

MINOR PROJECT

*Submitted in partial fulfilment of the
requirements for the award of degree
Of*

Bachelor of Technology

In

Computer Science & Engineering

By:

Japman Singh (49/CSE1/2017)

Japneet Singh (50/CSE1/2017)

Jaskaran Singh Kawatra (53/CSE1/2017)

Jaskirat Singh (54/CSE1/2017)



**Department of Computer Science & Engineering
Guru Tegh Bahadur Institute of Technology**

**Guru Gobind Singh Indraprastha University
Dwarka, New Delhi
Year 2017-2021**

SIMPLY PARKING

Duration

Aug, 2020 – Jan, 2021

By:

Japman Singh (49/CSE1/2017)

Japneet Singh (50/CSE1/2017)

Jaskaran Singh Kawatra (53/CSE1/2017)

Jaskirat Singh (54/CSE1/2017)

At

Guru Tegh Bahadur Institute of Technology

G-8 Area, Press Colony, Rajouri Garden, New Delhi, Delhi 110064

DECLARATION

We hereby declare that all the work presented in this Minor Project Report for the partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in **Computer Science & Engineering**, Guru Tegh Bahadur Institute of Technology, affiliated to Guru Gobind Singh Indraprastha University Delhi is an authentic record of our own work.

Date:

Japman Singh (49/CSE1/2017), Japneet Singh (50/CSE1/2017), Jaskaran Singh Kawatra (53/CSE1/2017), Jaskirat Singh (54/CSE1/2017)



ACKNOWLEDGEMENT

We have taken great efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. We would like to extend our sincere thanks to all of them.

With a deep sense of gratitude, we express our sincere thanks our mentor Ms. Basanti Pal Nandi, Senior faculty, CSE Department, for her active support and continuous guidance without which it would have been very difficult for us to complete this project. We would also like to take this opportunity to express our gratitude towards our HOD of CSE Department Dr. Ashish Bhardwaj for taking keen interest in our project and giving valuable suggestions and feedback while also helping us directly to complete this project.

Japman Singh (49/CSE1/2017)

Date:

Japneet Singh (50/CSE1/2017)

Jaskaran Singh Kawatra (53/CSE1/2017)

Jaskirat Singh (54/CSE1/2017)



ABSTRACT

Over the years, technology has revolutionized our world. Technology has created amazing tools and resources, like Airbnb, Twitter, Uber and countless other new technologies that put the world at our fingertips and help us coordinate activities at previously unimaginable speeds and scale.

With rapid advancements in technology, most daily processes have become automated. Even though rapid advancements in technology have solved many problems, the problem of parking is still a major concern. This is a very tedious work and people are made to waste a lot of time in the parking queues. Since the system is not automated and there's a person usually sitting on the toll booth, the manual nature of the current parking system makes this task inefficient.

Therefore, an automated system was needed to ensure that people aren't made to stand in queues for hours at a time to pay the parking fee. Our automated system will ensure that parking becomes seamless and hassle-free

This automated system consists of 4 major steps, namely:

- Image to text conversion of the number plate.
- Updating the backend with the detection.
- Departure from parking lot.

Brief description and the technologies used in all the above steps:

- **Image to text conversion of the number plate:**
 - When the user will arrive at the parking gate, a camera which is attached to the boom barrier will detect the number plate of the car and from that image the number plate is converted into text format with the help of Optical Character Recognition (OCR)
 - Python and OpenCv have been used in this step.
- **Updating the backend with the detection:**
 - After getting the number plate in the text format, entry timestamp is sent and updated at the Firebase backend which confirms that detection of the car has been completed and the boom barrier is uplifted for the car to enter.
 - Firebase and Python were the technologies used in this step.
- **Departure from parking lot:**
 - When the user will depart, another image to text conversion takes place to calculate final amount due and that amount is displayed on the user's mobile application and that amount is automatically deducted from the bank account of the user

- Flutter, Firebase, PyTesseract, TensorFlow and Python were the technologies used in this step.

The application which is used for automatic payment is built on **Flutter** (Flutter is an open-source UI software development kit created by Google. It is used to develop applications for Android, iOS, Windows, Mac, Linux and the web) as its framework and the language in which Source Code has been written is Dart (Dart is a client-optimized programming language for apps on multiple platforms. It is developed by Google and is used to build mobile, desktop, backend and web applications).

The backend for the entire automated system is made on **Firebase**. Firebase is a platform developed by Google for creating mobile and web applications. It was originally an independent company founded in 2011. In 2014, Google acquired the platform and it is now their flagship offering for app development.

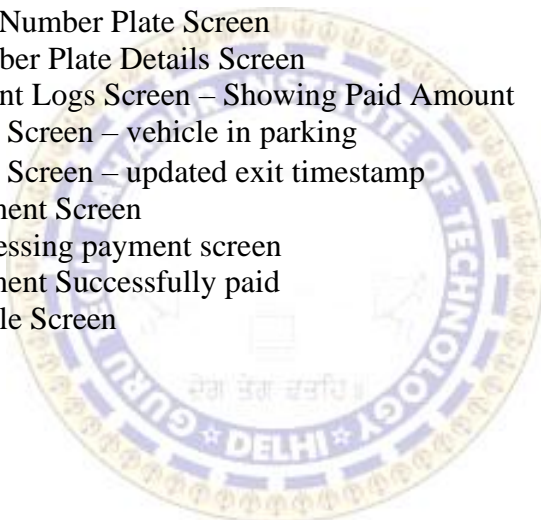
The detection is done with the help of **TensorFlow**. TensorFlow is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

Image to text conversion is done with the help **PyTesseract OCR**. Tesseract is an optical character recognition engine for various operating systems. It is free software, released under the Apache License. Originally developed by Hewlett-Packard as proprietary software in the 1980s, it was released as open source in 2005 and development has been sponsored by Google since 2006.

The entire source code for Number Plate Detection and Optical Character Recognition is written in **Python**. Python is an interpreted, high-level and general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

LIST OF FIGURES

Fig No.	Figure Name	Page
1.	Jupyter Notebook Layout	3
2.	Label Box Dashboard	4
3.	Single-Layered perceptron	8
4.	Tensor shapes	11
5.	Label Box Dashboard	39
6.	Label Box Image label tool	39
7.	Deep Learning model	40
8.	Trained model output	40
9.	Running Number plate detection script	41
10.	OCR Detection	41
11.	Login Screen	42
12.	Number plate dashboard	42
13.	Drawer	43
14.	Add Number Plate Screen	43
15.	Number Plate Details Screen	44
16.	Recent Logs Screen – Showing Paid Amount	44
17.	Logs Screen – vehicle in parking	45
18.	Logs Screen – updated exit timestamp	45
19.	Payment Screen	46
20.	Processing payment screen	46
21.	Payment Successfully paid	47
22.	Profile Screen	47



CONTENTS

Chapter No.	Page
Title Page	i
Declaration	iii
Acknowledgement	iv
Abstract	v
Tables and figures	vii
1. Introduction	1-20
1.1 Jupyter Notebook	
1.1.1 Introduction	
1.1.2 Components	
1.1.3 Anatomy of Jupyter Notebook	
1.2 Labelbox	
1.2.1 Technology	
1.2.2 Advantages	
1.3 Convolutional Neural Networks	
1.3.1 Introduction	
1.3.2 Architecture	
1.3.2.1 Convolutional	
1.3.2.2 Pooling	
1.3.2.3 Fully Connected	
1.3.2.4 Receptive Field	
1.3.2.5 Weights	
1.3.3 The Perceptron Model	
1.3.3.1 Transfer Learning	
1.3.3.2 Adam Optimizer	
1.3.3.3 Rectified Linear Unit	
1.3.3.4 Sigmoid Function	
1.3.3.5 Mean Squared Error	

1.4 Tensorflow

1.4.1 Introduction

1.4.2 Data Types

1.5 Opencv

1.5.1 Introduction

1.6 Pytesseract

1.6.1 Introduction

1.7 Flutter

1.7.1 Introduction

1.7.2 Flutter Framework Architecture

1.7.3 Dart Platform

1.7.4 Flutter Engine

1.7.5 Foundation Library

1.7.6 Widgets

1.7.6.1 Design-Specific Widgets

1.7.7 Hello World Program in Flutter

1.7.8 Introduction to Dart

1.7.8.1 Usage

1.7.8.2 Snapshots

1.8 Firebase

1.8.1 What is Firebase?

1.8.2 Features

2. Requirement Analysis with SRS

21-34

2.1 Introduction

2.1.1 Purpose

2.1.2 Scope

2.1.3 Definitions, Acronyms and Abbreviations

2.1.4 References

2.1.5 Overview

2.2 Overall Description

2.2.1 Product Perspective

2.2.1.2 User interface

2.2.1.2 Hardware Interfaces

2.2.1.3 Software Interfaces

2.2.1.4 Memory Constraints

2.2.1.5 Operations

2.2.1.6 Site Adaptation requirements

2.2.2 Product Functions

2.2.3 User Characteristics

2.2.4 Constraints

2.2.5 Assumptions and Dependencies

2.2.6 Apportioning of Requirements

2.3 Specific Requirements

2.3.1 External Interface Requirements

2.3.1.1 User Interfaces

2.3.1.2 Hardware Interfaces

2.3.1.3 Software Interfaces

2.3.2 System features

2.3.2.1 User Information Maintenance

2.3.3 Performance requirements

2.3.4 Design Constraints

2.3.5 Software System Attributes

2.3.5.1 Security

2.3.5.2 Maintainability

2.3.5.3 Portability	
2.3.5.4 Testability	
2.3.5.5 Availability	
2.3.5.6 Adaptability	
2.3.5.7 Time Limit	
2.4 Flow Diagram	
3. Test Cases	35
4. Result	43
5. Conclusion	45
6. Future scope	47
7. References	49
8. Appendix	52-131
A Screen Shots	53-61
B Source Code	62-131



Chapter 1- Introduction



1.1 JUPYTER NOTEBOOK

1.1.1 INTRODUCTION

Project Jupyter is a nonprofit organization created to "develop open-source software, open-standards, and services for interactive computing across dozens of programming languages". Spun off from IPython in 2014 by Fernando Pérez, Project Jupyter supports execution environments in several dozen languages. Project Jupyter's name is a reference to the three core programming languages supported by Jupyter, which are Julia, Python and R, and also a homage to Galileo's notebooks recording the discovery of the moons of Jupiter. Project Jupyter has developed and supported the interactive computing products Jupyter Notebook, JupyterHub, and JupyterLab.

The Jupyter Notebook is an interactive computing environment that enables users to author notebook documents that include: - Live code - Interactive widgets - Plots - Narrative text - Equations - Images - Video

These documents provide a complete and self-contained record of a computation that can be converted to various formats and shared with others using email, Dropbox, version control systems (like git/GitHub) or nbviewer.jupyter.org.

1.1.2 COMPONENTS

The Jupyter Notebook combines three components:

- The notebook web application: An interactive web application for writing and running code interactively and authoring notebook documents.
- Kernels: Separate processes started by the notebook web application that runs users' code in a given language and returns output back to the notebook web application. The kernel also handles things like computations for interactive widgets, tab completion and introspection.
- Notebook documents: Self-contained documents that contain a representation of all content visible in the notebook web application, including inputs and outputs of the computations, narrative text, equations, images, and rich media representations of objects. Each notebook document has its own kernel.

1.1.3 ANATOMY OF JUPYTER NOTEBOOK

FIG NO. (1) Jupyter Notebook Layout

```
[ ] 1 model = Sequential()
2 model.add(VGG16(weights="imagenet", include_top=False, input_shape=(HEIGHT, WIDTH, CHANNEL)))
3 model.add(Flatten())
4 model.add(Dense(128, activation="relu"))
5 model.add(Dense(64, activation="relu"))
6 model.add(Dense(64, activation="relu"))
7 model.add(Dense(4, activation="sigmoid"))
8
9 model.layers[-6].trainable = False
10
11 model.summary()
```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [=====] - 0s 0us/step
Model: "sequential_1"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 7, 7, 512)	14714688
flatten (Flatten)	(None, 25088)	0
dense_1 (Dense)	(None, 128)	3211392
dense_2 (Dense)	(None, 64)	8256
dense_3 (Dense)	(None, 64)	4160
dense_4 (Dense)	(None, 4)	260

Total params: 17,938,756
Trainable params: 3,224,068
Non-trainable params: 14,714,688

Notebook documents contain the inputs and outputs of an interactive session as well as narrative text that accompanies the code but is not meant for execution. Rich output generated by running code, including HTML, images, video, and plots, is embedded in the notebook, which makes it a complete and self-contained record of a computation.

When you run the notebook web application on your computer, notebook documents are just files on your local filesystem with a `.ipynb` extension. This allows you to use familiar workflows for organizing your notebooks into folders and sharing them with others.

Notebooks consist of a linear sequence of cells. There are three basic cell types:

- Code cells: Input and output of live code that is run in the kernel
- Markdown cells: Narrative text with embedded LaTeX equations
- Raw cells: Unformatted text that is included, without modification, when notebooks are converted to different formats using nbconvert

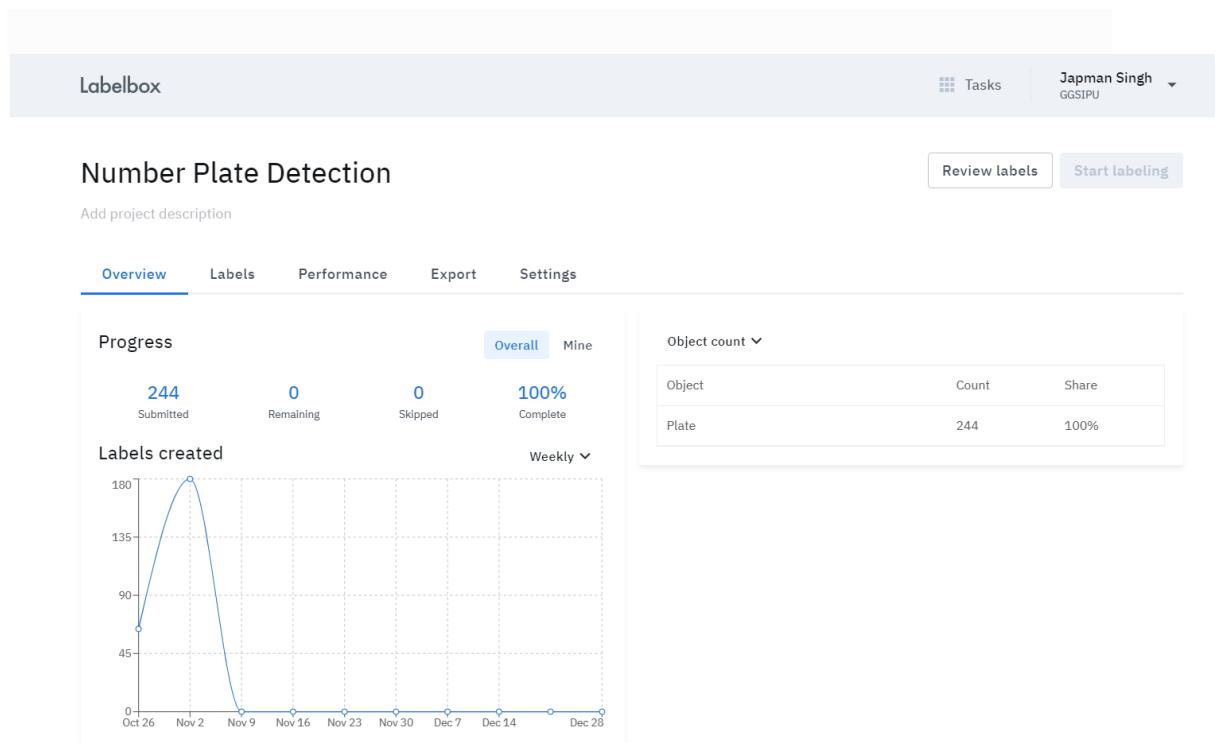
Internally, notebook documents are JSON data with binary values base64 encoded. This allows them to be read and manipulated programmatically by any programming language. Because JSON is a text format, notebook documents are version control friendly.

Notebooks can be exported to different static formats including HTML, reStructuredText, LaTeX, PDF, and slide shows ([reveal.js](#)) using Jupyter's `nbconvert` utility.

Furthermore, any notebook document available from a public URL or on GitHub can be shared via [nbviewer](#). This service loads the notebook document from the URL and renders it as a static web page. The resulting web page may thus be shared with others without their needing to install the Jupyter Notebook.

1.2 LABELBOX

FIG NO. (2) Label Box Dashboard



Labelbox is a training data platform used primarily for computer vision applications to provide labeled data to data scientists and machine learning engineers in their development of machine learning models.

The platform claims to offer automated labeling for image annotation by relying on predictions from an existing machine learning model to pre-label image data. The product also offers a Python SDK.

1.2.1 TECHNOLOGY

Automatic image annotation (also known as automatic image tagging or linguistic indexing) is the process by which a computer system automatically assigns metadata

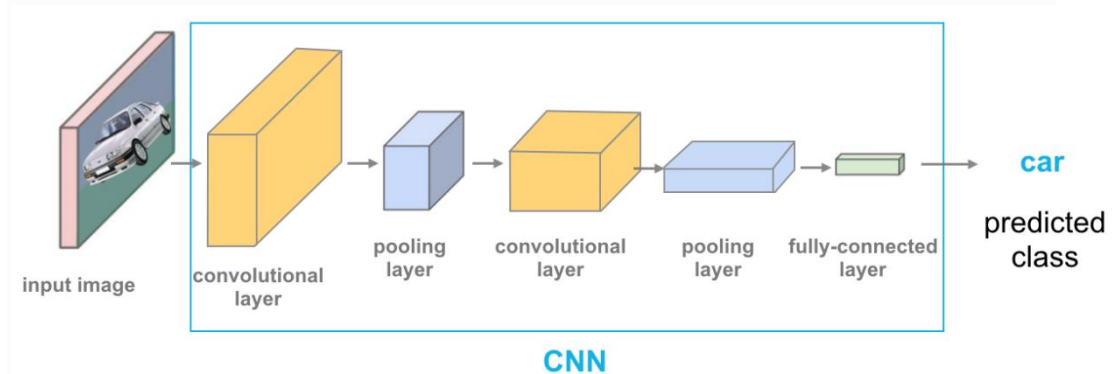
in the form of captioning or keywords to a digital image. This application of computer vision techniques is used in image retrieval systems to organize and locate images of interest from a database.

This method can be regarded as a type of multi-class image classification with a very large number of classes - as large as the vocabulary size. Typically, image analysis in the form of extracted feature vectors and the training annotation words are used by machine learning techniques to attempt to automatically apply annotations to new images. The first methods learned the correlations between image features and training annotations, then techniques were developed using machine translation to try to translate the textual vocabulary with the 'visual vocabulary', or clustered regions known as blobs. Work following these efforts have included classification approaches, relevance models and so on.

1.2.2 ADVANTAGES

The advantages of automatic image annotation versus content-based image retrieval (CBIR) are that queries can be more naturally specified by the user. CBIR generally (at present) requires users to search by image concepts such as color and texture, or finding example queries. Certain image features in example images may override the concept that the user is really focusing on. The traditional methods of image retrieval such as those used by libraries have relied on manually annotated images, which is expensive and time-consuming, especially given the large and constantly growing image databases in existence.

1.3 CONVOLUTIONAL NEURAL NETWORKS



1.3.1 INTRODUCTION

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics. They have applications in image and video recognition, recommender systems, image classification, medical image analysis, natural language processing, brain-computer interfaces, and financial time series.

CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks makes them prone to overfitting data. Typical ways of regularization include adding some form of magnitude measurement of weights to the loss function. CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme.

Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.

1.3.2 ARCHITECTURE

A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of a series of convolutional layers that convolve with a multiplication or other dot product. The activation function is commonly a ReLU layer, and is subsequently followed by additional convolutions such as pooling layers, fully connected layers and normalization layers, referred to as hidden layers because their inputs and outputs are masked by the activation function and final convolution.

