Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

Parkify

Project Engineering

Year 4

TAN MIN HAN

Bachelor of Engineering (Honours) in Software and Electronic Engineering

Atlantic Technological University

2022/2023

Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering (Honours) in Software and Electronic Engineering at Atlantic Technological University.

This project is my own work, also used Word Editor and Grammarly to proofread my writing and fix any grammar mistakes. except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

____

Acknowledgements

I would like to express my gratitude to Niall O'Keefe, my project supervisor, for his guidance throughout the project, his advice and assistance helped me a lot to overcome the problems that I have met and stay on track with my project timeline.

Furthermore, I also would like to thank the course lecturers, Michelle Lynch, Paul Lennon, and Brian O'Shea, who have been present and giving me guidance and support with my project throughout the year.

Finally, I am very grateful for the opportunity that I can pursue my studies overseas in Ireland. I would like to thank my parents for their support and encouragement throughout the academic year. The love and supports from them are my perseverance and determination for me to pursue my dream.

Table of Contents

1 Summary

Finding a parking spot has always been a frustrating experience for drivers as well as myself, especially when it comes to paying the parking fees with coins or cash. After conducting research and analysis online, I have developed the Parkify system as a solution for the issue at hand. The goal of Parkify is to create a smart parking system that makes parking effortless. The system should be able to bring convenience to the parking space that required a parking ticket and pay by cash or coins only.

The scope of the project includes Automatic License Plate Recognition technology which required to train the model and integrate it with the camera to allow the camera to detect the license plate. Additionally, the Parkify system includes a mobile application for users to check their parking time and pay for parking hassle-free through the application.

There are few key features of the project which are real-time parking availability information, payment processing for parking fees and creating a better experience for drivers.

The approach for developing the Parkify system involves using Google Cloud Vision to train the object detection model to be able to detect the license plate of a car. In addition, the mobile application will be using React Native and Expo to build the interface and integrate it with a cloud-based backend, Amazon Web Services (AWS).

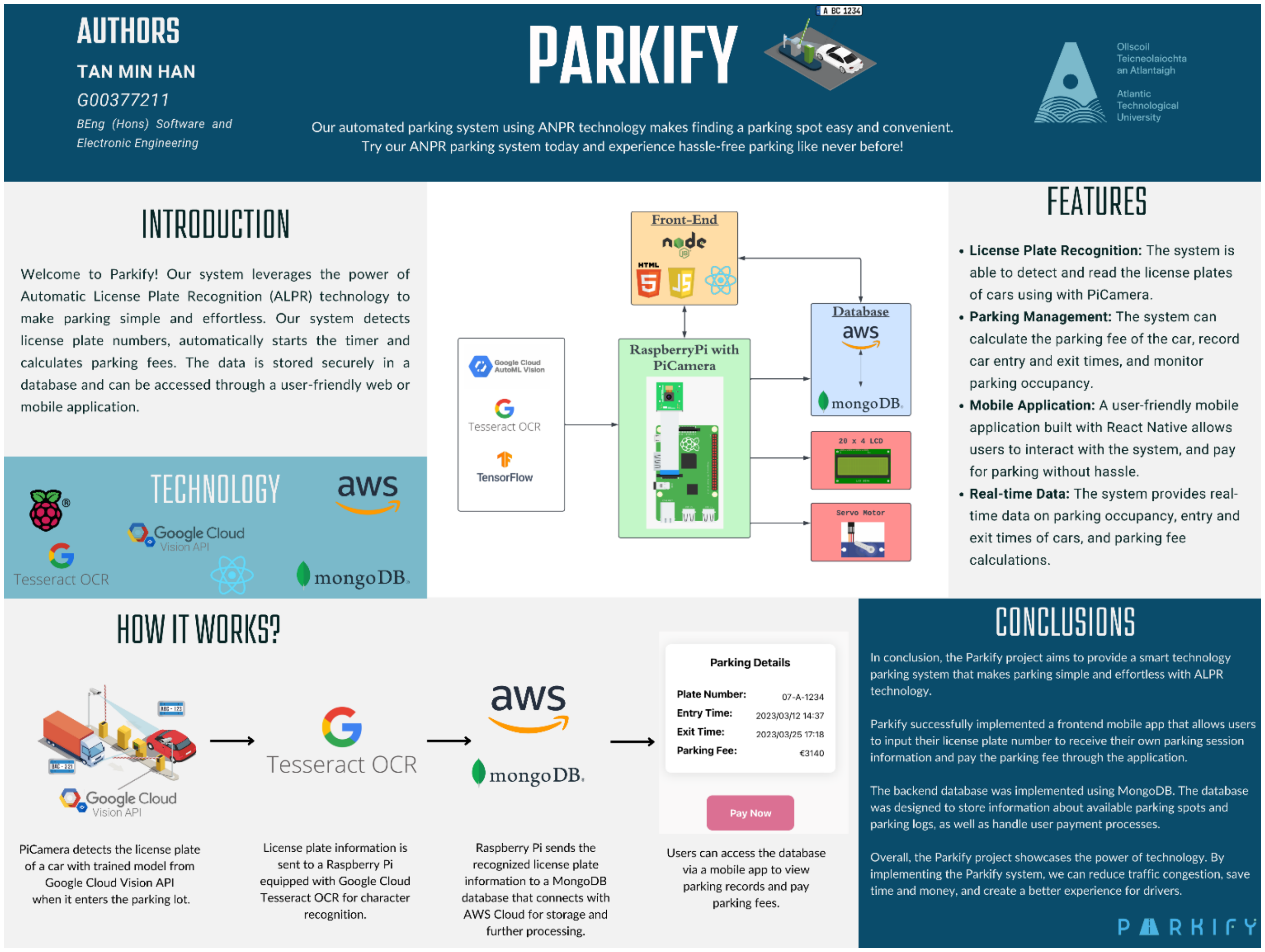
The Parkify system has developed using methods and technologies which involved Google Cloud Vision, and Google Cloud Tesseract OCR, these technologies are to train the model of detecting license plates and extracting the characters of the license plate. Moreover, React Native Framework is using to build a mobile application with the Expo developer tool. Other than that, the backend technology used in the Parkify system is MongoDB for the database to save the license plate that detected, update the entry, and exit time of the car also calculating the parking fee of the car according to the license plate.

The Parkify system has accomplished within the duration of the project timeline with the main goal of developing a smart parking system with the camera detecting the license plate and

storing it in the database. User can access their license plate in the mobile application and pay through the application with the payment processor.

By leveraging the power of technology, the Parkify system allows users to drive into the parking spaces without getting a ticket and paying parking fees by just using the mobile application integrated with the system. The results of the Parkify system have much more opportunity for further development, by integrating with private parking spots and shopping mall applications.

2 Poster



AUTHORS

TAN MIN HAN

G00377211

BEng (Hons) Software and Electronic Engineering

PARKIFY

Our automated parking system using ANPR technology makes finding a parking spot easy and convenient. Try our ANPR parking system today and experience hassle-free parking like never before!

Ollscoil Teicneolaíochta an Atlantaigh
Atlantic Technological University

INTRODUCTION

Welcome to Parkify! Our system leverages the power of Automatic License Plate Recognition (ALPR) technology to make parking simple and effortless. Our system detects license plate numbers, automatically starts the timer and calculates parking fees. The data is stored securely in a database and can be accessed through a user-friendly web or mobile application.

TECHNOLOGY

FEATURES

- **License Plate Recognition:** The system is able to detect and read the license plates of cars using with PiCamera.
- **Parking Management:** The system can calculate the parking fee of the car, record car entry and exit times, and monitor parking occupancy.
- **Mobile Application:** A user-friendly mobile application built with React Native allows users to interact with the system, and pay for parking without hassle.
- **Real-time Data:** The system provides real-time data on parking occupancy, entry and exit times of cars, and parking fee calculations.

HOW IT WORKS?

PiCamera detects the license plate of a car with trained model from Google Cloud Vision API when it enters the parking lot.

License plate information is sent to a Raspberry Pi equipped with Google Cloud Tesseract OCR for character recognition.

Raspberry Pi sends the recognized license plate information to a MongoDB database that connects with AWS Cloud for storage and further processing.

Users can access the database via a mobile app to view parking records and pay parking fees.

Parking Details

Plate Number:	07-A-1234
Entry Time:	2023/03/12 14:37
Exit Time:	2023/03/25 17:18
Parking Fee:	€3140

Pay Now

CONCLUSIONS

In conclusion, the Parkify project aims to provide a smart technology parking system that makes parking simple and effortless with ALPR technology.

Parkify successfully implemented a frontend mobile app that allows users to input their license plate number to receive their own parking session information and pay the parking fee through the application.

The backend database was implemented using MongoDB. The database was designed to store information about available parking spots and parking logs, as well as handle user payment processes.

Overall, the Parkify project showcases the power of technology. By implementing the Parkify system, we can reduce traffic congestion, save time and money, and create a better experience for drivers.

Figure 2.1: Poster

3 Introduction

Parkify is a smart parking system build with Raspberry Pi that integrated Automatic License Plate Recognition(ALPR) technology for a parking space. The development of Parkify aims to solve the problem of parking management as well as the parking system. With the rapid pace of technology nowadays, and the increasing number of vehicles on the roads, parking has become a challenge for many cities around the world. This motivated me to provide a smart parking solution that can construct a more convenient parking process and enhance the parking experience for drivers.

The scope of this project consists of the implementation of a PiCamera with a specified machine-learning trained model from Google Cloud to be able to detect the license plate of a car. The design and implementation of a mobile application also help users to check the parking duration and pay for their parking hassle-free.

4 Project Architecture

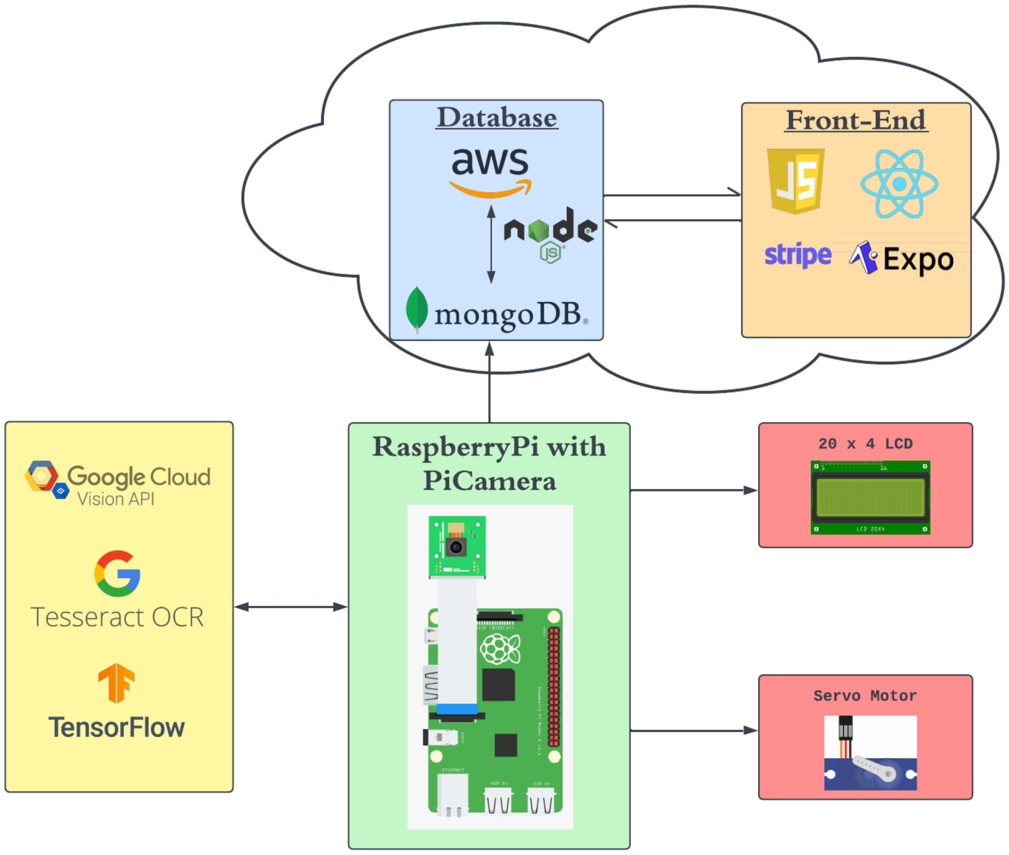
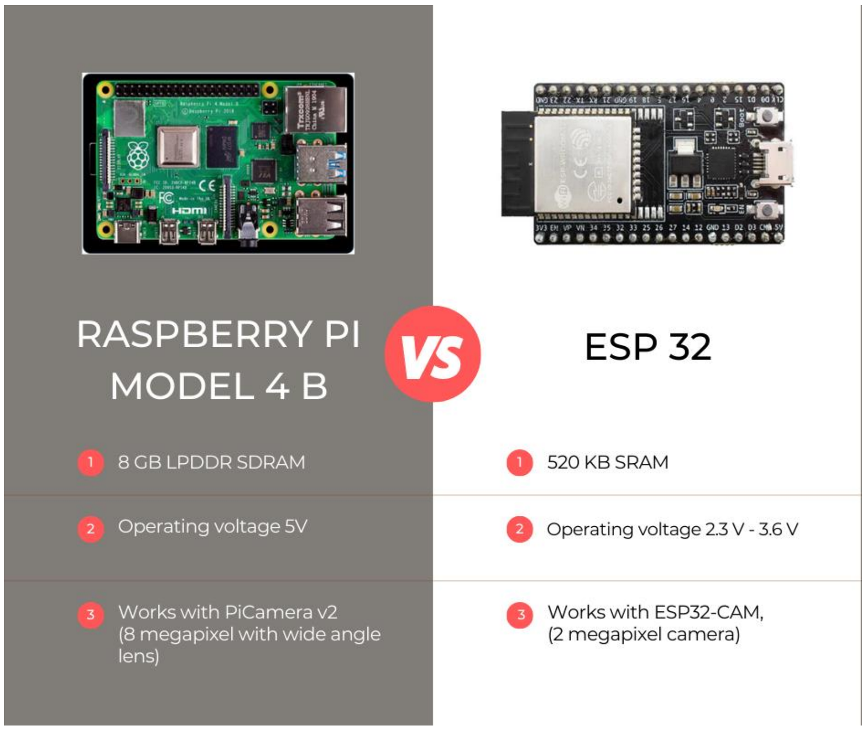


Figure 4.1: Architecture Diagram

5 Research

5.1 Hardware Selection

There are a few different options for hardware platforms that could be implemented in the Parkify system. The two hardware options that conducted research were Raspberry Pi and the ESP 32.



RASPBERRY PI MODEL 4 B	ESP 32
1 8 GB LPDDR SDRAM	1 520 KB SRAM
2 Operating voltage 5V	2 Operating voltage 2.3 V - 3.6 V
3 Works with PiCamera v2 (8 megapixel with wide angle lens)	3 Works with ESP32-CAM, (2 megapixel camera)

Figure 5.1: Raspberry PI vs ESP32

Comparing and researching the Raspberry Pi documentation[1] and ESP 32 documentation [2], which allows us to illustrate a comparison table as the figure shown above and determine which hardware is a better fit with the Parkify system.

The processing power of Raspberry Pi which has a higher operating voltage compared to ESP32 allows for faster and more efficient image processing.

On the other hand, the PiCamera v2 is a camera module specifically designed for Raspberry Pi[3] and it is a high-quality camera that features an 8-megapixel camera lens compared with ESP32-CAM designed for ESP32 features only a 2-megapixel camera lens[4].

Other than that, PiCamera is generally used for computer vision and robotics, and ESP32-CAM designed for small-scale projects such as smart home systems.

For the Parkify system, an excellent quality image is required to achieve precise outcome results and fast image processing for detecting the license plate region of a car. Based on the comparison between Raspberry Pi 4 Model B and ESP 32, the specifications of Raspberry Pi 4 Model B are the preferred hardware to use in the Parkify system. The Raspberry Pi has better processing power, more memory and a higher-quality camera as compared to the ESP 32. Additionally, the Raspberry Pi has a larger community and more available resources for developing the Parkify system.

5.2 Front End

For the Parkify front-end, there are several options for user interface such as website, web application or mobile application. However, the mobile application is the preferred user interface implemented with the Parkify system as mobile application development is a part of our course module, which provides the necessary skills and knowledge to design and develop a mobile application.

Besides mobile application development is a part of our course module, mobile applications also provide a more secure environment than web applications or websites. In mobile applications, we could implement features such as GPS, push notifications, and payment providers like Apple Pay and Google Pay, which will enhance the functionality and user experience of the Parkify system. Also, building mobile applications with React Native allows integration with third-party plugins.

5.3 Database

For the Parkify backend, there are also several options for the database such as MySQL, PostgreSQL, and MongoDB. MongoDB is the preferred database option implemented with the Parkify system as we have learned to set up the database in MongoDB with Create, Read, Update, Delete (CRUD) function. Moreover, MySQL and PostgreSQL are both relational database management systems (RDBMS) while MongoDB is a NoSQL database [5]. In other words, MongoDB is better suited for unstructured or semi-structured data which is a good option for Parkify systems that deal with complex data, such as image and video data.

In the case of Parkify, it requires storage that deals with image and video data, by learning the set-up procedures and implementing MongoDB in one of the course modules which also makes it more convenient and easier to set up the database with MongoDB.

5.4 Supervised Learning

Supervised learning, also known as supervised machine learning, is a subcategory of machine learning and artificial intelligence. It is defined by its use of labelled datasets to train algorithms that classify data or predict outcomes accurately.[6]

In traditional programming, a programmer must code or create rules manually, whereas, in Machine Learning, the algorithm will automatically create the rules from the data and finds the repetition of patterns from the data and formulates the rules from it.

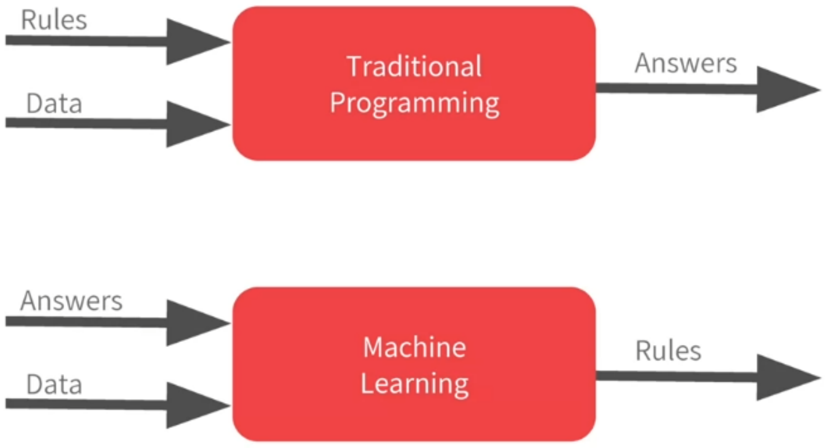
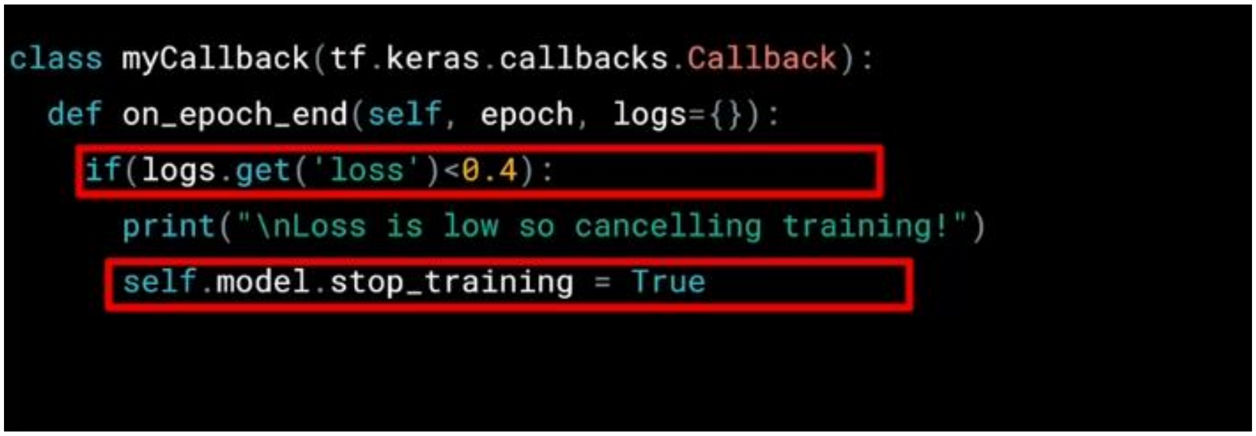


Figure 5.2: Traditional Programming and Machine Learning Difference [7]

This process of showing the computer what the data represents is called Labelling data. This labelling process is crucial and will be further discussed in a section. In machine learning, the term "dense" refers to a layer of connected neurons in a neural network model, which is a crucial component of machine learning algorithms. The loss function is also used to measure the accuracy of the current predictions, which tells how good the current value/guess is.

