align(
  alignment: AlignmentDirectional(0, 0),
  child: Padding(
    padding: EdgeInsetsDirectional.fromSTEB(16, 30, 16, 16),
    child: Column(
      mainAxisAlignment: MainAxisAlignment.max,
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Padding(
          padding: EdgeInsetsDirectional.fromSTEB(0, 0, 0, 16),
          child: Container(
            width: double.infinity,
            child: TextFormField(
              controller: _model.emailAddressTextController,
              focusNode: _model.emailAddressFocusNode,
              onChanged: (_) => EasyDebounce.debounce(
                '_model.emailAddressTextController',
                Duration(milliseconds: 2000),
                () => safeSetState(() {})),
            ),
            autofocus: true,
            autofillHints: [AutofillHints.email],
            textInputAction: TextInputAction.next,
            obscureText: false,

```

```

decoration: InputDecoration(
  labelText: 'Email',
  labelStyle: FlutterFlowTheme.of(context)
    .labelMedium
    .override(
      font: GoogleFonts.inter(
        fontWeight: FlutterFlowTheme.of(context)
          .labelMedium
          .fontWeight,
        fontStyle: FlutterFlowTheme.of(context)
          .labelMedium
          .fontStyle,
      ),
      letterSpacing: 0.0,
      fontWeight: FlutterFlowTheme.of(context)
        .labelMedium
        .fontWeight,
      fontStyle: FlutterFlowTheme.of(context)
        .labelMedium
        .fontStyle,
    ),
  enabledBorder: OutlineInputBorder(
    borderSide: BorderSide(
      color: FlutterFlowTheme.of(context).alternate,
      width: 2,
    ),
    borderRadius: BorderRadius.circular(12),
  ),
  focusedBorder: OutlineInputBorder(
    borderSide: BorderSide(
      color: FlutterFlowTheme.of(context).primary,
      width: 2,
    ),
  ),

```

```

borderRadius: BorderRadius.circular(12),
),
errorBorder: OutlineInputBorder(
borderSide: BorderSide(
color: FlutterFlowTheme.of(context).error,
width: 2,
),
borderRadius: BorderRadius.circular(12),
),
focusedErrorBorder: OutlineInputBorder(
borderSide: BorderSide(
color: FlutterFlowTheme.of(context).error,
width: 2,
),
borderRadius: BorderRadius.circular(12),
),
filled: true,
fillColor: FlutterFlowTheme.of(context)
.secondaryBackground,
contentPadding: EdgeInsets.all(24),
suffixIcon: _model.emailAddressTextController!
.text.isNotEmpty
? InkWell(
onTap: () async {
_model.emailAddressTextController
?.clear();
safeSetState(() {});
},
child: Icon(
Icons.clear,
color: Color(0xFF757575),
size: 22,
),

```

```

        )
        : null,
    ),
    style: FlutterFlowTheme.of(context)
        .bodyMedium
        .override(
            font: GoogleFonts.inter(
                fontWeight: FlutterFlowTheme.of(context)
                    .bodyMedium
                    .fontWeight,
                fontStyle: FlutterFlowTheme.of(context)
                    .bodyMedium
                    .fontStyle,
            ),
            letterSpacing: 0.0,
            fontWeight: FlutterFlowTheme.of(context)
                .bodyMedium
                .fontWeight,
            fontStyle: FlutterFlowTheme.of(context)
                .bodyMedium
                .fontStyle,
        ),
    keyboardType: TextInputType.emailAddress,
    validator: _model
        .emailAddressTextControllerValidator
        .asValidator(context),
    ),
    ),
    ),
    Padding(
        padding: EdgeInsetsDirectional.fromSTEB(0, 0, 0, 16),
        child: Container(
            width: double.infinity,

```

```

child: TextFormField(
  controller: _model.passwordTextController,
  focusNode: _model.passwordFocusNode,
  autofocus: false,
  autofillHints: [AutofillHints.password],
  textInputAction: TextInputAction.done,
  obscureText: !_model.passwordVisibility,
  decoration: InputDecoration(
    labelText: 'Password',
    labelStyle: FlutterFlowTheme.of(context)
      .labelMedium
      .override(
        font: GoogleFonts.inter(
          fontWeight: FlutterFlowTheme.of(context)
            .labelMedium
            .fontWeight,
          fontStyle: FlutterFlowTheme.of(context)
            .labelMedium
            .fontStyle,
        ),
        letterSpacing: 0.0,
        fontWeight: FlutterFlowTheme.of(context)
          .labelMedium
          .fontWeight,
        fontStyle: FlutterFlowTheme.of(context)
          .labelMedium
          .fontStyle,
      ),
    enabledBorder: OutlineInputBorder(
      borderSide: BorderSide(
        color: FlutterFlowTheme.of(context).alternate,
        width: 2,
      ),
    ),
  ),
)

```

```

borderRadius: BorderRadius.circular(12),
),
focusedBorder: OutlineInputBorder(
borderSide: BorderSide(
color: FlutterFlowTheme.of(context).primary,
width: 2,
),
borderRadius: BorderRadius.circular(12),
),
errorBorder: OutlineInputBorder(
borderSide: BorderSide(
color: FlutterFlowTheme.of(context).error,
width: 2,
),
borderRadius: BorderRadius.circular(12),
),
focusedErrorBorder: OutlineInputBorder(
borderSide: BorderSide(
color: FlutterFlowTheme.of(context).error,
width: 2,
),
borderRadius: BorderRadius.circular(12),
),
filled: true,
fillColor: FlutterFlowTheme.of(context)
.secondaryBackground,
contentPadding: EdgeInsets.all(24),
suffixIcon: InkWell(
onTap: () => safeSetState(
() => _model.passwordVisibility =
!_model.passwordVisibility,
),
focusNode: FocusNode(skipTraversal: true),

```

```

child: Icon(
  _model.passwordVisibility
    ? Icons.visibility_outlined
    : Icons.visibility_off_outlined,
  color: FlutterFlowTheme.of(context)
    .secondaryText,
  size: 24,
),
),
),
style: FlutterFlowTheme.of(context)
  .bodyMedium
  .override(
    font: GoogleFonts.inter(
      fontWeight: FlutterFlowTheme.of(context)
        .bodyMedium
        .fontWeight,
      fontStyle: FlutterFlowTheme.of(context)
        .bodyMedium
        .fontStyle,
    ),
    letterSpacing: 0.0,
    fontWeight: FlutterFlowTheme.of(context)
      .bodyMedium
      .fontWeight,
    fontStyle: FlutterFlowTheme.of(context)
      .bodyMedium
      .fontStyle,
  ),
validator: _model.passwordTextControllerValidator
  .asValidator(context),
),
),

```

```

),
Align(
  alignment: AlignmentDirectional(0, 0),
  child: Padding(
    padding: EdgeInsetsDirectional.fromSTEB(0, 0, 0, 16),
    child: StreamBuilder<List<UserRecord>>(
      stream: queryUserRecord(
        queryBuilder: (userRecord) => userRecord
          .where(
            'email',
            isEqualTo:
              _model.emailAddressTextController.text,
          )
          .where(
            'password',
            isEqualTo:
              _model.passwordTextController.text,
          ),
        singleRecord: true,
      ),
      builder: (context, snapshot) {
        // Customize what your widget looks like when it's loading.
        if (!snapshot.hasData) {
          return Center(
            child: SizedBox(
              width: 50,
              height: 50,
              child: CircularProgressIndicator(
                valueColor: AlwaysStoppedAnimation<Color>(
                  FlutterFlowTheme.of(context).primary,
                ),
              ),
            ),
          ),
        ),
      ),
    ),
  ),

```



```

    );
}
List<UserRecord> buttonUserRecordList =
    snapshot.data!;
final buttonUserRecord =
    buttonUserRecordList.isNotEmpty
    ? buttonUserRecordList.first
    : null;

return FFButtonWidget(
  onPressed: () async {
    if ((_model.emailAddressTextController.text ==
        buttonUserRecord?.email) &&
        (_model.passwordTextController.text ==
        buttonUserRecord?.password)) {
      context.pushNamed(
        HomepageWidget.routeName,
        queryParameters: {
          'username': serializeParam(
            valueOrDefault<String>(
              buttonUserRecord?.username,
              'guest',
            ),
            ParamType.String,
          ),
        }.withoutNulls,
      );

      return;
    } else {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text(

```

```

        'Invalid email/password',
        style: TextStyle(
          color: FlutterFlowTheme.of(context)
            .primaryText,
        ),
      ),
      duration: Duration(milliseconds: 4000),
      backgroundColor:
        FlutterFlowTheme.of(context)
          .secondary,
    ),
  );
  return;
}
},
text: 'Sign In',
options: FFButtonOptions(
  width: 230,
  height: 52,
  padding: EdgeInsetsDirectional.fromSTEB(
    0, 0, 0, 0),
  iconPadding: EdgeInsetsDirectional.fromSTEB(
    0, 0, 0, 0),
  color: Color(0xFF78A7D6),
  textStyle: FlutterFlowTheme.of(context)
    .titleSmall
    .override(
      font: GoogleFonts.interTight(
        fontWeight:
          FlutterFlowTheme.of(context)
            .titleSmall
            .fontWeight,
        fontStyle:

```

```

        FlutterFlowTheme.of(context)
        .titleSmall
        .fontStyle,
    ),
    color: Colors.white,
    letterSpacing: 0.0,
    fontWeight: FlutterFlowTheme.of(context)
        .titleSmall
        .fontWeight,
    fontStyle: FlutterFlowTheme.of(context)
        .titleSmall
        .fontStyle,
    ),
    elevation: 3,
    borderSide: BorderSide(
        color: Colors.transparent,
        width: 1,
    ),
    borderRadius: BorderRadius.circular(12),
    ),
    );
    },
    ),
    ),
    ),
    ],
    ).animateOnPageLoad(
        animationsMap['columnOnPageLoadAnimation']!),
    ),
    ),
    ],
    ),
    ),

```

## APPENDIX