1.3.2.1 CONVOLUTIONAL

When programming a CNN, the input is a tensor with shape (number of images) x (image height) x (image width) x (input channels). Then after passing through a convolutional layer, the image becomes abstracted to a feature map, with shape (number of images) x (feature map height) x (feature map width) x (feature map channels). A convolutional layer within a neural network should have the following attributes:

- Convolutional kernels defined by a width and height (hyper-parameters).
- The number of input channels and output channels (hyper-parameter).
- The depth of the Convolution filter (the input channels) must be equal to the number channels (depth) of the input feature map.

Convolutional layers convolve the input and pass its result to the next layer. This is similar to the response of a neuron in the visual cortex to a specific stimulus. Each convolutional neuron processes data only for its receptive field. Although fully connected feedforward neural networks can be used to learn features as well as classify data, it is not practical to apply this architecture to images. A very high number of neurons would be necessary, even in a shallow (opposite of deep) architecture, due to the very large input sizes associated with images, where each pixel is a relevant variable. For instance, a fully connected layer for a (small) image of size 100 x 100 has 10,000 weights for each neuron in the second layer. The convolution operation brings a solution to this problem as it reduces the number of free parameters, allowing the network to be deeper with fewer parameters. For instance, regardless of image size, tiling regions of size 5 x 5, each with the same shared weights, requires only 25 learnable parameters. By using regularized weights over fewer parameters, the vanishing gradient and exploding gradient problems seen during backpropagation in traditional neural networks are avoided.

1.3.2.2 POOLING

Convolutional networks may include local or global pooling layers to streamline the underlying computation. Pooling layers reduce the dimensions of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, typically 2 x 2. Global pooling acts on all the neurons of the convolutional layer. In addition, pooling may compute a max or an average. Max pooling uses the maximum value from each of a cluster of neurons at the prior layer. Average pooling uses the average value from each of a cluster of neurons at the prior layer.

1.3.2.3 FULLY CONNECTED

Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer perceptron neural network (MLP). The flattened matrix goes through a fully connected layer to classify the images.

1.3.2.4 RECEPTIVE FIELD

In neural networks, each neuron receives input from some number of locations in the previous layer. In a fully connected layer, each neuron receives input from every element of the previous layer. In a convolutional layer, neurons receive input from only a restricted subarea of the previous layer. Typically the subarea is of a square shape (e.g., size 5 by 5). The input area of a neuron is called its receptive field. So, in a fully connected layer, the receptive field is the entire previous layer. In a convolutional layer, the receptive area is smaller than the entire previous layer. The subarea of the original input image in the receptive field is increasingly growing as getting deeper in the network architecture. This is due to applying over and over again a convolution which takes into account the value of a specific pixel, but also some surrounding pixels.

1.3.2.5 WEIGHTS

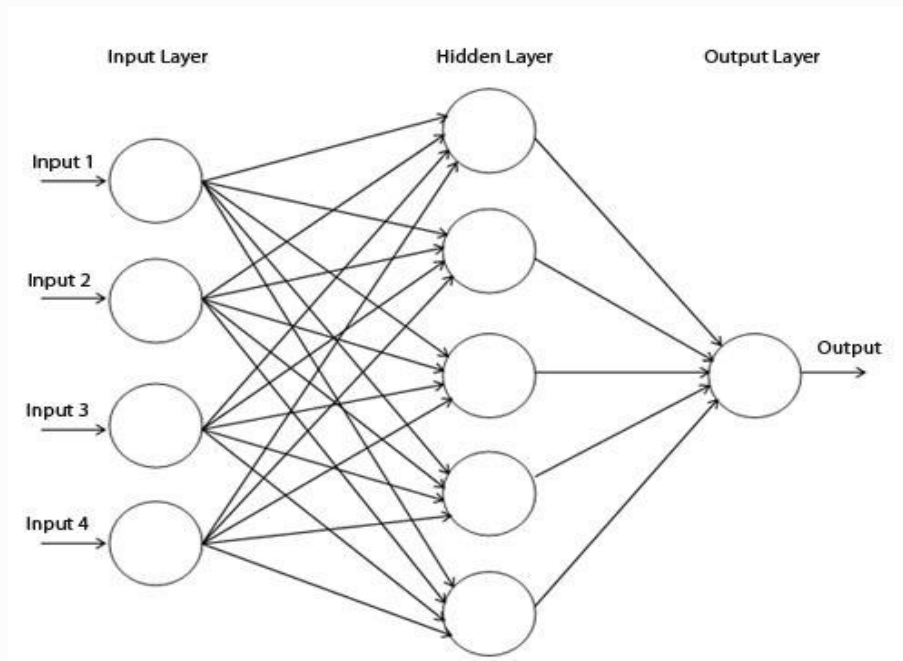
Each neuron in a neural network computes an output value by applying a specific function to the input values coming from the receptive field in the previous layer. The function that is applied to the input values is determined by a vector of weights and a bias (typically real numbers). Learning, in a neural network, progresses by making iterative adjustments to these biases and weights.

The vector of weights and the bias are called filters and represent particular features of the input (e.g., a particular shape). A distinguishing feature of CNNs is that many

neurons can share the same filter. This reduces memory footprint because a single bias and a single vector of weights are used across all receptive fields sharing that filter, as opposed to each receptive field having its own bias and vector weighting.

1.3.3 THE PERCEPTRON MODEL

FIG NO. (3) Single-Layered perceptron



A single-layer neural network represents the most simple form of neural network, in which there is only one layer of input nodes that send weighted inputs to a subsequent layer of receiving nodes, or in some cases, one receiving node. This single-layer design was part of the foundation for systems which have now become much more complex.

One of the early examples of a single-layer neural network was called a “perceptron.” The perceptron would return a function based on inputs, again, based on single neurons in the physiology of the human brain. In some senses, perceptron models are much like “logic gates” fulfilling individual functions: A perceptron will either send a signal, or not, based on the weighted inputs. Another type of single-layer neural network is the single-layer binary linear classifier, which can isolate inputs into one of two categories.

Single-layer neural networks can also be thought of as part of a class of feedforward neural networks, where information only travels in one direction, through the inputs,

to the output. Again, this defines these simple networks in contrast to immensely more complicated systems, such as those that use backpropagation or gradient descent to function.

1.3.5.1 TRANSFER LEARNING

Transfer learning (TL) is a research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. For example, knowledge gained while learning to recognize cars could apply when trying to recognize trucks.

1.3.5.2 ADAM OPTIMIZER

The Adam optimization algorithm is an extension to stochastic gradient descent that has recently seen broader adoption for deep learning applications in computer vision and natural language processing. Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data.

1.3.5.3 RECTIFIED LINEAR UNIT

The Rectified Linear Unit is the most commonly used activation function in deep learning models. The function returns 0 if it receives any negative input, but for any positive value x it returns that value back. So it can be written as $f(x)=\max(0,x)$.

1.3.5.4 SIGMOID FUNCTION

A sigmoid function is a mathematical function having a characteristic "S"-shaped curve or sigmoid curve.

1.3.5.5 MEAN SQUARED ERROR

MSE is the average of the squared error that is used as the loss function for least squares regression. It is the sum, over all the data points, of the square of the difference between the predicted and actual target variables, divided by the number of data points. RMSE is the square root of MSE. MSE is measured in units that are

the square of the target variable, while RMSE is measured in the same units as the target variable. Due to its formulation, MSE, just like the squared loss function that it derives from, effectively penalizes larger errors more severely.

1.4 TENSORFLOW

1.4.1 INTRODUCTION






TensorFlow is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

Tensorflow is a symbolic math library based on dataflow and differentiable programming. It is used for both research and production at Google. TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache License 2.0 in 2015.

1.4.2 DATA TYPES

The basic data type in this framework is a Tensor. A Tensor is an N-dimensional array of data. For instance, you can call a Scalar (a constant value such as integer 2) as a 0-dimension tensor. A vector is a 1-dimensional tensor and a matrix is a 2-dimensional tensor. The following graphic describes each of the dimensions of a tensor in detail.

FIG NO. (4) Tensor shapes

	Common name	Rank (Dimension)	Example	Shape of example
	Scalar	0	<code>x = tf.constant(3)</code>	<code>()</code>
	Vector	1	<code>x = tf.constant([3, 5, 7])</code>	<code>(3,)</code>
	Matrix	2	<code>x = tf.constant([[3, 5, 7], [4, 6, 8]])</code>	<code>(2, 3)</code>
	3D Tensor	3	<code>tf.constant([[[3, 5, 7],[4, 6, 8]], [[1, 2, 3],[4, 5, 6]]])</code>	<code>(2, 2, 3)</code>
	nD Tensor	n	<code>x1 = tf.constant([2, 3, 4])</code> <code>x2 = tf.stack([x1, x1])</code> <code>x3 = tf.stack([x2, x2, x2, x2])</code> <code>x4 = tf.stack([x3, x3])</code> <code>...</code>	<code>(3,)</code> <code>(2, 3)</code> <code>(4, 2, 3)</code> <code>(2, 4, 2, 3)</code>

1.5 OPENCV

1.5.1 INTRODUCTION

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 18 million. The library is used extensively in companies, research groups and by governmental bodies.

Along with well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many startups such as Applied Minds, VideoSurf, and Zeitera, that make extensive use of OpenCV. OpenCV's deployed uses span the range from stitching streetview images together, detecting intrusions in surveillance video in Israel, monitoring mine equipment in China, helping robots navigate and pick up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan.

It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDA and OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

1.6 PYTESSERACT

1.6.1 INTRODUCTION

Tesseract is an optical character recognition engine for various operating systems. It is free software, released under the Apache License. Originally developed by Hewlett-Packard as proprietary software in the 1980s, it was released as open source in 2005 and development has been sponsored by Google since 2006.

Tesseract is an open source text recognition (OCR) Engine. It can be used directly, or (for programmers) using an API to extract printed text from images. It supports a wide variety of languages. Tesseract doesn't have a built-in GUI, but there are several available from the 3rd Party page. Tesseract is compatible with many programming languages and frameworks through wrappers that can be found here. It can be used with the existing layout analysis to recognize text within a large document, or it can be used in conjunction with an external text detector to recognize text from an image of a single text line.

1.7 FLUTTER

1.7.1 INTRODUCTION

Flutter is an open-source UI software development kit created by Google. It is used to develop applications for Android, iOS, Windows, Mac, Linux, Google Fuchsia and the web.

The first version of Flutter was known as codename "Sky" and ran on the Android operating system. It was unveiled at the 2015 Dart developer summit, with the stated intent of being able to render consistently at 120 frames per second. During the keynote of Google Developer Days in Shanghai, Google announced Flutter Release Preview 2 which is the last big release before Flutter 1.0. On December 4th, 2018, Flutter 1.0 was released at the Flutter Live event, denoting the first "stable" version of the Framework.

1.7.2 FLUTTER FRAMEWORK ARCHITECTURE

The major components of Flutter include:

- Dart platform
- Flutter engine
- Foundation library
- Design-specific widgets

1.7.3 DART PLATFORM

Flutter apps are written in the Dart language and make use of many of the language's more advanced features.

On Windows, macOS and Linux via the semi-official Flutter Desktop Embedding project, Flutter runs in the Dart virtual machine which features a just-in-time execution engine. While writing and debugging an app, Flutter uses Just In Time compilation, allowing for "hot reload", with which modifications to source files can be injected into a running application. Flutter extends this with support for stateful hot reload, where in most cases changes to source code can be reflected immediately

in the running app without requiring a restart or any loss of state. This feature as implemented in Flutter has received widespread praise.

Release versions of Flutter apps are compiled with ahead-of-time (AOT) compilation on both Android and iOS, making Flutter's high performance on mobile devices possible.

1.7.4 FLUTTER ENGINE

Flutter's engine, written primarily in C++, provides low-level rendering support using Google's Skia graphics library. Additionally, it interfaces with platform-specific SDKs such as those provided by Android and iOS. The Flutter Engine is a portable runtime for hosting Flutter applications. It implements Flutter's core libraries, including animation and graphics, file and network I/O, accessibility support, plugin architecture, and a Dart runtime and compile toolchain. Most developers will interact with Flutter via the Flutter Framework, which provides a modern, reactive framework, and a rich set of platform, layout and foundation widgets.

1.7.5 FOUNDATION LIBRARY

The Foundation library, written in Dart, provides basic classes and functions which are used to construct applications using Flutter, such as APIs to communicate with the engine.

1.7.6 WIDGETS

UI design in Flutter involves using composition to assemble / create "Widgets" from other Widgets. The trick to understanding this is to realize that any tree of components (Widgets) that is assembled under a single build() method is also referred to as a single Widget. This is because those smaller Widgets are also made up of even smaller Widgets, and each has a build() method of its own. This is how Flutter makes use of Composition.

The docs say: " A widget is an immutable description of part of a user interface." A human being will tell you it's a Blueprint, which is a much easier way to think about it. However, one also needs to keep in mind there are many types of Widgets in Flutter, and you cannot see or touch all of them. Text is a Widget, but so is its TextStyle, which defines things like size, color, font family and weight. There are Widgets that represent things, ones that represent characteristics (like TextStyle) and even others that do things, like FutureBuilder and StreamBuilder.

Complex widgets can be created by combining many simpler ones, and an app is actually just the largest Widget of them all (often called "MyApp"). The MyApp Widget contains all the other Widgets, which can contain even smaller Widgets, and together they make up your app.

However, the use of widgets is not strictly required to build Flutter apps. An alternative option, usually only used by people who like to control every pixel drawn to the canvas, is to use the Foundation library's methods directly. These methods can be used to draw shapes, text, and imagery directly to the canvas. This ability of Flutter has been utilized in a few frameworks, such as the open-source Flame game engine.

1.7.6.1 DESIGN-SPECIFIC WIDGETS

The Flutter framework contains two sets of widgets which conform to specific design languages. Material Design widgets implement Google's design language of the same name, and Cupertino widgets implement Apple's Human Interface Guidelines iOS design.

1.7.7 HELLO WORLD PROGRAM IN FLUTTER

```
import 'package:flutter/material.dart';

void main() => runApp(HelloWorldApp());

class HelloWorldApp extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      title: 'Hello World App',

      home: Scaffold(

        appBar: AppBar(

          title: Text('Hello World App'),
```

```
    ),  
  
    body: Center(  
  
      child: Text('Hello World'),  
  
    ),  
  
  ),  
  
);  
  
}  
  
}
```

1.7.8 INTRODUCTION TO DART

Dart is a client-optimized programming language for apps on multiple platforms. It is developed by Google and is used to build mobile, desktop, backend and web applications.

Dart is an object-oriented, class defined, garbage-collected language using a C-style syntax that transcompiles optionally into JavaScript. It supports interfaces, mixins, abstract classes, reified generics, static typing, and a sound type system.

1.7.8.1 USAGE

There are three main ways to run Dart code:

Compiled as JavaScript

To run in mainstream web browsers, Dart relies on a source-to-source compiler to JavaScript. According to the project site, Dart was "designed to be easy to write development tools for, well-suited to modern app development, and capable of high-performance implementations. "When running Dart code in a web browser the code is precompiled into JavaScript using the dart2js compiler. Compiled as JavaScript, Dart code is compatible with all major browsers with no need for browsers to adopt Dart. Through optimizing the compiled JavaScript output to avoid expensive checks

and operations, code written in Dart can, in some cases, run faster than equivalent code hand-written using JavaScript idioms.

Stand-alone

The Dart software development kit (SDK) ships with a stand-alone Dart VM, allowing Dart code to run in a command-line interface environment. As the language tools included in the Dart SDK are written mostly in Dart, the stand-alone Dart VM is a critical part of the SDK. These tools include the dart2js compiler and a package manager called pub. Dart ships with a complete standard library allowing users to write fully working system apps, such as custom web servers.

Ahead-of-time compiled

Dart code can be AOT-compiled into machine code (native instruction sets). Apps built with Flutter, a mobile app SDK built with Dart, are deployed to app stores as AOT-compiled Dart code.

1.7.8.2 SNAPSHOTS

Snapshots are a core part of the Dart VM. Snapshots are files which store objects and other runtime data.

Script snapshots

Dart programs can be compiled into snapshot files. These files contain all of the program code and dependencies prepared and ready to execute. This allows fast start-ups.

Full snapshots

The Dart core libraries can be compiled into a snapshot file which allows fast loading of the libraries. Most standard distributions of the main Dart VM have a prebuilt snapshot for the core libraries which is loaded at runtime.

Object snapshots

Dart is a very asynchronous language. With this, it uses isolates for concurrency. Since these are workers which pass messages, it needs a way to serialize a message. This is done using a snapshot, which is generated from a given object, and then this is transferred to another isolate for deserializing.

1.8 FIREBASE

1.8.1 WHAT IS FIREBASE?

Firebase is a powerful platform for your mobile and web application. Firebase can power your app's backend, including data storage, user authentication, static hosting, and more. With Firebase, you can easily build mobile and web apps that scale from one user to one million.

1.8.2 FEATURES

1. Cloud Firestore: is a flexible, scalable database for mobile, web, and server development from Firebase and Google Cloud Platform. It is a NoSQL document database that lets you easily store, sync, and query data for your mobile and web apps — at a global scale. It's supporting for Android, iOS and Web Platform.

2. ML Kit: is a mobile SDK that brings Google's machine learning expertise to Android and iOS apps in a powerful yet easy-to-use package. Whether you're new or experienced in machine learning, you can implement the functionality you need in just a few lines of code. There's no need to have deep knowledge of neural networks or model optimization to get started. On the other hand, if you are an experienced ML developer, ML Kit provides convenient APIs that help you use your custom TensorFlow Lite models in your mobile apps. It's supporting for Android and iOS Platform.

3. Cloud Functions: for Firebase lets you automatically run backend code in response to events triggered by Firebase features and HTTPS requests. Your code is stored in Google's cloud and runs in a managed environment. There's no need to manage and scale your own servers. It's supporting for Android, iOS, C++, Unity and Web Platform.

4. Authentication: provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to your app. It supports authentication using passwords, phone numbers, popular federated identity providers like Google, Facebook and Twitter, and more. It's supporting for Android, iOS and Web Platform.

5. Hosting: is production-grade web content hosting for developers. With a single command, you can quickly deploy web apps and serve both static and dynamic content to a global CDN (content delivery network). You can also pair Firebase Hosting with Cloud Functions to build and host microservices on Firebase. It's supporting only Web Platform.

6. Cloud Storages: is for object storage service built for Google scale. The Firebase SDKs for Cloud Storage add Google security to file uploads and downloads for your Firebase apps, regardless of network quality. You can use our SDKs to store images, audio, video, or other user-generated content. On the server, you can use Google Cloud Storage, to access the same files. It's supporting for Android, iOS, C++, Unity and Web Platform.

7. Real-time Database: is a cloud-hosted NoSQL database that lets you store and sync between your users in real-time. The Real-time Database is really just one big JSON object that the developers can manage in real-time. It's supporting for Android, iOS, C++, Unity and Web Platform.

8. Crashlytics: is a lightweight, real-time crash reporter that helps you track, prioritize, and fix stability issues that erode your app quality. Crashlytics saves you troubleshooting time by intelligently grouping crashes and highlighting the circumstances that lead up to them. It's supporting for Android and iOS Platform.

9. Performance Monitoring: is a service that helps you to gain insight into the performance characteristics of your iOS and Android apps. You use the Performance Monitoring SDK to collect performance data from your app, and then review and analyze that data in the Firebase console. Performance Monitoring helps you to understand where and when the performance of your app can be improved so that you can use that information to fix performance issues. It's supporting for Android and iOS Platform.

10 In-App Messaging: helps you engage users who are actively using your app by sending them targeted and contextual messages that nudge them to complete key in-app actions — like beating a game level, buying an item, or subscribing to content. It's supporting for Android and iOS Platform.