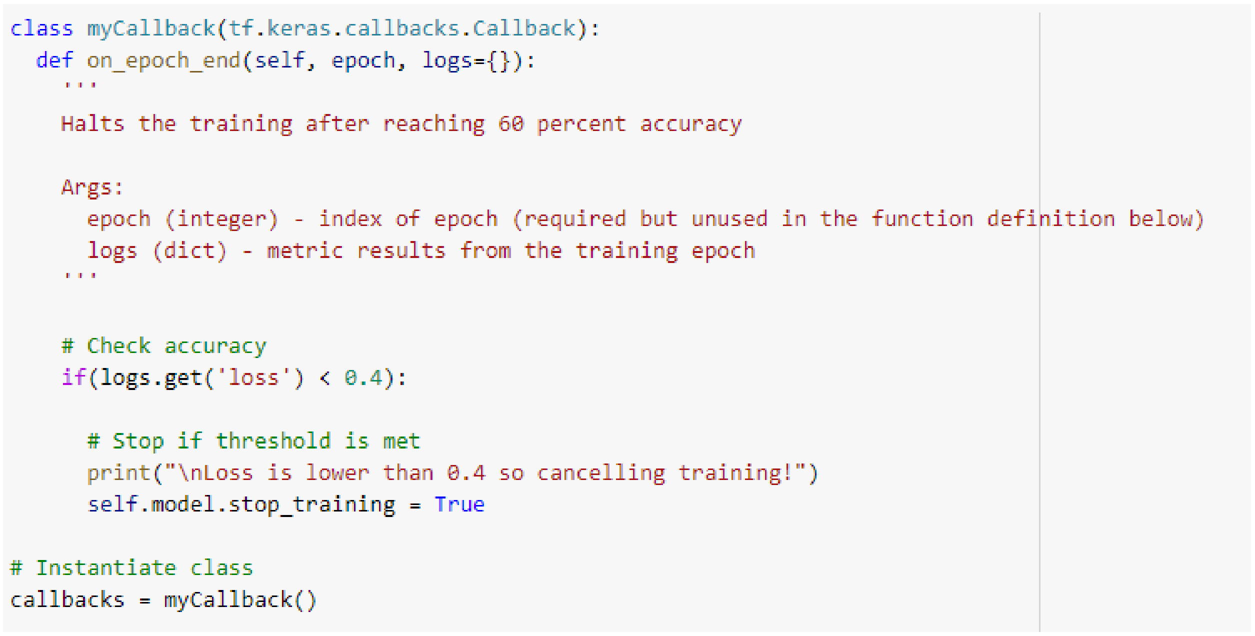
In the Parkify system, we also need to utilize the optimizer that could determine the next guess for the license plate recognition system based on the loss function. In this case, by using the model.fit function, we train the neural network to fit one set of values to another. [7]

Additionally, implementing a callbacks function that can be called on every epoch to monitor the loss and accuracy of the neural network until the expected accuracy/loss we expected is achieved. [7]



```
class myCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if(logs.get('loss')<0.4):
      print("\nLoss is low so cancelling training!")
      self.model.stop_training = True
```

Figure 5.3: Callback function example



```
class myCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    '''
    Halts the training after reaching 60 percent accuracy

    Args:
      epoch (integer) - index of epoch (required but unused in the function definition below)
      logs (dict) - metric results from the training epoch
    '''

    # Check accuracy
    if(logs.get('loss') < 0.4):

      # Stop if threshold is met
      print("\nLoss is lower than 0.4 so cancelling training!")
      self.model.stop_training = True

# Instantiate class
callbacks = myCallback()
```

Figure 5.4: Callback function usage

For the figure above[7] as an example, the callback function when the tests get loss lower than 0.4 and stop the training.

5.5 Image Processing

5.5.1 Convolutions

In machine learning, convolutions and poolings are techniques used in Convolutional Neural Networks(CNNs) for image recognition or computer vision tasks. The key concept of convolutions is the convolution operation, filter. Applying a filter to an image, which is an image processing procedure, to analyse and manipulate pixel values.

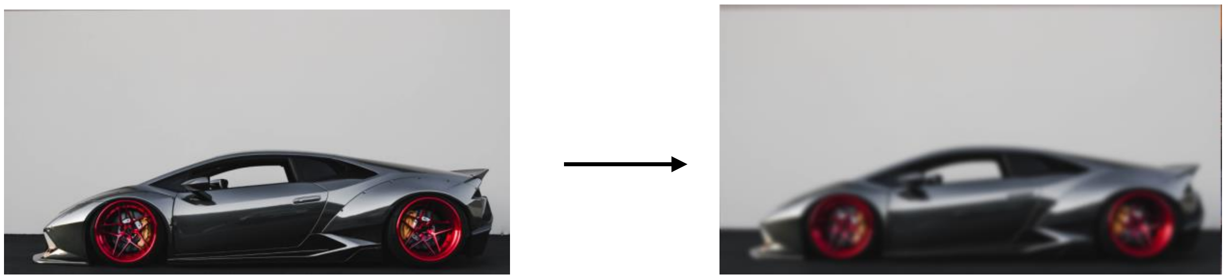
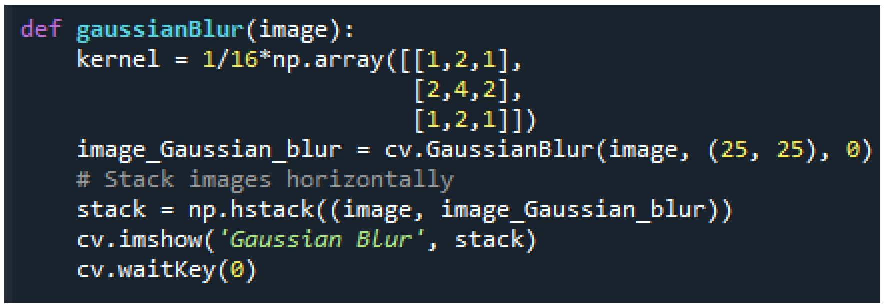


Figure 5.5: Implementation of Gaussian Blur Example

For the figure above, this is a process of mathematical filtering or convolution operation, I applied a Gaussian blur which is a low pass filter to the car.



```
def gaussianBlur(image):
    kernel = 1/16*np.array([[1,2,1],
                            [2,4,2],
                            [1,2,1]])
    image_Gaussian_blur = cv.GaussianBlur(image, (25, 25), 0)
    # Stack images horizontally
    stack = np.hstack((image, image_Gaussian_blur))
    cv.imshow('Gaussian Blur', stack)
    cv.waitKey(0)
```

Figure 5.6 Code snippet implementing Gaussian Blur

As the code snippet provided above[8] is using a kernel to apply a filter on the car image, resulting in a Gaussian blurred image. The Gaussian blur effect implemented by using the OpenCV function cv.GaussianBlur, which produces the image resulting in a Gaussian blur effect.

Convolution is a mathematical operation, an element-by-element multiplication of two matrices followed by a summation.[9]

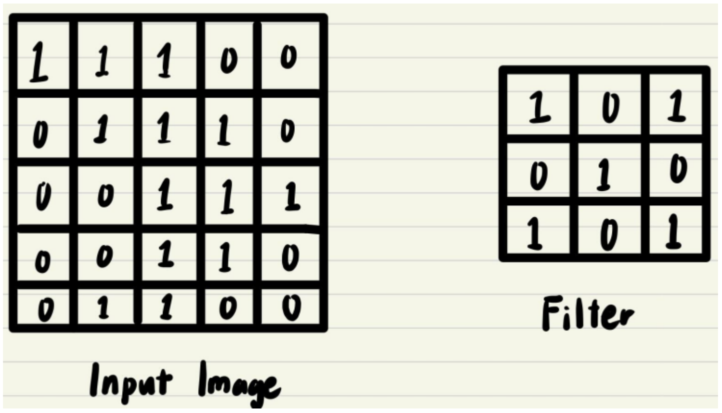


Figure 5.7 Convolution operation with input image and filter

The filter is applied to each pixel and its neighbouring pixels with the resulting values creating a new output.

In the Parkify system, TensorFlow's convolutional neural network is used to identify and recognize the license plate of a car in real time. TensorFlow can learn and try different filters and determine which filters work best when looking at the training data. As a result, this will greatly reduce the information passing through the network while also improving the accuracy of the license plate detection by isolating the key features of the license plate image.

5.5.2 Pooling

Pooling is also a technique used to enhance the power of convolutions. The concept of pooling is to take groups of pixels and perform an aggregation over them.[10]

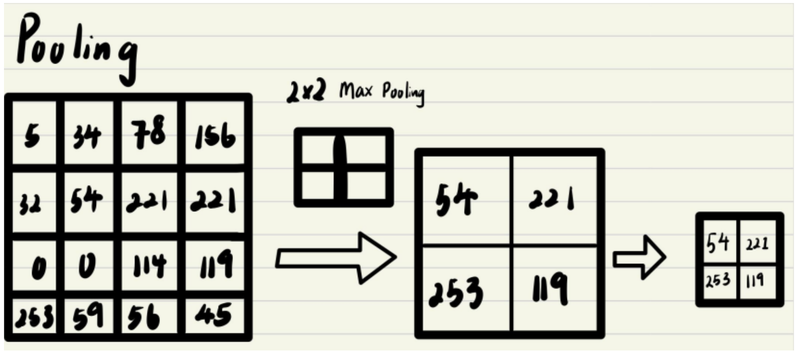


Figure 5.8: Pooling Technique

In a neural network, pooling is used to reduce the spatial size of the output from the CNN layer, thereby reducing the size of the image and the number of parameters required in the network. Pooling is generally done after the convolutional layer. This pooling process makes it easier to process further down the network and extract the most important features from the image which for Parkify is the license plate characters and number from the car image. The pooling process reduces the information without removing all the features instead of combining them.[7]

5.5.3 OCR Image Processing

In the Parkify system, the logic and process behind the TesseractOCR to be able to increase the accuracy of detected characters and numbers on the license plate is image processing. The following paragraphs will show and explain the logic and process involved in the OCR image process.

1. Get the bounding box coordinates and extract the image region within the bound of the box.
2. Scale the image up to 3x its original size.

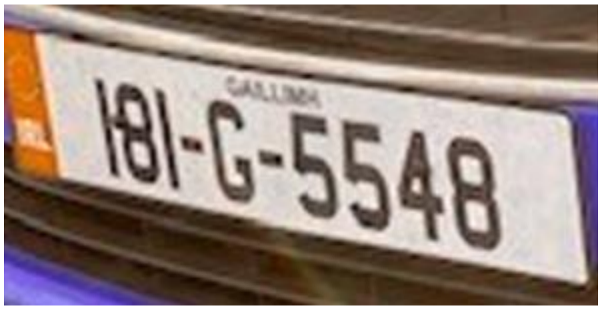


Figure 5.9: Scale Image

3. Convert the image into a grayscale and apply a small Gaussian blur to smooth it.

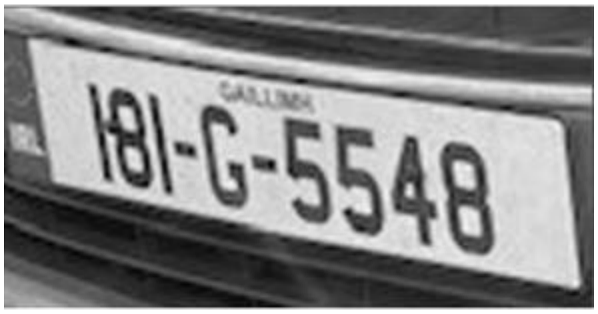


Figure 5.10: Applying Grayscale and Gaussian Blur

4. Set a threshold(Otsu Threshold Method) to white text with a black background. This will help to find the contours of the image with white text on black background.

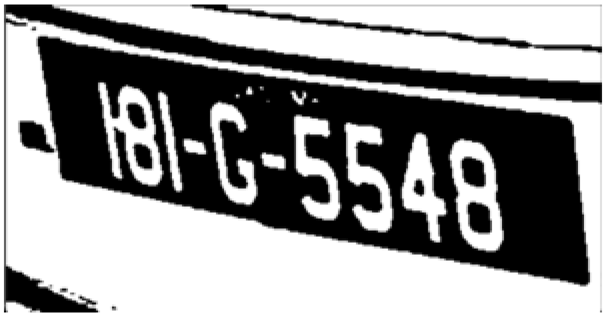


Figure 5.11: Otsu Threshold Method

5. By using OpenCV to find all rectangular-shaped contours on the image and sort them from LEFT to RIGHT.

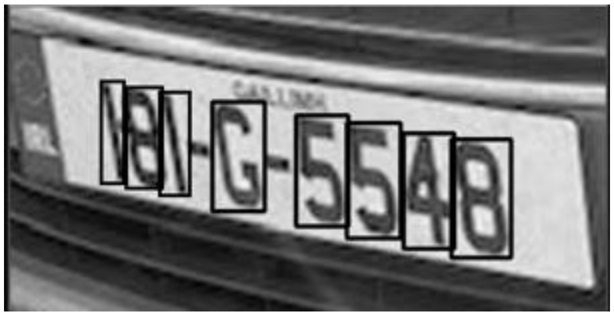


Figure 5.12: Contour Filtering

6. Flip the image to black text on a white background which TesseractOCR is more accurate with detecting the characters and numbers on it. Also, apply the median blur on the image and pass it to Tesseract to get the characters or numbers from it.

5.6 Labelling Data

In supervised machine learning, labelling data is a crucial process of assigning the labels or categories to a set of input data which lets the machine learn and train to formulate the rules of the dataset. These rules act as the machine learning algorithm, which learns to recognize the patterns and make predictions based on the labelled data.

Labelled data is normally split into two sets which are a training set and a test set. The training set is used to train and let the machine learn the algorithms of what we labelled. On the other hand, a test set is used to evaluate the performance. The test set is a data set used to provide an unbiased evaluation of a final model fit on the training data set. [11]

The Parkify system utilizes Google Cloud to store the database collected, and the labelling data process (labelling license plate of a car) is manually labelled in the Google Cloud platform.

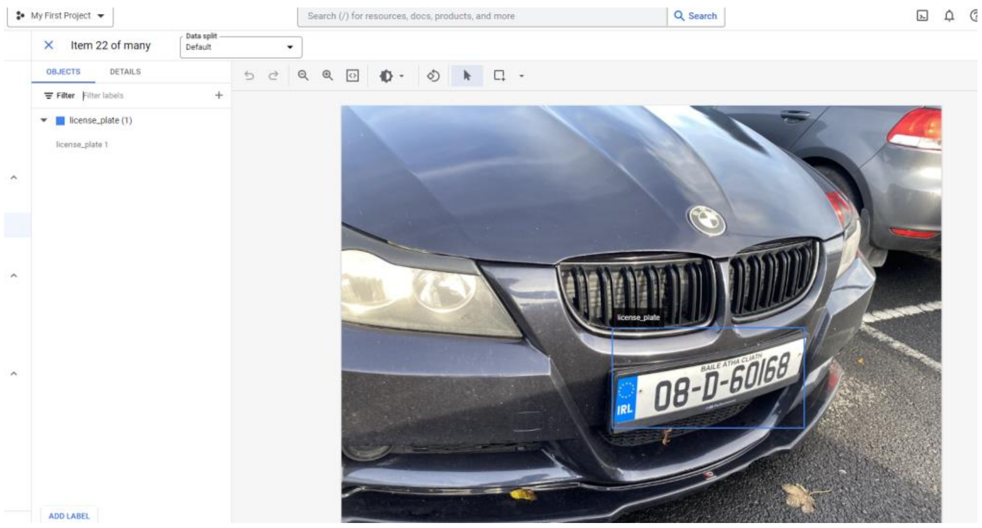


Figure 5.13: Labelling Data

5.7 Optical Character Recognition

Optical Character Recognition(OCR) is a process that converts physical printed texts in real life and converts it into digital texts or digital image files that are machine readable. There is various optical character recognition engine library that could use in the Parkify system such as TesseractOCR, EasyOCR, GOCR, et cetera. The most populated and precise OCR is TesseractOCR which will be implemented in the Parkify system to ensure high accuracy in detection and fast response.

The OCR process mostly involves several steps, including image pre-processing, feature extraction, classification, and post-processing. The accuracy of the OCR depends on various factors such as the quality of the image(pixels), the font of the characters used, and the complexity of the characters.

To achieve and implement object detection which specifically detects license plates and implements Optical Character Recognition(OCR) engine that handles detecting characters and numbers on the license plate, there are various algorithms specifically designed for it and will be discussed in the other sections.

5.8 YOLO Algorithm

At the beginning of the Parkify system are implementing the YOLOv4 algorithm to train our models and detect license plates. However, due to the limited accuracy of the local Tesseract OCR, the Parkify system switched to Google Cloud Auto Machine Learning with Cloud Vision API which we will be discussed the implementation and details on Google Cloud further in the other section.

YOLO(You Only Look Once) is a real-time object detection algorithm that uses neural networks to provide real-time detection. The algorithm is a popular object detection model known for its speed and accuracy. It was first introduced by Joseph Redmon et al. in 2016 and has since undergone several iterations, the latest being YOLO v7. [12]

Somehow the YOLO algorithm provides insight into how computer vision works as part of our research to gain a deeper understanding of how it works.

5.8.1 Residual Blocks

YOLO uses residual blocks, which are a type of Convolutional Neural Network(CNN) layer previously discussed in Supervised Machine Learning to help speed up training.

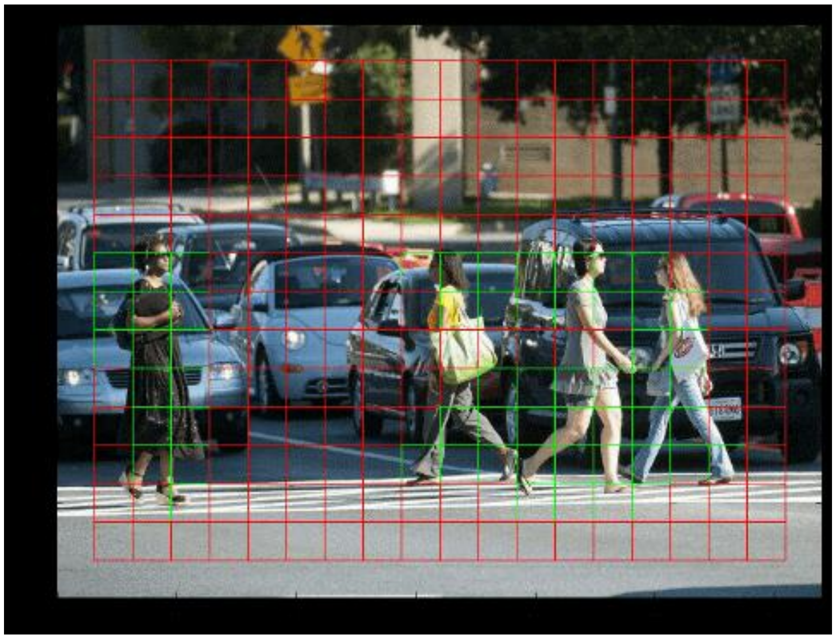


Figure 5.14: Residual Blocks[13]

The original image will be divided into grid boxes and each grid has an equal dimension. The figure above shows how an input image is divided into grids of equal dimensions. Every grid cell will detect objects that appear within the grid boxes. For example, if there is an object appears within a certain grid cell, then the cell will be responsible for detecting it.

5.8.2 Bounding box regression

Bounding box regression is a technique used to refine or predict the bounding boxes by the network. This step is to determine the bounding boxes which correspond to highlighted objects in the image. Every bounding box in the image consists of the following attributes: Width (bw), Height (bh), Class (for example, person, car, traffic light, etc. represented as letter c), and bounding box centre (bx, by).

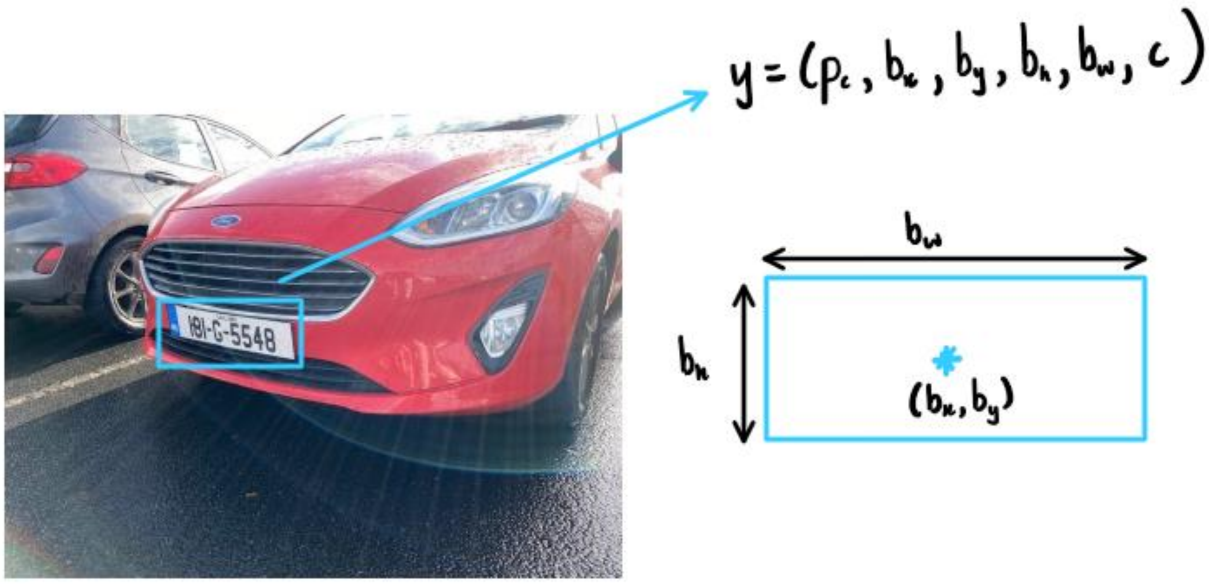


Figure 5.15: Bounding box overview

In our Parkify project, the class will be license plate which is the area of interest that the bounding box will be highlighted around with the given image like the example figure below.