```
),  
);  
}  
}
```

**Flutterflow (homepage):**

```

import '/backend/api_requests/api_calls.dart';
import '/backend/backend.dart';
import '/components/notification_widget.dart';
import '/flutter_flow/flutter_flow_animations.dart';
import '/flutter_flow/flutter_flow_theme.dart';
import '/flutter_flow/flutter_flow_util.dart';
import '/flutter_flow/flutter_flow_widgets.dart';
import 'dart:math';
import 'dart:ui';
import '/index.dart';
import 'package:flutter/material.dart';
import 'package:flutter/scheduler.dart';
import 'package:flutter_animate/flutter_animate.dart';
import 'package:google_fonts/google_fonts.dart';
import 'package:provider/provider.dart';

import 'homepage_model.dart';
export 'homepage_model.dart';

class HomepageWidget extends StatefulWidget {
  const HomepageWidget({
    super.key,
    String? username,
  }) : this.username = username ?? 'guest';

  final String username;

  static String routeName = 'homepage';
  static String routePath = '/homepage';

  @override
  State<HomepageWidget> createState() => _HomepageWidgetState();

```

## APPENDIX

```
}
```

```
class _HomepageWidgetState extends State<HomepageWidget>
```

```
  with TickerProviderStateMixin {
```

```
  late HomepageModel _model;
```

```
  final scaffoldKey = GlobalKey<ScaffoldState>();
```

```
  final animationsMap = <String, AnimationInfo>{};
```

```
  @override
```

```
  void initState() {
```

```
    super.initState();
```

```
    _model = createModel(context, () => HomepageModel());
```

```
    _model.tabBarController = TabController(
```

```
      vsync: this,
```

```
      length: 2,
```

```
      initialIndex: 0,
```

```
    )..addListener(() => safeSetState(() {}));
```

```
    animationsMap.addAll({
```

```
      'containerOnPageLoadAnimation1': AnimationInfo(
```

```
        trigger: AnimationTrigger.onPageLoad,
```

```
        effectsBuilder: () => [
```

```
          FadeEffect(
```

```
            curve: Curves.easeInOut,
```

```
            delay: 0.0.ms,
```

```
            duration: 600.0.ms,
```

```
            begin: 0.0,
```

```
            end: 1.0,
```

```
          ),
```

```
          MoveEffect(
```

```

        curve: Curves.easeInOut,
        delay: 0.0.ms,
        duration: 600.0.ms,
        begin: Offset(30.0, 0.0),
        end: Offset(0.0, 0.0),
    ),
],
),
'containerOnPageLoadAnimation2': AnimationInfo(
    trigger: AnimationTrigger.onPageLoad,
    effectsBuilder: () => [
        FadeEffect(
            curve: Curves.easeInOut,
            delay: 0.0.ms,
            duration: 600.0.ms,
            begin: 0.0,
            end: 1.0,
        ),
        MoveEffect(
            curve: Curves.easeInOut,
            delay: 0.0.ms,
            duration: 600.0.ms,
            begin: Offset(50.0, 0.0),
            end: Offset(0.0, 0.0),
        ),
    ],
),
});
setupAnimations(
    animationsMap.values.where((anim) =>
        anim.trigger == AnimationTrigger.onActionTrigger ||
        !anim.applyInitialState),
    this,

```

## APPENDIX

```
);  
}
```

```
@override  
void dispose() {  
  _model.dispose();  
  
  super.dispose();  
}
```

```
@override  
Widget build(BuildContext context) {  
  return FutureBuilder<ApiCallResponse>(  
    future: AvailabilityCall.call(),  
    builder: (context, snapshot) {  
      // Customize what your widget looks like when it's loading.  
      if (!snapshot.hasData) {  
        return Scaffold(  
          backgroundColor: FlutterFlowTheme.of(context).primaryBackground,  
          body: Center(  
            child: SizedBox(  
              width: 50,  
              height: 50,  
              child: CircularProgressIndicator(  
                valueColor: AlwaysStoppedAnimation<Color>(  
                  FlutterFlowTheme.of(context).primary,  
                ),  
              ),  
            ),  
          ),  
        );  
      }  
      final homepageAvailabilityResponse = snapshot.data!;
```



```

return GestureDetector(
  onTap: () {
    FocusScope.of(context).unfocus();
    FocusManager.instance.primaryFocus?.unfocus();
  },
  child: Scaffold(
    key: scaffoldKey,
    backgroundColor: FlutterFlowTheme.of(context).primaryBackground,
    body: SingleChildScrollView(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.max,
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Padding(
            padding: EdgeInsetsDirectional.fromSTEB(16, 44, 16, 12),
            child: Row(
              mainAxisAlignment: MainAxisAlignment.max,
              children: [
                Card(
                  clipBehavior: Clip.antiAliasWithSaveLayer,
                  color: FlutterFlowTheme.of(context).primary,
                  shape: RoundedRectangleBorder(
                    borderRadius: BorderRadius.circular(40),
                  ),
                  child: ClipRRect(
                    borderRadius: BorderRadius.circular(0),
                    child: Image.network(

```

'https://static.vecteezy.com/system/resources/previews/021/548/095/original/default-profile-picture-avatar-user-avatar-icon-person-icon-head-icon-profile-picture-icons-default-anonymous-user-male-and-female-businessman-photo-placeholder-social-network-avatar-portrait-free-vector.jpg',

```

        width: 58.6,
        height: 56.6,
        fit: BoxFit.fill,
      ),
    ),
  ),
  Padding(
    padding: EdgeInsetsDirectional.fromSTEB(12, 0, 0, 0),
    child: Column(
      mainAxisAlignment: MainAxisAlignment.max,
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Text(
          valueOrDefault<String>(
            widget!.username,
            'user',
          ),
          style: FlutterFlowTheme.of(context)
            .titleLarge
            .override(
              font: GoogleFonts.interTight(
                fontWeight: FlutterFlowTheme.of(context)
                  .titleLarge
                  .fontWeight,
                fontStyle: FlutterFlowTheme.of(context)
                  .titleLarge
                  .fontStyle,
              ),
              letterSpacing: 0.0,
              fontWeight: FlutterFlowTheme.of(context)
                .titleLarge
                .fontWeight,
              fontStyle: FlutterFlowTheme.of(context)

```

```

        .titleLarge
        .fontStyle,
    ),
),
],
),
),
],
),
),
Padding(
  padding: EdgeInsetsDirectional.fromSTEB(16, 0, 16, 0),
  child: Container(
    width: double.infinity,
    height: 50,
    decoration: BoxDecoration(
      color: FlutterFlowTheme.of(context).secondaryBackground,
      boxShadow: [
        BoxShadow(
          blurRadius: 4,
          color: Color(0x34090F13),
          offset: Offset(
            0.0,
            2,
          ),
        )
      ],
      borderRadius: BorderRadius.circular(40),
    ),
    child: FFButtonWidget(
      onPressed: (homepageAvailabilityResponse.bodyText != '0'
        ? true
        : false)
    )
  )
)

```

```

? null
: () async {
  await showModalBottomSheet(
    isScrollControlled: true,
    backgroundColor: Colors.transparent,
    enableDrag: false,
    context: context,
    builder: (context) {
      return GestureDetector(
        onTap: () {
          FocusScope.of(context).unfocus();
          FocusManager.instance.primaryFocus
            ?.unfocus();
        },
        child: Padding(
          padding:
            MediaQuery.viewInsetsOf(context),
          child: NotificationWidget(),
        ),
      );
    },
  ).then((value) => safeSetState(() {}));
},
text: 'New Notification ( 1 )',
options: FFButtonOptions(
  height: 40,
  padding: EdgeInsetsDirectional.fromSTEB(16, 0, 16, 0),
  iconPadding:
    EdgeInsetsDirectional.fromSTEB(0, 0, 0, 0),
  color: Color(0x93D3D3DA),
  textStyle:
    FlutterFlowTheme.of(context).titleSmall.override(
      font: GoogleFonts.interTight(

```

```

        fontWeight: FontWeight.w900,
        fontStyle: FlutterFlowTheme.of(context)
            .titleSmall
            .fontStyle,
    ),
    color: Color(0xFFCF4747),
    letterSpacing: 0.0,
    fontWeight: FontWeight.w900,
    fontStyle: FlutterFlowTheme.of(context)
        .titleSmall
        .fontStyle,
    ),
    elevation: 0,
    borderSide: BorderSide(
        color: Color(0x65249689),
    ),
    borderRadius: BorderRadius.circular(8),
    disabledColor: Color(0x93D3D3DA),
    disabledTextColor: Color(0x00D3D3DA),
    ),
    ),
    ),
    ),
    Padding(
        padding: EdgeInsetsDirectional.fromSTEB(16, 12, 0, 0),
        child: Text(
            'Parking Location',
            style:
                FlutterFlowTheme.of(context).headlineSmall.override(
                    font: GoogleFonts.interTight(
                        fontWeight: FlutterFlowTheme.of(context)
                            .headlineSmall
                            .fontWeight,

```

```

        fontStyle: FlutterFlowTheme.of(context)
          .headlineSmall
          .fontStyle,
      ),
      letterSpacing: 0.0,
      fontWeight: FlutterFlowTheme.of(context)
        .headlineSmall
        .fontWeight,
      fontStyle: FlutterFlowTheme.of(context)
        .headlineSmall
        .fontStyle,
    ),
  ),
),
Container(
  width: double.infinity,
  height: 250,
  decoration: BoxDecoration(
    color: FlutterFlowTheme.of(context).primaryBackground,
  ),
  child: ListView(
    padding: EdgeInsets.zero,
    primary: false,
    shrinkWrap: true,
    scrollDirection: Axis.horizontal,
    children: [
      Padding(
        padding:
          EdgeInsetsDirectional.fromSTEB(16, 12, 12, 12),
        child: Container(
          width: 230,
          height: 50,
          decoration: BoxDecoration(

```