11. Google Analytics: for Firebase is a free app measurement solution that provides insight on app usage and user engagement. At the heart of Firebase is Google Analytics for Firebase, a free and unlimited analytics solution. Analytics integrates across Firebase features and provides you with unlimited reporting for up to 500

distinct events that you can define using the Firebase SDK. Analytics reports help you understand clearly how your users behave, which enables you to make informed decisions regarding app marketing and performance optimizations. It's supporting for Android, iOS, C++ and Unity Platform.

12. Predictions: applies machine learning to your analytics data to create dynamic user segments based on the predicted behavior of users in your app. These predictions are automatically available for use with Firebase Remote Config, the Notifications composer, Firebase In-App Messaging, and A/B Testing. You can also link your app's Predictions data to BigQuery so you can get daily exports that you can further analyze or push to third party tools. It's supporting for Android, iOS, C++ and Unity Platform.

13. Cloud Messaging(FCM): provides a reliable and battery-efficient connection between your server and devices that allows you to deliver and receive messages and notifications on iOS, Android, and the web at no cost.

14. Remote Config: essentially allows us to publish updates to our users immediately. Whether we wish to change the color scheme for a screen, the layout for a particular section in our app or show promotional/seasonal options — this is completely doable using the server side parameters without the need to publish a new version. It's supporting for Android, iOS, C++ and Unity Platform.

The logo of Shaheed Bhagat Singh College, Delhi, is a circular emblem. It features a blue outer ring with the college's name in English and Hindi. Inside this is a yellow ring with a gear-like pattern. The center of the logo is white and contains a faint image of a building.

Chapter 2-Requirement Analysis with SRS

2.1 Introduction

This document aims at defining the overall software requirements for ‘Simply Parking’ mobile application. Efforts have been made to define the requirements exhaustively and accurately. The final product will be having only features functionalities mentioned in this document and assumptions for any additional functionality/feature should not be made by any of the parties involved in developing/testing/implementing/using this product. In case it is required to have some additional features, a formal change request will need to be raised and subsequently new release of this document and/or product will be produced

2.1.1 Purpose

This specification document describes the capabilities that will be provided by the software application for ‘Simply Parking’. The document also describes non-function requirement such as the user interface. It also states the various required constraints by which the system will abide. The purpose of this document is to present a detailed description of the Simply Parking mobile application. It explains the purpose and features of the application itself, the app interface through which a user can be a part of an automated parking system wherein the user’s car number plate can be identified and based on the plate number and the number of hours elapsed at the parking destination, the user is charged on the app itself. This document is intended for both the customer and the project development team.

2.1.2 Scope

The software required specification capture all the requirement in a single document. The software product ‘Simply Parking’ will be a mobile application that simulates an automated parking system wherein the user’s car number plate can be identified and based on the plate number and the number of hours elapsed at the parking destination, the user is charged on the app itself.

2.1.3 Definitions, Acronyms, and Abbreviations

Following abbreviations have been used throughout this document:

Py (Python)

2.1.4 References

IEEE Recommended Practice for Software Requirements Specifications-IEEE Std 830-1993

2.1.5 Overview

The rest of this SRS document describes the various system requirements, interfaces, features and functionalities in detail

2.2 Overall Description

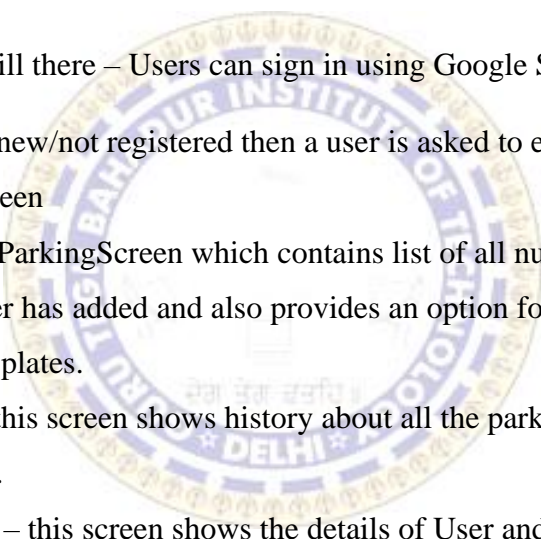
The ‘Simply Parking’ will simulate an automated parking system wherein the user’s car number plate can be identified and based on the plate number and the number of hours elapsed at the parking destination, the user is charged on the app itself. The purpose of this section is to present a detailed description of the product’s perspective giving information about the context and interface constraints. The product functions section outlines major functionality Simply Parking will perform. The user characteristics section explains the expectations Simply Parking has about the user. The constraints section contains detailed descriptions of constraints and safety critical properties pertaining to Simply Parking. The assumptions and dependencies section summarize any assumptions or dependencies the application has about the hardware, software, environment and user interactions associated with it. The proportioning of requirements section goes over the functionality customers want, but have been determined out of scope for the current iteration of the application

2.2.1 Product Perspective

The application will be an application-based, self-contained and independent software product. The Simply Parking mobile application consist of a User only. The User interacts with an interface which simulates an automated parking system wherein the user's car number plate can be identified and based on the plate number and the number of hours elapsed at the parking destination, the user is charged on the app itself. Moreover, the Simply Parking is especially made for automation purposes, therefore transactions done through this system are seamless and hassle-free.

2.2.1.1 User interface

The application will have a user-friendly and menu-based interface. Following screens will be provided:

- 
- i. User Login will there – Users can sign in using Google Sign in.
 - ii. If the User is new/not registered then a user is asked to enter the username on the signup screen
 - iii. Then there is ParkingScreen which contains list of all number plates and its details the user has added and also provides an option for the user to add more number plates.
 - iv. LogScreen – this screen shows history about all the parking transactions that user has done.
 - v. ProfileScreen – this screen shows the details of User and the number of cars added by the user.
 - vi. Drawer – contains : settings, saved locations, pending payments, support and logout.
 - vii. Payment interface – in this interface user can do secure and fast payments.

2.2.1.2 Hardware Interfaces

- i. Internal Storage of at least 75 MB required for proper and complete functionality of Application.
- ii. The application runs both in portrait as well as landscape mode.
- iii. internet is required to run this application.

2.2.1.3 Software Interfaces

- i. Python 3.7 and above.
- ii. Any android-based architecture system.
- iii. There must be at least 75 MB free in App Space to run Application properly.

2.2.1.4 Memory Constraints

At least 64 MB RAM and 110 MB space on internal storage will be required for running the application.

2.2.1.5 Operations

This product release will not cover any automated housekeeping aspects of the data.

The User can easily install application if the Hardware and Software Interfaces specified in above section is fulfilled. It can also be easily uninstalled by User.

2.2.1.6 Site Adaptation Requirements

The android device at client site will have to support the hardware and software interfaces in above sections.

2.2.2 Product Functions

The application will allow access to Users who have installed application in their devices. Depending upon the user's choice, he will be able to access all modules of the application.

A summary of the major functions that the application will perform:

- i. The menu from where the User check the contents, and information of the application.

- ii. The user can be a part of an automated parking system wherein the user's car number plate can be identified and based on the plate number and the number of hours elapsed at the parking destination, the user is charged on the app itself.

2.2.3 User Characteristics

- Educational level: At least graduate should be comfortable with English language
- Experience: Should be well versed informed about automated systems.
- Technical expertise: Should be comfortable using general-purpose application on a Android mobile phone.

2.2.4 Constraints

The User can easily access the application. Internet is compulsory for the application to work properly.

2.2.5 Assumptions and Dependencies

- i. Detection of number plates using camera must be correct.
- ii. Bank Balance should be greater than the fees of parking.
- iii. Internet should function properly at the parking booths.

2.2.6 Apportioning of Requirements

Not Required

2.3 Specific Requirements

This section contains the software requirements to a level of detail sufficient to enable designers to design the system, and testers to test that system.

2.3.1 External Interface Requirements

2.3.1.1 User Interfaces

The following screens will be provided:

- **Login Screen**
- **Parking Screen**
- **Add Number Plate Screen**
- **Number Plate Details Screen**
- **Logs Screen**
- **Profile Screen**
- **Drawer**

Login screen:

This will be the first screen that will be displayed. It allows the user to sign in using google or register new users.

Parking Screen:

It will allow users to see the number plate list and to check its details by navigating to Number plate details screen.

Add Number Plate Screen:

It will allow the user to add number plates and its various details.

Number Plate Details Screen:

It will allow the user to check and see number plates and its various details like:

- Engine Number
- Fuel Type
- Owner name
- Fitness upto
- Chassis number
- Registraion Date
- Vehicle class
- Registration number

- Insurance expiry
- Registering authority
- Model

It allows to user to see the Logs related to that particular number plate.

Logs Screen:

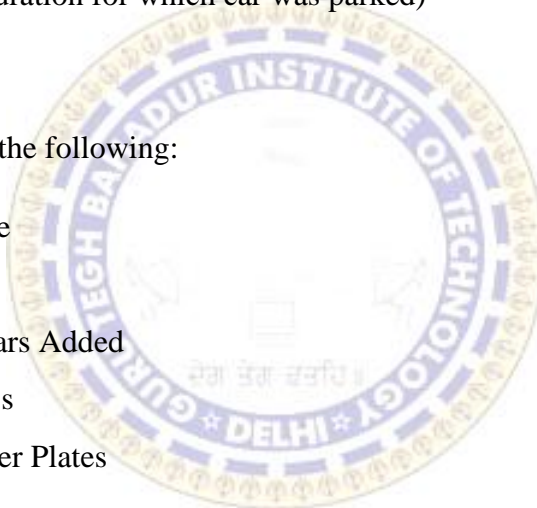
This screen shows car number plate and location it has been parked and status of its transaction (i.e whether bill is paid or not). It shows the following details:

- Place
- Entry timestamp
- Exit timestamp
- Parked for (duration for which car was parked)

Profile Screen:

This screen contains the following:

- Profile Picture
- Name
- Number of Cars Added
- Email Address
- Added Number Plates



Drawer:

This screen contains the following:

- Settings
- saved locations
- pending payments
- support
- logout.

2.3.1.2 Hardware Interfaces

- i. Internal Storage of at least 50 MB required for proper and complete functionality of Application.

- ii. The application runs both in portrait as well as landscape mode.
- iii. internet is required to run this application.

2.3.1.3 Software Interfaces

- i. Python 3.7 and above.
- ii. Any android-based architecture system.
- iii. There must be at least 75 MB free in App Space to run Application properly.

2.3.2 System Features

2.3.2.1 User Information Maintenance

Description

The following information would be maintained:

User Name, User Profile Picture, User E-mail, User Google ID, User Number Plates, User_car details.

The system will allow modification of new/existing users.

Validity checks

- i. User Name cannot be blank.
- ii. Email cannot be null.
- iii. Profile picture cannot be null.

Sequencing information

User Info for a particular module will have to be entered in the system before any result of module is displayed on the screen.

Error handling response to abnormal situations

If any of the above validations/sequencing flow does not hold true, appropriate error messages will be prompted to the user for doing the needful

2.3.3 Performance Requirements

- i. The software does not break down suddenly.
- ii. The performance of function should be well.
- iii. The input of one phase is output to the next phase and this make it more reliable and accurate.
- iv. This application will be supported by other embedded software.
- v. The overall performance of the software will be reliable and enable user to use it efficiently.

2.3.4 Design Constraints

None

2.3.5 Software System Attributes

2.3.5.1 Security

The whole application is secure. User Data will be stored in the secure backend of Firebase(Google) and the application will be using an Internet connection to run.

2.3.5.2 Maintainability

The application will be designed in a maintainable manner. Deployment of the project if any error that can be maintained by software developer.

2.3.5.3 Portability

The application will be easily portable on any android-based system that has software and hardware interfaces fulfilled.

2.3.5.4 Testability

The application will be tested:

- i. Alpha testing.
- ii. Beta testing.

2.3.5.5 Availability

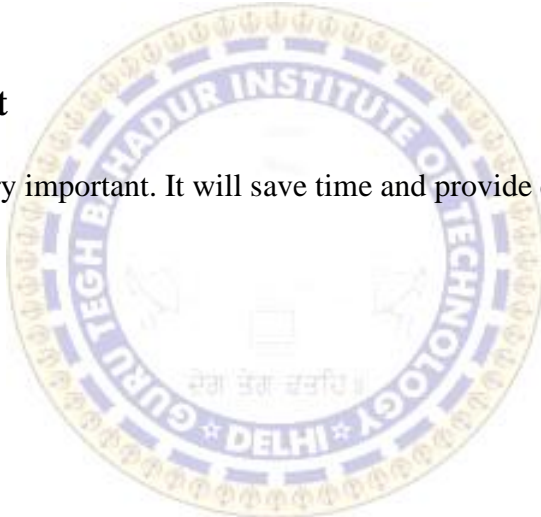
The application will be available every time.

2.3.5.6 Adaptability

The application is adaptable by any User.

2.3.5.7 Time limit

The Time limit is very important. It will save time and provide quick access to the information.



2.4 Diagrams

2.4.1 Flow Diagram

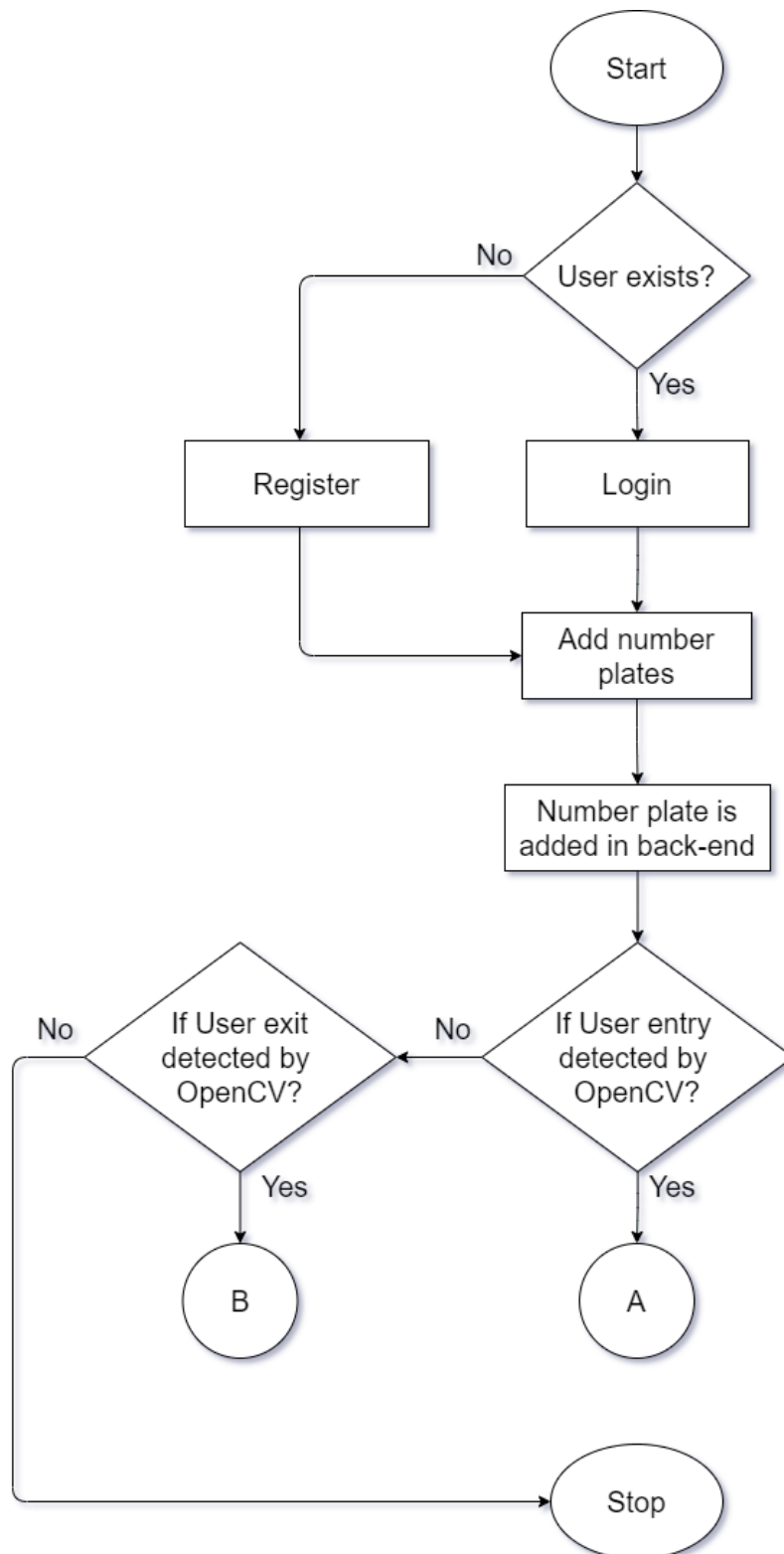


Fig – Application flow Simply Parking (1)

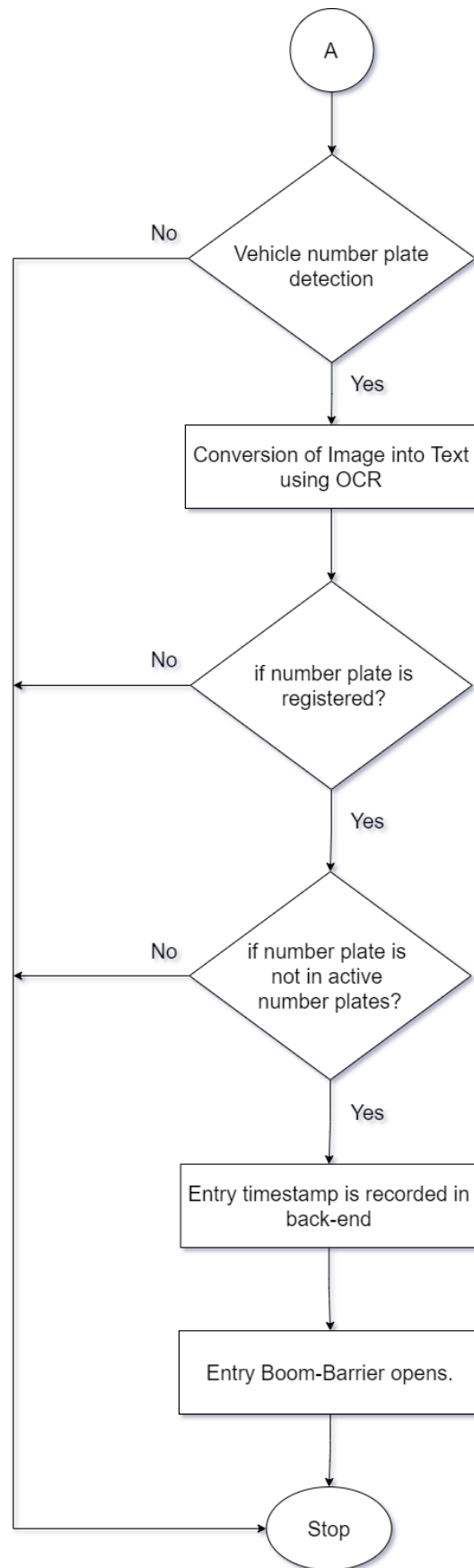


Fig – Application flow Simply Parking (2)