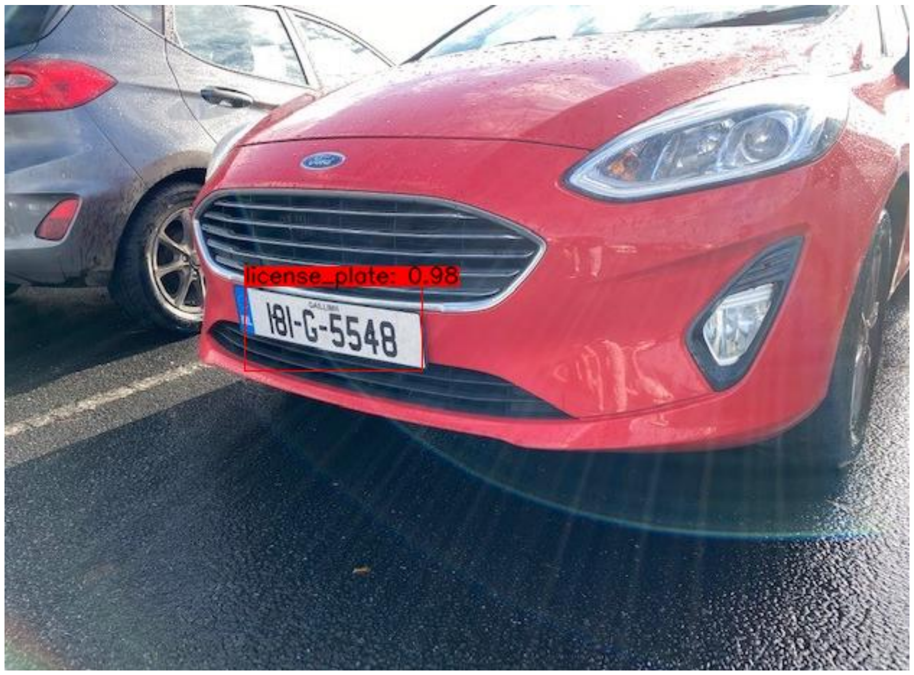


Figure 5.16: Example of the area of interest of bounding box

5.8.3 Intersect Over Unions (IOU)

Intersect Over Union(IOU) is a measure of how well the predicted bounding box overlaps. It is used to evaluate the accuracy of the predictions. Most of the time, a single object in an image can have multiple grid box candidates for prediction, even though not all of them are relevant.[14]

Each of the grid cells is predicting the bounding boxes and their confidence scores. The IOU value is 1 when the predicted bounding box matches the actual real box. This method will help to reduce the size of the box and eliminates the bounding boxes that do not equal the real actual box. The figure shown below is an example explanation of the IOU.[14]

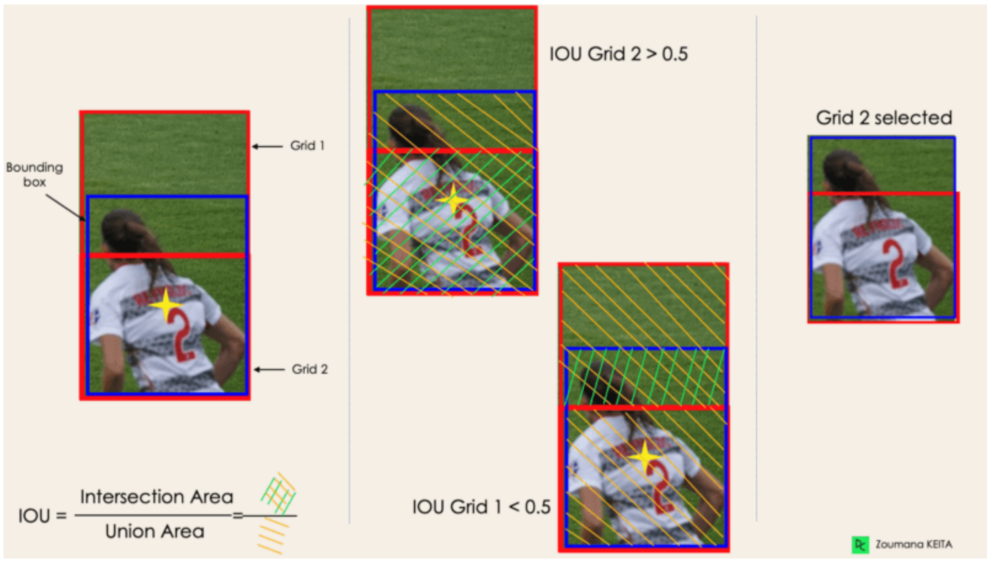


Figure 5.17: Intersect Over Union Grid

5.8.4 Combinations

With the combinations of the three techniques stated above, the following image shows and provides the results of it.[15]

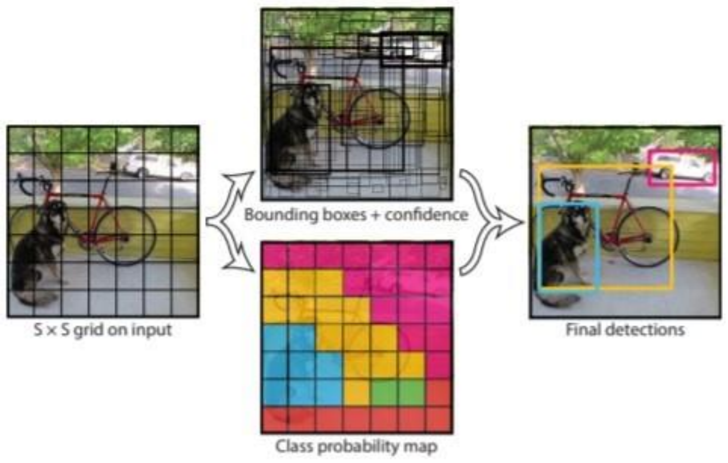


Figure 5.18: Combination of the techniques

The image is first divided into S x S grid cells on the input image. Each grid cell will predicts the bounding boxes and provides their confidence score then predict the class probabilities.

As the figure above, the observation is that the car is surrounded by the pink bounding box, the bicycle is surrounded by the yellow bounding box, and the dog has been highlighted and surrounded by the blue bounding box. In this case, IOU ensures that the final detection consists of unique bounding boxes that fit the objects perfectly.

The following figure below presents the results obtained from online sources.[16]

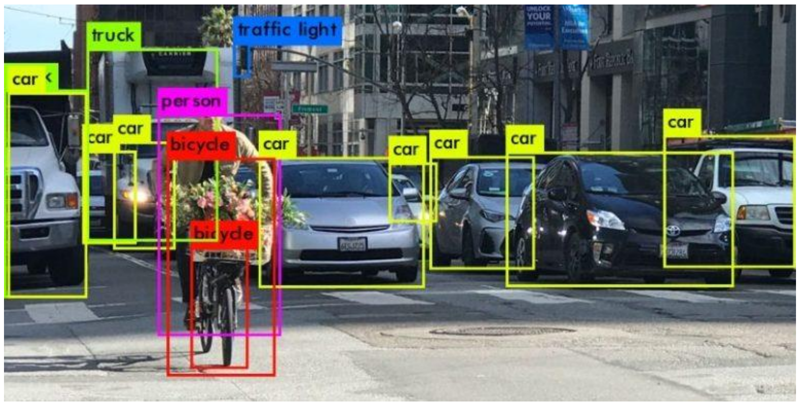


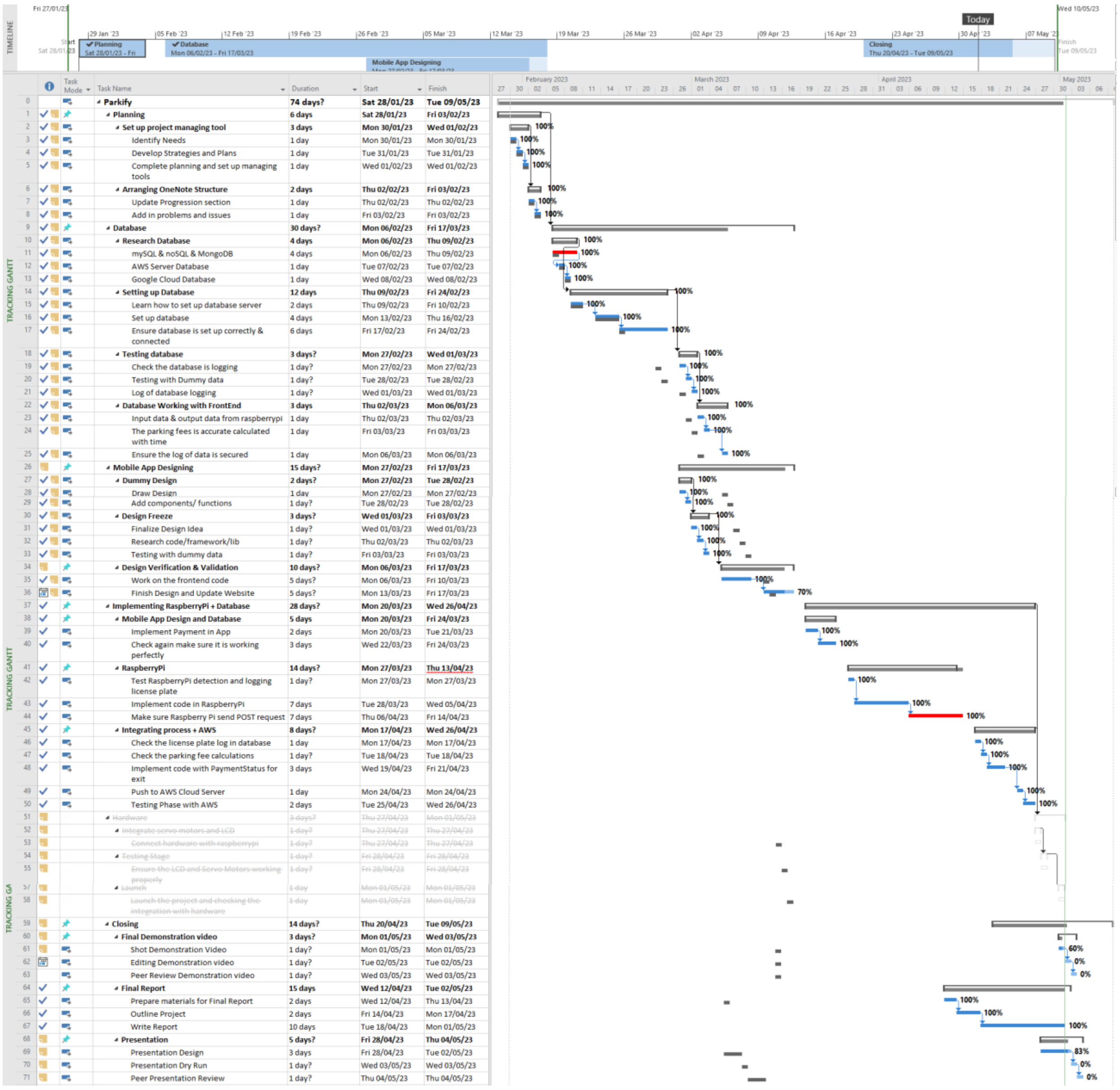
Figure 5.19: Example YOLO Results

6 Project Plan

The project management tool using in this project is Microsoft Project and Microsoft OneNote, which Project could set the baseline of the project date and track the project progression while OneNote to log my progression details, the problems met and the solutions to the problems.

6.1 Microsoft Project Gantt Chart

The Gantt chart of my project in Microsoft Project shown in figure below:



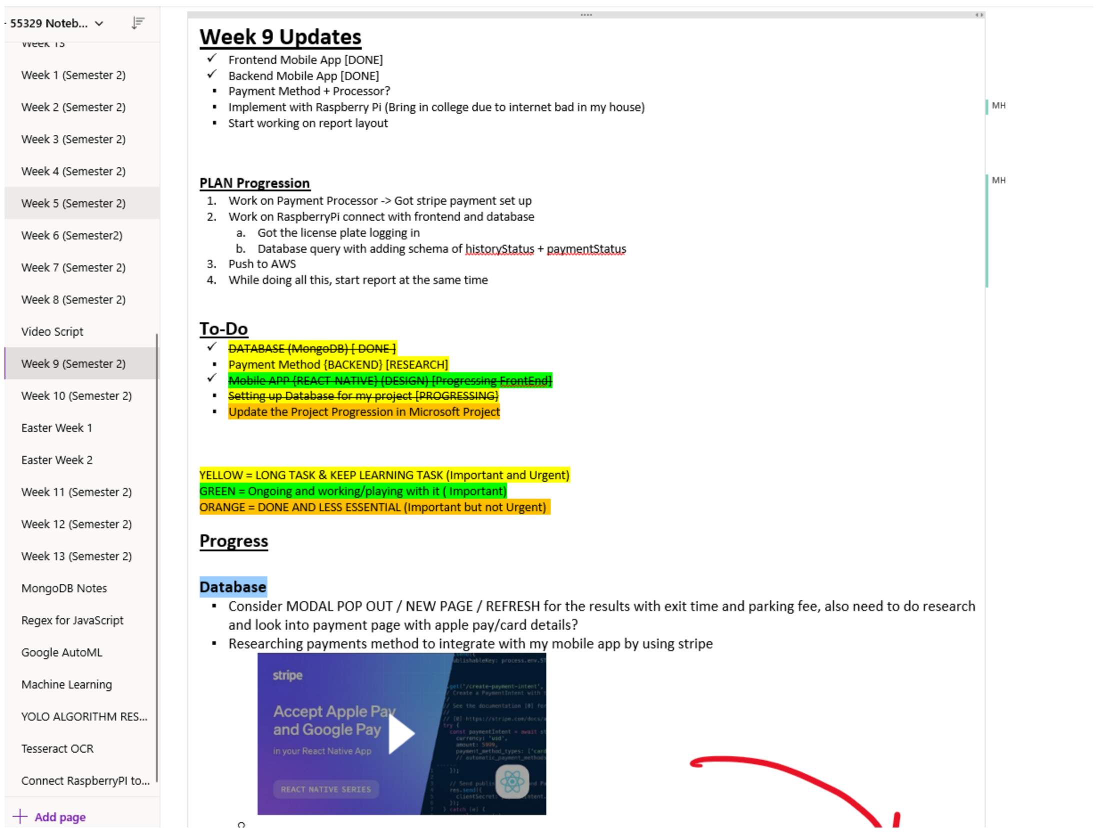
		Task Name	Duration	Start	Finish
0		Parkify	74 days?	Sat 28/01/23	Tue 09/05/23
1	✓	Planning	6 days	Sat 28/01/23	Fri 03/02/23
2	✓	Set up project managing tool	3 days	Mon 30/01/23	Wed 01/02/23
3	✓	Identify Needs	1 day	Mon 30/01/23	Mon 30/01/23
4	✓	Develop Strategies and Plans	1 day	Tue 31/01/23	Tue 31/01/23
5	✓	Complete planning and set up managing tools	1 day	Wed 01/02/23	Wed 01/02/23
6	✓	Arranging OneNote Structure	2 days	Thu 02/02/23	Fri 03/02/23
7	✓	Update Progression section	1 day	Thu 02/02/23	Thu 02/02/23
8	✓	Add in problems and issues	1 day	Fri 03/02/23	Fri 03/02/23
9	✓	Database	30 days?	Mon 06/02/23	Fri 17/03/23
10	✓	Research Database	4 days	Mon 06/02/23	Thu 09/02/23
11	✓	mySQL & noSQL & MongoDB	4 days	Mon 06/02/23	Thu 09/02/23
12	✓	AWS Server Database	1 day	Tue 07/02/23	Tue 07/02/23
13	✓	Google Cloud Database	1 day	Wed 08/02/23	Wed 08/02/23
14	✓	Setting up Database	12 days	Thu 09/02/23	Fri 24/02/23
15	✓	Learn how to set up database server	2 days	Thu 09/02/23	Fri 10/02/23
16	✓	Set up database	4 days	Mon 13/02/23	Thu 16/02/23
17	✓	Ensure database is set up correctly & connected	6 days	Fri 17/02/23	Fri 24/02/23
18	✓	Testing database	3 days?	Mon 27/02/23	Wed 01/03/23
19	✓	Check the database is logging	1 day?	Mon 27/02/23	Mon 27/02/23
20	✓	Testing with Dummy data	1 day?	Tue 28/02/23	Tue 28/02/23
21	✓	Log of database logging	1 day?	Wed 01/03/23	Wed 01/03/23
22	✓	Database Working with FrontEnd	3 days	Thu 02/03/23	Mon 06/03/23
23	✓	Input data & output data from raspberrypi	1 day	Thu 02/03/23	Thu 02/03/23
24	✓	The parking fees is accurate calculated with time	1 day	Fri 03/03/23	Fri 03/03/23
25	✓	Ensure the log of data is secured	1 day	Mon 06/03/23	Mon 06/03/23
26		Mobile App Designing	15 days?	Mon 27/02/23	Fri 17/03/23
27	✓	Dummy Design	2 days?	Mon 27/02/23	Tue 28/02/23
28	✓	Draw Design	1 day	Mon 27/02/23	Mon 27/02/23
29	✓	Add components/ functions	1 day?	Tue 28/02/23	Tue 28/02/23
30	✓	Design Freeze	3 days?	Wed 01/03/23	Fri 03/03/23
31	✓	Finalize Design Idea	1 day?	Wed 01/03/23	Wed 01/03/23
32	✓	Research code/framework/lib	1 day?	Thu 02/03/23	Thu 02/03/23
33	✓	Testing with dummy data	1 day?	Fri 03/03/23	Fri 03/03/23
34		Design Verification & Validation	10 days?	Mon 06/03/23	Fri 17/03/23
35	✓	Work on the frontend code	5 days?	Mon 06/03/23	Fri 10/03/23
36		Finish Design and Update Website	5 days?	Mon 13/03/23	Fri 17/03/23
37	✓	Implementing RaspberryPi + Database	28 days?	Mon 20/03/23	Wed 26/04/23
38	✓	Mobile App Design and Database	5 days	Mon 20/03/23	Fri 24/03/23
39	✓	Implement Payment in App	2 days	Mon 20/03/23	Tue 21/03/23
40	✓	Check again make sure it is working perfectly	3 days	Wed 22/03/23	Fri 24/03/23
41	✓	RaspberryPi	14 days?	Mon 27/03/23	Thu 13/04/23
42	✓	Test RaspberryPi detection and logging license plate	1 day?	Mon 27/03/23	Mon 27/03/23
43	✓	Implement code in RaspberryPi	7 days	Tue 28/03/23	Wed 05/04/23
44	✓	Make sure Raspberry Pi send POST request	7 days	Thu 06/04/23	Fri 14/04/23
45	✓	Integrating process + AWS	8 days?	Mon 17/04/23	Wed 26/04/23
46	✓	Check the license plate log in database	1 day	Mon 17/04/23	Mon 17/04/23
47	✓	Check the parking fee calculations	1 day?	Tue 18/04/23	Tue 18/04/23
48	✓	Implement code with PaymentStatus for exit	3 days	Wed 19/04/23	Fri 21/04/23
49	✓	Push to AWS Cloud Server	1 day	Mon 24/04/23	Mon 24/04/23
50	✓	Testing Phase with AWS	2 days	Tue 25/04/23	Wed 26/04/23
51		~~Hardware~~	~~3 days?~~	~~Thu 27/04/23~~	~~Mon 01/05/23~~
52		~~Integrate servo motors and LCD~~	~~1 day?~~	~~Thu 27/04/23~~	~~Thu 27/04/23~~
53		~~Connect hardware with raspberrypi~~	~~1 day?~~	~~Thu 27/04/23~~	~~Thu 27/04/23~~
54		~~Testing Stage~~	~~1 day?~~	~~Fri 28/04/23~~	~~Fri 28/04/23~~
55		~~Ensure the LCD and Servo Motors working properly~~	~~1 day?~~	~~Fri 28/04/23~~	~~Fri 28/04/23~~
57		~~Launch~~	~~1 day~~	~~Mon 01/05/23~~	~~Mon 01/05/23~~
58		~~Launch the project and checking the integration with hardware~~	~~1 day~~	~~Mon 01/05/23~~	~~Mon 01/05/23~~
59		Closing	14 days?	Thu 20/04/23	Tue 09/05/23
60		Final Demonstration video	3 days?	Mon 01/05/23	Wed 03/05/23
61		Shot Demonstration Video	1 day?	Mon 01/05/23	Mon 01/05/23
62		Editing Demonstration video	1 day?	Tue 02/05/23	Tue 02/05/23
63		Peer Review Demonstration video	1 day?	Wed 03/05/23	Wed 03/05/23
64	✓	Final Report	15 days	Wed 12/04/23	Tue 02/05/23
65	✓	Prepare materials for Final Report	2 days	Wed 12/04/23	Thu 13/04/23
66	✓	Outline Project	2 days	Fri 14/04/23	Mon 17/04/23
67	✓	Write Report	10 days	Tue 18/04/23	Mon 01/05/23
68		Presentation	5 days?	Fri 28/04/23	Thu 04/05/23
69		Presentation Design	3 days	Fri 28/04/23	Tue 02/05/23
70		Presentation Dry Run	1 day?	Wed 03/05/23	Wed 03/05/23
71		Peer Presentation Review	1 day?	Thu 04/05/23	Thu 04/05/23

Figure 6.1: Gantt Chart

6.2 Microsoft OneNote

Microsoft OneNote using in the Parkify project to record and log the progress and researches that did during the project weekly.