```

color: FlutterFlowTheme.of(context)
    .secondaryBackground,
boxShadow: [
    BoxShadow(
        blurRadius: 4,
        color: Color(0x34090F13),
        offset: Offset(
            0.0,
            2,
        ),
    )
],
borderRadius: BorderRadius.circular(12),
),
child: Column(
    mainAxisAlignment: MainAxisAlignment.max,
    children: [
        Container(
            width: double.infinity,
            height: 140,
            decoration: BoxDecoration(
                color: Color(0xFF79799B),
                borderRadius: BorderRadius.only(
                    bottomLeft: Radius.circular(0),
                    bottomRight: Radius.circular(0),
                    topLeft: Radius.circular(12),
                    topRight: Radius.circular(12),
                ),
            ),
        ),
        child: Padding(
            padding: EdgeInsets.all(12),
            child: Column(
                mainAxisAlignment: MainAxisAlignment.max,

```

```

mainAxisAlignment:
  MainAxisAlignment.spaceBetween,
crossAxisAlignment:
  CrossAxisAlignment.start,
children: [
  Container(
    width: 36,
    height: 36,
    decoration: BoxDecoration(
      color: Color(0x98FFFFFF),
      borderRadius:
        BorderRadius.circular(12),
    ),
    alignment: AlignmentDirectional(0, 0),
    child: Icon(
      Icons.desktop_windows,
      color: Colors.white,
      size: 20,
    ),
  ),
  InkWell(
    splashColor: Colors.transparent,
    focusColor: Colors.transparent,
    hoverColor: Colors.transparent,
    highlightColor: Colors.transparent,
    onTap: () async {
      context.pushNamed(
        BlockNWidget.routeName);
    },
    child: Text(
      'Block N',
      style: FlutterFlowTheme.of(context)
        .titleMedium

```



```

.override(
  font: GoogleFonts.interTight(
    fontWeight:
      FlutterFlowTheme.of(
        context)
        .titleMedium
        .fontWeight,
    fontStyle:
      FlutterFlowTheme.of(
        context)
        .titleMedium
        .fontStyle,
  ),
  color: Colors.white,
  letterSpacing: 0.0,
  fontWeight:
    FlutterFlowTheme.of(
      context)
      .titleMedium
      .fontWeight,
  fontStyle:
    FlutterFlowTheme.of(
      context)
      .titleMedium
      .fontStyle,
),
),
),
Text(
  'Zone A to Zone D',
  style: FlutterFlowTheme.of(context)
    .bodyMedium
    .override(

```

```

font: GoogleFonts.inter(
  fontWeight:
    FlutterFlowTheme.of(
      context)
      .bodyMedium
      .fontWeight,
  fontStyle:
    FlutterFlowTheme.of(
      context)
      .bodyMedium
      .fontStyle,
),
color: Colors.white,
letterSpacing: 0.0,
fontWeight:
  FlutterFlowTheme.of(context)
    .bodyMedium
    .fontWeight,
fontStyle:
  FlutterFlowTheme.of(context)
    .bodyMedium
    .fontStyle,
),
),
],
),
),
),
),
Padding(
  padding: EdgeInsetsDirectional.fromSTEB(
    12, 12, 12, 0),
  child: Row(
    mainAxisAlignment: MainAxisAlignment.max,

```

```

mainAxisAlignment:
  MainAxisAlignment.spaceBetween,
children: [
  Padding(
    padding: EdgeInsetsDirectional.fromSTEB(
      0, 0, 8, 0),
    child: Text(
      '${homepageAvailabilityResponse.bodyText} empty spaces',
      style: FlutterFlowTheme.of(context)
        .bodyMedium
        .override(
          font: GoogleFonts.inter(
            fontWeight:
              FlutterFlowTheme.of(
                context)
                .bodyMedium
                .fontWeight,
            fontStyle:
              FlutterFlowTheme.of(
                context)
                .bodyMedium
                .fontStyle,
          ),
          letterSpacing: 0.0,
          fontWeight:
            FlutterFlowTheme.of(context)
              .bodyMedium
              .fontWeight,
          fontStyle:
            FlutterFlowTheme.of(context)
              .bodyMedium
              .fontStyle,
        ),
    ),
  ),
  ),

```

```

        ),
      ),
    ],
  ),
),
],
),
).animateOnPageLoad(
  animationsMap['containerOnPageLoadAnimation1']!),
),
Padding(
  padding:
    EdgeInsetsDirectional.fromSTEB(0, 12, 16, 12),
  child: Container(
    width: 230,
    height: 50,
    decoration: BoxDecoration(
      color: FlutterFlowTheme.of(context)
        .secondaryBackground,
      boxShadow: [
        BoxShadow(
          blurRadius: 4,
          color: Color(0x34090F13),
          offset: Offset(
            0.0,
            2,
          ),
        )
      ],
      borderRadius: BorderRadius.circular(12),
    ),
    child: Column(
      mainAxisAlignment: MainAxisAlignment.max,

```

```

children: [
  Container(
    width: double.infinity,
    height: 140,
    decoration: BoxDecoration(
      color: Color(0xFF596E61),
      borderRadius: BorderRadius.only(
        bottomLeft: Radius.circular(0),
        bottomRight: Radius.circular(0),
        topLeft: Radius.circular(12),
        topRight: Radius.circular(12),
      ),
    ),
    child: Padding(
      padding: EdgeInsets.all(12),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Container(
            width: 36,
            height: 36,
            decoration: BoxDecoration(
              color: Color(0x98FFFFFF),
              borderRadius:
                BorderRadius.circular(12),
            ),
            alignment: AlignmentDirectional(0, 0),
            child: Icon(
              Icons.design_services_outlined,

```

```

        color: Colors.white,
        size: 20,
      ),
    ),
    Text(
      'Block A',
      style: FlutterFlowTheme.of(context)
        .titleMedium
        .override(
          font: GoogleFonts.interTight(
            fontWeight:
              FlutterFlowTheme.of(
                context)
                .titleMedium
                .fontWeight,
            fontStyle:
              FlutterFlowTheme.of(
                context)
                .titleMedium
                .fontStyle,
          ),
          color: Colors.white,
          letterSpacing: 0.0,
          fontWeight:
            FlutterFlowTheme.of(context)
              .titleMedium
              .fontWeight,
          fontStyle:
            FlutterFlowTheme.of(context)
              .titleMedium
              .fontStyle,
        ),
    ),
  ),

```

```

Text(
  'Coming Soon',
  style: FlutterFlowTheme.of(context)
    .bodyMedium
    .override(
      font: GoogleFonts.inter(
        fontWeight:
          FlutterFlowTheme.of(
            context)
            .bodyMedium
            .fontWeight,
        fontStyle:
          FlutterFlowTheme.of(
            context)
            .bodyMedium
            .fontStyle,
      ),
      color: Colors.white,
      letterSpacing: 0.0,
      fontWeight:
        FlutterFlowTheme.of(context)
          .bodyMedium
          .fontWeight,
      fontStyle:
        FlutterFlowTheme.of(context)
          .bodyMedium
          .fontStyle,
    ),
),
],
),
),
),
),

```

```

Padding(
  padding: EdgeInsetsDirectional.fromSTEB(
    12, 12, 12, 0),
  child: Row(
    mainAxisAlignment: MainAxisAlignment.max,
    mainAxisAlignment:
      MainAxisAlignment.spaceBetween,
    children: [
      Padding(
        padding: EdgeInsetsDirectional.fromSTEB(
          0, 0, 8, 0),
        child: Text(
          'Coming Soon',
          style: FlutterFlowTheme.of(context)
            .bodyMedium
            .override(
              font: GoogleFonts.inter(
                fontWeight:
                  FlutterFlowTheme.of(
                    context)
                    .bodyMedium
                    .fontWeight,
                fontStyle:
                  FlutterFlowTheme.of(
                    context)
                    .bodyMedium
                    .fontStyle,
              ),
              letterSpacing: 0.0,
              fontWeight:
                FlutterFlowTheme.of(context)
                  .bodyMedium
                  .fontWeight,

```



```

        fontStyle:
          FlutterFlowTheme.of(context)
            .bodyMedium
            .fontStyle,
      ),
    ),
  ),
],
),
),
],
),
).animateOnPageLoad(
  animationsMap['containerOnPageLoadAnimation2']!),
),
],
),
),
Padding(
  padding: EdgeInsetsDirectional.fromSTEB(16, 12, 0, 0),
  child: Text(
    'Preview',
    style:
      FlutterFlowTheme.of(context).headlineSmall.override(
        font: GoogleFonts.interTight(
          fontWeight: FlutterFlowTheme.of(context)
            .headlineSmall
            .fontWeight,
          fontStyle: FlutterFlowTheme.of(context)
            .headlineSmall
            .fontStyle,
        ),
        letterSpacing: 0.0,

```

```

fontWeight: FlutterFlowTheme.of(context)
    .headlineSmall
    .fontWeight,
fontStyle: FlutterFlowTheme.of(context)
    .headlineSmall
    .fontStyle,
),
),
),
Padding(
padding: EdgeInsetsDirectional.fromSTEB(0, 20, 0, 0),
child: Container(
width: double.infinity,
height: 400,
decoration: BoxDecoration(
color: FlutterFlowTheme.of(context).secondaryBackground,
boxShadow: [
BoxShadow(
blurRadius: 6,
color: Color(0x1B090F13),
offset: Offset(
0.0,
-2,
),
)
],
borderRadius: BorderRadius.only(
bottomLeft: Radius.circular(0),
bottomRight: Radius.circular(0),
topLeft: Radius.circular(16),
topRight: Radius.circular(16),
),
),
),

```

```

child: Padding(
  padding: EdgeInsetsDirectional.fromSTEB(0, 8, 0, 0),
  child: Column(
    children: [
      Align(
        alignment: Alignment(0, 0),
        child: TabBar(
          isScrollable: true,
          labelColor:
            FlutterFlowTheme.of(context).primary,
          unselectedLabelColor:
            FlutterFlowTheme.of(context).secondaryText,
          labelStyle: FlutterFlowTheme.of(context)
            .bodyMedium
            .override(
              font: GoogleFonts.inter(
                fontWeight: FlutterFlowTheme.of(context)
                  .bodyMedium
                  .fontWeight,
                fontStyle: FlutterFlowTheme.of(context)
                  .bodyMedium
                  .fontStyle,
              ),
              letterSpacing: 0.0,
              fontWeight: FlutterFlowTheme.of(context)
                .bodyMedium
                .fontWeight,
              fontStyle: FlutterFlowTheme.of(context)
                .bodyMedium
                .fontStyle,
            ),
          unselectedLabelStyle: TextStyle(),
          indicatorColor:

```