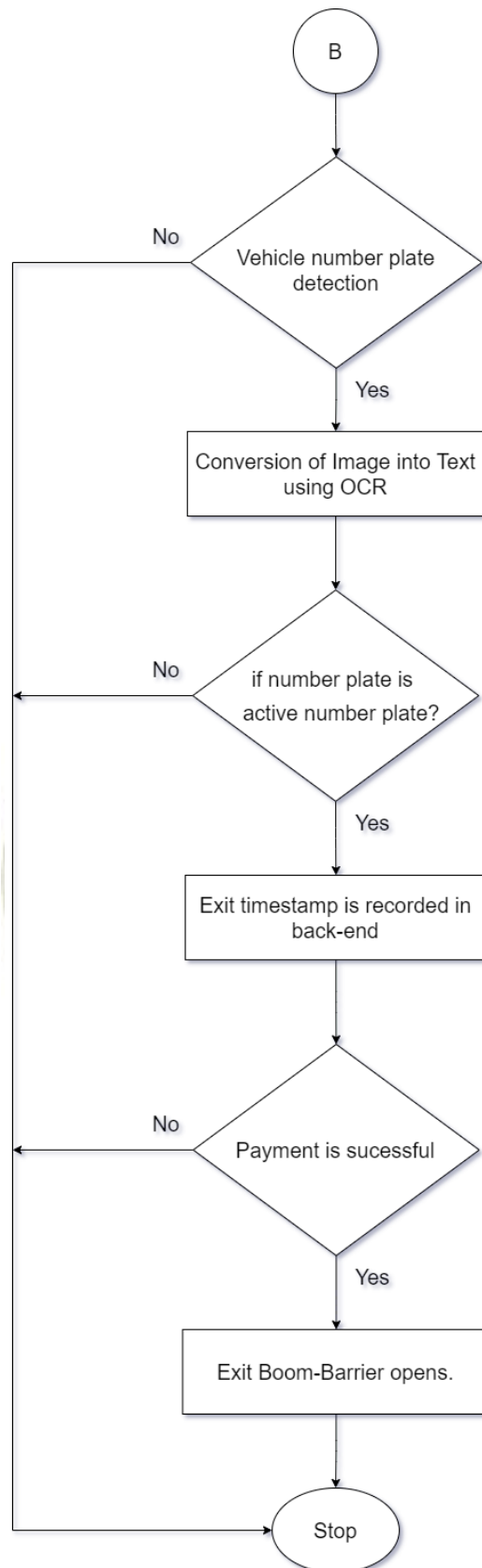
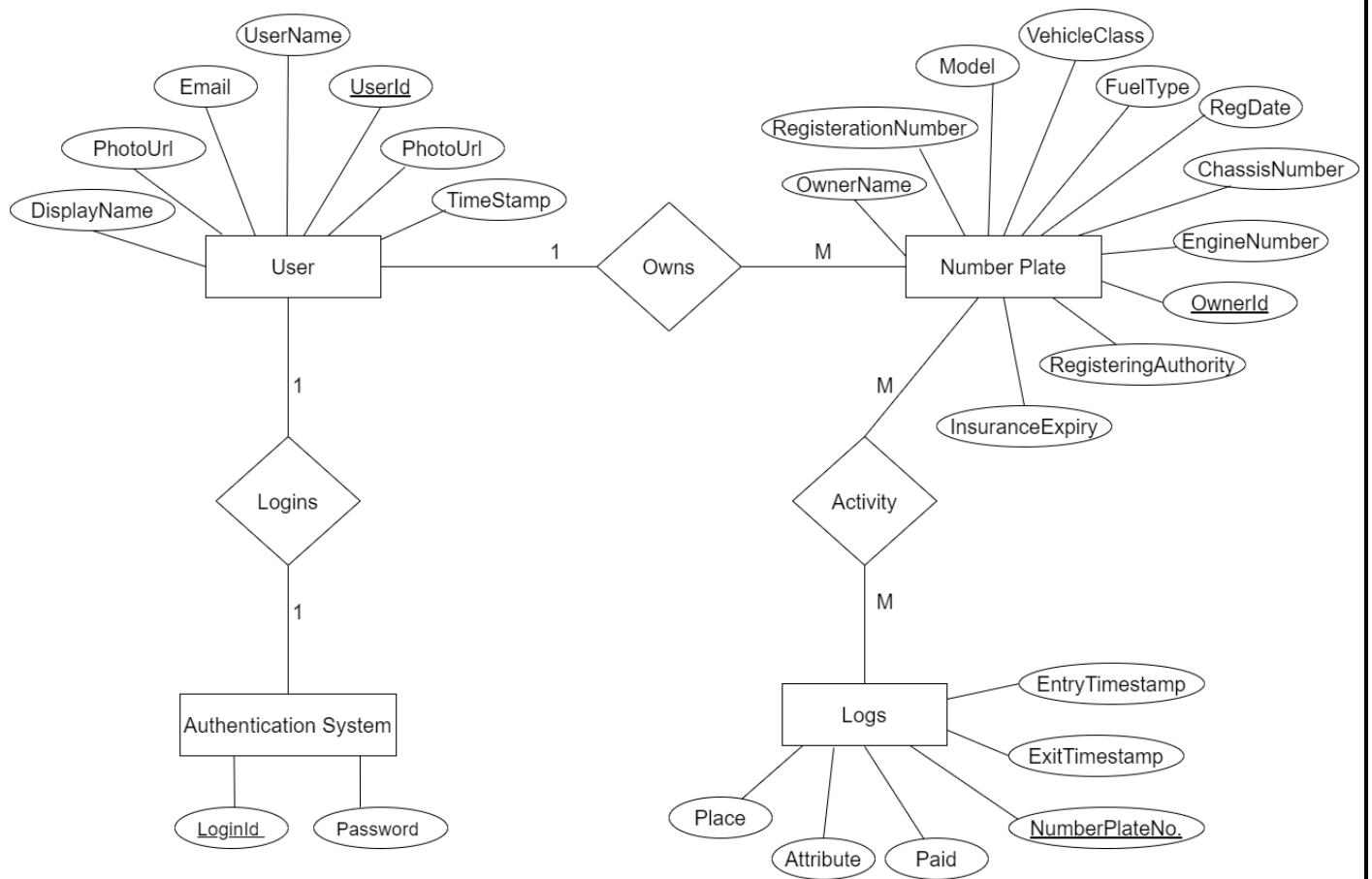
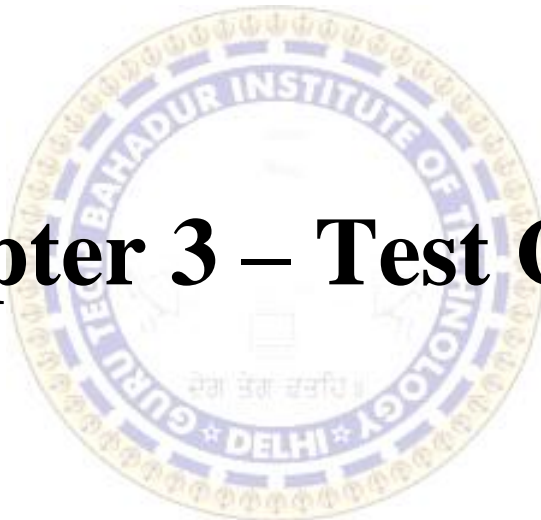


Fig – Application flow Simply Parking (3)

2.4.2 ER Diagram

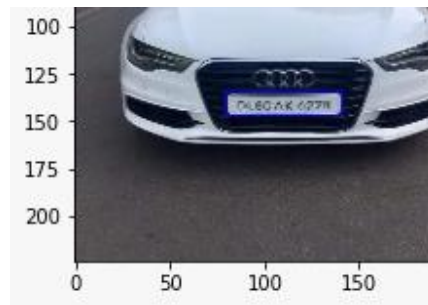


Chapter 3 – Test Cases



Test Cases

The system has been tested on a series of different test cases based on various types of number plates in India and also Globally.



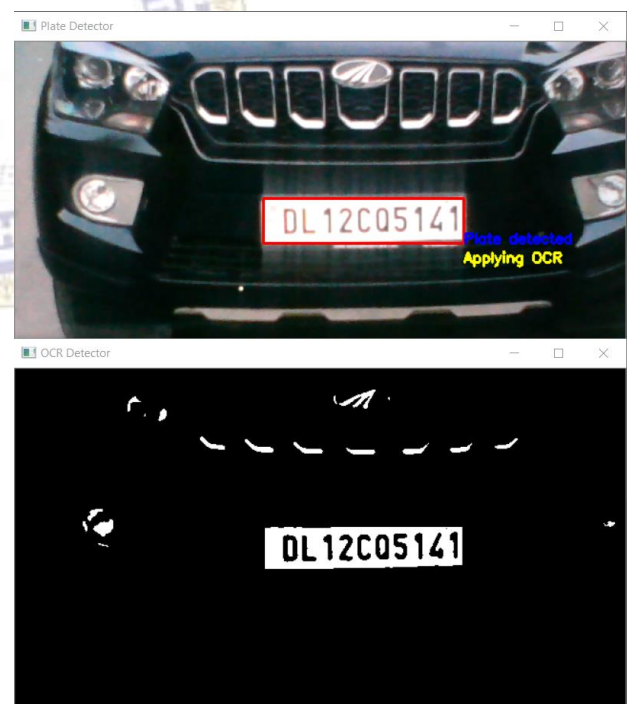
Expected Output Sample (For Reference)

Test Case(1) – Plate Detection

Status - **Success**



Input Camera Feed



Detected Plate and Mask

Test Case(2) – Plate Detection

Status - **Success**



Input Camera Feed



Detected Plate and Mask

Test Case(3) – Plate Detection

Status - **Success**



Input Camera Feed



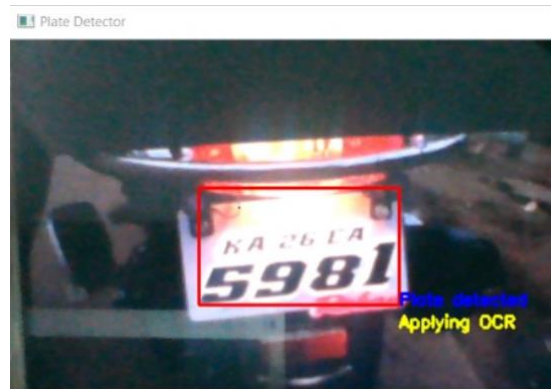
Detected Plate and Mask

Test Case(4) – Plate Detection

Status - **Success**



Input Camera Feed



Detected Plate and Mask

Test Case(5) – Plate Detection

Status - **Success**



Input Camera Feed



Detected Plate and Mask

Test Case(6) – Plate Detection

Status - **Success**



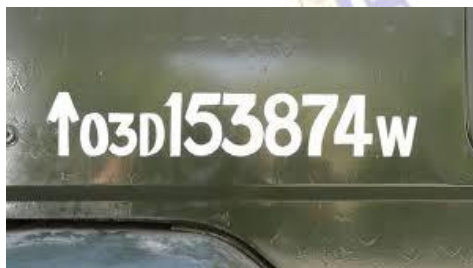
Input Camera



Detected Plate and Mask

Test Case(7) – Plate Detection

Status - **Failed**



Input Camera



Detected Plate and Mask

Test Case(1) – OCR

Status - **Success**



```
Windows PowerShell
PS C:\Users\Japman\Desktop\Final project> python entry_backup.py
Welcome to Simply Parking : ['DL3CCC6508']
```

Test Case(2) – OCR

Status - **Success**

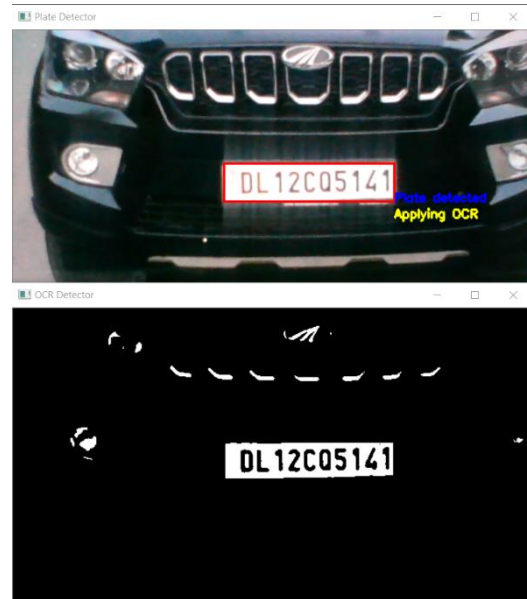


```
Windows PowerShell
PS C:\Users\Japman\Desktop\Final project> python entry_backup.py
Welcome to Simply Parking : ['DL3C BA 5115']
```

Test Case(3) – OCR

Status - Failed

Reason – Due to Q similar to 0, the system detects last 4 digits as 0514 instead of 5141



```
Windows PowerShell
PS C:\Users\Japman\Desktop\Final project> python entry_backup.py
Welcome to Simply Parking : ['DL 12C0514']
```

Test Case(4) – OCR

Status - Failed

Reason – Due to Q similar to O, the system detects Q as O



```
Windows PowerShell
PS C:\Users\jaska> python detection.py
Welcome to Simply Parking : ['DL2C0 9816']
```

Test Case(5) – OCR

Status - **Failed**

Reason –

Due to Regex based on Indian format, detection fails



Detected Plate and Mask

```
Select Windows PowerShell
PS C:\Users\jaska> python detection.py
OCR Failed
```

Test Case(6) – Plate Detection

Status - **Failed**

Reason –

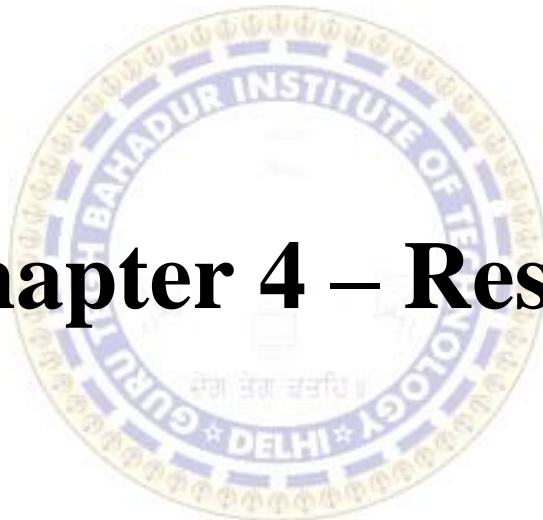
Due to Regex based on Indian format, detection fails



Detected Plate and Mask

```
Select Windows PowerShell
PS C:\Users\jaska> python detection.py
OCR Failed
```

Chapter 4 – Result



Result

The following results can be deduced from the development of application.

- The application is fully accurate in number plate detection for all legal plates in India as well as abroad.
- The application can accurately detect plate boundaries in almost all test cases
- For illegal plates too, the application is accurate except for painted number plates that cause errors
- OCR is however not as accurate and shows inaccuracies in similar looking alphabets and digits
- OCR only works for English language and has been limited to detect Indian plates format by using regex.



Chapter 5 – Conclusion



CONCLUSION

The system was designed in such a way that future modifications can be done easily. The following conclusions can be deduced from the development of application.

- It provides a friendly graphical user interface which proves to be better for user.
- The application has adequate scope for modification if it is necessary.
- It will help users to avoid long queues at parking and avoid the wastage of time.
- It makes the parking system less tedious and hassle-free.
- When the user will arrive at the parking gate, a camera which is attached to the boom barrier will detect the number plate of the car and from that image the number plate is converted into text format with the help of Optical Character Recognition (OCR)
- After getting the number plate in the text format, entry timestamp is sent and updated at the Firebase backend which confirms that detection of the car has been completed and the boom barrier is uplifted for the car to enter.
- When the user will depart, another image to text conversion takes place to calculate final amount due and that amount is displayed on the user's mobile application and that amount is automatically deducted from the bank account of the user
- It is a simple and efficient solution for the users to experience an automated parking system.

The User interacts with an interface which simulates an automated parking system wherein the user's car number plate can be identified and based on the plate number and the number of hours elapsed at the parking destination, the user is charged on the app itself

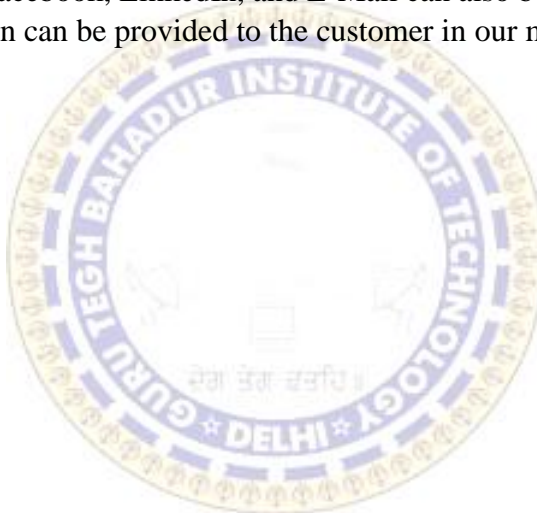
Chapter 6 - Future Scope



FUTURE SCOPE

There are many additional features that are planned to be incorporated during the future enhancements of this project. Although all the main objectives according to SRS document have been achieved, there is still room for improvement.

- This software can be easily upgraded in the future and also include many more features for the existing system.
- The software will be further enhanced with the addition of more training sets.
- The OCR in number plate is not 100% accurate. In future, we can work on increasing the accuracy.
- Quality of dataset can be increased in the future.
- UI changes will be made to make the software more interactive
- Efficiency of the software will improve as better algorithms will be implemented.
- Payment gateway needs to be implemented in our mobile application using secure payment channels.
- Login from Facebook, LinkedIn, and E-Mail can also be implemented.
- Support option can be provided to the customer in our mobile application in the future.