The examples of the weekly log are shown below:



Week 9 Updates

- ✓ Frontend Mobile App [DONE]
- ✓ Backend Mobile App [DONE]
- Payment Method + Processor?
- Implement with Raspberry Pi (Bring in college due to internet bad in my house)
- Start working on report layout

PLAN Progression

1. Work on Payment Processor -> Got stripe payment set up
2. Work on RaspberryPi connect with frontend and database
    a. Got the license plate logging in
    b. Database query with adding schema of historyStatus + paymentStatus
3. Push to AWS
4. While doing all this, start report at the same time

To-Do

- ✓ ~~DATABASE (MongoDB) [DONE]~~
- Payment Method {BACKEND} [RESEARCH]
- ✓ ~~Mobile APP {REACT NATIVE} (DESIGN) [Progressing FrontEnd]~~
- ~~Setting up Database for my project [PROGRESSING}~~
- Update the Project Progression in Microsoft Project

YELLOW = LONG TASK & KEEP LEARNING TASK (Important and Urgent)
GREEN = Ongoing and working/playing with it (Important)
ORANGE = DONE AND LESS ESSENTIAL (Important but not Urgent)

Progress

Database

- Consider MODAL POP OUT / NEW PAGE / REFRESH for the results with exit time and parking fee, also need to do research and look into payment page with apple pay/card details?
- Researching payments method to integrate with my mobile app by using stripe

Figure 6.2: Weekly Log Example

7 Raspberry Pi Setup

7.1 Pi Camera Module

The first step is to establish the connection between the Raspberry Pi and the PiCamera v2 module. By locating the camera module port on the Raspberry Pi and inserting the camera module ribbon cable into the port to ensure that the Raspberry Pi is connected with the PiCamera v2 module as shown below.

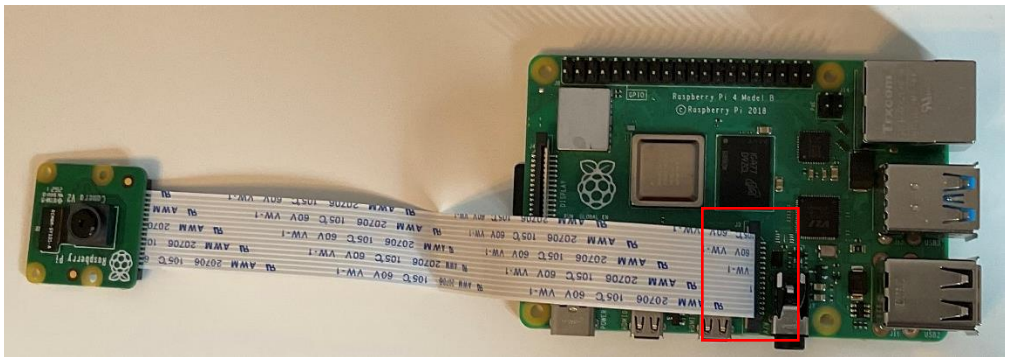


Figure 7.1: Raspberry Pi with Pi Camera v2

7.2 Raspberry Pi Imager

The Raspberry Pi Imager tool enables the quick formatting of the operating system and writing onto a micro SD card.

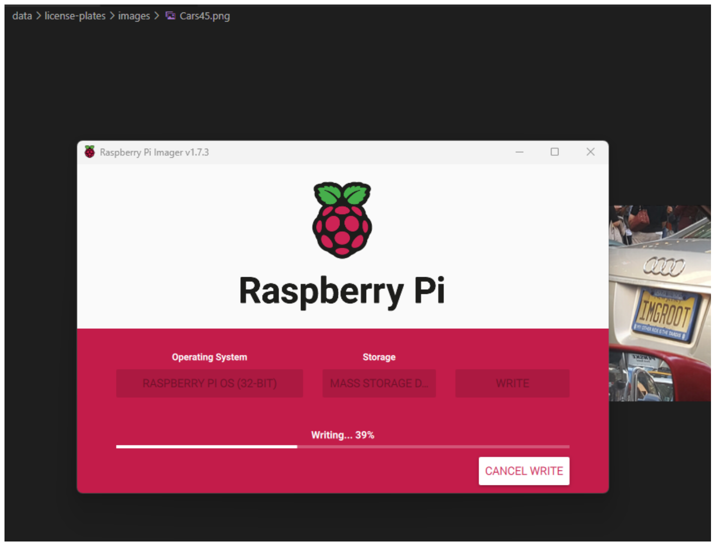
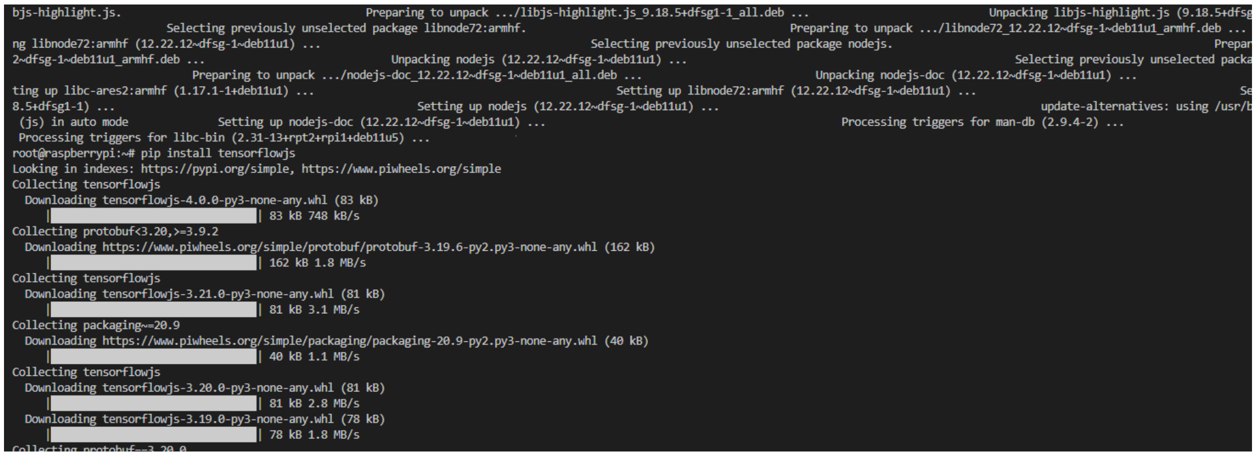


Figure 7.2: Raspberry Pi Imager

The Raspberry Pi Imager tool also allows us to pre-config the Wi-Fi network SSID with a password and enable the connection of SSH. For the operating system, the Parkify system uses the recommended operating system which is Raspberry Pi OS (32-bit). The final part of installing the operating system and pre-configuration (Wi-Fi and SSH settings) of Raspberry Pi is to select the SD card that is used for storage. Lastly, we will just need to insert the formatted micro SD card into the designated slot on the Raspberry Pi.

7.3 Installing dependencies

After configuring Raspberry Pi, the next step is installing the necessary dependencies and packages that are required for the Parkify system which includes NodeJS and TensorFlow.js.



```
bjs-highlight.js.                                   Preparing to unpack .../libjs-highlight.js_9.18.5+dfsg1-1_all.deb ...                          Unpacking libjs-highlight.js (9.18.5+dfsg
                        Selecting previously unselected package libnode72:armhf.                                        Preparing to unpack .../libnode72_12.22.12~dfsg-1~deb11u1_armhf.deb ...
ng libnode72:armhf (12.22.12~dfsg-1~deb11u1) ...                                    Selecting previously unselected package nodejs.                                           Prepar
2~dfsg-1~deb11u1_armhf.deb ...                         Unpacking nodejs (12.22.12~dfsg-1~deb11u1) ...                                         Selecting previously unselected packa
                            Preparing to unpack .../nodejs-doc_12.22.12~dfsg-1~deb11u1_all.deb ...                           Unpacking nodejs-doc (12.22.12~dfsg-1~deb11u1) ...
ting up libc-ares2:armhf (1.17.1-1+deb11u1) ...                                         Setting up libnode72:armhf (12.22.12~dfsg-1~deb11u1) ...                                    Se
8.5+dfsg1-1) ...                                             Setting up nodejs (12.22.12~dfsg-1~deb11u1) ...                                           update-alternatives: using /usr/b
 (js) in auto mode              Setting up nodejs-doc (12.22.12~dfsg-1~deb11u1) ...                                   Processing triggers for man-db (2.9.4-2) ...
 Processing triggers for libc-bin (2.31-13+rpt2+rpi1+deb11u5) ...
root@raspberrypi:~# pip install tensorflowjs
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting tensorflowjs
  Downloading tensorflowjs-4.0.0-py3-none-any.whl (83 kB)
     |████████████████████████████████| 83 kB 748 kB/s
Collecting protobuf<3.20,>=3.9.2
  Downloading https://www.piwheels.org/simple/protobuf/protobuf-3.19.6-py2.py3-none-any.whl (162 kB)
     |████████████████████████████████| 162 kB 1.8 MB/s
Collecting tensorflowjs
  Downloading tensorflowjs-3.21.0-py3-none-any.whl (81 kB)
     |████████████████████████████████| 81 kB 3.1 MB/s
Collecting packaging~=20.9
  Downloading https://www.piwheels.org/simple/packaging/packaging-20.9-py2.py3-none-any.whl (40 kB)
     |████████████████████████████████| 40 kB 1.1 MB/s
Collecting tensorflowjs
  Downloading tensorflowjs-3.20.0-py3-none-any.whl (81 kB)
     |████████████████████████████████| 81 kB 2.8 MB/s
  Downloading tensorflowjs-3.19.0-py3-none-any.whl (78 kB)
     |████████████████████████████████| 78 kB 1.8 MB/s
Collecting protobuf==3.20.0
```

Figure 7.3: Dependencies and packages installation

7.4 Testing the PiCamera

This step involves testing the PiCamera functionality by running the command libcamera-jpeg -o test.jpg, it allows us to make sure the camera is operating properly. The image quality with shutter -10000 is shown in the image result below.

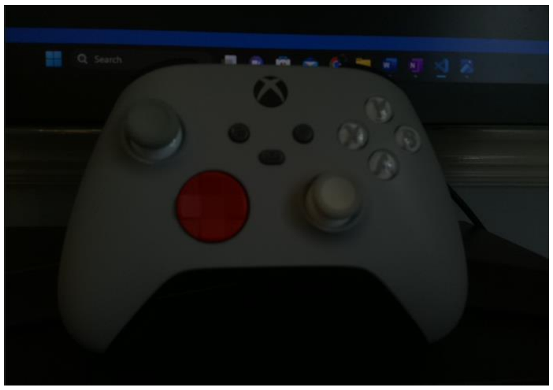
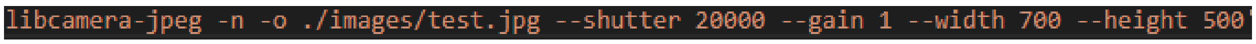


Figure 7.4: Example image before adjusting

7.4.1 Configuring PiCamera settings

During this stage, the camera exposure was too long, and the exposure brightness was affecting the quality of the image, resulting in the image doesn't show clearly. By referring to the picamera documents and reviewing the Raspberry Pi forum, I was able to adjust to the preferred settings to capture a clear image of the license plate. The code snippet with the camera settings and result image is shown below:



```
libcamera-jpeg -n -o ./images/test.jpg --shutter 20000 --gain 1 --width 700 --height 500
```

Figure 7.5: Pi Camera settings code snippet

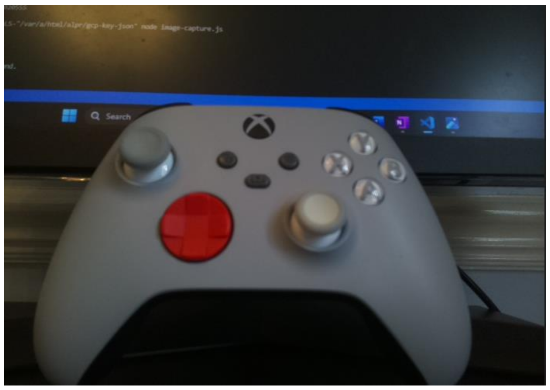


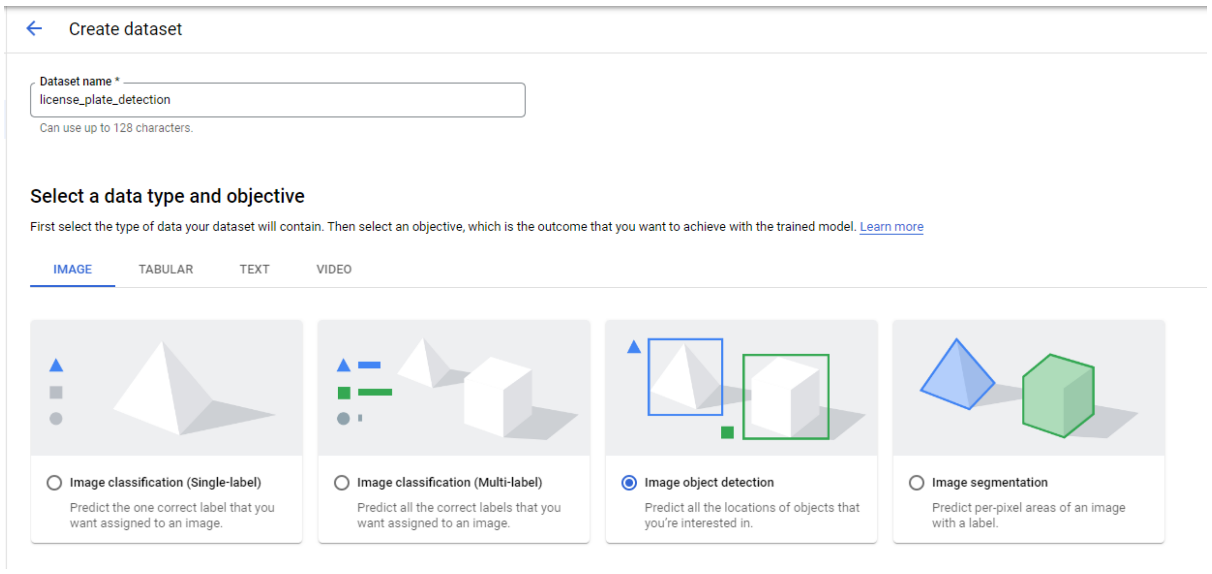
Figure 7.6: Example image after adjusting

8 Machine Learning Model Training

The Parkify system implemented with Google Cloud Vertex AI which is a machine-learning platform that offers an easy and convenient data training model for users.[17]

8.1 Creating Datasets

In Vertex AI there is a variety of options for selecting the type of dataset required for training the model.



Create dataset

Dataset name *
license_plate_detection
Can use up to 128 characters.

Select a data type and objective

First select the type of data your dataset will contain. Then select an objective, which is the outcome that you want to achieve with the trained model. Learn more

IMAGE TABULAR TEXT VIDEO

( ) Image classification (Single-label)
Predict the one correct label that you want assigned to an image.

( ) Image classification (Multi-label)
Predict all the correct labels that you want assigned to an image.

[✓ Image object detection]
Predict all the locations of objects that you're interested in.

( ) Image segmentation
Predict per-pixel areas of an image with a label.

Figure 8.1: Creating Dataset in Google Cloud Platform

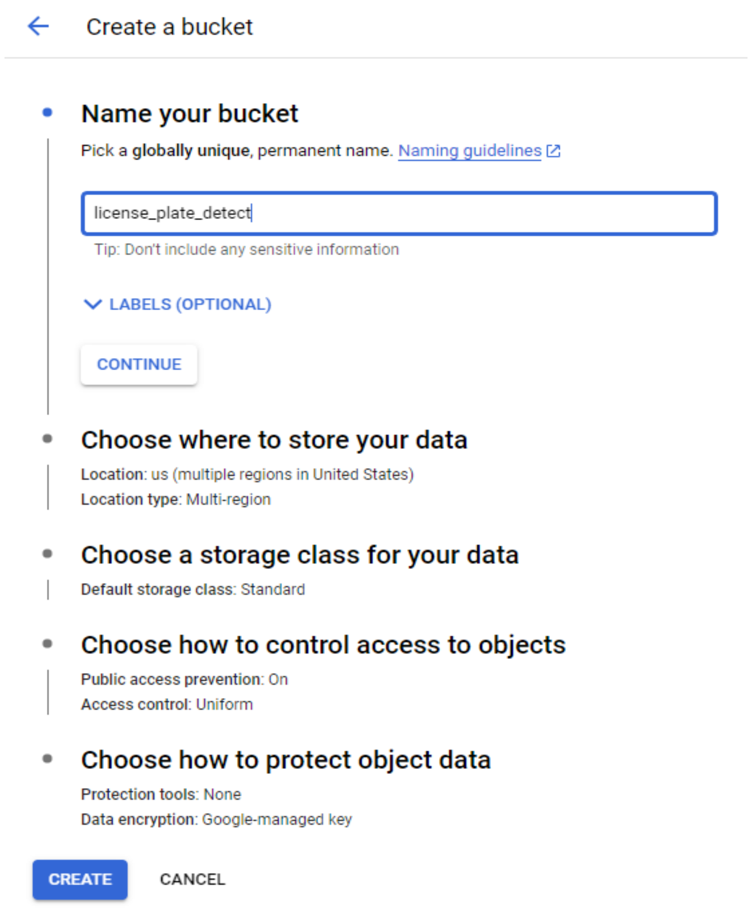
The figure above shows the four different data type which is Image, Tabular, Text and Video that could train. In the Parkify system, ALPR technology has relied on images and extracting the characters and numbers from the license plate region in the image.

The figure also shows four objectives that could be trained to achieve the expected outcomes with the trained model. Parkify system is focusing on detecting the location of license plates in images. Therefore, image object detection is considered the optimal approach as it predicts the locations of objects that are interested in the image.

On the other hand, image classification involves identifying the class or category of an entire image to predict the correct label (single/multi). Also, image segmentation is used to identify the segment of different regions in an image. Therefore, based on the specific requirements of the Parkify system, image object detection is the most appropriate approach for detecting license plates in the images.

8.2 Importing training data

Before training the dataset on Google Cloud, the training must import into Google Cloud Storage. This can be achieved by creating a bucket and uploading the data to the bucket, while also configuring the privacy control access settings so that allow the bucket to access for training.



Create a bucket

Name your bucket

Pick a **globally unique**, permanent name. Naming guidelines

license_plate_detect

Tip: Don't include any sensitive information

Choose where to store your data

Location: us (multiple regions in United States)
Location type: Multi-region

Choose a storage class for your data

Default storage class: Standard

Choose how to control access to objects

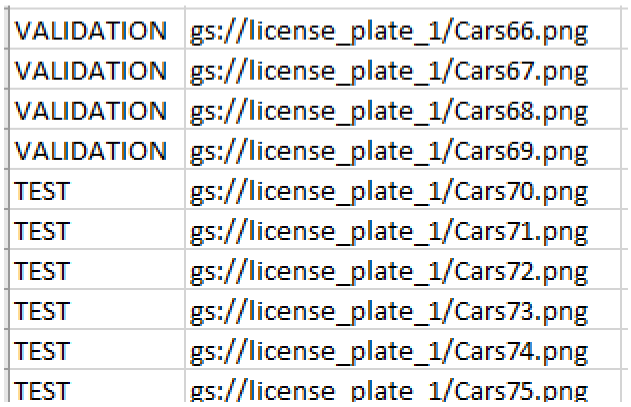
Public access prevention: On
Access control: Uniform

Choose how to protect object data

Protection tools: None
Data encryption: Google-managed key

Figure 8.2: Creating bucket storage in Google Cloud Platform

Additionally, a CSV file needs to create specifying the data used for training, validation or testing as the figure is shown below.



VALIDATION	gs://license_plate_1/Cars66.png
VALIDATION	gs://license_plate_1/Cars67.png
VALIDATION	gs://license_plate_1/Cars68.png
VALIDATION	gs://license_plate_1/Cars69.png
TEST	gs://license_plate_1/Cars70.png
TEST	gs://license_plate_1/Cars71.png
TEST	gs://license_plate_1/Cars72.png
TEST	gs://license_plate_1/Cars73.png
TEST	gs://license_plate_1/Cars74.png
TEST	gs://license_plate_1/Cars75.png

Figure 8.3: Importing data

The process of dividing the data into training, validation and testing subsets is called the hold-out method which involves using a portion of data for training the model (usually the largest percentage) and using the remaining subsets for validating and testing the model.

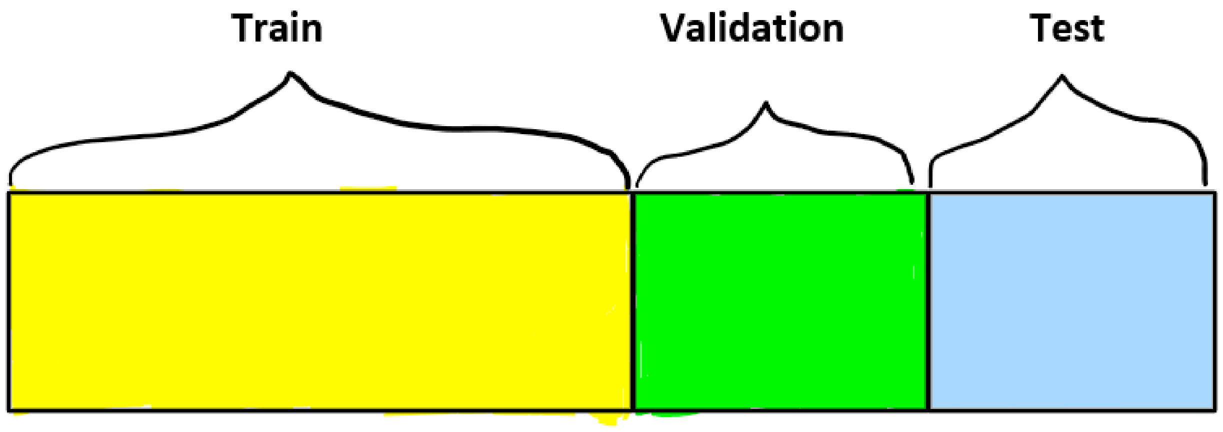


Figure 8.4: Dataset portion

The training dataset is the set of data used to train a machine-learning model.