```

FlutterFlowTheme.of(context).primary,
indicatorWeight: 2,
tabs: [
  Tab(
    text: 'Available',
  ),
  Tab(
    text: 'Entry Record',
  ),
],
controller: _model.tabBarController,
onTap: (i) async {
  [() async {}, () async {}][i]();
},
),
),
Expanded(
  child: TabBarView(
    controller: _model.tabBarController,
    children: [
      Padding(
        padding: EdgeInsetsDirectional.fromSTEB(
          16, 12, 16, 12),
        child: ListView(
          padding: EdgeInsets.zero,
          primary: false,
          shrinkWrap: true,
          scrollDirection: Axis.vertical,
          children: [
            ClipRRect(
              borderRadius:
                BorderRadius.circular(8),
              child: Image.asset(

```

```

        'assets/images/map.jpg',
        width: 333.3,
        height: 200,
        fit: BoxFit.cover,
      ),
    ),
    Padding(
      padding:
        EdgeInsetsDirectional.fromSTEB(
          16, 12, 16, 12),
      child: ListView(
        padding: EdgeInsets.zero,
        primary: false,
        shrinkWrap: true,
        scrollDirection: Axis.vertical,
        children: [
          Container(
            width: double.infinity,
            decoration: BoxDecoration(
              color: FlutterFlowTheme.of(
                context)
                .secondaryBackground,
            borderRadius:
              BorderRadius.circular(12),
            border: Border.all(
              color: FlutterFlowTheme.of(
                context)
                .alternate,
              width: 2,
            ),
          ),
          child: Padding(
            padding: EdgeInsets.all(12),

```

```

child: Column(
  mainAxisAlignment:
    MainAxisAlignment.max,
  crossAxisAlignment:
    CrossAxisAlignment
      .start,
  children: [
    Align(
      alignment:
        AlignmentDirectional(
          0, -1),
      child: Text(
        'Zone Availability',
        style: FlutterFlowTheme
          .of(context)
          .headlineSmall
          .override(
            font: GoogleFonts
              .interTight(
                fontWeight: FlutterFlowTheme.of(
                  context)
                    .headlineSmall
                    .fontWeight,
                fontStyle: FlutterFlowTheme.of(
                  context)
                    .headlineSmall
                    .fontStyle,
              ),
            letterSpacing:
              0.0,
            fontWeight: FlutterFlowTheme.of(
              context)
                .headlineSmall

```

```

        .fontWeight,
        fontStyle: FlutterFlowTheme.of(
            context)
        .headlineSmall
        .fontStyle,
    ),
),
),
Divider(
    height: 24,
    thickness: 1,
    color:
        FlutterFlowTheme.of(
            context)
        .alternate,
),
Padding(
    padding:
        EdgeInsetsDirectional
        .fromSTEB(
            0, 0, 0, 5),
    child: Row(
        mainAxisAlignment:
            MainAxisAlignment.max,
        mainAxisAlignment:
            MainAxisAlignment
            .spaceBetween,
        children: [
            Expanded(
                child: Column(
                    mainAxisAlignment:
                        MainAxisAlignment
                        .min,

```

```

crossAxisAlignment:
  CrossAxisAlignment
    .start,
children: [
  Text(
    'Zone A',
    style: FlutterFlowTheme.of(
      context)
    .bodyLarge
    .override(
      font:
        GoogleFonts.inter(
          fontWeight:
            FontWeight.w600,
          fontStyle:
FlutterFlowTheme.of(context).bodyLarge.fontStyle,
    ),
    letterSpacing:
      0.0,
    fontWeight:
      FontWeight.w600,
    fontStyle: FlutterFlowTheme.of(context)
      .bodyLarge
      .fontStyle,
    ),
  ),
  Text(
    '${homepageAvailabilityResponse.bodyText}
spaces available',
    style: FlutterFlowTheme.of(
      context)
    .bodyMedium

```



```

        .override(
          font:
            GoogleFonts.inter(
              fontWeight:

FlutterFlowTheme.of(context).bodyMedium.fontWeight,
              fontStyle:

FlutterFlowTheme.of(context).bodyMedium.fontStyle,
            ),
          color:

FlutterFlowTheme.of(context).secondaryText,
          letterSpacing:
            0.0,
          fontWeight: FlutterFlowTheme.of(context)
            .bodyMedium
            .fontWeight,
          fontStyle: FlutterFlowTheme.of(context)
            .bodyMedium
            .fontStyle,
        ),
      ),
    ],
  ),
),
Container(
  width: 60,
  height: 60,
  decoration:
    BoxDecoration(
      color: Color(
        0xFFEDE5FE),

```

```

borderRadius:
  BorderRadius
    .circular(
      8),
),
child: Align(
  alignment:
    AlignmentDirectional(
      0, 0),
  child: Padding(
    padding:
      EdgeInsets
        .all(
          8),
    child: Text(
      homepageAvailabilityResponse
        .bodyText,
      style: FlutterFlowTheme.of(
        context)
        .headlineMedium
        .override(
          font:
            GoogleFonts.interTight(
              fontWeight:
                FlutterFlowTheme.of(context).headlineMedium.fontWeight,
                fontStyle:
                FlutterFlowTheme.of(context).headlineMedium.fontStyle,
            ),
          color:
            Color(0xFF862EFF),
          letterSpacing:

```

```

        0.0,
        fontWeight: FlutterFlowTheme.of(context)
            .headlineMedium
            .fontWeight,
        fontStyle: FlutterFlowTheme.of(context)
            .headlineMedium
            .fontStyle,
    ),
),
),
),
),
],
),
),
Padding(
  padding:
    EdgeInsetsDirectional
      .fromSTEB(
        0, 0, 0, 5),
  child: Row(
    mainAxisAlignment:
      MainAxisAlignment.max,
    mainAxisAlignment:
      MainAxisAlignment
        .spaceBetween,
    children: [
      Expanded(
        child: Column(
          mainAxisAlignment:
            MainAxisAlignment
              .min,
          crossAxisAlignment:

```

```

CrossAxisAlignment
    .start,
children: [
  Text(
    'Zone B',
    style: FlutterFlowTheme.of(
      context)
    .bodyLarge
    .override(
      font:
        GoogleFonts.inter(
          fontWeight:
            FontWeight.w600,
          fontStyle:
FlutterFlowTheme.of(context).bodyLarge.fontStyle,
        ),
      letterSpacing:
        0.0,
      fontWeight:
        FontWeight.w600,
      fontStyle: FlutterFlowTheme.of(context)
        .bodyLarge
        .fontStyle,
    ),
  ),
  Text(
    'Coming Soon',
    style: FlutterFlowTheme.of(
      context)
    .bodyMedium
    .override(
      font:

```

```

        GoogleFonts.inter(
          fontWeight:
FlutterFlowTheme.of(context).bodyMedium.fontWeight,
          fontStyle:
FlutterFlowTheme.of(context).bodyMedium.fontStyle,
        ),
        color:
FlutterFlowTheme.of(context).secondaryText,
        letterSpacing:
          0.0,
        fontWeight: FlutterFlowTheme.of(context)
          .bodyMedium
          .fontWeight,
        fontStyle: FlutterFlowTheme.of(context)
          .bodyMedium
          .fontStyle,
      ),
    ),
  ],
),
),
Container(
  width: 60,
  height: 60,
  decoration:
    BoxDecoration(
      color: Color(
        0x55509558),
      borderRadius:
        BorderRadius

```

```

        .circular(
          8),
      ),
      child: Align(
        alignment:
          AlignmentDirectional(
            0, 0),
        child: Padding(
          padding:
            EdgeInsets
              .all(
                8),
          child: Text(
            '-',
            style: FlutterFlowTheme.of(
              context)
              .headlineMedium
              .override(
                font:
                  GoogleFonts.interTight(
                    fontWeight:
FlutterFlowTheme.of(context).headlineMedium.fontWeight,
                    fontStyle:
FlutterFlowTheme.of(context).headlineMedium.fontStyle,
                ),
                color:
                  Color(0xFF2D622C),
                letterSpacing:
                  0.0,
                fontWeight: FlutterFlowTheme.of(context)
                  .headlineMedium

```

```

        .fontWeight,
        fontStyle: FlutterFlowTheme.of(context)
        .headlineMedium
        .fontStyle,
    ),
),
),
),
),
],
),
),
Padding(
  padding:
    EdgeInsetsDirectional
      .fromSTEB(
        0, 0, 0, 5),
  child: Row(
    mainAxisAlignment:
      MainAxisAlignment.max,
    mainAxisAlignment:
      MainAxisAlignment
        .spaceBetween,
    children: [
      Expanded(
        child: Column(
          mainAxisAlignment:
            MainAxisAlignment
              .min,
          crossAxisAlignment:
            CrossAxisAlignment
              .start,
          children: [

```

```

Text(
  'Zone C',
  style: FlutterFlowTheme.of(
    context)
    .bodyLarge
    .override(
      font:
        GoogleFonts.inter(
          fontWeight:
            FontWeight.w600,
          fontStyle:
FlutterFlowTheme.of(context).bodyLarge.fontStyle,
        ),
      letterSpacing:
        0.0,
      fontWeight:
        FontWeight.w600,
      fontStyle: FlutterFlowTheme.of(context)
        .bodyLarge
        .fontStyle,
    ),
),
Text(
  'Coming Soon',
  style: FlutterFlowTheme.of(
    context)
    .bodyMedium
    .override(
      font:
        GoogleFonts.inter(
          fontWeight:

```



```
FlutterFlowTheme.of(context).bodyMedium.fontWeight,
                                fontStyle:

FlutterFlowTheme.of(context).bodyMedium.fontStyle,
                                ),
                                color:

FlutterFlowTheme.of(context).secondaryText,
                                letterSpacing:
                                    0.0,
                                fontWeight: FlutterFlowTheme.of(context)
                                    .bodyMedium
                                    .fontWeight,
                                fontStyle: FlutterFlowTheme.of(context)
                                    .bodyMedium
                                    .fontStyle,
                                ),
                                ),
                                ],
                                ),
                                ),
                                Container(
                                    width: 60,
                                    height: 60,
                                    decoration:
                                        BoxDecoration(
                                            color: Color(
                                                0x55509558),
                                            borderRadius:
                                                BorderRadius
                                                    .circular(
                                                        8),
```

```

),
child: Align(
  alignment:
    AlignmentDirectional(
      0, 0),
child: Padding(
  padding:
    EdgeInsets
      .all(
        8),
child: Text(
  '-',
  style: FlutterFlowTheme.of(
    context)
    .headlineMedium
    .override(
      font:
        GoogleFonts.interTight(
          fontWeight:

FlutterFlowTheme.of(context).headlineMedium.fontWeight,
          fontStyle:

FlutterFlowTheme.of(context).headlineMedium.fontStyle,
        ),
      color:
        Color(0xFF2D622C),
      letterSpacing:
        0.0,
      fontWeight: FlutterFlowTheme.of(context)
        .headlineMedium
        .fontWeight,
      fontStyle: FlutterFlowTheme.of(context)

```

```

        .headlineMedium
        .fontStyle,
    ),
),
),
),
),
],
),
),
Padding(
  padding:
    EdgeInsetsDirectional
      .fromSTEB(
        0, 0, 0, 5),
  child: Row(
    mainAxisAlignment:
      MainAxisAlignment.max,
    mainAxisAlignment:
      MainAxisAlignment
        .spaceBetween,
    children: [
      Expanded(
        child: Column(
          mainAxisAlignment:
            MainAxisAlignment
              .min,
          crossAxisAlignment:
            CrossAxisAlignment
              .start,
          children: [
            Text(
              'Zone D',

```

```

style: FlutterFlowTheme.of(
  context)
.bodyLarge
.override(
  font:
    GoogleFonts.inter(
      fontWeight:
        FontWeight.w600,
      fontStyle:
FlutterFlowTheme.of(context).bodyLarge.fontStyle,
    ),
    letterSpacing:
      0.0,
    fontWeight:
      FontWeight.w600,
    fontStyle: FlutterFlowTheme.of(context)
      .bodyLarge
      .fontStyle,
  ),
),
Text(
  'Coming Soon',
  style: FlutterFlowTheme.of(
    context)
    .bodyMedium
    .override(
      font:
        GoogleFonts.inter(
          fontWeight:
FlutterFlowTheme.of(context).bodyMedium.fontWeight,
          fontStyle:

```

```
FlutterFlowTheme.of(context).bodyMedium.fontStyle,
    ),
    color:

FlutterFlowTheme.of(context).secondaryText,
    letterSpacing:
        0.0,
    fontWeight: FlutterFlowTheme.of(context)
        .bodyMedium
        .fontWeight,
    fontStyle: FlutterFlowTheme.of(context)
        .bodyMedium
        .fontStyle,
    ),
),
],
),
),
Container(
    width: 60,
    height: 60,
    decoration:
        BoxDecoration(
            color: Color(
                0x55509558),
            borderRadius:
                BorderRadius
                    .circular(
                        8),
        ),
    child: Align(
        alignment:
```

```

        AlignmentDirectional(
          0, 0),
        child: Padding(
          padding:
            EdgeInsets
              .all(
                8),
          child: Text(
            '-',
            style: FlutterFlowTheme.of(
              context)
              .headlineMedium
              .override(
                font:
                  GoogleFonts.interTight(
                    fontWeight:
                      FlutterFlowTheme.of(context).headlineMedium.fontWeight,
                      fontStyle:
                        FlutterFlowTheme.of(context).headlineMedium.fontStyle,
                      ),
                color:
                  Color(0xFF2D622C),
                letterSpacing:
                  0.0,
                fontWeight: FlutterFlowTheme.of(context)
                  .headlineMedium
                  .fontWeight,
                fontStyle: FlutterFlowTheme.of(context)
                  .headlineMedium
                  .fontStyle,
              ),

```

```
),  
    ),  
    ),  
    ),  
  ],  
),  
),  
],  
),  
),  
],  
),  
),  
],  
),  
),  
],  
),  
),  
Padding(  
  padding: EdgeInsetsDirectional.fromSTEB(  
    16, 12, 16, 12),  
  child: ListView(  
    padding: EdgeInsets.zero,  
    primary: false,  
    shrinkWrap: true,  
    scrollDirection: Axis.vertical,  
    children: [  
      Container(  
        width: double.infinity,  
        decoration: BoxDecoration(  
          color: FlutterFlowTheme.of(context)  
            .secondaryBackground,  
          borderRadius:  
            BorderRadius.circular(12),
```

```

border: Border.all(
  color:
    FlutterFlowTheme.of(context)
      .alternate,
  width: 2,
),
),
child: Padding(
  padding: EdgeInsets.all(12),
  child: Column(
    mainAxisAlignment: MainAxisAlignment.max,
    crossAxisAlignment:
      CrossAxisAlignment.start,
    children: [
      Align(
        alignment:
          AlignmentDirectional(
            0, -1),
        child: Text(
          'Car Entry Record',
          style: FlutterFlowTheme.of(
            context)
            .headlineSmall
            .override(
              font: GoogleFonts
                .interTight(
                  fontWeight:
                    FlutterFlowTheme.of(
                      context)
                      .headlineSmall
                      .fontWeight,
                  fontStyle:
                    FlutterFlowTheme.of(

```



```

        context)
        .headlineSmall
        .fontStyle,
    ),
    letterSpacing: 0.0,
    fontWeight:
        FlutterFlowTheme.of(
            context)
            .headlineSmall
            .fontWeight,
    fontStyle:
        FlutterFlowTheme.of(
            context)
            .headlineSmall
            .fontStyle,
    ),
),
),
Align(
    alignment:
        AlignmentDirectional(
            0, 0),
    child: Padding(
        padding:
            EdgeInsetsDirectional
                .fromSTEB(
                    0, 4, 0, 0),
        child: Text(
            dateTimeFormat(
                "MMMMEEEEd",
                getTimestamp),
            style:
                FlutterFlowTheme.of(

```

```

        context)
      .labelMedium
      .override(
        font:
          GoogleFonts
            .inter(
              fontWeight: FlutterFlowTheme.of(
                context)
                .labelMedium
                .fontWeight,
              fontStyle: FlutterFlowTheme.of(
                context)
                .labelMedium
                .fontStyle,
            ),
        letterSpacing:
          0.0,
        fontWeight: FlutterFlowTheme.of(
          context)
          .labelMedium
          .fontWeight,
        fontStyle: FlutterFlowTheme.of(
          context)
          .labelMedium
          .fontStyle,
      ),
    ),
  ),
  Divider(
    height: 24,
    thickness: 1,
    color: FlutterFlowTheme.of(

```

```

        context)
    .alternate,
),
Row(
  mainAxisAlignment:
    MainAxisAlignment.max,
  mainAxisAlignment:
    MainAxisAlignment.start,
  children: [
    Expanded(
      child: Padding(
        padding:
          EdgeInsetsDirectional
            .fromSTEB(16, 0,
              16, 0),
      child: Container(
        width:
          double.infinity,
        decoration:
          BoxDecoration(
            color: FlutterFlowTheme
              .of(context)
              .secondaryBackground,
          boxShadow: [
            BoxShadow(
              blurRadius: 3,
              color: Color(
                0x33000000),
              offset: Offset(
                0,
                1,
              ),
              spreadRadius: 0,

```

```

    )
  ],
  borderRadius:
    BorderRadius
      .circular(
        12),
  ),
  child: Padding(
    padding:
      EdgeInsetsDirectional
        .fromSTEB(
          16,
          16,
          16,
          16),
    child: Column(
      mainAxisAlignment:
        MainAxisAlignment
          .min,
      children: [
        Row(
          mainAxisAlignment:
            MainAxisAlignment
              .max,
          mainAxisAlignment:
            MainAxisAlignment
              .spaceBetween,
          children: [
            Text(
              'License Plate',
              style: FlutterFlowTheme.of(
                context)
                .bodyMedium

```

```

        .override(
          font:
            GoogleFonts.inter(
              fontWeight: FontWeight.w600,
              fontStyle:
FlutterFlowTheme.of(context).bodyMedium.fontStyle,
            ),
          letterSpacing:
            0.0,
          fontWeight:
            FontWeight.w600,
          fontStyle:
FlutterFlowTheme.of(context).bodyMedium.fontStyle,
        ),
      ),
      Text(
        'Entry Time',
        style: FlutterFlowTheme.of(
          context)
          .bodyMedium
          .override(
            font:
              GoogleFonts.inter(
                fontWeight: FontWeight.w600,
                fontStyle:
FlutterFlowTheme.of(context).bodyMedium.fontStyle,
              ),
            letterSpacing:
              0.0,
            fontWeight:
              FontWeight.w600,
            fontStyle:

```

```
FlutterFlowTheme.of(context).bodyMedium.fontStyle,
    ),
  ),
  Text(
    'Status',
    style: FlutterFlowTheme.of(
      context)
      .bodyMedium
      .override(
        font:
          GoogleFonts.inter(
            fontWeight: FontWeight.w600,
            fontStyle:
FlutterFlowTheme.of(context).bodyMedium.fontStyle,
          ),
        letterSpacing:
          0.0,
        fontWeight:
          FontWeight.w600,
        fontStyle:

FlutterFlowTheme.of(context).bodyMedium.fontStyle,
    ),
  ),
],
),
Container(
  width: double
    .infinity,
  height: 1,
  decoration:
    BoxDecoration(
```

```

        color: FlutterFlowTheme.of(
            context)
        .alternate,
    ),
),
StreamBuilder<
    List<
        UserRecord>>(
stream:
    queryUserRecord(
queryBuilder:
    (userRecord) =>
        userRecord.where(
            'plate_number',
            isEqualTo:
                _model
                .carPlate1,
        ),
    singleRecord:
        true,
    ),
builder: (context,
    snapshot) {
    // Customize what your widget looks like when it's
loading.

    if (!snapshot
        .hasData) {
    return Center(
        child:
            SizedBox(
                width:
                    50,
                height:

```