Chapter 7 - References



REFERENCES

Books, Journals and Articles

- [1] O'Reilly - Hands On Machine Learning with ScikitLearn, Keras and TensorFlow, Pg 80-165
- [2] Allibai, E. (2018, November 21). Building a deep learning model using Keras. Medium. <https://towardsdatascience.com/building-a-deep-learning-model-using-keras-1548ca149d37>
- [3] Zhang, Wei (1988). "Shift-invariant pattern recognition neural network and its optical architecture". Proceedings of Annual Conference of the Japan Society of Applied Physics.
- [4] Vishal Gupta – Restful API Guide - <https://shrouded-falls-48764.herokuapp.com/docs/readme.html>
- [5] Venkatachalam, M. (2020, May 14). Introduction to object detection. Medium. <https://towardsdatascience.com/introduction-to-object-detection-943f21e26063>
- [6] Labenz, C. (2020, October 15). Testable flutter and cloud Firestore. Medium. <https://medium.com/flutter/testable-flutter-and-cloud-firestore-1cf2fbbce97b>
- [7] Uther, . (2017). Temporal difference learning. Encyclopedia of Machine Learning and Data Mining, 1233-1240. https://doi.org/10.1007/978-1-4899-7687-1_817
- [8] Wright, S. (2013). App logic components. Pro SharePoint 2013 App Development, 245-282. https://doi.org/10.1007/978-1-4302-5885-8_9

Online References

- [1] <https://flutter.dev/docs/reference/tutorials>
- [2]. <https://www.w3adda.com/flutter-tutorial>
- [3] <https://stackoverflow.com/questions/49072957/flutter-dart-open-a-view-after-a-delay>

Chapter 8 - APPENDIX



APPENDIX A: SCREENSHOTS

FIG NO. (5) Label Box Dashboard

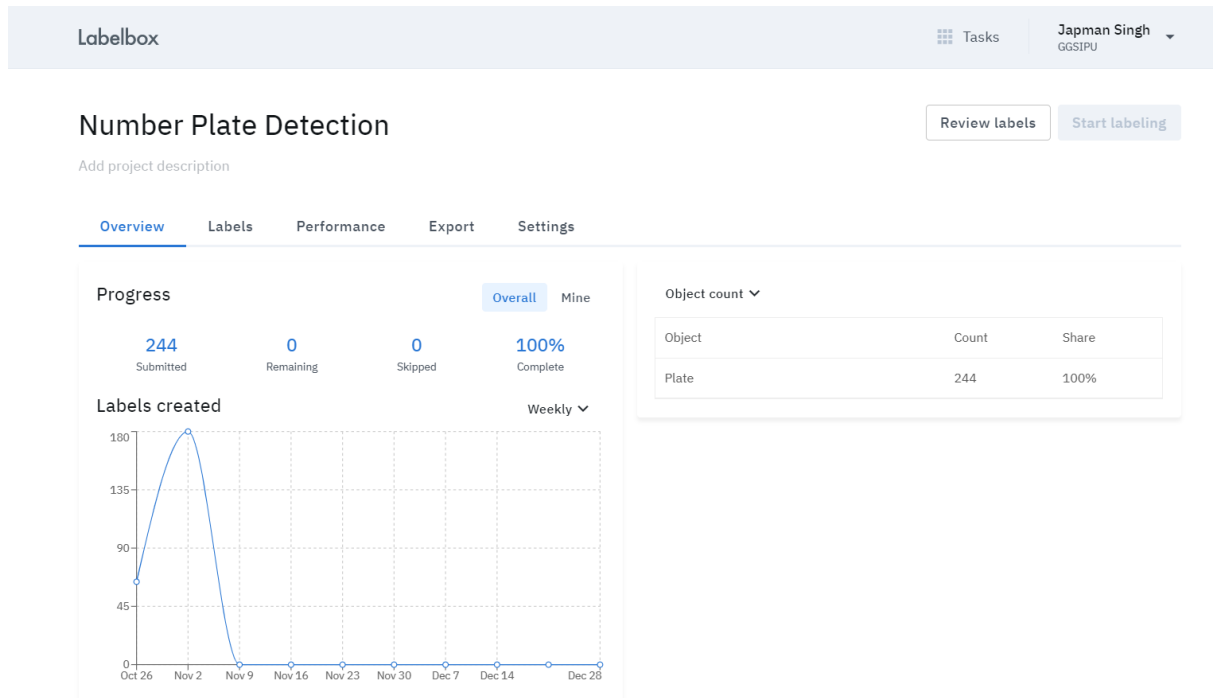


FIG NO. (6) Label Box Image label tool



FIG NO. (7) Deep Learning model

```
[ ] 1 model = Sequential()
2 model.add(VGG16(weights="imagenet", include_top=False, input_shape=(HEIGHT, WIDTH, CHANNEL)))
3 model.add(Flatten())
4 model.add(Dense(128, activation="relu"))
5 model.add(Dense(64, activation="relu"))
6 model.add(Dense(64, activation="relu"))
7 model.add(Dense(4, activation="sigmoid"))
8
9 model.layers[-6].trainable = False
10
11 model.summary()
```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [=====] - 0s 0us/step
Model: "sequential_1"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 7, 7, 512)	14714688
flatten (Flatten)	(None, 25088)	0
dense_1 (Dense)	(None, 128)	3211392
dense_2 (Dense)	(None, 64)	8256
dense_3 (Dense)	(None, 64)	4160
dense_4 (Dense)	(None, 4)	260

Total params: 17,938,756
Trainable params: 3,224,068
Non-trainable params: 14,714,688

FIG NO. (8) Trained model output

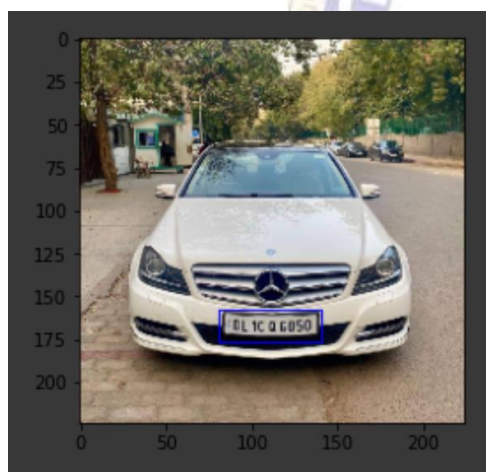


FIG NO. (9) Running Number plate detection script

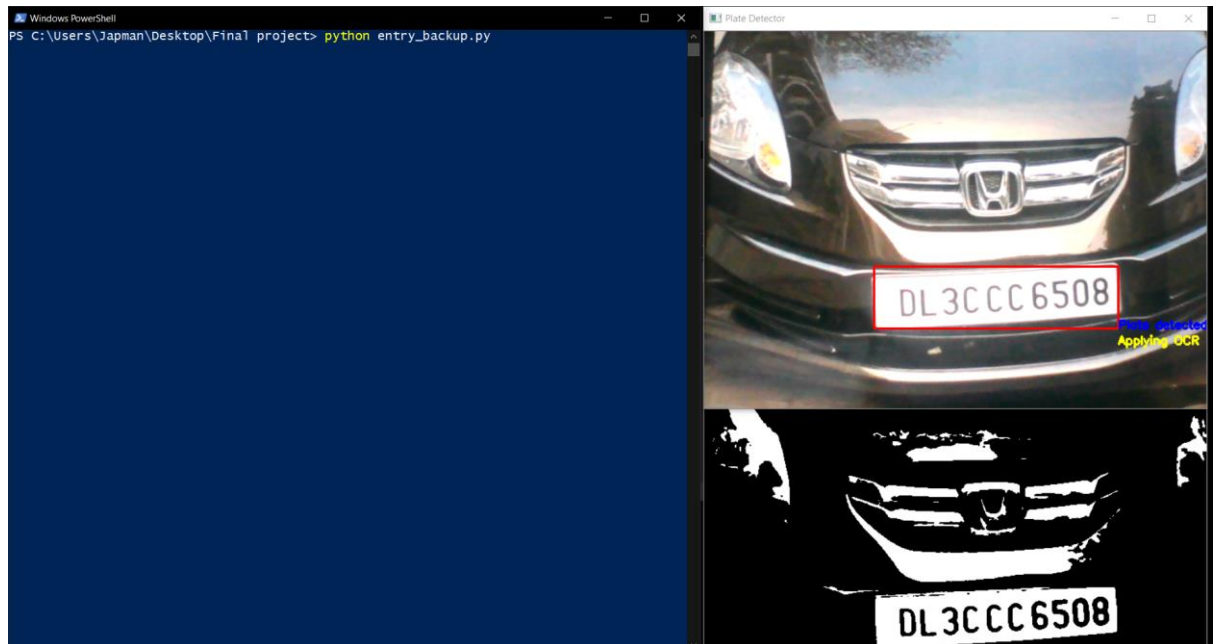


FIG NO. (10) OCR Detection

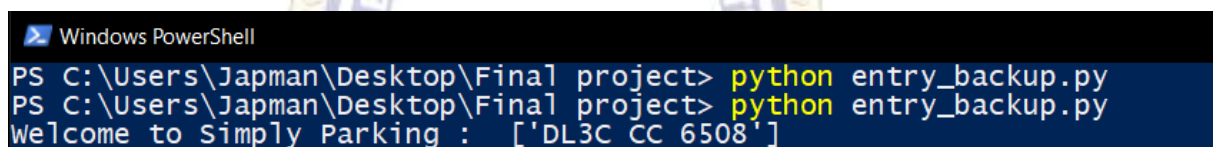


FIG NO. (11) Login Screen

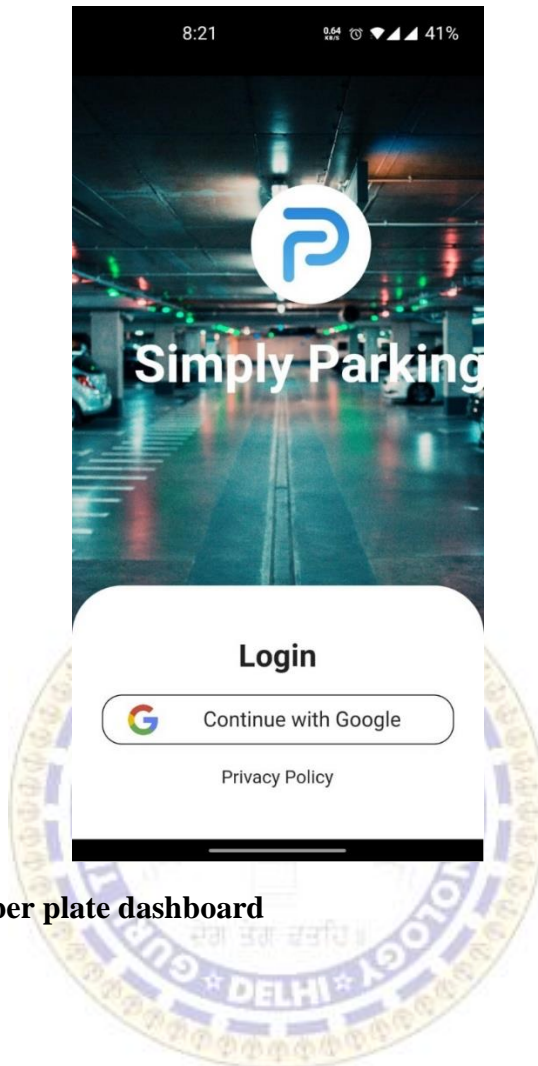


FIG NO. (12) Number plate dashboard

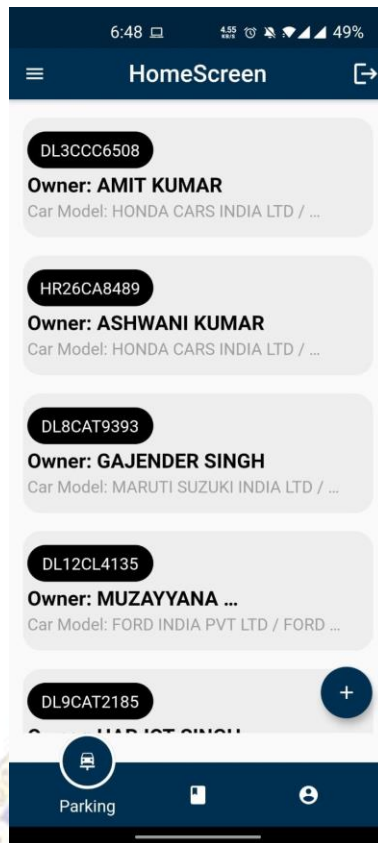


FIG NO. (13) Drawer

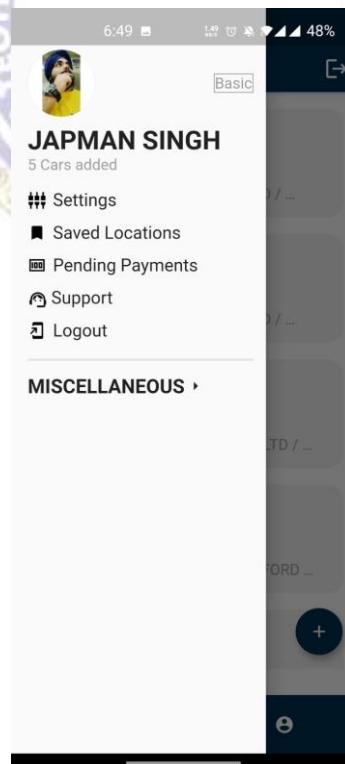


FIG NO. (14) Add Number Plate Screen



FIG NO. (15) Number Plate Details Screen

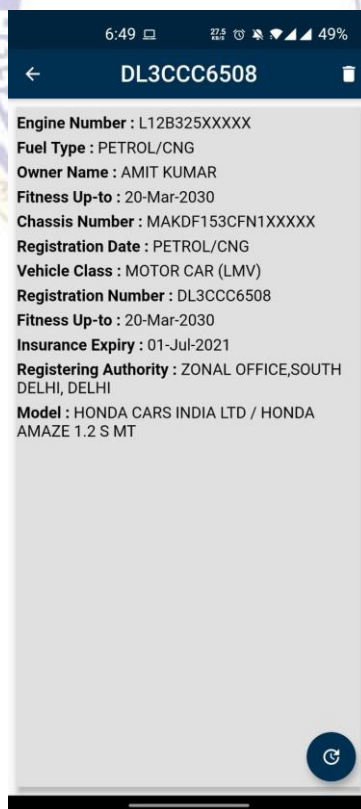


FIG NO. (16) Recent Logs Screen – Showing Paid Amount

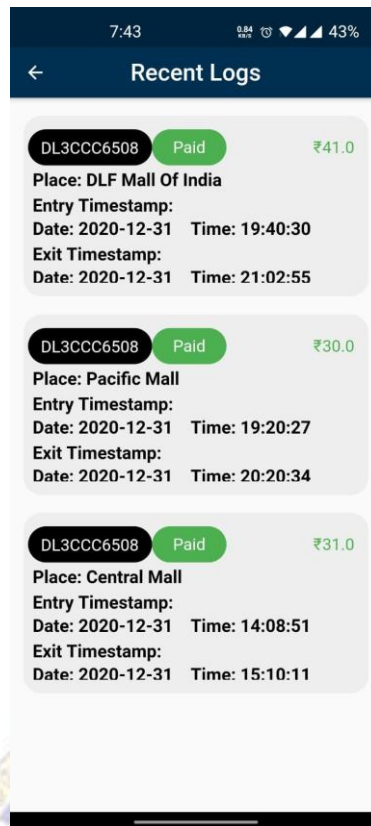


FIG NO. (17) Logs Screen – Showing vehicle in parking

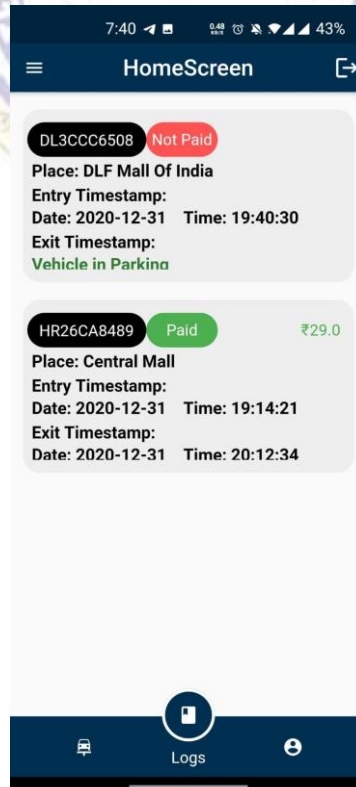


FIG NO. (18) Logs Screen – updated exit timestamp

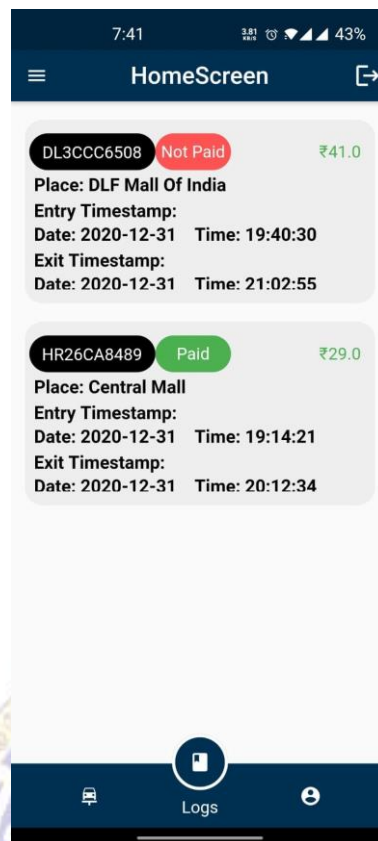


FIG NO. (19) Payment Screen



FIG NO. (20) Processing payment screen

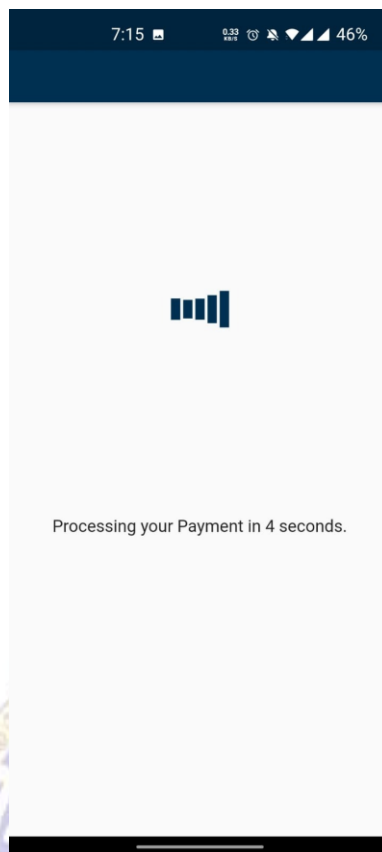


FIG NO. (21) Payment Successfully paid

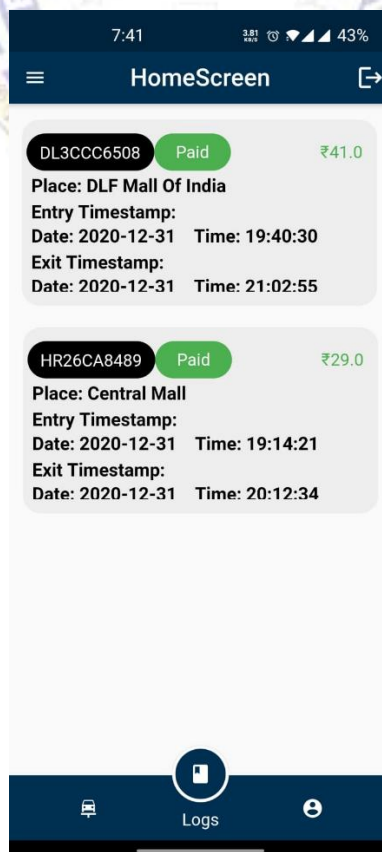
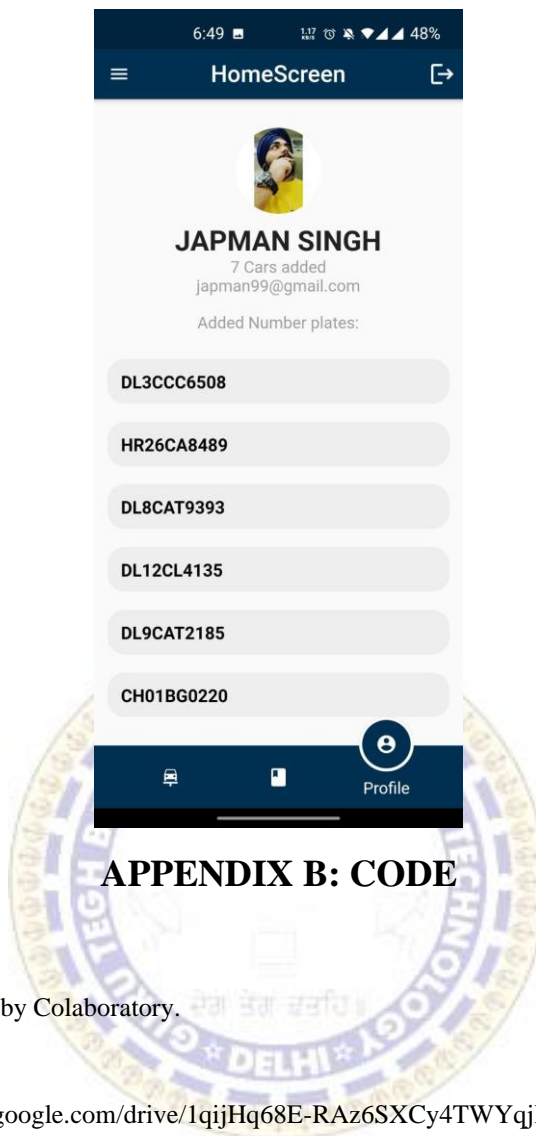


FIG NO. (22) Profile Screen



APPENDIX B: CODE

-*- coding: utf-8 -*-

Automatically generated by Colaboratory.

Original file is located at

<https://colab.research.google.com/drive/1qijHq68E-RAz6SXCy4TWYqjRd2x1F0rE>

Introduction

This is a model for object detection to detect number plate using CNN, LSTM along with transfer learning.