“The model sees and learns from this data”.[18]

The validation dataset is the dataset that is used to tune the model and select the best model.

The testing dataset is the dataset that is used to evaluate the final performance of the model.

The “training” data set is the general term for the samples used to create the model, while the “test” or “validation” data set is used to qualify performance.[11]

8.3 Labelling Data

After importing all the datasets for training, the next step before starting to train the model is labelling data. Google Cloud Platform provides bounding box tools which let us label the data by drawing the bounding boxes around the area of interest which is the license plate region in the Parkify system.

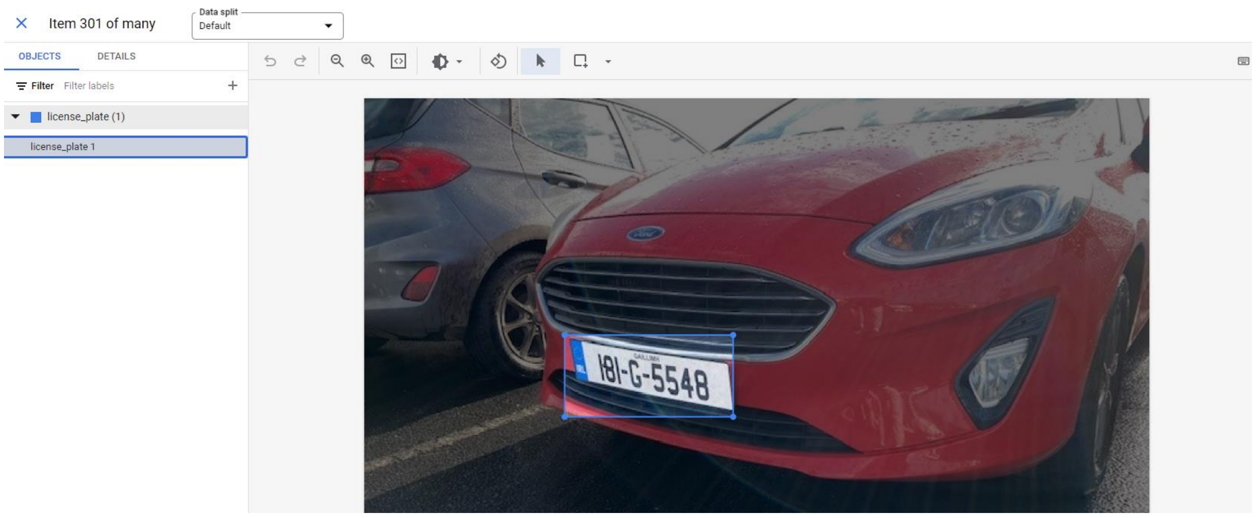
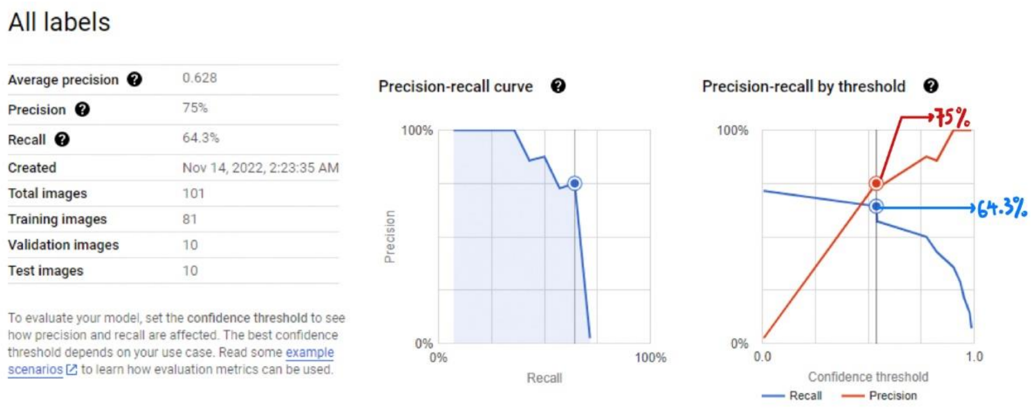


Figure 8.5: Labelling data on Google Cloud Platform

In the labelling process, it is necessary to classify the region of interest as an object which in the Parkify system is license_plate as shown in the figure above. This helps in training the model to identify and distinguish the license plate more accurately within the image.

8.4 Training Data

The next and significant step involves training the model with the datasets and labelled data that were previously provided on Google Cloud.



All labels

Average precision	0.628
Precision	75%
Recall	64.3%
Created	Nov 14, 2022, 2:23:35 AM
Total images	101
Training images	81
Validation images	10
Test images	10

To evaluate your model, set the confidence threshold to see how precision and recall are affected. The best confidence threshold depends on your use case. Read some example scenarios to learn how evaluation metrics can be used.

Figure 8.6: Trained Model Performance Graph

After training the data, the Google Cloud platform displays the graphs and statistical details of the trained model as shown figure above. Adjusting the confidence threshold and IoU threshold, will evaluate the final performance and results of the model and improve the precision of the label results.

The data previously shows that the model got a precision accuracy percentage of 75% and a recall accuracy percentage of 64.3%. By implementing this model, it can identify the true and false negative results in license plate detection. Despite the detection of license plates, the case below observed in the figure below was classified as false negatives and the low confidence score of the true positives license plate. The false negative refers to the model failing to detect the license plate that is actually present in the image as shown below:

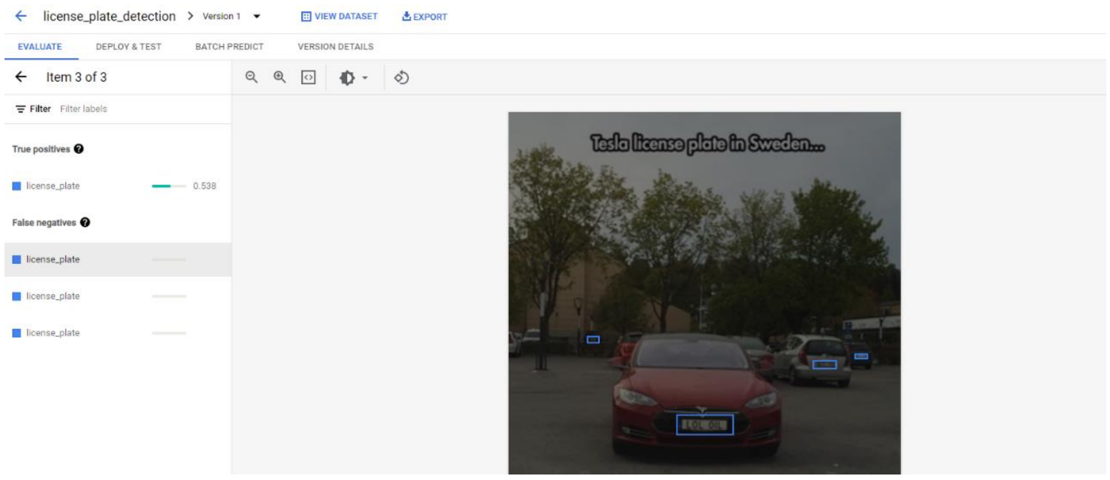
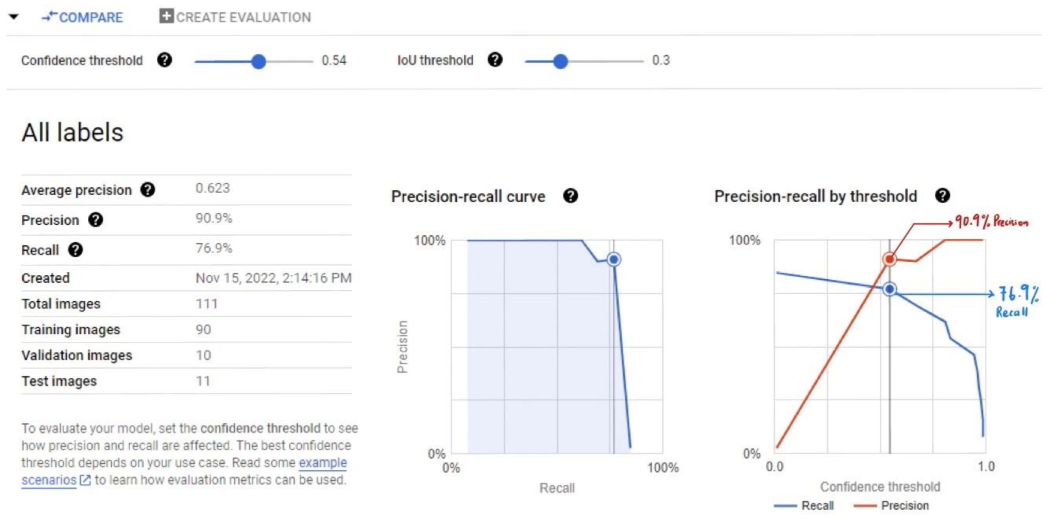


Figure 8.7: Trained Model Test

8.4.1 Model Performance Evaluation

In the Parkify system, detecting accurate license plates is crucial. However, the previous model did not meet the expected performance. Hence, this step focuses on improving the precision and recall percentages of the model.



Confidence threshold 0.54
IoU threshold 0.3

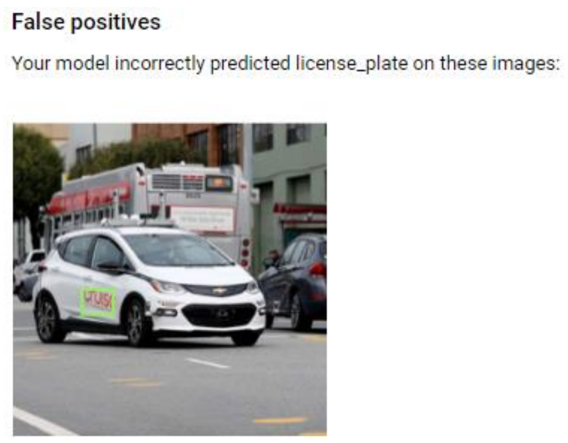
Average precision	0.623
Precision	90.9%
Recall	76.9%
Created	Nov 15, 2022, 2:14:16 PM
Total images	111
Training images	90
Validation images	10
Test images	11

To evaluate your model, set the confidence threshold to see how precision and recall are affected. The best confidence threshold depends on your use case. Read some example scenarios to learn how evaluation metrics can be used.

Figure 8.8: Improved Trained Model Performance Graph

As the figure shows above, the confidence threshold and Intersection Over Union (IoU) threshold are adjusted to a certain value which will achieve the precision accuracy from 75% to 90.9% and the recall percentage from 64.3% to 76.9% for the license plate detection model.

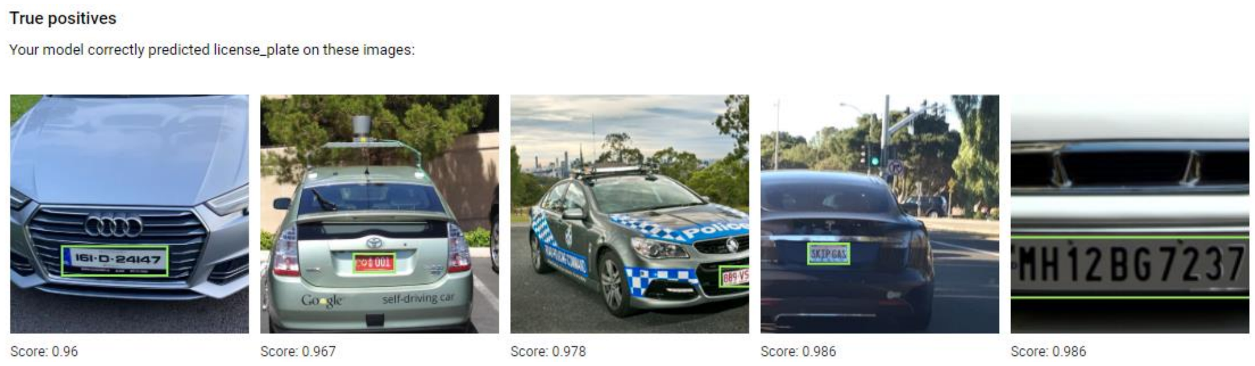
After improving the trained model, the test results on the image with false positive which tells the situation where the model predicts the presence of an object likely license plate when there is actually no license plate present in the image. The image shown below is an example from the improved model with detected area interest of the advertisement on the side of the car which identifies as the false positive of the model.


False positives

Your model incorrectly predicted license_plate on these images:

Figure 8.9: Example of false positive detected

On the other hand, the improved model has a higher confidence score for true positives results as shown in the figure below:


True positives

Your model correctly predicted license_plate on these images:

Score: 0.96

Score: 0.967

Score: 0.978

Score: 0.986

Score: 0.986

Figure 8.10: True Positives Results

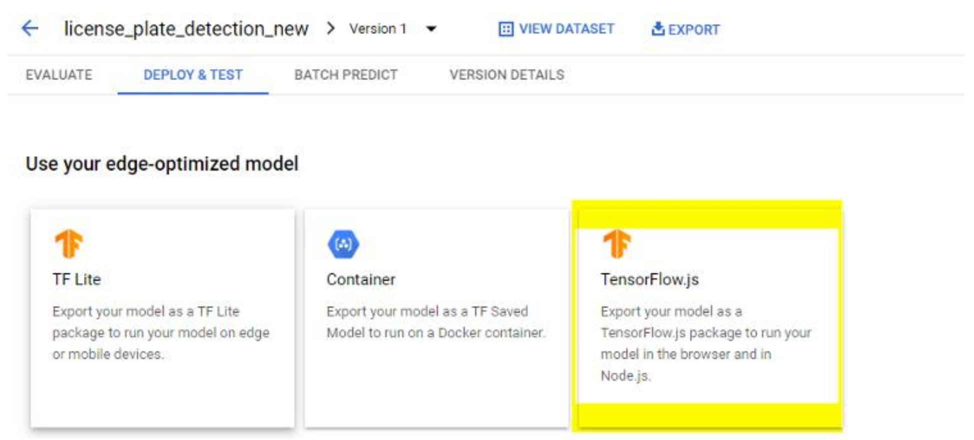
In machine learning models, it is particularly challenging and rare to achieve 100% precision and recall, especially in real-world applications. However, with the adjustments and test results of the improved model, the observed accuracy of 90.9% is satisfactory and dependable for the license plate detection model.

9 Model Deployment

This section involves the steps required to deploy the trained model into Raspberry Pi and test the model with Raspberry Pi to ensure the camera and OCR functionalities are working.

9.1 Exporting Model

After training the model, it allows to deploy as different packages depending on the platform of the project. In the Parkify system, the preferred model to export is TensorFlow.js which can run the model in the browser and Node.js.



Use your edge-optimized model

TF Lite

Export your model as a TF Lite package to run your model on edge or mobile devices.

Container

Export your model as a TF Saved Model to run on a Docker container.

TensorFlow.js

Export your model as a TensorFlow.js package to run your model in the browser and in Node.js.

Figure 9.1: Export Trained Model Options

To implement the license plate detection model, the model will add to the folder with the code. Then we will implement our code by loading the pre-trained object detection model with the directory as shown in the figure below.

```
// Localize the License Plate Model
const model = await tf.automl.loadObjectDetection('http://raspberrypi.local/alpr/model/model.json');
```

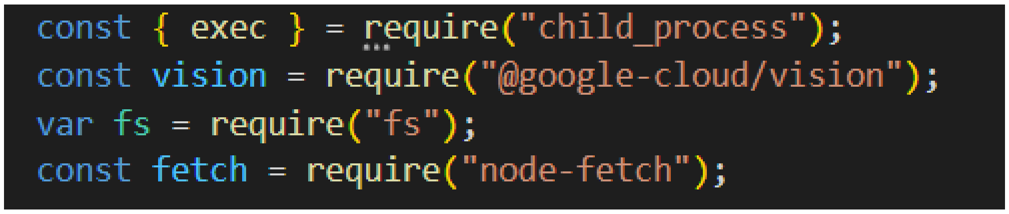
Figure 9.2: Load Object Detection Model Code Snippet

After the model is loaded, the Raspberry Pi will be able to use it to perform license plate detection on an image or video stream.

9.2 License Plate Detection Code Implementation

This section below will show how the code implementation demonstrates capturing an image using a Pi Camera module and running the OCR text detection on the image using Google Cloud Vision API with the Node.js environment.

9.2.1 Libraries



```
const { exec } = require("child_process");
const vision = require("@google-cloud/vision");
var fs = require("fs");
const fetch = require("node-fetch");
```

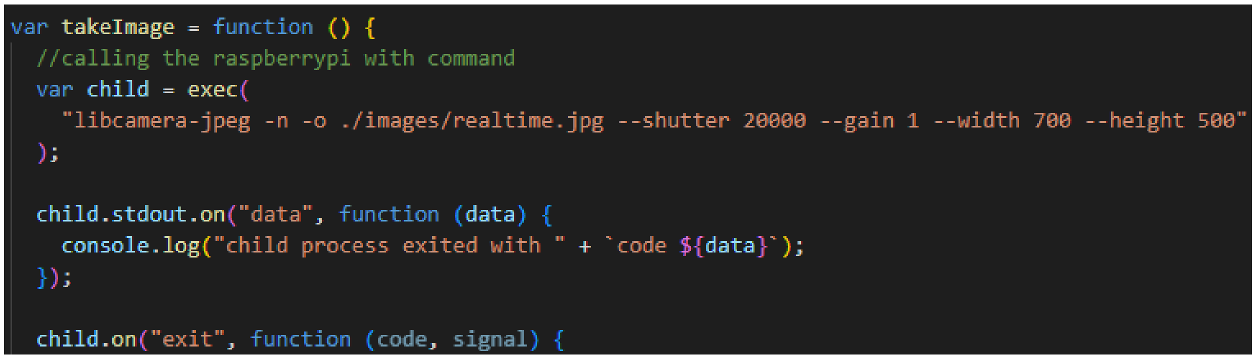
Figure 9.3: Libraries and Packages

The code snippet above shows the necessary libraries that are imported into our code.

Importing the 'exec' from the 'child_process' package allows us to execute shell commands within the Node.js environment.[19]

In addition, the '@google-cloud/vision' is the package that provides access to Google Cloud Vision API and is responsible for the handling of the OCR text detection process while the 'fs' package allows us to read and write files to the HTML documents. Finally, the 'node-fetch' package provides an interface for making HTTP requests.

9.2.2 Image Capturing



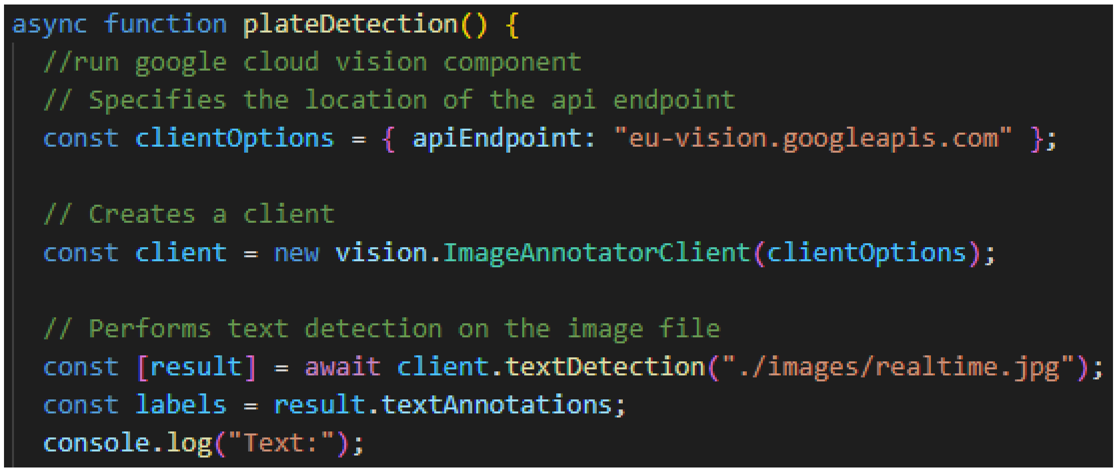
```
var takeImage = function () {
  //calling the raspberrypi with command
  var child = exec(
    "libcamera-jpeg -n -o ./images/realtime.jpg --shutter 20000 --gain 1 --width 700 --height 500"
  );

  child.stdout.on("data", function (data) {
    console.log("child process exited with " + `code ${data}`);
  });

  child.on("exit", function (code, signal) {
```

Figure 9.4: Image Capture Function

The code snippets above show the main function of the script which captures an image using PiCamera by running the executing the command with the camera setting parameters.