```

        50,
      child:
        CircularProgressIndicator(
          valueColor:
            AlwaysStoppedAnimation<Color>(
              FlutterFlowTheme.of(context).primary,
            ),
          ),
        ),
      );
    }
  List<UserRecord>
    rowUserRecordList =
      snapshot
        .data!;
  final rowUserRecord = rowUserRecordList
    .isEmpty
      ? rowUserRecordList
        .first
      : null;

  return Row(
    mainAxisAlignment:
      MainAxisAlignment
        .max,
    mainAxisAlignment:
      MainAxisAlignment
        .spaceBetween,
    children: [
      Text(
        _model
          .carPlate1,
        style: FlutterFlowTheme.of(context)

```



```

        .bodyMedium
        .override(
            font: GoogleFonts.inter(
                fontWeight:
FlutterFlowTheme.of(context).bodyMedium.fontWeight,
                fontStyle:
FlutterFlowTheme.of(context).bodyMedium.fontStyle,
            ),
            letterSpacing: 0.0,
            fontWeight:
FlutterFlowTheme.of(context).bodyMedium.fontWeight,
            fontStyle:
FlutterFlowTheme.of(context).bodyMedium.fontStyle,
        ),
    ),
    Text(
        '08:15 AM',
        style: FlutterFlowTheme.of(context)
            .bodyMedium
            .override(
                font: GoogleFonts.inter(
                    fontWeight:
FlutterFlowTheme.of(context).bodyMedium.fontWeight,
                    fontStyle:
FlutterFlowTheme.of(context).bodyMedium.fontStyle,
                ),
                letterSpacing: 0.0,
                fontWeight:
FlutterFlowTheme.of(context).bodyMedium.fontWeight,
                fontStyle:
FlutterFlowTheme.of(context).bodyMedium.fontStyle,
            ),
    ),

```

```

Padding(
  padding: EdgeInsetsDirectional.fromSTEB(
    4,
    2,
    4,
    2),
  child:
    Container(
      decoration:
        BoxDecoration(
          color: Color(0xFFE6FFF2),
          borderRadius: BorderRadius.circular(4),
        ),
      child:
        Padding(
          padding: EdgeInsets.all(8),
          child: Text(
            rowUserRecord != null ? 'Authorized' :
'Unauthorized',
            style:
FlutterFlowTheme.of(context).labelSmall.override(
          font: GoogleFonts.inter(
            fontWeight:
FlutterFlowTheme.of(context).labelSmall.fontWeight,
            fontStyle:
FlutterFlowTheme.of(context).labelSmall.fontStyle,
          ),
          color:
FlutterFlowTheme.of(context).success,
          letterSpacing: 0.0,
          fontWeight:
FlutterFlowTheme.of(context).labelSmall.fontWeight,

```

```

                                fontStyle:
FlutterFlowTheme.of(context).labelSmall.fontStyle,
                                ),
                                ),
                                ),
                                ),
                                ),
                                ],
                                );
                                },
                                ),
StreamBuilder<
    List<
        UserRecord>>(<
stream:
    queryUserRecord(
queryBuilder:
    (userRecord) =>
        userRecord.where(
            'plate_number',
isEqualTo:
            _model
                .carPlate2,
        ),
    singleRecord:
        true,
    ),
builder: (context,
    snapshot) {
    // Customize what your widget looks like when it's
loading.

    if (!snapshot
        .hasData) {

```

```

return Center(
  child:
    SizedBox(
      width:
        50,
      height:
        50,
      child:
        CircularProgressIndicator(
          valueColor:
            AlwaysStoppedAnimation<Color>(
              FlutterFlowTheme.of(context).primary,
            ),
          ),
        ),
      );
}
List<UserRecord>
  rowUserRecordList =
    snapshot
      .data!;
final rowUserRecord = rowUserRecordList
  .isEmpty
    ? rowUserRecordList
      .first
    : null;

return Row(
  mainAxisAlignment:
    MainAxisAlignment
      .max,
  mainAxisAlignment:
    MainAxisAlignment

```

```

        .spaceBetween,
children: [
  Text(
    _model
    .carPlate2,
    style: FlutterFlowTheme.of(context)
    .bodyMedium
    .override(
      font: GoogleFonts.inter(
        fontWeight:
FlutterFlowTheme.of(context).bodyMedium.fontWeight,
        fontStyle:
FlutterFlowTheme.of(context).bodyMedium.fontStyle,
      ),
      letterSpacing: 0.0,
      fontWeight:
FlutterFlowTheme.of(context).bodyMedium.fontWeight,
      fontStyle:
FlutterFlowTheme.of(context).bodyMedium.fontStyle,
    ),
  ),
  Text(
    '08:32 AM',
    style: FlutterFlowTheme.of(context)
    .bodyMedium
    .override(
      font: GoogleFonts.inter(
        fontWeight:
FlutterFlowTheme.of(context).bodyMedium.fontWeight,
        fontStyle:
FlutterFlowTheme.of(context).bodyMedium.fontStyle,
      ),
      letterSpacing: 0.0,

```

```

fontWeight:
FlutterFlowTheme.of(context).bodyMedium.fontWeight,
fontWeight:
FlutterFlowTheme.of(context).bodyMedium.fontStyle,
),
),
Padding(
padding: EdgeInsetsDirectional.fromSTEB(
4,
2,
4,
2),
child:
Container(
decoration:
BoxDecoration(
color: Color(0xFFE6FFF2),
borderRadius: BorderRadius.circular(4),
),
child:
Padding(
padding: EdgeInsets.all(8),
child: Text(
rowUserRecord != null ? 'Authorized' :
'Unauthorized',
style:
FlutterFlowTheme.of(context).labelSmall.override(
font: GoogleFonts.inter(
fontWeight:
FlutterFlowTheme.of(context).labelSmall.fontWeight,
fontStyle:
FlutterFlowTheme.of(context).labelSmall.fontStyle,
),

```

```

                                color:
FlutterFlowTheme.of(context).success,
                                letterSpacing: 0.0,
                                fontWeight:
FlutterFlowTheme.of(context).labelSmall.fontWeight,
                                fontStyle:
FlutterFlowTheme.of(context).labelSmall.fontStyle,
                                ),
                                ),
                                ),
                                ),
                                ],
                                );
                                },
                                ),
StreamBuilder<
  List<
    UserRecord>>(<
stream:
  queryUserRecord(
queryBuilder:
  (userRecord) =>
    userRecord.where(
      'plate_number',
      isEqualTo:
        _model
          .carPlate3,
    ),
  singleRecord:
    true,
  ),
  builder: (context,

```

loading.

```

    snapshot) {
// Customize what your widget looks like when it's

    if (!snapshot
        .hasData) {
    return Center(
    child:
        SizedBox(
    width:
        50,
    height:
        50,
    child:
        CircularProgressIndicator(
    valueColor:
        AlwaysStoppedAnimation<Color>(
        FlutterFlowTheme.of(context).primary,
        ),
        ),
        ),
        );
    }
    List<UserRecord>
    rowUserRecordList =
    snapshot
    .data!;
    final rowUserRecord = rowUserRecordList
    .isEmpty
    ? rowUserRecordList
    .first
    : null;

    return Row(

```



```

mainAxisSize:
  MainAxisSize
  .max,
mainAxisAlignment:
  MainAxisAlignment
  .spaceBetween,
children: [
  Text(
    _model
    .carPlate3,
    style: FlutterFlowTheme.of(context)
      .bodyMedium
      .override(
        font: GoogleFonts.inter(
          fontWeight:
FlutterFlowTheme.of(context).bodyMedium.fontWeight,
          fontStyle:
FlutterFlowTheme.of(context).bodyMedium.fontStyle,
        ),
        letterSpacing: 0.0,
        fontWeight:
FlutterFlowTheme.of(context).bodyMedium.fontWeight,
        fontStyle:
FlutterFlowTheme.of(context).bodyMedium.fontStyle,
      ),
    ),
  Text(
    '09:23 AM',
    style: FlutterFlowTheme.of(context)
      .bodyMedium
      .override(
        font: GoogleFonts.inter(

```

```

fontWeight:
FlutterFlowTheme.of(context).bodyMedium.fontWeight,
fontWeight:
FlutterFlowTheme.of(context).bodyMedium.fontWeight,
),
letterSpacing: 0.0,
fontWeight:
FlutterFlowTheme.of(context).bodyMedium.fontWeight,
fontWeight:
FlutterFlowTheme.of(context).bodyMedium.fontWeight,
),
),
Padding(
padding: EdgeInsetsDirectional.fromSTEB(
4,
2,
4,
2),
child:
Container(
decoration:
BoxDecoration(
color: Color(0xFFFFFE6E6),
borderRadius: BorderRadius.circular(4),
),
child:
Padding(
padding: EdgeInsets.all(8),
child: Text(
rowUserRecord != null ? 'Authorized' :
'Unauthorized',
style:
FlutterFlowTheme.of(context).labelSmall.override(

```

```

font: GoogleFonts.inter(
fontWeight:
FlutterFlowTheme.of(context).labelSmall.fontWeight,
fontStyle:
FlutterFlowTheme.of(context).labelSmall.fontStyle,
),
color: Color(0xFFFFF5963),
letterSpacing: 0.0,
fontWeight:
FlutterFlowTheme.of(context).labelSmall.fontWeight,
fontStyle:
FlutterFlowTheme.of(context).labelSmall.fontStyle,
),
),
),
),
),
],
);
},
),
].divide(SizedBox(
height: 12)),
),
),
),
),
),
],
),
],
),
),

```

## APPENDIX

```
        ),  
        ],  
        ),  
        ),  
        ],  
        ),  
        ),  
        ],  
        ),  
        ),  
        ),  
        ),  
        ],  
        ),  
        ),  
        ),  
        ),  
        );  
    },  
);  
}  
}
```

**Flutterflow (BlockN):**