"""

```
import cv2
import os
import numpy as np
import pandas as pd
import urllib
import matplotlib.pyplot as plt
```

```
from PIL import Image
from keras.applications.vgg16 import VGG16
from keras.layers import Flatten, Dense
from keras.models import Model, Sequential
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam
```

```

for dirname, _, filenames in os.walk('/content/'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

"""# Step 1

We download the data we collected manually and labelled using labelbox, from our github repository.
"""

!wget https://raw.githubusercontent.com/japmansingh/plate_detection/main/plates.json

"""#Viewing Data"""

df = pd.read_json("/content/plates.json")
df.head()

"""#Create a Directory for JSON data to be extracted to"""

try:
    os.mkdir("Dataset")
except:
    print("Already Exists")

"""
#Check if data in JSON file has all required parameters
"""

count = 0
error = 0
import json
with open('plates.json') as json_file:
    data = json.load(json_file)
    for item in data:
        if(item["Label"]["objects"][0]["bbox"]["top"]):
            if(item["Label"]["objects"][0]["bbox"]["left"]):
                if(str(item["Label"]["objects"][0]["bbox"]["height"]):
                    if(str(item["Label"]["objects"][0]["bbox"]["width"]):
                        count+=1

        else:
            error+=1

    print(count, "fully labelled images found and there are " ,error, "incomplete images that need
processing")

"""#Initialising lists to append image parameters"""

dataset = dict()

dataset["image_name"] = list()
dataset["image_width"] = list()

```



```

dataset["image_height"] = list()
dataset["top"] = list()
dataset["left"] = list()

counter = 0
tt = item["Label"]["objects"][0]["bbox"]["top"]
ll = item["Label"]["objects"][0]["bbox"]["left"]
hh = item["Label"]["objects"][0]["bbox"]["height"]
ww = item["Label"]["objects"][0]["bbox"]["width"]

#import json
#with open('plates.json') as json_file:
#    data = json.load(json_file)

#    for item in data:
#        dataset["image_width"].append(ww)
#        dataset["image_height"].append(hh)
#        dataset["left"].append(ll)
#        dataset["top"].append(tt)

"""#Parsing JSON file and obtaining target parameters"""

import json
with open('plates.json') as json_file:
    data = json.load(json_file)
    for item in data:
        dataset["image_height"].append(item["Label"]["objects"][0]["bbox"]["height"])
        dataset["top"].append(item["Label"]["objects"][0]["bbox"]["top"])
        dataset["left"].append(item["Label"]["objects"][0]["bbox"]["left"])
        dataset["image_width"].append(item["Label"]["objects"][0]["bbox"]["width"])

"""###Checking if correct values were appended"""

print(dataset['image_height'])

"""###Appending image names from dataframe that was initially created"""

for index, row in df.iterrows():
    img = urllib.request.urlopen(row["Labeled Data"])
    img = Image.open(img)
    img = img.convert('RGB')
    img.save("Dataset/licensed_car{ }.jpeg".format(counter), "JPEG")

    dataset["image_name"].append("licensed_car{ }".format(counter))

    #dataset["image_height"].append(hh)
    #dataset["top"].append(tt)
    #dataset["left"].append(ll)
    #dataset["image_width"].append(ww)

    counter += 1

```

```

print("Downloaded {} car images.".format(counter))

"""###Verifying all lengths to prevent error during df generation"""

len(dataset["image_name"])

len(dataset["left"])

len(dataset["image_height"])

len(dataset["top"])

len(dataset["image_width"])

df = pd.DataFrame(dataset)
df.head()

"""#Saving in csv format for later use."""

df.to_csv("plates.csv", index=False)

df = pd.read_csv("plates.csv")
df["image_name"] = df["image_name"] + ".jpeg"
#df.drop(["image_width", "image_height"], axis=1, inplace=True)
df.head()

"""###As evident in the dataframe, we only have the top and left coordinate of the plates, so we
calculate all 4 coordinates using the following formulae"""

dataset["width"] = list()
dataset["height"] = list()
dataset["top_x"] = list()
dataset["top_y"] = list()
dataset["bottom_x"] = list()
dataset["bottom_y"] = list()

from PIL import Image
for name in df["image_name"]:
    img = Image.open("Dataset/" + str(name))
    width, height = img.size
    #print(width)
    #print(height)
    dataset["width"].append(width)
    dataset["height"].append(height)

    topx = float((float(df["left"].iloc[index])/width))
    topy = float((float(df["top"].iloc[index])/height))
    botx = topx + float(float(df["image_width"].iloc[index])/width)
    boty = topy + float(float(df["image_height"].iloc[index])/height)

    dataset["top_x"].append(topx)
    dataset["top_y"].append(topy)

```

```

dataset["bottom_x"].append(botx)
dataset["bottom_y"].append(boty)

len(dataset["top_x"])

len(dataset["bottom_x"])

len(dataset["top_y"])

len(dataset["bottom_y"])

df = pd.DataFrame(dataset)
df.head()

df.to_csv("plates.csv", index=False)
df = pd.read_csv("plates.csv")
df["image_name"] = df["image_name"] + ".jpeg"
#df.drop(["image_width", "image_height"], axis=1, inplace=True)
df.head()

#"""dataset["top_x"] = list()
#dataset["top_y"] = list()
#dataset["bottom_x"] = list()
#dataset["bottom_y"] = list()

#for data in df:
#    topx = float((float(df["left"].iloc[index])/float(df["width"].iloc[index])))
#    topy = float((float(df["top"].iloc[index])/float(df["height"].iloc[index])))
#    botx = tx + float(float(df["image_width"].iloc[index])/float(df["width"].iloc[index]))
#    boty = ty + float(float(df["image_height"].iloc[index])/float(df["height"].iloc[index]))
#    print(topx)
#    dataset["top_x"].append(tx)
#    dataset["top_y"].append(ty)
#    dataset["bottom_x"].append(bx)
#    dataset["bottom_y"].append(by)"""

#"""print(dataset["top_x"])"""

"""I picked five random records from the dataframe for a later visual inspection of predictions. I
dropped these records from the original dataframe by aiming to prevent the model to be trained on
them."""

df = pd.DataFrame(dataset)
df.head()

df.to_csv("plates.csv", index=False)
df = pd.read_csv("plates.csv")
df["image_name"] = df["image_name"] + ".jpeg"
#df.drop(["image_width", "image_height"], axis=1, inplace=True)
df.head()

dropped_test_samples = np.random.randint(0, len(df), 5)

```

```

#reduced_df = df.drop(dropped_test_samples, axis=0)
reduced_df=df

WIDTH = 224
HEIGHT = 224
CHANNEL = 3

def show_img(index):
    image = cv2.imread("Dataset/" + df["image_name"].iloc[index])
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = cv2.resize(image, dsize=(WIDTH, HEIGHT))

    #tx = int(df["left"].iloc[index])
    tx = int(float((float(df["left"].iloc[index])/float(df["width"].iloc[index])))*WIDTH)
    #int(df["top_x"].iloc[index] * WIDTH)
    #ty = int(df["top"].iloc[index])
    ty = int(float((float(df["top"].iloc[index])/float(df["height"].iloc[index])))*HEIGHT)
    #int(df["top_y"].iloc[index] * HEIGHT)
    #bx = tx + int(df["image_width"].iloc[index])
    bx = tx + int(float(float(df["image_width"].iloc[index])/float(df["width"].iloc[index]))*WIDTH)
    #int(df["image_height"].iloc[index])
    #int(df["bottom_x"].iloc[index] * WIDTH)
    #by = ty + int(df["image_height"].iloc[index])
    by = ty + int(float(float(df["image_height"].iloc[index])/float(df["height"].iloc[index]))*HEIGHT)
    #int(df["image_width"].iloc[index])
    #int(df["bottom_y"].iloc[index] * HEIGHT)

    image = cv2.rectangle(image, (tx, ty), (bx, by), (0, 0, 255), 1)
    plt.imshow(image)
    plt.show()

"""Here, you can see a sample image from the dataset with a bounding box over the license plate."""

show_img(19)

show_img(25)

show_img(35)

datagen = ImageDataGenerator(rescale=1./255, validation_split=0.1)

train_generator = datagen.flow_from_dataframe(
    reduced_df,
    directory="Dataset/",
    x_col="image_name",
    y_col=["top_x", "top_y", "bottom_x", "bottom_y"],
    target_size=(WIDTH, HEIGHT),
    batch_size=32,
    class_mode="other",
    subset="training")

validation_generator = datagen.flow_from_dataframe(

```

```

reduced_df,
directory="Dataset/",
x_col="image_name",
y_col=["top_x", "top_y", "bottom_x", "bottom_y"],
target_size=(WIDTH, HEIGHT),
batch_size=32,
class_mode="other",
subset="validation")

"""#To be Reviewed"""

model = Sequential()
model.add(VGG16(weights="imagenet", include_top=False, input_shape=(HEIGHT, WIDTH,
CHANNEL)))
model.add(Flatten())
model.add(Dense(128, activation="relu"))
model.add(Dense(64, activation="relu"))
model.add(Dense(64, activation="relu"))
model.add(Dense(4, activation="sigmoid"))

model.layers[-6].trainable = False

model.summary()

model = Sequential()
model.add(VGG16(weights="imagenet", include_top=False, input_shape=(HEIGHT, WIDTH,
CHANNEL)))

model.add(Flatten())
model.add(Dense(128, activation="relu"))
model.add(Dense(64, activation="relu"))
model.add(Dense(64, activation="relu"))
model.add(Dense(4, activation="sigmoid"))

model.layers[-6].trainable = False

model.summary()

STEP_SIZE_TRAIN = int(np.ceil(train_generator.n / train_generator.batch_size))
STEP_SIZE_VAL = int(np.ceil(validation_generator.n / validation_generator.batch_size))

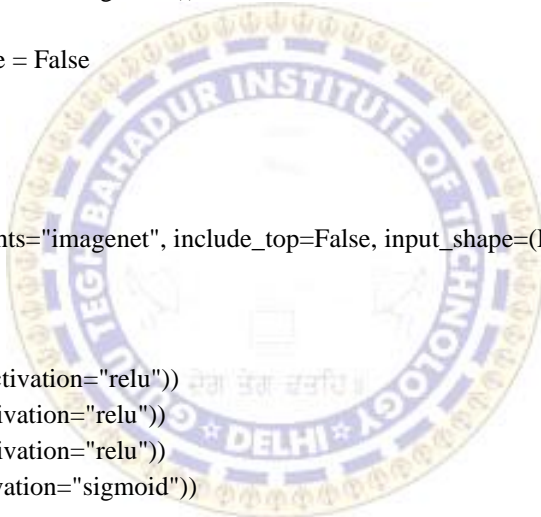
print("Train step size:", STEP_SIZE_TRAIN)
print("Validation step size:", STEP_SIZE_VAL)

train_generator.reset()
validation_generator.reset()

"""## Training"""

adam = Adam(lr=0.0005)
model.compile(optimizer=adam, loss="mse")

```




```

history = model.fit_generator(train_generator,
                             steps_per_epoch=STEP_SIZE_TRAIN,
                             validation_data=validation_generator,
                             validation_steps=STEP_SIZE_VAL,
                             epochs=15)

```

```

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

```

""""The model's success over the validation data is 80%. However, you can see that from the above figure, the training is stopped after 30th epoch. This may because of the low number of training samples or my model is not capable of learning such data. If you have an idea, please comment below.

```

## Tests
"""

```

```

model.evaluate_generator(validation_generator, steps=STEP_SIZE_VAL)

```

```

model.save_weights("weights.hdf5", overwrite = True)

```

""""Remember that, we had picked five dropped test samples for visual inspection. Here they are.""""

```

for idx, row in df.iloc[dropped_test_samples].iterrows():
    img = cv2.resize(cv2.imread("Dataset/" + row[0]) / 255.0, dsize=(WIDTH, HEIGHT))

    y_hat = model.predict(img.reshape(1, WIDTH, HEIGHT, 3)).reshape(-1) * WIDTH

    xt, yt = y_hat[0], y_hat[1]
    xb, yb = y_hat[2], y_hat[3]

    img = cv2.cvtColor(img.astype(np.float32), cv2.COLOR_BGR2RGB)
    image = cv2.rectangle(img, (xt, yt), (xb, yb), (0, 0, 255), 1)
    plt.imshow(image)
    plt.show()

```

```

""""#The End""""

```

OPENCV AND FIREBASE

```

import cv2
import numpy as np
import time
from time import sleep
from copy import deepcopy
from PIL import Image
import pytesseract as tess
import re

```

```

model = r"D:\Program Files\Minor\weights.h5"
plate_det_mask = r"D:\Program Files\Minor\plate_data.xml"

ocr = ""
tess.pytesseract.tesseract_cmd = r'D:\Program Files\Tesseract-OCR\tesseract.exe'
cap = cv2.VideoCapture(0)
while True:
    _, frame = cap.read()
    blurred_frame = cv2.GaussianBlur(frame, (5, 5), 0)
    hsv = cv2.cvtColor(blurred_frame, cv2.COLOR_BGR2HSV)
    offset = 15
    lower_white = np.array([0,0,255-offset])
    upper_white = np.array([255,offset,255])
    #lower_white = np.array([0, 0, 0])
    #upper_white = np.array([0, 0, 255])
    mask = cv2.inRange(hsv, lower_white, upper_white)
    #out = np.zeros_like(frame)
    #out[mask == 255] = frame[mask == 255]
    contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
    color1 = (255, 0, 0)
    color2 = (0, 255, 255)
    color3 = (0, 0, 255)

    ""
    idx = ... # The index of the contour that surrounds your object
    mask = np.zeros_like(frame) # Create mask where white is what we want, black otherwise
    cv2.drawContours(mask, contours, idx, 255, -1) # Draw filled contour in mask
    out = np.zeros_like(frame) # Extract out the object and place into output image
    out[mask == 255] = img[mask == 255]
    ""

# Show the output image
#cv2.imshow('Output', out)
#cv2.waitKey(0)
#cv2.destroyAllWindows()

max_area=0
for contour in contours:
    #print(contour)
    area = cv2.contourArea(contour)
    if area>max_area and area>8500:
        (x, y, w, h) = cv2.boundingRect(contour)
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)
        cv2.rectangle(mask, (x, y), (x + w, y + h), (0, 0, 255), 2)

        #cv2.drawContours(frame, contour, -1, (255, 0, 0), 3)
        cv2.putText(frame, "Plate detected", (x+w,y+h), cv2.FONT_HERSHEY_SIMPLEX,0.5,
color1, 2)
        cv2.putText(frame, "Applying OCR", (x+w,y+h+20), cv2.FONT_HERSHEY_SIMPLEX,0.5,
color2, 2)

```

```

buffer = 0
crop_ocr = Image.fromarray(frame[x - buffer : x+w+buffer, y- buffer: y+h+buffer])
full_ocr = Image.fromarray(frame)
found=False
print("waiting for pytesseract")
try:
    print("entered tess loop")
    text1=tess.image_to_string(crop_ocr)
    check1=re.findall(r"[A-Z][A-Z]\s*[A-Z0-9]+\s*[A-Z0-9]+\s*[0-9]{4}",text1)
    text2=tess.image_to_string(full_ocr)
    ocr=re.findall( r"[A-Z][A-Z]\s*[A-Z0-9]+\s*[A-Z0-9][A-Z]\s*[0-9]{4}",ocr)
    #print("1",check1)
    print(ocr)

    if check2:
        found = True
        #print("Detected Text : ",check2)
        ocr = check2
        ocr = ocr.replace(" ", "")
        print("Welcome, Your plate no. is ",ocr)

    #else:
    #    print("Detected Text : ",check1)
    #    ocr = check1
    #    ocr = ocr.replace(" ", "")
    #    print(ocr)
    #    found = True
    if found==True:
        print("Printing OCR on feed")
        cv2.putText(frame, "Successful, uploading to firebase", (x+w,y+h+40),
cv2.FONT_HERSHEY_SIMPLEX,0.5, color3, 2)
        cv2.putText(frame, "Welcome to Simply Park", (x,y+h+40),
cv2.FONT_HERSHEY_SIMPLEX,0.5, color3, 2)
        cv2.putText(frame, ocr, (x,y+h+60), cv2.FONT_HERSHEY_SIMPLEX,0.5, color3, 2)
        break
    else:
        continue

except:
    break
    print("OCR Failed")
max_area=area
print("uploading")
import firebase_admin
from firebase_admin import credentials, firestore
import datetime

cred = credentials.Certificate("./serviceAccountKey.json")
app = firebase_admin.initialize_app(cred)

db = firestore.client()

```

```

user_id = "

# Japman - 104885248578748957962
# Japneet - 102559740660975769254

num_plt = ocr

# 1. DL8CAT9393
# 2. DL12CE7693
# 3.
# 4. DL8CAK6278

# => returns user id to corresponding number plate
docs = db.collection('number_plate').where(num_plt, '==', num_plt).limit(1).stream()

for doc in docs:
    print(f'{doc.id} => {doc.to_dict()}')
    user_id = doc.id
    print(user_id)

#adding first data
doc_ref =
db.collection('logs').document(user_id).collection('logs').document('Q12w'+num_plt)

# *****----- On Entrance: -----*****

doc_ref.set({
    'entry_timestamp': str(datetime.datetime.now()),
    'exit_timestamp': "",
    'number_plate': num_plt ,
    'place': 'Central Mall1',
    'paid': ""
})
#cropped = cap[70:170, 440:540]
#cv2.imshow("cropped", cropped)
#cv2.imwrite("plate.png", cropped)
#cv2.waitKey(0)
#print(cv2.drawContours(frame, contour, -1, (0, 255, 0), 3))
cv2.imshow("Plate Detector", frame)
cv2.imshow("OCR Detector", mask)
key = cv2.waitKey(1)
if key == 27:
    cv2.waitKey(0)
    break

cap.release()
cv2.destroyAllWindows()

```

App code:

main.dart

```
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:minor_project/Pages/AuthenticationPage.dart';
```

```
Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(MyApp());
}
```

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Simply Parking',
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        primarySwatch: Colors.deepPurple,
        accentColor: Colors.teal,
      ),
      home: Login(),
    );
  }
}
```

Models

credit_card.dart

```
import 'dart:async';
class CreditCardBloc {
  final ccInputController = StreamController<String>();
  final expInputController = StreamController<String>();
  final cvvInputController = StreamController<String>();
  final nameInputController = StreamController<String>();

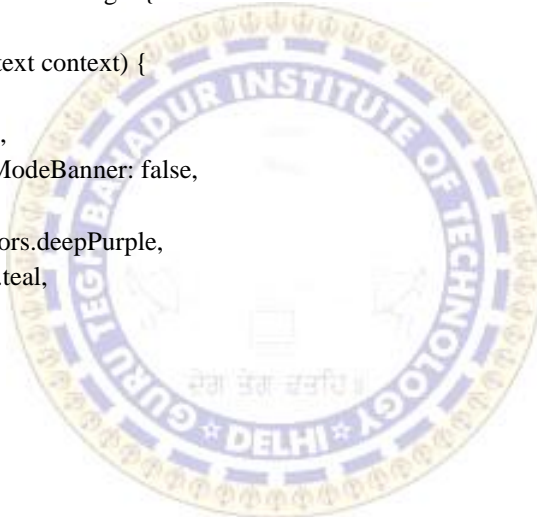
  Sink<String> get ccInputSink => ccInputController.sink;

  Sink<String> get expInputSink => expInputController.sink;

  Sink<String> get cvvInputSink => cvvInputController.sink;

  Sink<String> get nameInputSink => nameInputController.sink;

  final ccOutputController = StreamController<String>();
  final expOutputController = StreamController<String>();
```



```

final cvvOutputController = StreamController<String>();
final nameOutputController = StreamController<String>();

Stream<String> get ccOutputStream => ccOutputController.stream;

Stream<String> get expOutputStream => expOutputController.stream;

Stream<String> get cvvOutputStream => cvvOutputController.stream;

Stream<String> get nameOutputStream => nameOutputController.stream;