9.2.3 Plate Detection

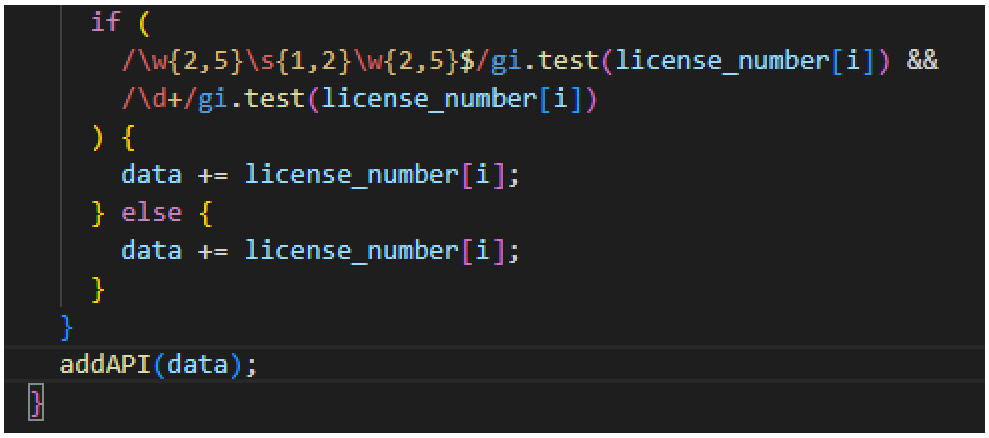
After the image is captured, the image will be passed to the 'plateDetection' function to perform license plate recognition with Google Cloud Vision API. The code snippet of the function is shown below.



```
async function plateDetection() {
  //run google cloud vision component
  // Specifies the location of the api endpoint
  const clientOptions = { apiEndpoint: "eu-vision.googleapis.com" };

  // Creates a client
  const client = new vision.ImageAnnotatorClient(clientOptions);

  // Performs text detection on the image file
  const [result] = await client.textDetection("./images/realtime.jpg");
  const labels = result.textAnnotations;
  console.log("Text:");
```

Figure 9.5: License Plate Detection Function

The 'clientOptions' object will set the location of the API endpoint which is "eu-vision.googleapis.com". The 'textDetection' method 'ImageAnnotatorClient' is used to extract the text detected on the image.



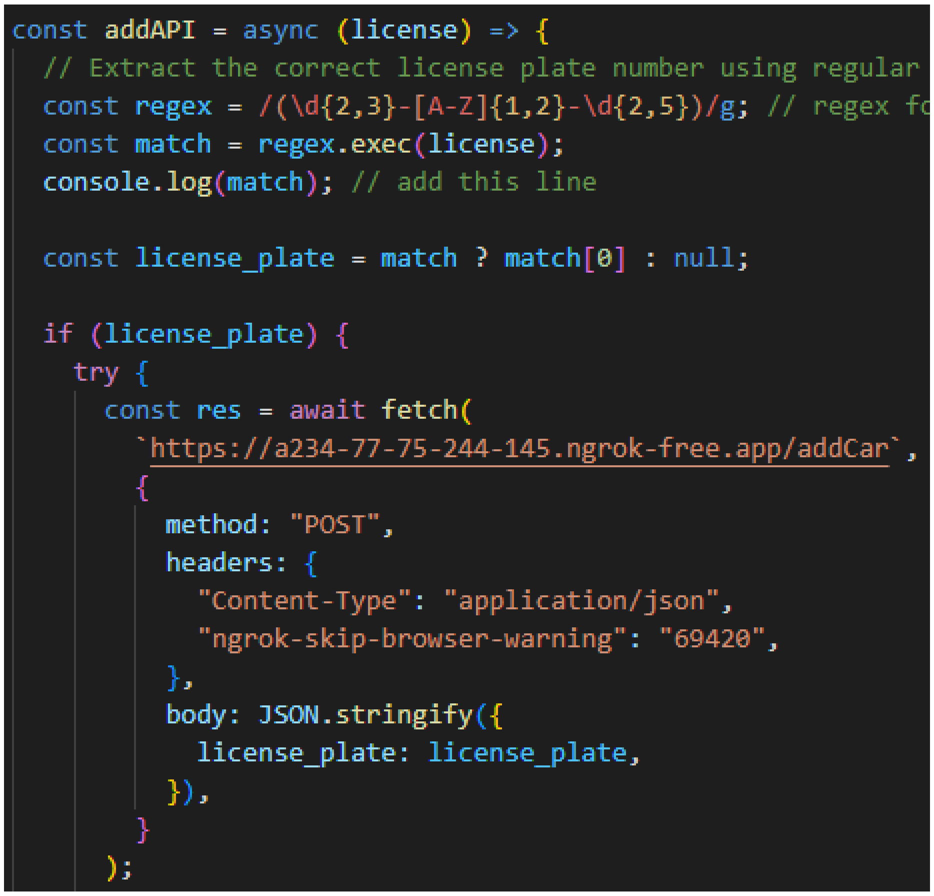
```
      if (
        /\w{2,5}\s{1,2}\w{2,5}$/gi.test(license_number[i]) &&
        /\d+/gi.test(license_number[i])
      ) {
        data += license_number[i];
      } else {
        data += license_number[i];
      }
    }
    addAPI(data);
  }
```

Figure 9.6: Regular Expression to extract license plate characters code snippet

The recognized text will then be looped through to extract the license plate number using regular expressions and call the addAPI function which to POST the request to save in the database.

9.2.4 POST Request

The addAPI function call with the data storing the license plate will send a POST request to the backend server. Before the addAPI function send the POST request it will perform another loop to check and ensure that the license plate number extracted matches the format.



```
const addAPI = async (license) => {
  // Extract the correct license plate number using regular
  const regex = /(\d{2,3}-[A-Z]{1,2}-\d{2,5})/g; // regex fo
  const match = regex.exec(license);
  console.log(match); // add this line

  const license_plate = match ? match[0] : null;

  if (license_plate) {
    try {
      const res = await fetch(
        `https://a234-77-75-244-145.ngrok-free.app/addCar`,
        {
          method: "POST",
          headers: {
            "Content-Type": "application/json",
            "ngrok-skip-browser-warning": "69420",
          },
          body: JSON.stringify({
            license_plate: license_plate,
          }),
        }
      );
```

Figure 9.7: addAPI function code snippet

10 Mobile Application Development

The Parkify mobile application is developed with React Native, Expo which is an open source platform that allows us to build applications with cross-platforms such as Android, and iOS.

10.1 React Native

React Native is an open source JavaScript framework that is used to develop mobile applications which allow Parkify system to develop mobile applications with cross-platform such as Android or IOS. The cross-platform capabilities will also enhance user experience and accessibility.

10.2 React Navigation

React Navigation is a library which widely used in React Native applications for navigating and routing between the screens in React Native mobile applications. It is used for managing, passing data, and switching between screens.

10.3 Expo

Expo is also an open source platform and framework that allows us to build and deploy mobile applications using React Native. Thus, Expo provides services and tools to simplify the development process, making it more efficient which allows us to build and debug the mobile application interface at the same time. For the Parkify system, the application is implemented with the expo notifications service to push notifications to users.

10.4 Stripe

Stripe is a payment processing platform that provides APIs to process payment requests and is integrated with websites and applications which makes online payments easier and faster. It also could allow developers to use mock cards to test the payment process and review it in the Stripe dashboard. Additionally, it also supports various payment methods and banks such as Google Pay, Apple Pay and American Express Card instead of just only Visa debit cards.

The Parkify system is utilises Stripe to process payment for parking fees which are calculated from the database. Stripe was chosen as the third-party payment processor due to their compatibility and the payment details is not processed or saved in the Parkify database. The Parkify's system database sends a request to initiate a payment process through sending a POST payment intent, and Stripe will return a client_secret key which to process the payment. Hence, the actual payment process is handled by the Stripe platform.

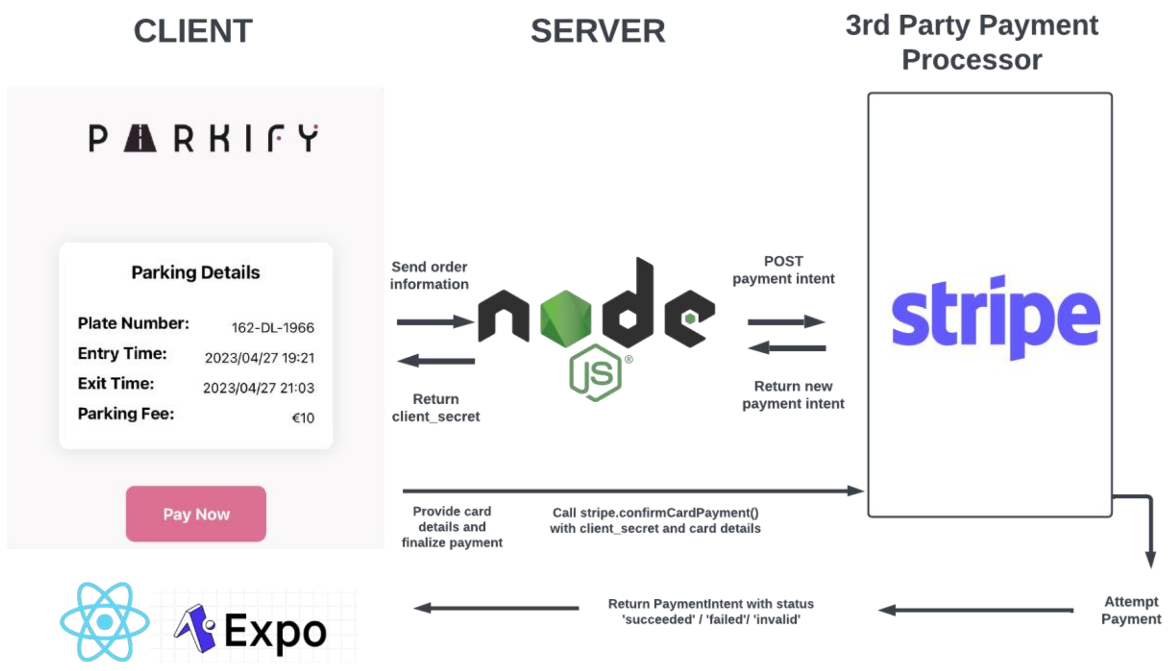
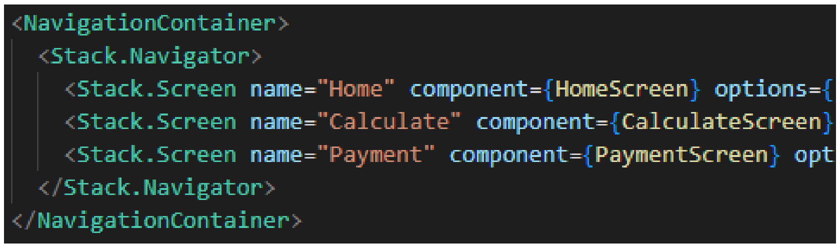

CLIENT
SERVER
3rd Party Payment Processor
PARKIFY
Parking Details
Plate Number: 162-DL-1966
Entry Time: 2023/04/27 19:21
Exit Time: 2023/04/27 21:03
Parking Fee: €10
Pay Now
Send order information
Return client_secret
POST payment intent
Return new payment intent
stripe
Provide card details and finalize payment
Call stripe.confirmCardPayment() with client_secret and card details
Attempt Payment
Return PaymentIntent with status 'succeeded' / 'failed'/ 'invalid'
Expo

Figure 10.1: Stripe payment process overview

10.5 Code Overview

The Parkify mobile application is using stack navigator to pass data and navigate between three screens as shown in the code snippet below.

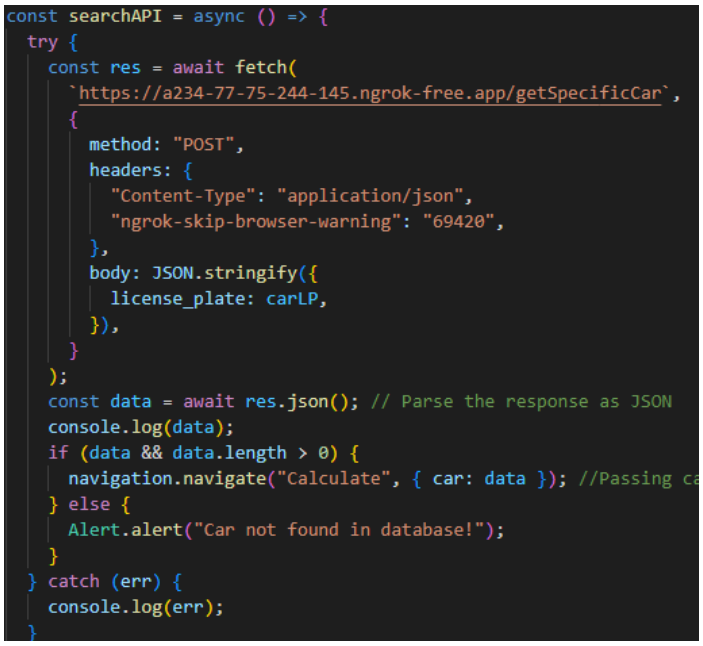


```
<NavigationContainer>
  <Stack.Navigator>
    <Stack.Screen name="Home" component={HomeScreen} options={
    <Stack.Screen name="Calculate" component={CalculateScreen}
    <Stack.Screen name="Payment" component={PaymentScreen} opt
  </Stack.Navigator>
</NavigationContainer>
```

Figure 10.2: Screen Components

10.5.1 Home Screen

The Home screen is the first screen component that the user will show up when starting the application and the user will be able to search for the license plate and will navigate to the CalculateScreen component by passing the values of the license plate.



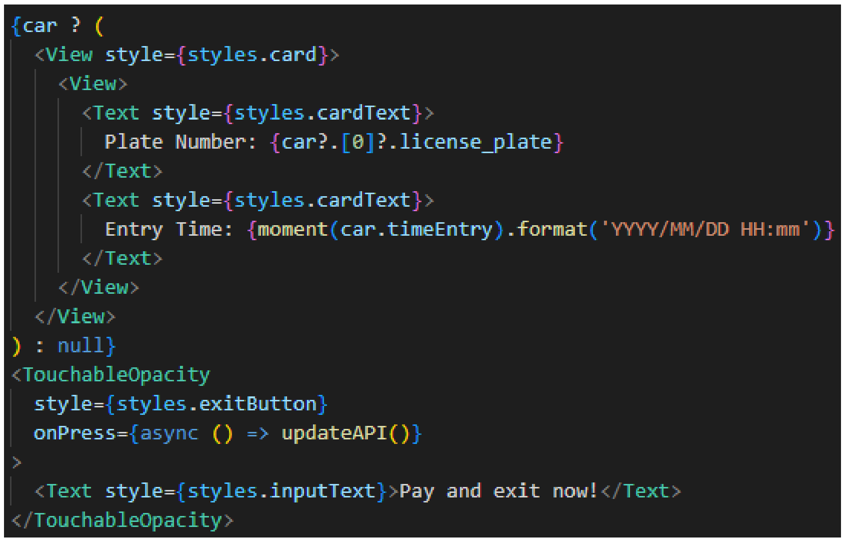
```
const searchAPI = async () => {
  try {
    const res = await fetch(
      `https://a234-77-75-244-145.ngrok-free.app/getSpecificCar`,
      {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
          "ngrok-skip-browser-warning": "69420",
        },
        body: JSON.stringify({
          license_plate: carLP,
        }),
      }
    );
    const data = await res.json(); // Parse the response as JSON
    console.log(data);
    if (data && data.length > 0) {
      navigation.navigate("Calculate", { car: data }); //Passing ca
    } else {
      Alert.alert("Car not found in database!");
    }
  } catch (err) {
    console.log(err);
  }
```

Figure 10.3: searchAPI function code snippet

This is the searchAPI function which after the user enters the license plate and clicks on the 'Search' button will call this searchAPI function to send a POST request and find the according license plate that is stored in the database. Also, the license plate value will pass to the Calculate screen accordingly.

10.5.2 Calculate Screen

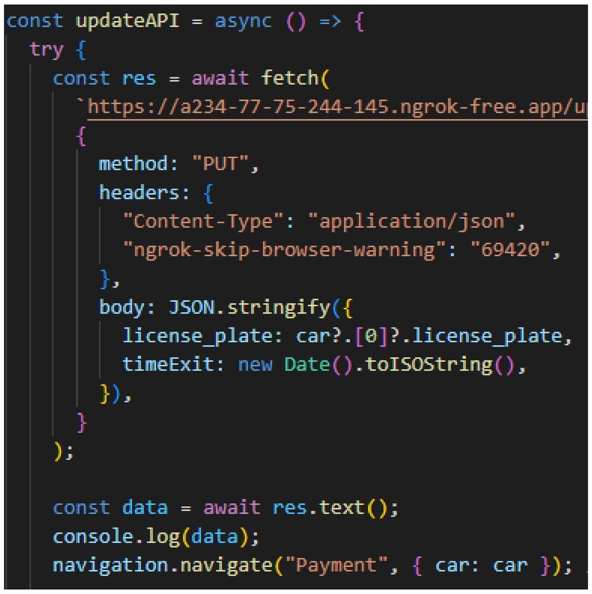
In Calculate Screen of the Parkify application, users will be able to view their license plate with the entry time to the parking space. The time format here uses the moment library to show the time format as ‘YYYY/MM/DD HH:mm’ instead of the ISO format which is difficult to read.



```
{car ? (
  <View style={styles.card}>
    <View>
      <Text style={styles.cardText}>
        Plate Number: {car?.[0]?.license_plate}
      </Text>
      <Text style={styles.cardText}>
        Entry Time: {moment(car.timeEntry).format('YYYY/MM/DD HH:mm')}
      </Text>
    </View>
  </View>
) : null}
<TouchableOpacity
  style={styles.exitButton}
  onPress={async () => updateAPI()}
>
  <Text style={styles.inputText}>Pay and exit now!</Text>
</TouchableOpacity>
```

Figure 10.4: Calculate screen code snippet

In the meantime, when the user clicks on the ‘Pay and exit now!’ button, it will call the updateAPI function to send a PUT request which will update the exit time field, perform the parking fee calculation on the backend database and navigate to the Payment Screen.

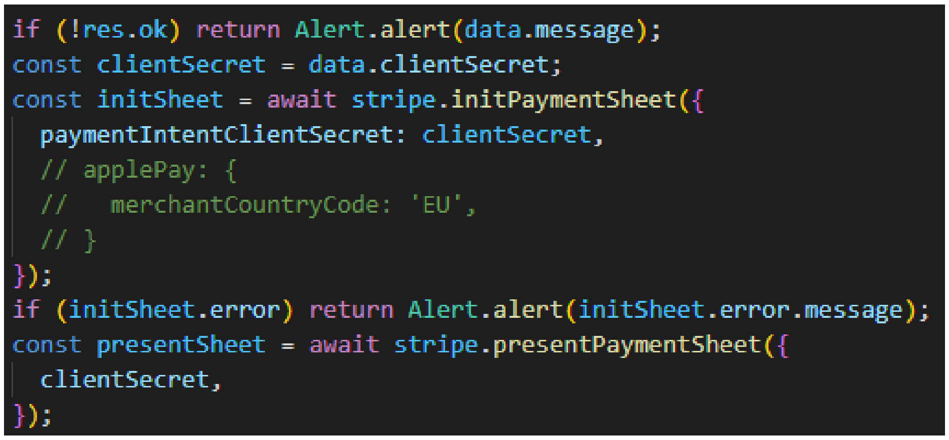


```
const updateAPI = async () => {
  try {
    const res = await fetch(
      `https://a234-77-75-244-145.ngrok-free.app/u
      {
        method: "PUT",
        headers: {
          "Content-Type": "application/json",
          "ngrok-skip-browser-warning": "69420",
        },
        body: JSON.stringify({
          license_plate: car?.[0]?.license_plate,
          timeExit: new Date().toISOString(),
        }),
      }
    );

    const data = await res.text();
    console.log(data);
    navigation.navigate("Payment", { car: car });
```

Figure 10.5: updateAPI function code snippet

10.5.3 Payment Screen

The payment screen will display the updated exit time of the car based on the license plate and the corresponding parking fee. When the user clicks on the 'Pay Now' button, it will call the payAPI function.



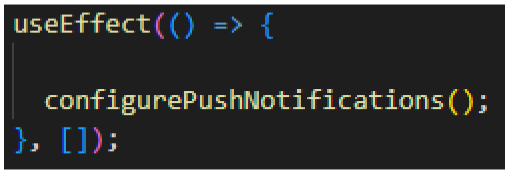
```
if (!res.ok) return Alert.alert(data.message);
const clientSecret = data.clientSecret;
const initSheet = await stripe.initPaymentSheet({
  paymentIntentClientSecret: clientSecret,
  // applePay: {
  //   merchantCountryCode: 'EU',
  // }
});
if (initSheet.error) return Alert.alert(initSheet.error.message);
const presentSheet = await stripe.presentPaymentSheet({
  clientSecret,
});
```

Figure 10.6: payAPI function code snippet

When the payAPI function is called, it will get the clientSecret key from the backend of the database which then will send to Stripe to initiate the payment sheet and present the payment sheet with the corresponding parking fee.

10.5.4 Notifications

The Parkify application will configure the push notifications by using the effect hooks as shown below.

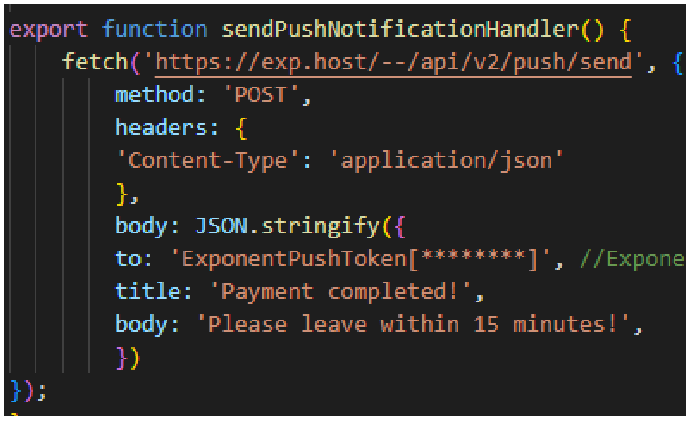


```
useEffect(() => {

  configurePushNotifications();
}, []);
```

Figure 10.7: useEffect Hook code snippet

The configurePushNotifications function is used to request permission from the user device to grant permission for pushing notifications.