```

import '/backend/api_requests/api_calls.dart';
import '/flutter_flow/flutter_flow_icon_button.dart';
import '/flutter_flow/flutter_flow_theme.dart';
import '/flutter_flow/flutter_flow_util.dart';
import '/flutter_flow/flutter_flow_widgets.dart';
import 'dart:ui';
import '/index.dart';
import 'package:flutter/material.dart';
import 'package:google_fonts/google_fonts.dart';
import 'package:provider/provider.dart';

import 'block_n_model.dart';
export 'block_n_model.dart';

class BlockNWidget extends StatefulWidget {
  const BlockNWidget({super.key});

  static String routeName = 'BlockN';
  static String routePath = '/blockN';

  @override
  State<BlockNWidget> createState() => _BlockNWidgetState();
}

class _BlockNWidgetState extends State<BlockNWidget>
  with TickerProviderStateMixin {
  late BlockNModel _model;

  final scaffoldKey = GlobalKey<ScaffoldState>();

  @override
  void initState() {

```

## APPENDIX

```
super.initState();
_model = createModel(context, () => BlockNModel());

_model.tabBarController = TabController(
  vsync: this,
  length: 4,
  initialIndex: 0,
)..addListener(() => safeSetState(() {}));
}

@override
void dispose() {
  _model.dispose();

  super.dispose();
}

@override
Widget build(BuildContext context) {
  return GestureDetector(
    onTap: () {
      FocusScope.of(context).unfocus();
      FocusManager.instance.primaryFocus?.unfocus();
    },
    child: Scaffold(
      key: scaffoldKey,
      backgroundColor: FlutterFlowTheme.of(context).primaryBackground,
      appBar: AppBar(
        backgroundColor: Color(0xFFCF39EF),
        automaticallyImplyLeading: false,
        leading: FlutterFlowIconButton(
          borderColor: Colors.transparent,
          borderRadius: 30,
```

```

borderWidth: 1,
buttonSize: 60,
icon: Icon(
  Icons.arrow_back_rounded,
  color: Colors.white,
  size: 30,
),
onPressed: () async {
  context.pop();
},
),
title: Text(
  'Block N',
  style: FlutterFlowTheme.of(context).headlineMedium.override(
    font: GoogleFonts.interTight(
      fontWeight:
        FlutterFlowTheme.of(context).headlineMedium.fontWeight,
      fontStyle:
        FlutterFlowTheme.of(context).headlineMedium.fontStyle,
    ),
    color: Colors.white,
    fontSize: 22,
    letterSpacing: 0.0,
    fontWeight:
      FlutterFlowTheme.of(context).headlineMedium.fontWeight,
    fontStyle:
      FlutterFlowTheme.of(context).headlineMedium.fontStyle,
  ),
),
actions: [],
centerTitle: true,
elevation: 2,
),

```

```

body: SafeArea(
  top: true,
  child: Column(
    children: [
      Align(
        alignment: Alignment(0, 0),
        child: TabBar(
          isScrollable: true,
          labelColor: FlutterFlowTheme.of(context).primaryText,
          unselectedLabelColor:
            FlutterFlowTheme.of(context).secondaryText,
          labelStyle: FlutterFlowTheme.of(context).titleMedium.override(
            font: GoogleFonts.interTight(
              fontWeight: FlutterFlowTheme.of(context)
                .titleMedium
                .fontWeight,
              fontStyle: FlutterFlowTheme.of(context)
                .titleMedium
                .fontStyle,
            ),
            letterSpacing: 0.0,
            fontWeight:
              FlutterFlowTheme.of(context).titleMedium.fontWeight,
            fontStyle:
              FlutterFlowTheme.of(context).titleMedium.fontStyle,
          ),
          unselectedLabelStyle: FlutterFlowTheme.of(context)
            .titleMedium
            .override(
              font: GoogleFonts.interTight(
                fontWeight: FlutterFlowTheme.of(context)
                  .titleMedium
                  .fontWeight,

```



```

        fontStyle: FlutterFlowTheme.of(context)
            .titleMedium
            .fontStyle,
    ),
    letterSpacing: 0.0,
    fontWeight:
        FlutterFlowTheme.of(context).titleMedium.fontWeight,
    fontStyle:
        FlutterFlowTheme.of(context).titleMedium.fontStyle,
    ),
    indicatorColor: FlutterFlowTheme.of(context).primary,
    tabs: [
        Tab(
            text: 'Zone A',
        ),
        Tab(
            text: 'Zone B',
        ),
        Tab(
            text: 'Zone C',
        ),
        Tab(
            text: 'Zone D',
        ),
    ],
    controller: _model.tabBarController,
    onTap: (i) async {
        [(() async {}, () async {}, () async {}, () async {})[i]()];
    },
    ),
    ),
    Expanded(
        child: TabBarView(

```

```

controller: _model.tabBarController,
children: [
  SingleChildScrollView(
    primary: false,
    child: Column(
      mainAxisAlignment: MainAxisAlignment.max,
      children: [
        SingleChildScrollView(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.max,
            children: [
              Column(
                mainAxisAlignment: MainAxisAlignment.max,
                children: [
                  Material(
                    color: Colors.transparent,
                    elevation: 2,
                    shape: RoundedRectangleBorder(
                      borderRadius: BorderRadius.circular(12),
                    ),
                    child: Container(
                      width: double.infinity,
                      decoration: BoxDecoration(
                        color: FlutterFlowTheme.of(context)
                          .secondaryBackground,
                        borderRadius:
                          BorderRadius.circular(12),
                      ),
                      child: Padding(
                        padding:
                          EdgeInsetsDirectional.fromSTEB(
                            12, 12, 12, 12),
                        child: Column(

```

```

mainAxisSize: MainAxisSize.min,
children: [
  Text(
    'Zone A (1-7)',
    style:
      FlutterFlowTheme.of(context)
        .titleMedium
        .override(
          font: GoogleFonts
            .interTight(
              fontWeight:
                FlutterFlowTheme.of(
                  context)
                  .titleMedium
                  .fontWeight,
              fontStyle:
                FlutterFlowTheme.of(
                  context)
                  .titleMedium
                  .fontStyle,
            ),
          color: FlutterFlowTheme
            .of(context)
            .primary,
          letterSpacing: 0.0,
          fontWeight:
            FlutterFlowTheme.of(
              context)
              .titleMedium
              .fontWeight,
          fontStyle:
            FlutterFlowTheme.of(
              context)

```

```

        .titleMedium
        .fontStyle,
    ),
),
FutureBuilder<ApiCallResponse>(
  future: GetBaseImageCall.call(),
  builder: (context, snapshot) {
    // Customize what your widget looks like when it's loading.
    if (!snapshot.hasData) {
      return Center(
        child: SizedBox(
          width: 50,
          height: 50,
          child:
            CircularProgressIndicator(
              valueColor:
                AlwaysStoppedAnimation<
                  Color>(
                    FlutterFlowTheme.of(
                      context)
                      .primary,
                ),
            ),
        ),
      );
    }
    final activeViewGetBaseImageResponse =
      snapshot.data!;

    return ClipRRect(
      borderRadius:
        BorderRadius.circular(
          8),

```

```

        child: Image.network(
          activeViewGetBaseImageResponse
            .jsonBody
            .toString(),
          width: 355,
          height: 174.5,
          fit: BoxFit.cover,
        ),
      );
    },
  ),
  Row(
    mainAxisAlignment: MainAxisAlignment.max,
    mainAxisAlignment:
      MainAxisAlignment
        .spaceBetween,
    children: [
      FutureBuilder<
        ApiCallResponse>(
        future:
          AvailabilityCall.call(),
        builder:
          (context, snapshot) {
            // Customize what your widget looks like when it's loading.
            if (!snapshot.hasData) {
              return Center(
                child: SizedBox(
                  width: 50,
                  height: 50,
                  child:
                    CircularProgressIndicator(
                      valueColor:
                        AlwaysStoppedAnimation<

```

```

        Color>(
        FlutterFlowTheme.of(
            context)
        .primary,
    ),
    ),
    ),
);
}
final textAvailabilityResponse =
    snapshot.data!;

return Text(
    'Available Spaces:
    ${textAvailabilityResponse.bodyText}/7',
    style:
        FlutterFlowTheme.of(
            context)
        .bodyMedium
        .override(
            font:
                GoogleFonts
                .inter(
                    fontWeight: FlutterFlowTheme.of(
                        context)
                    .bodyMedium
                    .fontWeight,
                    fontStyle: FlutterFlowTheme.of(
                        context)
                    .bodyMedium
                    .fontStyle,
                ),
            color: FlutterFlowTheme.of(

```

```

        context)
        .success,
letterSpacing:
    0.0,
fontWeight: FlutterFlowTheme.of(
    context)
    .bodyMedium
    .fontWeight,
fontStyle: FlutterFlowTheme.of(
    context)
    .bodyMedium
    .fontStyle,
    ),
);
},
),
Text(
    dateTimeFormat("jms",
        getCurrentTimestamp),
    style: FlutterFlowTheme.of(
        context)
        .labelSmall
        .override(
            font:
                GoogleFonts.inter(
                    fontWeight:
                        FlutterFlowTheme.of(
                            context)
                            .labelSmall
                            .fontWeight,
                    fontStyle:
                        FlutterFlowTheme.of(
                            context)

```

```

        .labelSmall
        .fontStyle,
    ),
    color: FlutterFlowTheme
        .of(context)
        .secondaryText,
    letterSpacing: 0.0,
    fontWeight:
        FlutterFlowTheme.of(
            context)
        .labelSmall
        .fontWeight,
    fontStyle:
        FlutterFlowTheme.of(
            context)
        .labelSmall
        .fontStyle,
    ),
    ),
    ],
    ),
    ].divide(SizedBox(height: 8)),
    ),
    ),
    ),
    ),
    Material(
        color: Colors.transparent,
        elevation: 2,
        shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(12),
        ),
        child: Container(

```



```

width: double.infinity,
decoration: BoxDecoration(
  color: FlutterFlowTheme.of(context)
    .secondaryBackground,
  borderRadius:
    BorderRadius.circular(12),
),
child: Padding(
  padding:
    EdgeInsetsDirectional.fromSTEB(
      12, 12, 12, 12),
  child: Column(
    mainAxisAlignment: MainAxisAlignment.min,
    children: [
      Text(
        'Zone A (8-17)',
        style:
          FlutterFlowTheme.of(context)
            .titleMedium
            .override(
              font: GoogleFonts
                .interTight(
                  fontWeight:
                    FlutterFlowTheme.of(
                      context)
                        .titleMedium
                        .fontWeight,
                  fontStyle:
                    FlutterFlowTheme.of(
                      context)
                        .titleMedium
                        .fontStyle,
                ),

```