CreditCardBloc() {
  ccInputController.stream.listen(onCCInput);
  expInputController.stream.listen(onExpInput);
  cvvInputController.stream.listen(onCvvInput);
  nameInputController.stream.listen(onNameInput);
}

onCCInput(String input) {
  ccOutputController.add(input.toString());
}

onExpInput(String input) {
  expOutputController.add(input);
}

onCvvInput(String input) {
  cvvOutputController.add(input);
}

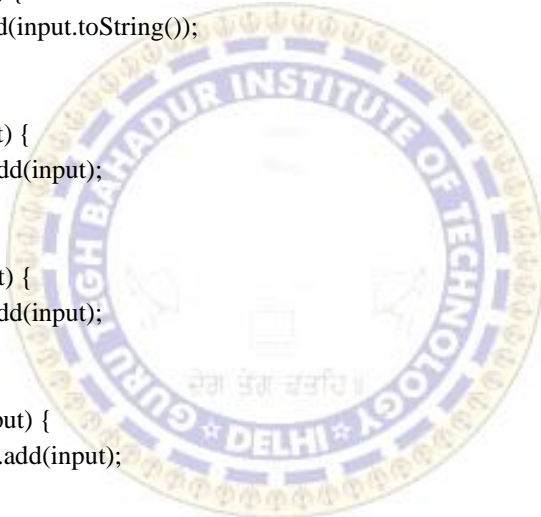
onNameInput(String input) {
  nameOutputController.add(input);
}

void ccFormat(String s) {
  print(s);
  ccInputSink.add(s);
}

void dispose() {
  ccInputController?.close();
  cvvInputController?.close();
  expInputController?.close();
  nameInputController?.close();
}
}

numberPlateDetails.dart
import 'package:flutter/material.dart';
class NumberPlateDetails {
  String ownerName;
  String registrationNumber;

```



```

String model;
String vehicleClass;
String fuelType;
String regDate;
String chassisNumber;
String engineNumber;
String fitnessUpto;
String insuranceExpiry;
String registeringAuthority;

```

```

NumberPlateDetails({this.ownerName, this.registrationNumber, this.model, this.vehicleClass,
this.fuelType, this.regDate, this.chassisNumber, this.engineNumber, this.fitnessUpto,
this.insuranceExpiry, this.registeringAuthority});

```

```

NumberPlateDetails.fromJson(Map<String, dynamic> json) {
  ownerName = json['Owner Name'];
  registrationNumber = json['Registration Number'];
  model = json['Model'];
  vehicleClass = json['Class'];
  fuelType = json['Fuel Type'];
  regDate = json['Reg Date'];
  chassisNumber = json['Chassis Number'];
  engineNumber = json['Engine Number'];
  fitnessUpto = json['Fitness Upto'];
  insuranceExpiry = json['Insurance expiry'];
  registeringAuthority = json['Registering Authority'];
}

```

```

Map<String, dynamic> toJson() {
  final Map<String, dynamic> data = new Map<String, dynamic>();
  data['Owner Name'] = this.ownerName;
  data['Registration Number'] = this.registrationNumber;
  data['Model'] = this.model;
  data['Class'] = this.vehicleClass;
  data['Fuel Type'] = this.fuelType;
  data['Reg Date'] = this.regDate;
  data['Chassis Number'] = this.chassisNumber;
  data['Engine Number'] = this.engineNumber;
  data['Fitness Upto'] = this.fitnessUpto;
  data['Insurance expiry'] = this.insuranceExpiry;
  data['Registering Authority'] = this.registeringAuthority;
  return data;
}
}

```

```

uidata.dart
import 'dart:ui';
import 'package:flutter/material.dart';
class UIData {
  //fonts
  static const String quickFont = "Quicksand";
  static const String ralewayFont = "Raleway";
}

```



```
static const String quickBoldFont = "Quicksand_Bold.otf";
static const String quickNormalFont = "Quicksand_Book.otf";
static const String quickLightFont = "Quicksand_Light.otf";
```

```
//colors
static List<Color> kitGradients = [
  // new Color.fromRGBO(103, 218, 255, 1.0),
  // new Color.fromRGBO(3, 169, 244, 1.0),
  // new Color.fromRGBO(0, 122, 193, 1.0),
  Colors.blueGrey.shade800,
  Colors.black87,
];
static List<Color> kitGradients2 = [
  Colors.cyan.shade600,
  Colors.blue.shade900
];
}
```

```
user.dart
import 'package:cloud_firestore/cloud_firestore.dart';
```

```
class User {
  final String id;
  final String username;
  final String email;
  final String photoUrl;
  final String displayName;
  final String bio;
```

```
User({
  this.id,
  this.username,
  this.email,
  this.photoUrl,
  this.displayName,
  this.bio,
});
```

```
factory User.fromDocument(DocumentSnapshot doc) {
  return User(
    id: doc.id,
    email: doc['email'],
    username: doc['username'],
    photoUrl: doc['photoUrl'],
    displayName: doc['displayName'],
    bio: doc['bio'],
  );
}
}
```

Pages




```

addNumberPlate.dart
import 'dart:convert';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:http/http.dart';
import 'package:minor_project/Models/numberPlateDetails.dart';
import 'package:minor_project/Pages/AuthenticationPage.dart';
import 'package:minor_project/widgets/progress.dart';

class AddNumberPlate extends StatefulWidget {
  @override
  _AddNumberPlateState createState() => _AddNumberPlateState();
}

class _AddNumberPlateState extends State<AddNumberPlate> {
  final numberPlateRef = FirebaseFirestore.instance
    .collection('number_plate')
    .doc(user.id)
    .collection("number_plate");
  final numberPlateDocRef =
    FirebaseFirestore.instance.collection('number_plate').doc(user.id);
  final _formKey = GlobalKey<FormState>();
  bool isLoading = false;
  String numberPlate;
  NumberPlateDetails _numberPlateDetails = NumberPlateDetails();

  Future<void> _getNumberPlateDetails() async {
    setState() {
      isLoading = true;
    });
    print("Making get request");
    // HR29AB3211
    String url =
      'http://shrouded-falls-48764.herokuapp.com/vehicle-info/$numberPlate';
    Map<String, String> headers = {
      "API-Key": "df81c10a960344fa87c287296b4f5c47"
    };
    print(url);
    try {
      Response response = await get(url, headers: headers);
      int statusCode = response.statusCode;
      Map<String, dynamic> data = json.decode(response.body);
      _numberPlateDetails = NumberPlateDetails.fromJson(data);
      print(response.body);
      print(_numberPlateDetails.ownerName);
      switch (statusCode) {
        case 200:
          print("LIST OF USER FILTERS ${_numberPlateDetails.ownerName}");
          if (_numberPlateDetails.ownerName.isNotEmpty) {
            final DateTime timestamp = DateTime.now();
            //navigate to other screen

```

```

numberPlateDocRef.set({ "$numberPlate": "$numberPlate" });
numberPlateRef.doc(numberPlate).set({
  "numberPlateNumber": numberPlate,
  "userid": user.id,
  "ownername": _numberPlateDetails.ownerName,
  "username": user.displayName,
  "registrationNumber": _numberPlateDetails.registrationNumber,
  "model": _numberPlateDetails.model,
  "vehicleClass": _numberPlateDetails.vehicleClass,
  "fuelType": _numberPlateDetails.fuelType,
  "regDate": _numberPlateDetails.fuelType,
  "chassisNumber": _numberPlateDetails.chassisNumber,
  "engineNumber": _numberPlateDetails.engineNumber,
  "fitnessUpto": _numberPlateDetails.fitnessUpto,
  "insuranceExpiry": _numberPlateDetails.insuranceExpiry,
  "registeringAuthority": _numberPlateDetails.registeringAuthority,
  "timestamp": timestamp
}).whenComplete() {
  Navigator.pop(context, numberPlate);
});
}
break;
default:
  Navigator.pop(context, numberPlate);
}
} catch (e) {
  print(e);
  Navigator.pop(context, numberPlate);
}
}

_showDialogForRelevance(BuildContext context, String msg) {
  Scaffold.of(context).showSnackBar(SnackBar(
    content: Text(msg),
    duration: Duration(seconds: 1),
  ));
}

@override
Widget build(BuildContext context) {
  // Build a Form widget using the _formKey created above.
  return Scaffold(
    appBar: AppBar(
      title: Text('Add Number Plate'),
      backgroundColor: Color(0xFF003051),
    ),
    body: Stack(
      children: [
        Container(
          height: double.infinity,
          width: double.infinity,
          child: Image(

```



```

width: double.infinity,
child: RaisedButton(
  color: Colors.green,
  onPressed: () {
    final form = _formKey.currentState;
    if (form.validate()) {
      form.save();
      _getNumberPlateDetails();
    }
  },
child: Text(
  'Submit',
  style: TextStyle(color: Colors.white),
),
shape: RoundedRectangleBorder(
  borderRadius:
    new BorderRadius.circular(18.0),
  side: BorderSide(color: Colors.green)),
),
)),
],
),
),
),
],
));
}
}

```



AuthenticationPage.dart
import 'dart:async';
import 'dart:ui';

```

import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:material_design_icons_flutter/material_design_icons_flutter.dart';
import 'package:google_sign_in/google_sign_in.dart';
import 'package:minor_project/Pages/create_account.dart';
import 'package:custom_bottom_navigation_bar/custom_bottom_navigation_bar.dart';
import 'package:custom_bottom_navigation_bar/custom_bottom_navigation_bar_item.dart';
import 'package:minor_project/Pages/logs.dart';
import 'package:minor_project/Pages/parking.dart';
import 'package:minor_project/Pages/profile.dart';
import 'package:minor_project/widgets/drawer.dart';

```

```

class Login extends StatefulWidget {
  @override
  _LoginState createState() => _LoginState();
}

```

```

final usersRef = FirebaseFirestore.instance.collection('users');
final DateTime timestamp = DateTime.now();

```

```

final GoogleSignIn googleSignIn = GoogleSignIn();
final GoogleSignInAccount user = googleSignIn.currentUser;

class _LoginState extends State<Login> {
  bool isAuth = false;
  PageController _pageController = PageController();

  // bool isShow = false;

  @override
  void initState() {
    // Detects when user signed in
    googleSignIn.onCurrentUserChanged.listen((account) {
      handleSignIn(account);
    }, onError: (err) {
      print('Error signing in: $err');
    });
    // Reauthenticate user when app is opened
    googleSignIn.signInSilently(suppressErrors: false).then((account) {
      handleSignIn(account);
    }).catchError((err) {
      print('Error signing in: $err');
    });
    // Timer(
    //   Duration(
    //     seconds: 3,
    //   ), () {
    //     if (this.mounted)
    //       setState(() {
    //         isShow = true;
    //       });
    //   });
    super.initState();
  }

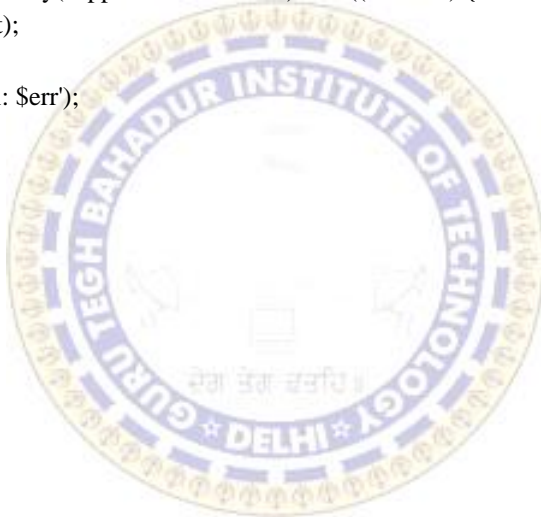
  createUserInFirestore() async {
    // 1) check if user exists in users collection in database (according to their id)
    DocumentSnapshot doc = await usersRef.doc(user.id).get();

    if (!doc.exists) {
      // 2) if the user doesn't exist, then we want to take them to the create account page
      final mapData = await Navigator.push(
        context, MaterialPageRoute(builder: (context) => CreateAccount()));

      String username = mapData["username"];
      String phoneNumber = mapData["phoneNumber"];

      // 3) get username from create account, use it to make new user document in users collection
      usersRef.doc(user.id).set({
        "id": user.id,
        "username": username,
        "phoneNumber": phoneNumber,
      });
    }
  }
}

```



```

        "photoUrl": user.photoUrl,
        "email": user.email,
        "displayName": user.displayName,
        "timestamp": timestamp
    });

    doc = await usersRef.doc(user.id).get();
  }

  // currentUser = User.fromDocument(doc);
}

handleSignIn(GoogleSignInAccount account) async {
  if (account != null) {
    await createUserInFirestore();
    if (this.mounted)
      setState(() {
        isAuth = true;
      });
  } else {
    if (this.mounted)
      setState(() {
        isAuth = false;
        // isShow = true;
      });
  }
}

login() {
  googleSignIn.signIn();
}

logout() {
  googleSignIn.signOut();
}

Scaffold buildAuthScreen({double width, double height}) {
  return Scaffold(
    backgroundColor: Colors.blueAccent,
    body: Container(
      width: width,
      height: height,
      alignment: Alignment.center,
      child: Stack(
        children: <Widget>[
          Container(
            width: width,
            height: height,
            color: Colors.grey,
            child: Image.asset(
              "assets/images/parking.jpg",
              fit: BoxFit.cover,

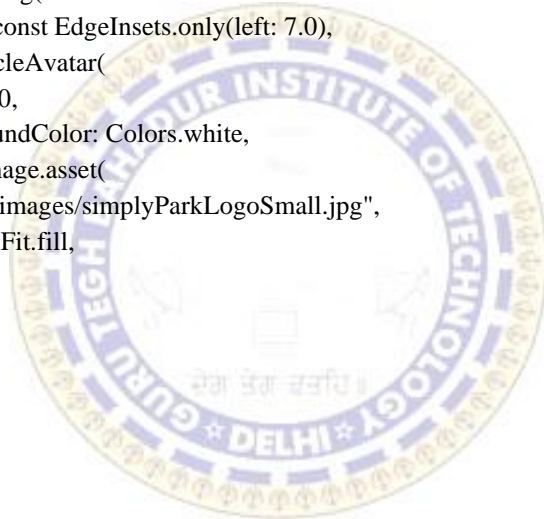
```



```

    ),
  ),
  Positioned(
    top: height * 0.2,
    left: width * 0.15,
    child: Column(
      mainAxisAlignment: MainAxisAlignment.min,
      crossAxisAlignment: CrossAxisAlignment.center,
      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
        Container(
          height: 120,
          width: 140,
          alignment: Alignment.center,
          decoration: BoxDecoration(
            color: Colors.white, shape: BoxShape.circle),
          child: ClipOval(
            clipBehavior: Clip.hardEdge,
            child: Padding(
              padding: const EdgeInsets.only(left: 7.0),
              child: CircleAvatar(
                radius: 50,
                backgroundColor: Colors.white,
                child: Image.asset(
                  "assets/images/simplyParkLogoSmall.jpg",
                  fit: BoxFit.fill,
                ),
              ),
            ),
          ),
        ),
        SizedBox(
          height: 30,
        ),
        Text(
          'Simply Parking',
          style: TextStyle(
            color: Colors.white,
            fontSize: 40,
            fontWeight: FontWeight.bold),
        )
      ],
    ),
  ),
  Positioned(
    top: height * 0.65,
    child: Container(
      width: width,
      height: height * 0.35,
      decoration: BoxDecoration(
        color: Colors.white,
        borderRadius: BorderRadius.vertical(

```



```

    top: Radius.circular(50),
  ),
),
child: Center(
  child: Column(
    mainAxisAlignment: MainAxisAlignment.min,
    children: <Widget>[
      Text(
        'Login',
        style: TextStyle(
          fontWeight: FontWeight.bold, fontSize: 24),
      ),
      SizedBox(
        height: 20,
      ),
      GestureDetector(
        onTap: () {
          login();
        },
        child: Container(
          height: 50,
          margin: const EdgeInsets.symmetric(horizontal: 30),
          decoration: BoxDecoration(
            borderRadius: BorderRadius.circular(20),
            border: Border.all(color: Colors.black),
          ),
        ),
        child: Row(
          mainAxisAlignment: MainAxisAlignment.start,
          children: <Widget>[
            SizedBox(width: 20),
            Container(
              width: 40,
              height: 40,
              child: Image.asset(
                "assets/images/google_logo.jpg"),
            ),
            SizedBox(
              width: 40,
            ),
            Text(
              'Continue with Google',
              style: TextStyle(fontSize: 16),
            ),
          ],
        ),
      ),
      SizedBox(
        height: 10,
      ),
      GestureDetector(
        onTap: () {},

```



```

child: Container(
  height: 50,
  margin: const EdgeInsets.symmetric(horizontal: 30),
  decoration: BoxDecoration(
    color: Colors.blue,
    borderRadius: BorderRadius.circular(20),
    border: Border.all(color: Colors.black),
  ),
  child: Row(
    mainAxisAlignment: MainAxisAlignment.start,
    children: <Widget>[
      SizedBox(width: 20),
      Icon(
        MdiIcons.facebook,
        color: Colors.white,
        size: 30,
      ),
      SizedBox(
        width: 40,
      ),
      Text(
        'Continue with Facebook',
        style: TextStyle(
          color: Colors.white, fontSize: 16),
      ),
    ],
  ),
),
),
),
),
),
SizedBox(
  height: 20,
),
Text(
  'Privacy Policy',
  style: TextStyle(
    fontSize: 14,
  ),
),
),
),
),
),
),
),
);
}

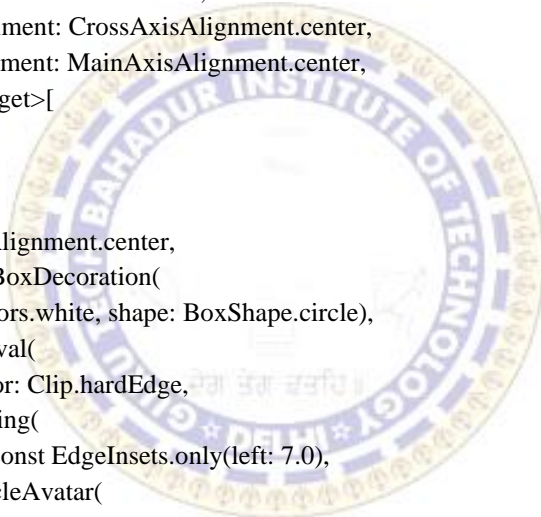
```

```
Scaffold buildAuthScreen2({double width, double height}) {
  return Scaffold(
    body: Container(
```

```

width: width,
height: height,
alignment: Alignment.center,
child: Stack(
  children: <Widget>[
    Container(
      width: width,
      height: height,
      color: Colors.grey,
      child: Image.asset(
        "assets/images/parking.jpg",
        fit: BoxFit.cover,
      ),
    ),
    Positioned(
      top: height * 0.2,
      left: width * 0.15,
      child: Column(
        mainAxisAlignment: MainAxisAlignment.min,
        crossAxisAlignment: CrossAxisAlignment.center,
        mainAxisSize: MainAxisSize.min,
        children: <Widget>[
          Container(
            height: 120,
            width: 140,
            alignment: Alignment.center,
            decoration: BoxDecoration(
              color: Colors.white, shape: BoxShape.circle,
            ),
            child: ClipOval(
              clipBehavior: Clip.hardEdge,
              child: Padding(
                padding: const EdgeInsets.only(left: 7.0),
                child: CircleAvatar(
                  radius: 50,
                  backgroundColor: Colors.white,
                  child: Image.asset(
                    "assets/images/simplyParkLogoSmall.jpg",
                    fit: BoxFit.fill,
                  ),
                ),
              ),
            ),
          ),
          SizedBox(
            height: 30,
          ),
          Text(
            'SimplyParking',
            style: TextStyle(
              color: Colors.white,
              fontSize: 40,
              fontWeight: FontWeight.bold),
          ),
        ],
      ),
    ),
  ],
),

```