When the payment is successful, the sendPushNotificationHandler function will call and push a notification to the user as shown below:



```
export function sendPushNotificationHandler() {
    fetch('https://exp.host/--/api/v2/push/send', {
        method: 'POST',
        headers: {
        'Content-Type': 'application/json'
        },
        body: JSON.stringify({
        to: 'ExponentPushToken[********]', //Expone
        title: 'Payment completed!',
        body: 'Please leave within 15 minutes!',
        })
});
```

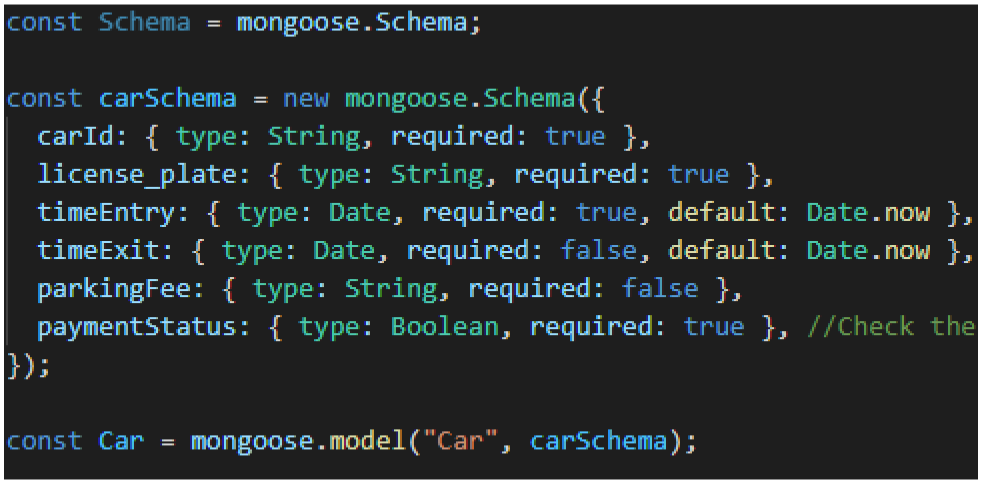
Figure 10.8: sendPushNotificationHandler function code snippet

11 Database Design and Implementation

The Parkify backend database is developed with MongoDB to perform CREATE, READ and UPDATE license plate details functions. The database includes ‘addCar’, ‘getSpecificCar’, ‘updateSpecificCar’, ‘pay’ and ‘updatePaymentStatus’ which are the 5 main API endpoints that are utilized in our smart parking system, Parkify database.

11.1 Mongoose Schema

The ‘carSchema’ is a Mongoose Schema which logs the car id, license plate, entry time, exit time, parking fee and payment status of the car corresponding with the license plate.



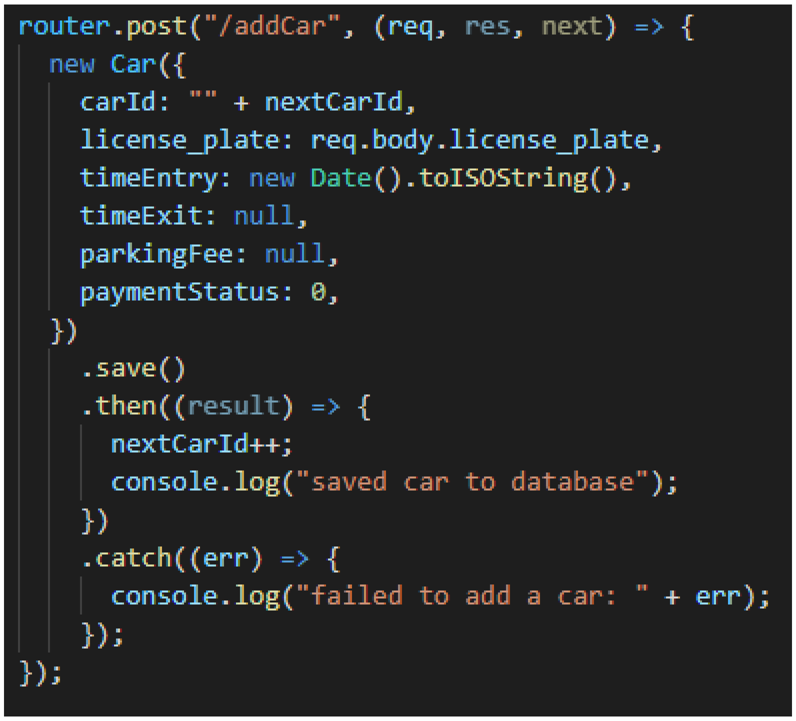
```
const Schema = mongoose.Schema;

const carSchema = new mongoose.Schema({
  carId: { type: String, required: true },
  license_plate: { type: String, required: true },
  timeEntry: { type: Date, required: true, default: Date.now },
  timeExit: { type: Date, required: false, default: Date.now },
  parkingFee: { type: String, required: false },
  paymentStatus: { type: Boolean, required: true }, //Check the
});

const Car = mongoose.model("Car", carSchema);
```

Figure 11.1: Car Schema

The car id which purposed to log and check how many cars will be in the parking spaces. When a car’s license plate is detected, it will require the car id, license plate, time entry, time exit and the payment status of the car.

11.2 Storing data into database

The endpoint ‘addCar’ adds a new car to the database with the provided license plate and sets the time entry to the current date and time. The payment status will also be initialised to ‘0’ which is a flag in Boolean type to check whether the parking fee is paid or not. The code snippet for storing data in the database is shown below:

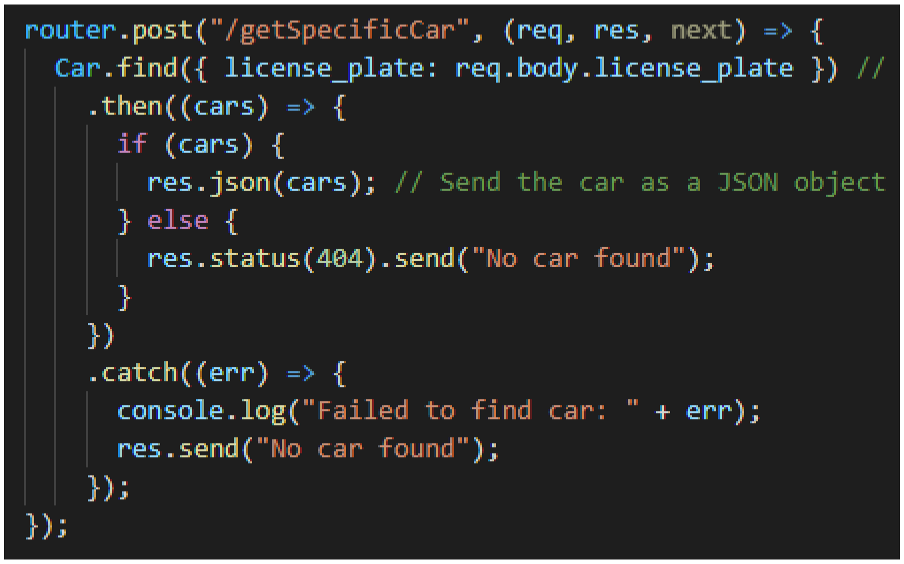


```
router.post("/addCar", (req, res, next) => {
  new Car({
    carId: "" + nextCarId,
    license_plate: req.body.license_plate,
    timeEntry: new Date().toISOString(),
    timeExit: null,
    parkingFee: null,
    paymentStatus: 0,
  })
    .save()
    .then((result) => {
      nextCarId++;
      console.log("saved car to database");
    })
    .catch((err) => {
      console.log("failed to add a car: " + err);
    });
});
```

Figure 11.2: addCar endpoint code snippet

11.3 Retrieving data from database

The following endpoint 'getSpecificCar' retrieves a specific car from the MongoDB database using the license plate that the user entered in the front end and returns the car details.

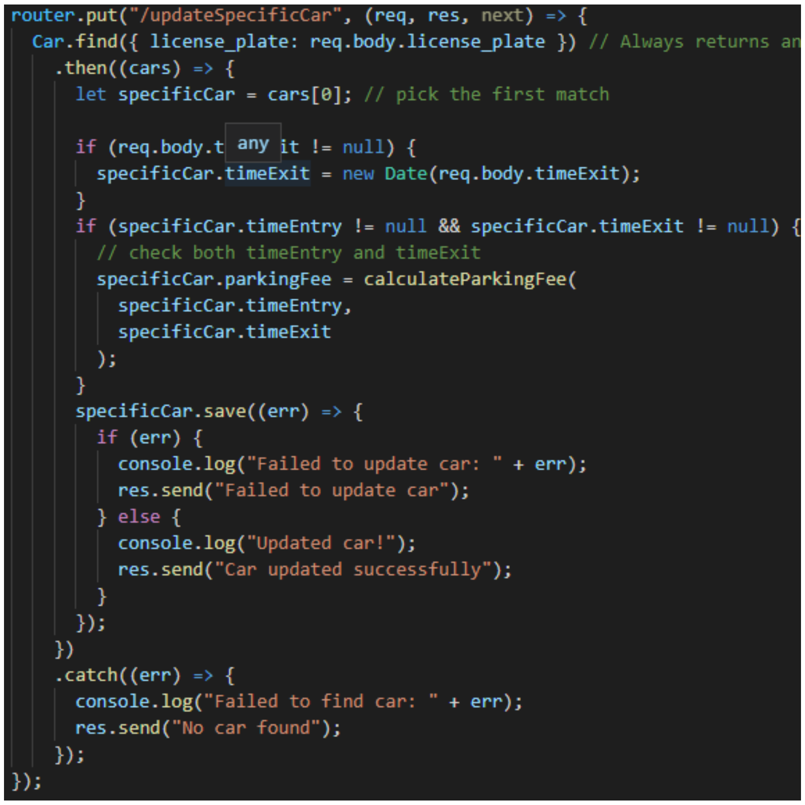


```
router.post("/getSpecificCar", (req, res, next) => {
  Car.find({ license_plate: req.body.license_plate }) //
    .then((cars) => {
      if (cars) {
        res.json(cars); // Send the car as a JSON object
      } else {
        res.status(404).send("No car found");
      }
    })
    .catch((err) => {
      console.log("Failed to find car: " + err);
      res.send("No car found");
    });
});
```

Figure 11.3: getSpecificCar endpoint code snippet

11.4 Updating database

Furthermore, the ‘updateSpecificCar’ endpoint updates the car details such as the exit time and parking fee based on the license plate provided. There is an if statement to check if the exit time and the entry time of the car are available, it will then calculate the parking fee by calling the calculateParkingFee function with the values.

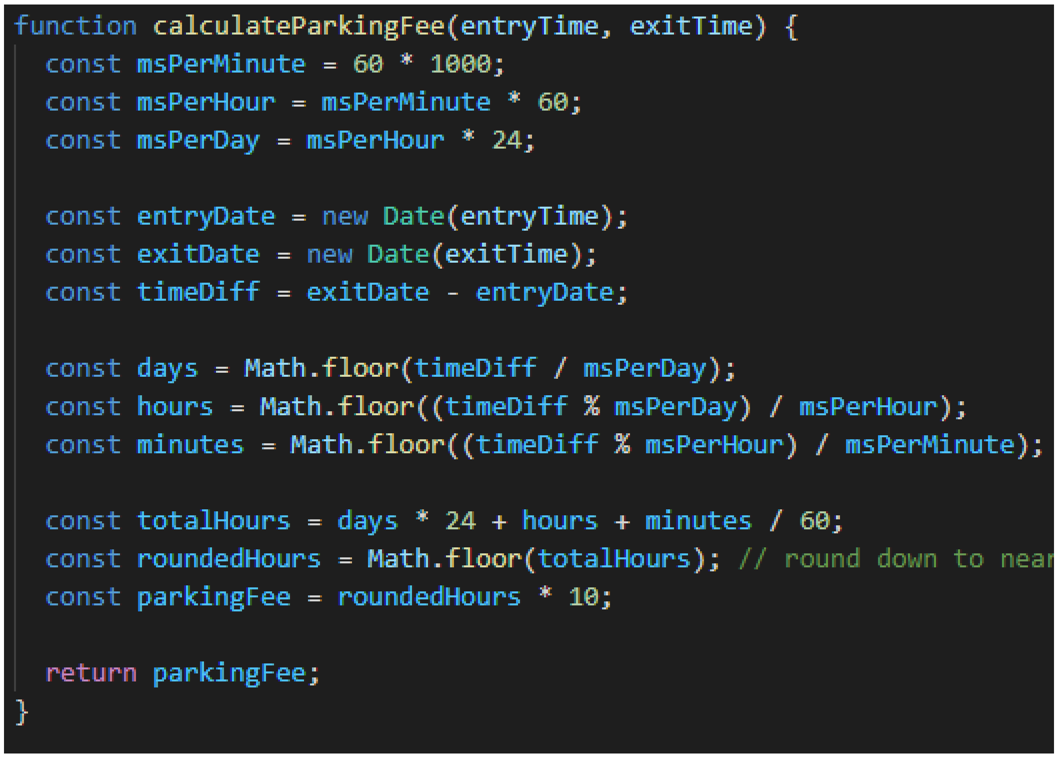


```
router.put("/updateSpecificCar", (req, res, next) => {
  Car.find({ license_plate: req.body.license_plate }) // Always returns an
    .then((cars) => {
      let specificCar = cars[0]; // pick the first match

      if (req.body.t any it != null) {
        specificCar.timeExit = new Date(req.body.timeExit);
      }
      if (specificCar.timeEntry != null && specificCar.timeExit != null) {
        // check both timeEntry and timeExit
        specificCar.parkingFee = calculateParkingFee(
          specificCar.timeEntry,
          specificCar.timeExit
        );
      }
      specificCar.save((err) => {
        if (err) {
          console.log("Failed to update car: " + err);
          res.send("Failed to update car");
        } else {
          console.log("Updated car!");
          res.send("Car updated successfully");
        }
      });
    })
    .catch((err) => {
      console.log("Failed to find car: " + err);
      res.send("No car found");
    });
});
```

Figure 11.4: updateSpecificCar endpoint code snippet

11.4.1 Parking Fee Calculation

The ‘calculateParkingFee’ function calculates the parking fee based on the entry and exit time parameters by converting the time difference into hours and rounding down to the nearest hours by using Math.floor function. The parking fee is calculated with roundedHours and the parking fee rate which is 10 in the example below.

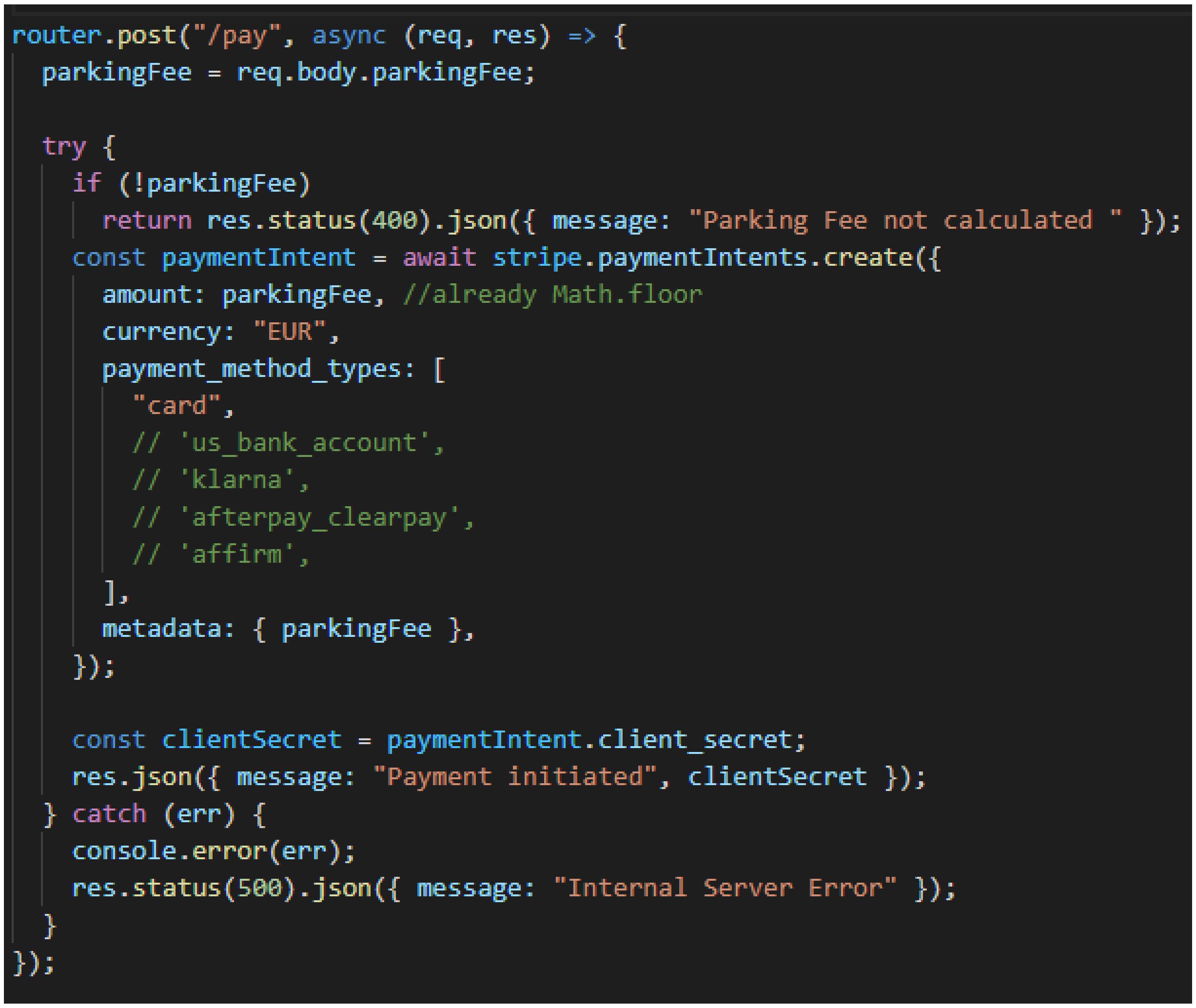


```javascript
function calculateParkingFee(entryTime, exitTime) {
  const msPerMinute = 60 * 1000;
  const msPerHour = msPerMinute * 60;
  const msPerDay = msPerHour * 24;

  const entryDate = new Date(entryTime);
  const exitDate = new Date(exitTime);
  const timeDiff = exitDate - entryDate;

  const days = Math.floor(timeDiff / msPerDay);
  const hours = Math.floor((timeDiff % msPerDay) / msPerHour);
  const minutes = Math.floor((timeDiff % msPerHour) / msPerMinute);

  const totalHours = days * 24 + hours + minutes / 60;
  const roundedHours = Math.floor(totalHours); // round down to near
  const parkingFee = roundedHours * 10;

  return parkingFee;
}
```

Figure 11.2: Parking Fee Calculation Code Snippet

11.5 Payment

The ‘pay’ endpoint initiates the payment request using the Stripe API for the provided parking fee. Integrating Stripe API also allows payment method types such as Klarna, Afterpay, American Express et cetera, which brings users a better experience.



```
router.post("/pay", async (req, res) => {
  parkingFee = req.body.parkingFee;

  try {
    if (!parkingFee)
      return res.status(400).json({ message: "Parking Fee not calculated " });
    const paymentIntent = await stripe.paymentIntents.create({
      amount: parkingFee, //already Math.floor
      currency: "EUR",
      payment_method_types: [
        "card",
        // 'us_bank_account',
        // 'klarna',
        // 'afterpay_clearpay',
        // 'affirm',
      ],
      metadata: { parkingFee },
    });

    const clientSecret = paymentIntent.client_secret;
    res.json({ message: "Payment initiated", clientSecret });
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: "Internal Server Error" });
  }
});
```

Figure 11.3: Payment endpoint code snippet

11.5.1 Payment Review

The Stripe company also provides a dashboard on the website, which the administrator could be able to review payments made by the users through the Stripe payment gateway.