```

        color: FlutterFlowTheme
          .of(context)
          .primary,
        letterSpacing: 0.0,
        fontWeight:
          FlutterFlowTheme.of(
            context)
            .titleMedium
            .fontWeight,
        fontStyle:
          FlutterFlowTheme.of(
            context)
            .titleMedium
            .fontStyle,
      ),
    ),
    ClipRRect(
      borderRadius:
        BorderRadius.circular(8),
      child: Image.asset(
        'assets/images/pic2_with_boundary.png',
        width: 355,
        height: 174.5,
        fit: BoxFit.cover,
      ),
    ),
    Row(
      mainAxisAlignment: MainAxisAlignment.max,
      mainAxisAlignment:
        MainAxisAlignment
          .spaceBetween,
      children: [
        Text(

```

```

'Available Spaces: 5/6',
style: FlutterFlowTheme.of(
  context)
.bodyMedium
.override(
  font:
    GoogleFonts.inter(
      fontWeight:
        FlutterFlowTheme.of(
          context)
          .bodyMedium
          .fontWeight,
      fontStyle:
        FlutterFlowTheme.of(
          context)
          .bodyMedium
          .fontStyle,
    ),
  color: FlutterFlowTheme
    .of(context)
    .warning,
  letterSpacing: 0.0,
  fontWeight:
    FlutterFlowTheme.of(
      context)
      .bodyMedium
      .fontWeight,
  fontStyle:
    FlutterFlowTheme.of(
      context)
      .bodyMedium
      .fontStyle,
),

```

```
),  
Text(  
  dateTimeFormat("jms",  
    getCurrentTimestamp),  
  style: FlutterFlowTheme.of(  
    context)  
  ).labelSmall  
  .override(  
    font:  
      GoogleFonts.inter(  
        fontWeight:  
          FlutterFlowTheme.of(  
            context)  
            .labelSmall  
            .fontWeight,  
        fontStyle:  
          FlutterFlowTheme.of(  
            context)  
            .labelSmall  
            .fontStyle,  
      ),  
    color: FlutterFlowTheme  
      .of(context)  
      .secondaryText,  
    letterSpacing: 0.0,  
    fontWeight:  
      FlutterFlowTheme.of(  
        context)  
        .labelSmall  
        .fontWeight,  
    fontStyle:  
      FlutterFlowTheme.of(  
        context)
```

```

        .labelSmall
        .fontStyle,
    ),
),
],
),
].divide(SizedBox(height: 8)),
),
),
),
),
Material(
  color: Colors.transparent,
  elevation: 2,
  shape: RoundedRectangleBorder(
    borderRadius: BorderRadius.circular(12),
  ),
  child: Container(
    width: double.infinity,
    decoration: BoxDecoration(
      color: FlutterFlowTheme.of(context)
        .secondaryBackground,
      borderRadius:
        BorderRadius.circular(12),
    ),
    child: Padding(
      padding:
        EdgeInsetsDirectional.fromSTEB(
          12, 12, 12, 12),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.min,
        children: [
          Text(

```

```

'Zone A (18-20)',
style:
  FlutterFlowTheme.of(context)
    .titleMedium
    .override(
      font: GoogleFonts
        .interTight(
          fontWeight:
            FlutterFlowTheme.of(
              context)
              .titleMedium
              .fontWeight,
          fontStyle:
            FlutterFlowTheme.of(
              context)
              .titleMedium
              .fontStyle,
        ),
      color: FlutterFlowTheme
        .of(context)
        .primary,
      letterSpacing: 0.0,
      fontWeight:
        FlutterFlowTheme.of(
          context)
          .titleMedium
          .fontWeight,
      fontStyle:
        FlutterFlowTheme.of(
          context)
          .titleMedium
          .fontStyle,
    ),

```

```

    ),
    ClipRRect(
      borderRadius:
        BorderRadius.circular(8),
      child: Image.asset(
        'assets/images/pic3_with_boundary.png',
        width: 355,
        height: 174.5,
        fit: BoxFit.cover,
      ),
    ),
  ),
  Row(
    mainAxisAlignment: MainAxisAlignment.max,
    mainAxisAlignment:
      MainAxisAlignment
        .spaceBetween,
    children: [
      Text(
        'Available Spaces: 6/7',
        style: FlutterFlowTheme.of(
          context)
          .bodyMedium
          .override(
            font:
              GoogleFonts.inter(
                fontWeight:
                  FlutterFlowTheme.of(
                    context)
                    .bodyMedium
                    .fontWeight,
                fontStyle:
                  FlutterFlowTheme.of(
                    context)

```

```

        .bodyMedium
        .fontStyle,
    ),
    color: FlutterFlowTheme
        .of(context)
        .error,
    letterSpacing: 0.0,
    fontWeight:
        FlutterFlowTheme.of(
            context)
        .bodyMedium
        .fontWeight,
    fontStyle:
        FlutterFlowTheme.of(
            context)
        .bodyMedium
        .fontStyle,
    ),
),
Text(
    dateTimeFormat("jms",
        getCurrentTimestamp),
    style: FlutterFlowTheme.of(
        context)
        .labelSmall
        .override(
            font:
                GoogleFonts.inter(
                    fontWeight:
                        FlutterFlowTheme.of(
                            context)
                            .labelSmall
                            .fontWeight,

```



```

        fontStyle:
          FlutterFlowTheme.of(
            context)
          .labelSmall
          .fontStyle,
      ),
      color: FlutterFlowTheme
        .of(context)
        .secondaryText,
      letterSpacing: 0.0,
      fontWeight:
        FlutterFlowTheme.of(
          context)
          .labelSmall
          .fontWeight,
      fontStyle:
        FlutterFlowTheme.of(
          context)
          .labelSmall
          .fontStyle,
    ),
  ),
],
),
].divide(SizedBox(height: 8)),
),
),
),
),
],
),
Padding(
  padding: EdgeInsetsDirectional.fromSTEB(

```

```

    16, 0, 16, 0),
child: Row(
  mainAxisAlignment: MainAxisAlignment.max,
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    FFButtonWidget(
      onPressed: () async {
        if (Navigator.of(context).canPop()) {
          context.pop();
        }
        context.pushNamed(
          BlockNWWidget.routeName,
          extra: <String, dynamic>{
            kTransitionInfoKey:
              TransitionInfo(
                hasTransition: true,
                transitionType:
                  PageTransitionType.fade,
              ),
          },
        );
      },
      text: 'Refresh Now',
      icon: Icon(
        Icons.refresh_rounded,
        color: Colors.white,
        size: 15,
      ),
      options: FFButtonOptions(
        width: 200,
        height: 50,
        padding: EdgeInsets.all(8),
        iconPadding:

```

```

        EdgeInsetsDirectional.fromSTEB(
            0, 0, 0, 0),
        color: FlutterFlowTheme.of(context)
            .primary,
        textStyle: FlutterFlowTheme.of(
            context)
            .titleSmall
            .override(
                font: GoogleFonts.interTight(
                    fontWeight:
                        FlutterFlowTheme.of(
                            context)
                                .titleSmall
                                .fontWeight,
                    fontStyle:
                        FlutterFlowTheme.of(
                            context)
                                .titleSmall
                                .fontStyle,
                ),
                color: Colors.white,
                letterSpacing: 0.0,
                fontWeight:
                    FlutterFlowTheme.of(context)
                        .titleSmall
                        .fontWeight,
                fontStyle:
                    FlutterFlowTheme.of(context)
                        .titleSmall
                        .fontStyle,
            ),
        elevation: 2,
    ),

```

```

    ),
  ],
),
),
],
),
),
],
),
),
Container(
  width: 100,
  height: 100,
  decoration: BoxDecoration(
    color: Color(0x4D39D27F),
  ),
  child: Align(
    alignment: AlignmentDirectional(0, 0),
    child: Text(
      'Coming Soon~',
      style:
        FlutterFlowTheme.of(context).bodyMedium.override(
          font: GoogleFonts.inter(
            fontWeight: FontWeight.w900,
            fontStyle: FlutterFlowTheme.of(context)
              .bodyMedium
              .fontStyle,
          ),
          letterSpacing: 0.0,
          fontWeight: FontWeight.w900,
          fontStyle: FlutterFlowTheme.of(context)
            .bodyMedium
            .fontStyle,

```

```

        ),
    ),
),
),
Container(
  width: 100,
  height: 100,
  decoration: BoxDecoration(
    color: Color(0x4D39D27F),
  ),
  child: Align(
    alignment: AlignmentDirectional(0, 0),
    child: Text(
      'Coming Soon~',
      style:
        FlutterFlowTheme.of(context).bodyMedium.override(
          font: GoogleFonts.inter(
            fontWeight: FontWeight.w900,
            fontStyle: FlutterFlowTheme.of(context)
              .bodyMedium
              .fontStyle,
          ),
          letterSpacing: 0.0,
          fontWeight: FontWeight.w900,
          fontStyle: FlutterFlowTheme.of(context)
            .bodyMedium
            .fontStyle,
        ),
    ),
  ),
),
Container(
  width: 100,

```

```

height: 100,
decoration: BoxDecoration(
  color: Color(0x4D39D27F),
),
child: Align(
  alignment: AlignmentDirectional(0, 0),
  child: Text(
    'Coming Soon~',
    style:
      FlutterFlowTheme.of(context).bodyMedium.override(
        font: GoogleFonts.inter(
          fontWeight: FontWeight.w900,
          fontStyle: FlutterFlowTheme.of(context)
            .bodyMedium
            .fontStyle,
        ),
        letterSpacing: 0.0,
        fontWeight: FontWeight.w900,
        fontStyle: FlutterFlowTheme.of(context)
          .bodyMedium
          .fontStyle,
      ),
  ),
),
),
],
),
),
],
),
),
),
);

```

## APPENDIX

}

}