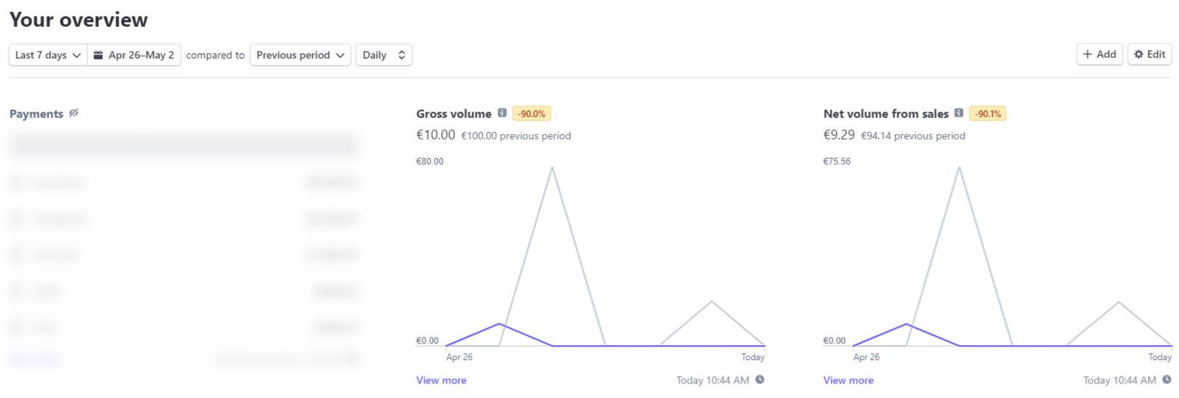
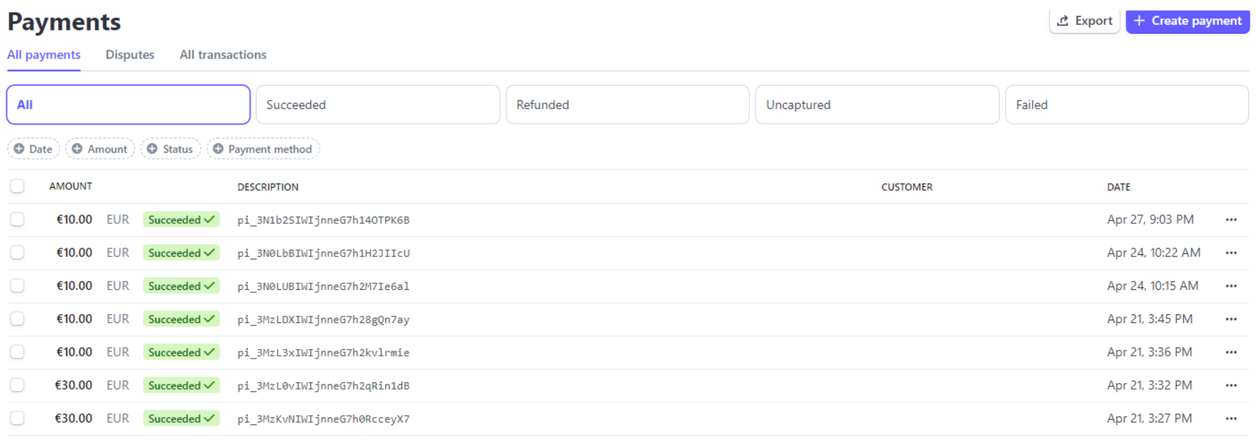


Figure 11.4: Stripe Dashboard Overview



AMOUNT			DESCRIPTION	CUSTOMER	DATE
€10.00	EUR	Succeeded ✓	pi_3N1b2SIWIjnneG7h14OTPK6B		Apr 27, 9:03 PM
€10.00	EUR	Succeeded ✓	pi_3N0LbBIWIjnneG7h1H2JIIcU		Apr 24, 10:22 AM
€10.00	EUR	Succeeded ✓	pi_3N0LUBIWIjnneG7h2M7Ie6al		Apr 24, 10:15 AM
€10.00	EUR	Succeeded ✓	pi_3MzLDXIWIjnneG7h28gQn7ay		Apr 21, 3:45 PM
€10.00	EUR	Succeeded ✓	pi_3MzL3xIWIjnneG7h2kvlrmie		Apr 21, 3:36 PM
€30.00	EUR	Succeeded ✓	pi_3MzL0vIWIjnneG7h2qRin1dB		Apr 21, 3:32 PM
€30.00	EUR	Succeeded ✓	pi_3MzKvNIWIjnneG7h0RcceyX7		Apr 21, 3:27 PM

Figure 11.5: Stripe Payment History Review

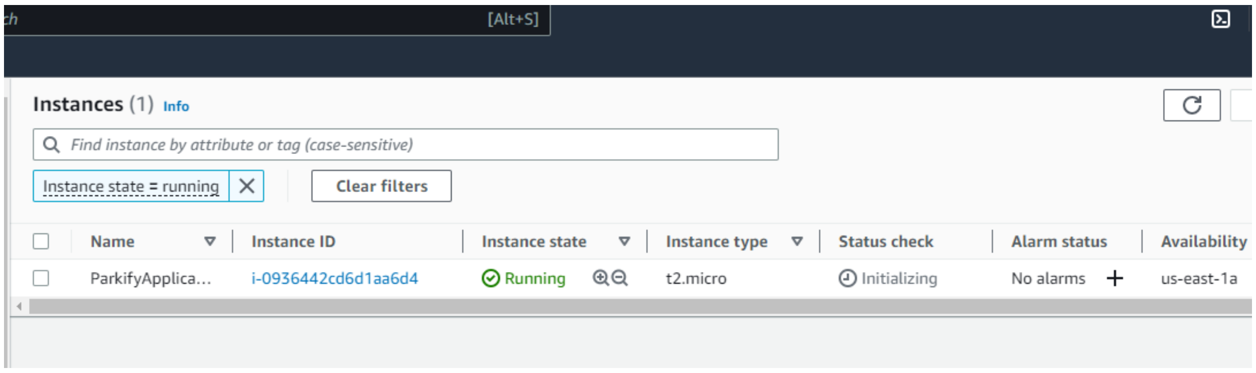
In the Parkify application, the payment demonstration is using mock cards which provided testing cards to simulate different scenarios such as card declines, fraud or authentication with 3D secure and PINs, the list of testing mock cards can be found in the stripe documentation.[20]

12 Cloud Deployment

The Parkify system also has integrated cloud services to deploy mobile applications and a backend database to manage the data. This section below will discuss the deployment of Amazon Web Services (AWS) and MongoDB Atlas in the Parkify system.

12.1 Amazon Web Services (AWS)

To deploy and integrate the Parkify system on the cloud, a new AWS EC2 instance that hosts both the front-end and back-end code is created as shown below.



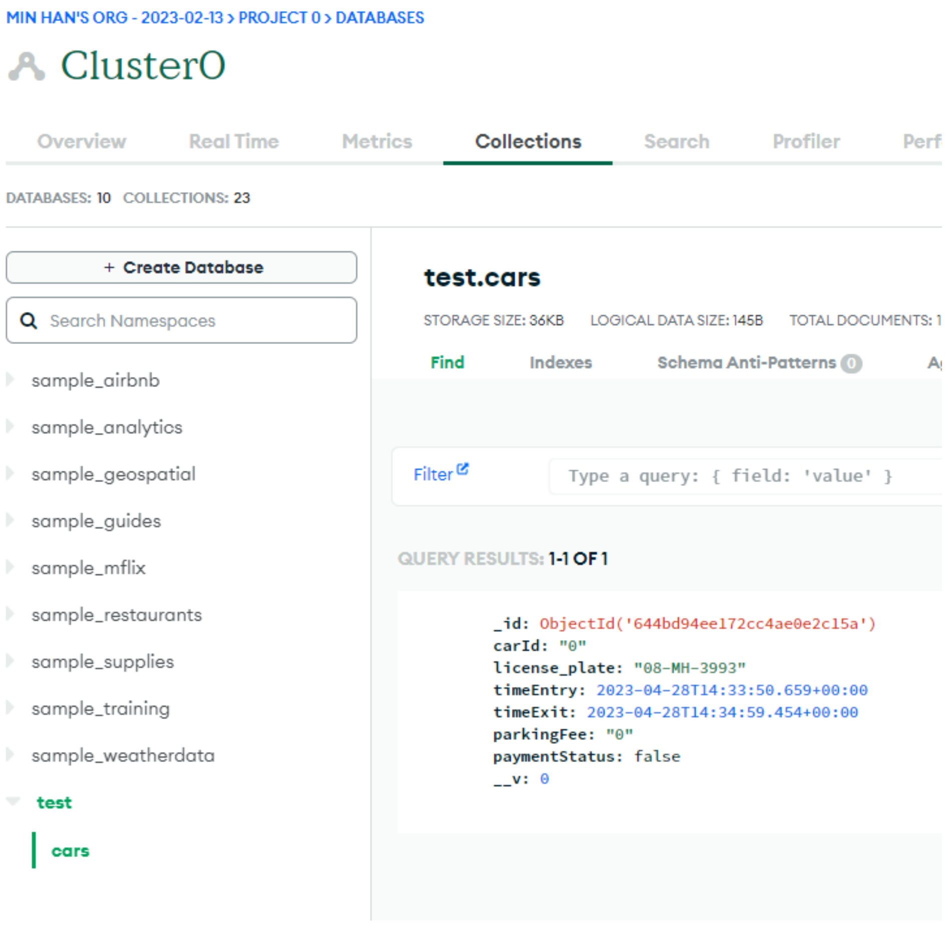
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability
ParkifyApplica...	i-0936442cd6d1aa6d4	Running	t2.micro	Initializing	No alarms	us-east-1a

Figure 12.1: Amazon Web Services instance overview

AWS also allows us to allocate an Elastic IP address to the instance, which provides a consistent IP address that helps create a more accessible system for the Parkify mobile application and backend database from the internet.

12.2 MongoDB Atlas

In the final stage of the Parkify system have integrated MongoDB Atlas to view and manage the database from the cloud. MongoDB Atlas provides an easy and convenient platform for managing and viewing the Parkify system data. The Parkify system data stores and shown in the MongoDB Atlas platform as shown below:



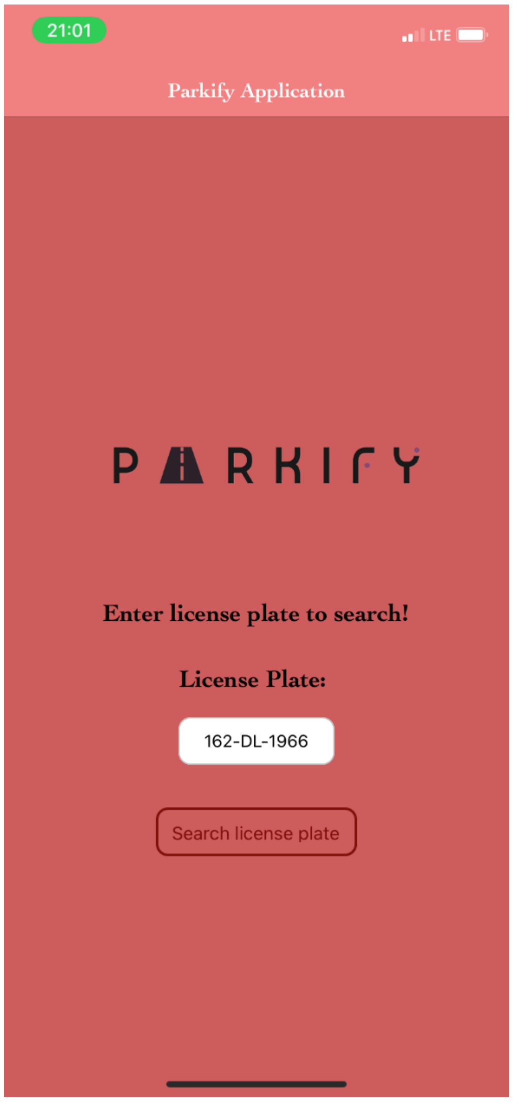
```
_id: ObjectId('644bd94ee172cc4ae0e2c15a')
carId: "0"
license_plate: "08-MH-3993"
timeEntry: 2023-04-28T14:33:50.659+00:00
timeExit: 2023-04-28T14:34:59.454+00:00
parkingFee: "0"
paymentStatus: false
__v: 0
```

Figure 12.2: MongoDB Atlas database overview

Also, MongoDB Atlas provides other features to ensure the data's safety and reliability.

13 Mobile Application Overview

The Parkify mobile application demo overview is shown below:



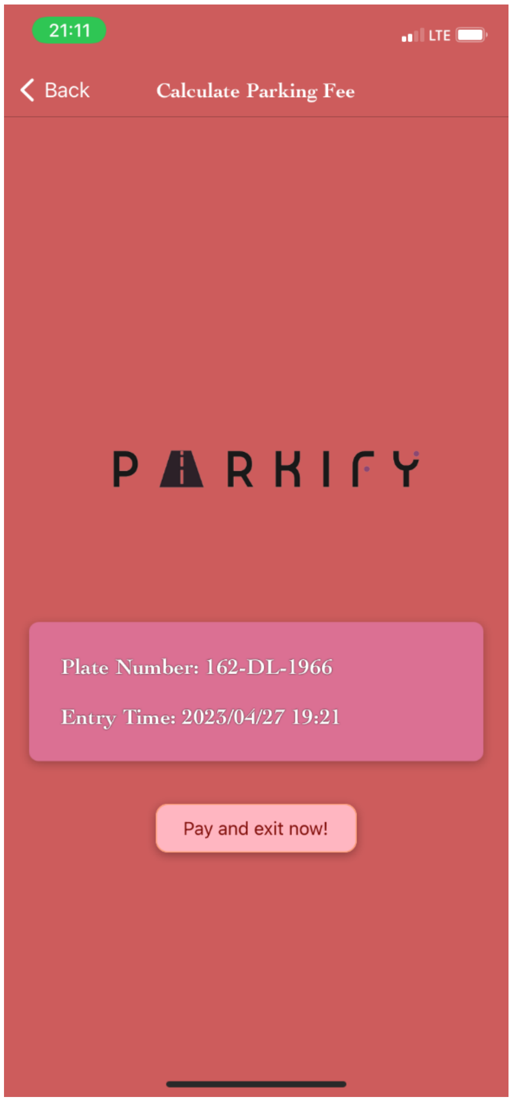
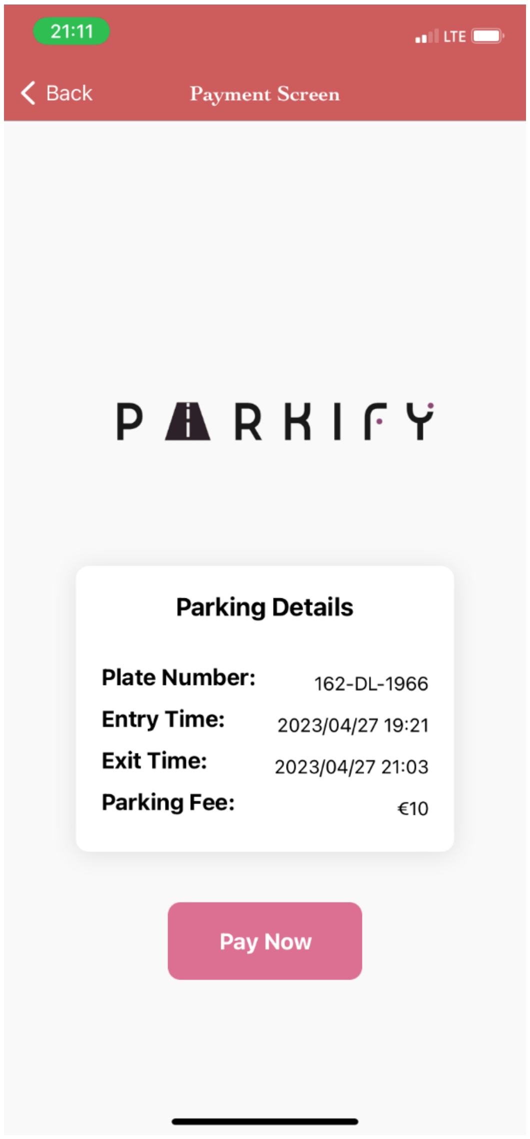


Plate Number: 162-DL-1966

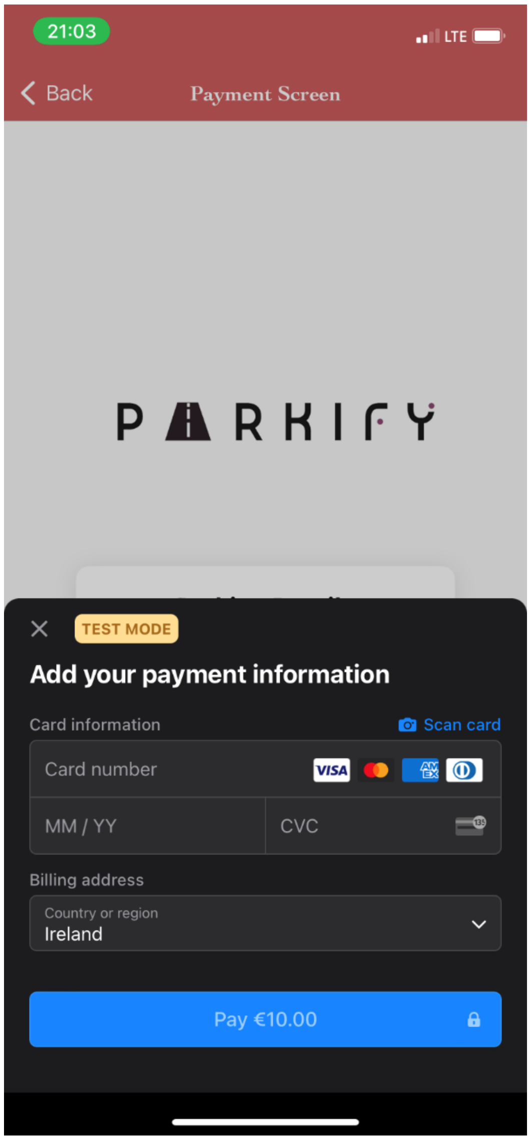
Entry Time: 2023/04/27 19:21

Figure 13.1 & Figure 13.2: Home Screen and Calculate Screen Overview



Parking Details

Plate Number:	162-DL-1966
Entry Time:	2023/04/27 19:21
Exit Time:	2023/04/27 21:03
Parking Fee:	€10



Add your payment information

Card information

Billing address

Country or region: Ireland

Figure 13.3 & Figure 13.4: Payment Screen and Payment Modal Overview

14 Ethics

As technology grows rapidly and plays an important role in our daily lives, it is essential to approach the development and implementation of ethics. In the Parkify system, ethical considerations such as privacy concerns about payment details will be discussed in this section.

14.1 Privacy Data Concerns

This section will be explaining how the user data was collected, processed, and stored.

In the Parkify system, the license plate numbers, and driver information are very important to ensure that user data is collected and stored securely. The system should comply Personal Data Privacy Act(PDPA) and General Data Protection Regulation (GDPR) to protect users' personal information.

Additionally, Stripe is a third-party payment processor, and the card details won't save on the Parkify back-end server side. In Parkify server just passes the client_secret and the payment process are securely processed on the Stripe server.

15 Further Development

The Parkify system has a lot of future opportunities for further development including improving the accuracy of ALPR technology as well as expanding the features of the mobile application to add more parking-related conveniences for users. Other than that, the Parkify system could also implement with a private car parking space as a security feature.

16 Conclusion

In conclusion, Parkify has successfully developed a working prototype of a smart parking system which integrated Raspberry Pi with Automatic License Plate Recognition(ALPR) technology and detects license plates with PiCamera. The mobile application built with React Native that integrated with the Parkify system also allows users to access their parking details with a convenient and user-friendly platform. The MongoDB Atlas database was also successfully set up to store the data corresponding with license plates and the calculations of the parking fees in the backend are precise.

The Parkify system also successfully integrated and demonstrated the Stripe payment processor which will handle the user payment process securely. Furthermore, the Parkify system was also integrated with the Amazon Web Services(AWS) EC2 instance server, which allows us to host our mobile application and database on the cloud server instead of the local server.

The demonstration of Parkify successfully demonstrates the ALPR technology that could implement in the real world, providing a more convenient and better user experience as a parking solution.

17 References

[1] 'Raspberry Pi 4 Computer Model B', 2020, [Online]. Available: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf

[2] 'ESP32 Series Datasheet Including', 2023, [Online]. Available: https://www.espressif.com/en/support/download/documents.

[3] 'The PiCamera2 Module', 2022, [Online]. Available: https://datasheets.raspberrypi.com/camera/picamera2-manual.pdf

[4] 'ESP32-CAM Development Board', [Online]. Available: https://media.digikey.com/pdf/Data%20Sheets/DFRobot%20PDFs/DFR0602_Web.pdf

[5] 'MongoDB vs MySQL: Know The Difference - InterviewBit'. https://www.interviewbit.com/blog/mongodb-vs-mysql/

[6] 'What is Supervised Learning? | IBM'. https://www.ibm.com/topics/supervised-learning

[7] 'Introduction to TensorFlow for Artificial Intelligence, Machine Learning, and Deep Learning Course (DeepLearning.AI) | Coursera'. https://www.coursera.org/learn/introduction-tensorflow

[8] M. Lynch, 'Kernels Lab'. https://vlegalwaymayo.atu.ie/pluginfile.php/860828/mod_resource/content/4/KernelsLab.py

[9] M. Lynch, 'Image Processing'.

[10] 'Understanding Convolutions and Pooling in Neural Networks: a simple explanation | by Miguel Fernández Zafra | Towards Data Science'. https://towardsdatascience.com/understanding-convolutions-and-pooling-in-neural-networks-a-simple-explanation-885a2d78f211

[11] ‘What is the Difference Between Test and Validation Datasets? - MachineLearningMastery.com’. https://machinelearningmastery.com/difference-test-validation-datasets/

[12] ‘YOLO Algorithm for Object Detection Explained [+Examples]’. https://www.v7labs.com/blog/yolo-object-detection

[13] ‘Residual Block Image Source’. https://www.guidetomlandai.com/assets/img/computer_vision/grid.png

[14] ‘YOLO Object Detection Explained: A Beginner’s Guide | DataCamp’. https://www.datacamp.com/blog/yolo-object-detection-explained

[15] ‘YOLO Combination Technique’. https://www.guidetomlandai.com/assets/img/computer_vision/YOLO.PNG

[16] ‘YOLO Example Image Source’. https://miro.medium.com/v2/resize:fit:720/0*xfXdebLeaMXt3Vct

[17] ‘Vertex AI | Google Cloud’. https://cloud.google.com/vertex-ai

[18] ‘About Train, Validation and Test Sets in Machine Learning | by Tarang Shah | Towards Data Science’. https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7

[19] ‘Child process | Node.js v20.0.0 Documentation’. https://nodejs.org/api/child_process.html#child-process

[20] ‘Test cards | Stripe Documentation’. https://stripe.com/docs/testing