```

        )
      ],
    ),
  ),
],
),
),
);
}

```

```

Scaffold homeScreen() {
  return Scaffold(
    appBar: AppBar(
      backgroundColor: Color(0XFF003051),
      title: Text("HomeScreen"),
      centerTitle: true,
      actions: [
        Padding(
          padding: const EdgeInsets.all(8.0),
          child: InkWell(
            onTap: () {
              logout();
              setState() {
                isAuth = false;
              };
            },
            child: Icon(
              Icons.logout,
              size: 30.0,
            )),
        )
      ],
    ),
    drawer: NavigationDrawer(),
    body: PageView(
      controller: _pageController,
      physics: NeverScrollableScrollPhysics(),
      children: <Widget>[
        Parking(),
        Logs(),
        Profile(),
      ],
    ),
    bottomNavigationBar: CustomBottomNavigationBar(
      items: [
        CustomBottomNavigationBarItem(
          icon: Icons.car_repair,
          title: "Parking",
        ),
        CustomBottomNavigationBarItem(
          icon: Icons.book,
          title: "Logs",

```



```

    ),
    CustomBottomNavigationBarItem(
      icon: Icons.account_circle,
      title: "Profile",
    ),
  ],
  onTap: (index) {
    _pageController.animateToPage(index,
      curve: Curves.fastLinearToSlowEaseIn,
      duration: Duration(milliseconds: 600));
  },
),
);
}

@override
Widget build(BuildContext context) {
  double width = MediaQuery.of(context).size.width;
  double height = MediaQuery.of(context).size.height;
  return WillPopScope(
    onWillPop: () async => showDialog(
      context: context,
      builder: (context) => AlertDialog(
        title: Text('Are you sure you want to quit?'),
        actions: <Widget>[
          FlatButton(
            child: Text(
              'Yes',
              style: TextStyle(color: Colors.red),
            ),
            onPressed: () => Navigator.of(context).pop(true)),
          FlatButton(
            child: Text('No',
              style: TextStyle(color: Color(0XFF003051))),
            onPressed: () => Navigator.of(context).pop(false)),
        ]),
    child: isAuth
      ? homeScreen()
      :
      // : isShow ?
      buildAuthScreen(width: width, height: height),
  );
  // : buildAuthScreen2(width: width, height: height);
}
}

create_account.dart
import 'dart:async';
import 'package:flutter/material.dart';

class CreateAccount extends StatefulWidget {
  @override

```

```

_CreateAccountState createState() => _CreateAccountState();
}

class _CreateAccountState extends State<CreateAccount> {
  final _scaffoldKey = GlobalKey<ScaffoldState>();
  final _formKey = GlobalKey<FormState>();
  String username;
  String phoneNumber;

  submit() {
    final form = _formKey.currentState;

    if (form.validate()) {
      form.save();
      SnackBar snackbar = SnackBar(content: Text("Welcome $username!"));
      _scaffoldKey.currentState.showSnackBar(snackbar);
      Timer(Duration(seconds: 2), () {
        Navigator.pop(context, {"username":username,"phoneNumber":phoneNumber});
      });
    }
  }

  @override
  Widget build(BuildContext parentContext) {
    return Scaffold(
      key: _scaffoldKey,
      // appBar: header(context,
      //   titleText: "Set up your profile", removeBackButton: true),
      body: ListView(
        children: <Widget>[
          Container(
            child: Column(
              children: <Widget>[
                Padding(
                  padding: EdgeInsets.only(top: 25.0),
                  child: Center(
                    child: Text(
                      "Form",
                      style: TextStyle(fontSize: 25.0),
                    ),
                  ),
                ),
                ),
                Padding(
                  padding: EdgeInsets.all(16.0),
                  child: Container(
                    child: Form(
                      key: _formKey,
                      autovalidate: true,
                      child: Column(
                        crossAxisAlignment: CrossAxisAlignment.start,
                        children: [
                          SizedBox(

```

```

        height: 20.0,
      ),
      Padding(
        padding: const EdgeInsets.only(bottom: 10.0),
        child: Text("Enter Username"),
      ),
      TextFormField(
        validator: (val) {
          if (val.trim().length < 3 || val.isEmpty) {
            return "Username too short";
          } else if (val.trim().length > 12) {
            return "Username too long";
          } else {
            return null;
          }
        },
        onSave: (val) => username = val,
        decoration: InputDecoration(
          border: OutlineInputBorder(),
          labelText: "Username",
          labelStyle: TextStyle(fontSize: 15.0),
          hintText: "Must be at least 3 characters",
        ),
      ),
      SizedBox(
        height: 20.0,
      ),
      Padding(
        padding: const EdgeInsets.only(bottom: 10.0),
        child: Text("Enter Phone Number"),
      ),
      TextFormField(
        keyboardType: TextInputType.number,
        validator: (val) {
          if (val.trim().length < 10 || val.isEmpty) {
            return "Please enter valid phone number";
          } else if (val.trim().length > 10) {
            return "Please enter valid phone number";
          } else {
            return null;
          }
        },
        onSave: (val) => phoneNumber = val,
        decoration: InputDecoration(
          border: OutlineInputBorder(),
          labelText: "Phone Number",
          labelStyle: TextStyle(fontSize: 15.0),
          hintText: "Must be at least 10 characters",
        ),
      ),
    ],
  ),
),

```



```

appBar: AppBar(
  title: Text('${ widget.data['numberPlateNumber']}'),
  backgroundColor: Color(0XFF003051),
  centerTitle: true,
  actions: [
    IconButton(
      icon: Icon(
        Icons.delete,
        color: Colors.white,
      ),
      onPressed: () {
        FirebaseFirestore.instance
          .collection('number_plate')
          .doc(user.id)
          .collection("number_plate")
          .doc(widget.data['numberPlateNumber'])
          .delete();

        FirebaseFirestore.instance
          .collection('logs')
          .doc(user.id)
          .collection("logs")
          .doc('Q12w' + widget.data['numberPlateNumber'])
          .delete();

        FirebaseFirestore.instance
          .collection('logs')
          .doc(user.id)
          .collection(widget.data['numberPlateNumber'])
          .get()
          .then((snapshot) {
            for (DocumentSnapshot ds in snapshot.docs) {
              ds.reference.delete();
            }
          });

        // final delNumPlate = FirebaseFirestore.instance
        //   .collection('logs')
        //   .doc(user.id)
        //   .collection(widget.data['numberPlateNumber'])
        //   .where("number_plate",
        //     isEqualTo: widget.data['numberPlateNumber'])
        //   .snapshots();
        // delNumPlate.forEach((ele) {
        //   print(ele.docs);
        //   ele.docs.forEach((element) {
        //     print(element.id);
        //     FirebaseFirestore.instance
        //       .collection('logs')
        //       .doc(user.id)
        //       .collection("logs")
        //       .doc(element.id)

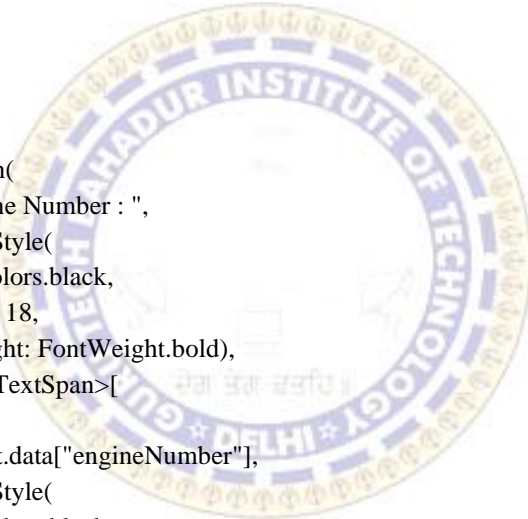
```



```

        //      .delete();
        //    });
        //  });
        Navigator.pop(context);
      })
    ],
  ),
  body: Padding(
    padding: const EdgeInsets.all(10.0),
    child: Container(
      height: MediaQuery.of(context).size.height,
      child: Material(
        elevation: 5.0,
        color: Colors.grey[300],
        child: Column(
          mainAxisAlignment: MainAxisAlignment.start,
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            SizedBox(
              height: 7,
              width: 10.0,
            ),
            RichText(
              text: TextSpan(
                text: "Engine Number : ",
                style: TextStyle(
                  color: Colors.black,
                  fontSize: 18,
                  fontWeight: FontWeight.bold),
                children: <TextSpan>[
                  TextSpan(
                    text: widget.data["engineNumber"],
                    style: TextStyle(
                      color: Colors.black,
                      fontSize: 18,
                      fontWeight: FontWeight.normal),
                  )
                ]),
            SizedBox(
              height: 7,
              width: 10.0,
            ),
            RichText(
              text: TextSpan(
                text: "Fuel Type : ",
                style: TextStyle(
                  color: Colors.black,
                  fontSize: 18,
                  fontWeight: FontWeight.bold),
                children: <TextSpan>[
                  TextSpan(
                    text: widget.data["fuelType"],

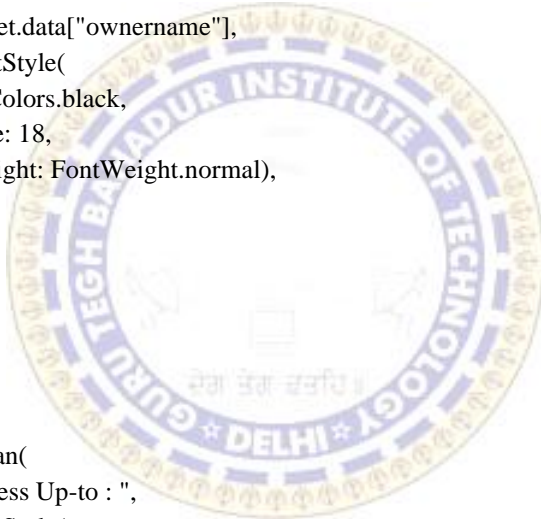
```



```

        style: TextStyle(
          color: Colors.black,
          fontSize: 18,
          fontWeight: FontWeight.normal),
      ),
    )),
  SizedBox(
    height: 7,
    width: 10.0,
  ),
  RichText(
    text: TextSpan(
      text: "Owner Name : ",
      style: TextStyle(
        color: Colors.black,
        fontSize: 18,
        fontWeight: FontWeight.bold),
      children: <TextSpan>[
        TextSpan(
          text: widget.data["ownername"],
          style: TextStyle(
            color: Colors.black,
            fontSize: 18,
            fontWeight: FontWeight.normal),
        ),
      ],
    ),
    SizedBox(
      height: 7,
      width: 10.0,
    ),
    RichText(
      text: TextSpan(
        text: "Fitness Up-to : ",
        style: TextStyle(
          color: Colors.black,
          fontSize: 18,
          fontWeight: FontWeight.bold),
        children: <TextSpan>[
          TextSpan(
            text: widget.data["fitnessUpto"],
            style: TextStyle(
              color: Colors.black,
              fontSize: 18,
              fontWeight: FontWeight.normal),
          ),
        ],
      ),
      SizedBox(
        height: 7,
        width: 10.0,
      ),
      RichText(
        text: TextSpan(

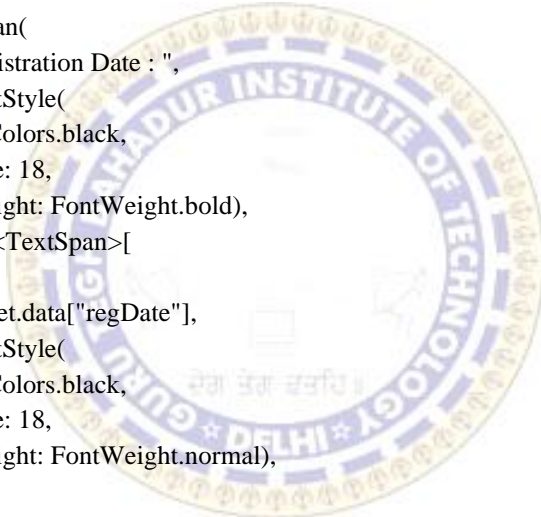
```



```

      text: "Chassis Number : ",
      style: TextStyle(
        color: Colors.black,
        fontSize: 18,
        fontWeight: FontWeight.bold),
      children: <TextSpan>[
        TextSpan(
          text: widget.data["chassisNumber"],
          style: TextStyle(
            color: Colors.black,
            fontSize: 18,
            fontWeight: FontWeight.normal),
        )
      ])),
    SizedBox(
      height: 7,
      width: 10.0,
    ),
    RichText(
      text: TextSpan(
        text: "Registration Date : ",
        style: TextStyle(
          color: Colors.black,
          fontSize: 18,
          fontWeight: FontWeight.bold),
        children: <TextSpan>[
          TextSpan(
            text: widget.data["regDate"],
            style: TextStyle(
              color: Colors.black,
              fontSize: 18,
              fontWeight: FontWeight.normal),
          )
        ])),
    SizedBox(
      height: 7,
      width: 10.0,
    ),
    RichText(
      text: TextSpan(
        text: "Vehicle Class : ",
        style: TextStyle(
          color: Colors.black,
          fontSize: 18,
          fontWeight: FontWeight.bold),
        children: <TextSpan>[
          TextSpan(
            text: widget.data["vehicleClass"],
            style: TextStyle(
              color: Colors.black,
              fontSize: 18,
              fontWeight: FontWeight.normal),
          )
        ]
      )
    )
  ),

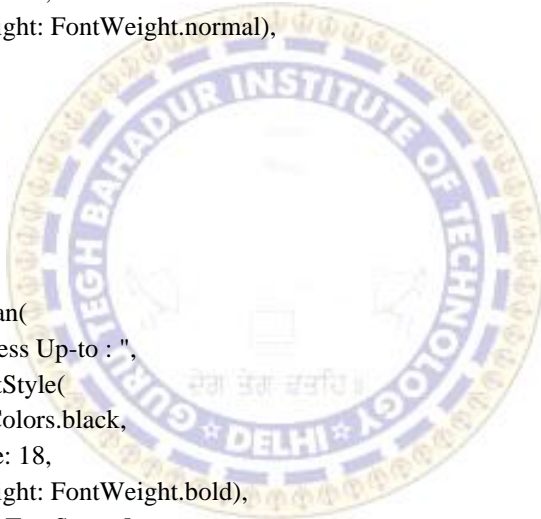
```



```

    )
  )),
  SizedBox(
    height: 7,
    width: 10.0,
  ),
  RichText(
    text: TextSpan(
      text: "Registration Number : ",
      style: TextStyle(
        color: Colors.black,
        fontSize: 18,
        fontWeight: FontWeight.bold),
      children: <TextSpan>[
        TextSpan(
          text: widget.data["registrationNumber"],
          style: TextStyle(
            color: Colors.black,
            fontSize: 18,
            fontWeight: FontWeight.normal),
        )
      ]
    )),
  SizedBox(
    height: 7,
    width: 10.0,
  ),
  RichText(
    text: TextSpan(
      text: "Fitness Up-to : ",
      style: TextStyle(
        color: Colors.black,
        fontSize: 18,
        fontWeight: FontWeight.bold),
      children: <TextSpan>[
        TextSpan(
          text: widget.data["fitnessUpto"],
          style: TextStyle(
            color: Colors.black,
            fontSize: 18,
            fontWeight: FontWeight.normal),
        )
      ]
    )),
  SizedBox(
    height: 7,
    width: 10.0,
  ),
  RichText(
    text: TextSpan(
      text: "Insurance Expiry : ",
      style: TextStyle(
        color: Colors.black,
        fontSize: 18,

```



```

        fontWeight: FontWeight.bold),
        children: <TextSpan>[
        TextSpan(
            text: widget.data["insuranceExpiry"],
            style: TextStyle(
                color: Colors.black,
                fontSize: 18,
                fontWeight: FontWeight.normal),
        )
        ])),
    SizedBox(
        height: 7,
        width: 10.0,
    ),
    RichText(
        text: TextSpan(
            text: "Registering Authority : ",
            style: TextStyle(
                color: Colors.black,
                fontSize: 18,
                fontWeight: FontWeight.bold),
            children: <TextSpan>[
            TextSpan(
                text: widget.data["registeringAuthority"],
                style: TextStyle(
                    color: Colors.black,
                    fontSize: 18,
                    fontWeight: FontWeight.normal),
            )
            ])),
    SizedBox(
        height: 7,
        width: 10.0,
    ),
    RichText(
        text: TextSpan(
            text: "Model : ",
            style: TextStyle(
                color: Colors.black,
                fontSize: 18,
                fontWeight: FontWeight.bold),
            children: <TextSpan>[
            TextSpan(
                text: widget.data["model"],
                style: TextStyle(
                    color: Colors.black,
                    fontSize: 18,
                    fontWeight: FontWeight.normal),
            )
            ])),
    ],
),

```

```

    ),
    ),
  ),
  floatingActionButton: FloatingActionButton(
    onPressed: () {
      Navigator.push(
        context,
        MaterialPageRoute(
          builder: (context) => RecentLogs(
            numberPlate: widget.data['numberPlateNumber'],
          )),
      );
    },
    child: Icon(Icons.update),
    backgroundColor: Color(0XFF003051),
  ),
);
}
}

```

logs.dart

```

import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:minor_project/Pages/AuthenticationPage.dart';
import 'package:minor_project/widgets/log_tile.dart';

```

```

class Logs extends StatefulWidget {
  @override
  _LogsState createState() => _LogsState();
}

```

```

class _LogsState extends State<Logs> {
  // int _payRate = 30;
  // double calChg = 0;
  // bool _notParked = true;
  final logsRef = FirebaseFirestore.instance
    .collection('logs')
    .doc(user.id)
    .collection("logs");

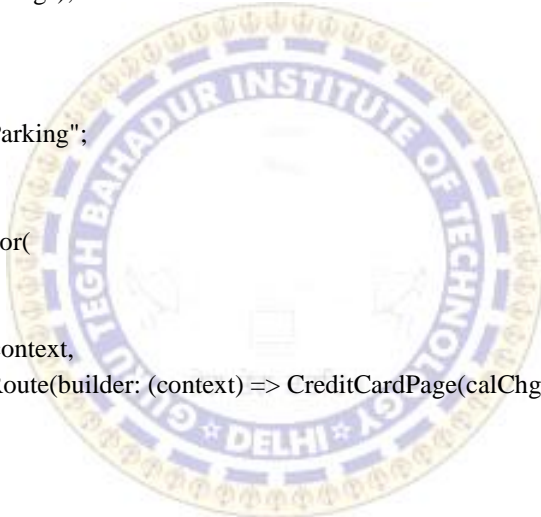
  // Widget listItem(BuildContext context, DocumentSnapshot document) {
  //   // String entry =
  //   //   DateTime.parse(document["entry_timestamp"].toDate().toString())
  //   //     .toString()
  //   //     .substring(0, 19);
  //   // String exit = "";
  //   // try {
  //   //   exit = DateTime.parse(document["exit_timestamp"]?.toDate().toString())
  //   //     .toString()
  //   //     .substring(0, 19);
  //   //   _notParked = true;
  //   // } catch (e) {
  //   //   print("false false*****");
  // }

```

```

// // exit = "Vehicle in Parking";
// // _notParked = false;
// // }
// // print("*****$entry");
// String entry = document["entry_timestamp"];
// print("*****$entry");
//
// int entryHr = int.parse(entry.substring(11, 13));
// int entryMn = int.parse(entry.substring(14, 16));
// String exit = "";
// int exitHr = 0;
// int exitMn = 0;
// try {
//   exit = document["exit_timestamp"];
//   exitHr = int.parse(exit.substring(11, 13));
//   exitMn = int.parse(exit.substring(14, 16));
//   calChg = (exitHr - entryHr) * _payRate +
//     ((exitMn - entryMn) * (_payRate / 60));
//   print("Charge : $calChg");
//   _notParked = true;
// } catch (e) {
//   print(e);
//   exit = "Vehicle in Parking";
//   _notParked = false;
// }
// return GestureDetector(
//   onTap: () {
//     if (_notParked) {
//       Navigator.push(context,
//         MaterialPageRoute(builder: (context) => CreditCardPage(calChg)));
//     }
//   },
//   child: Padding(
//     padding: EdgeInsets.only(left: 15.0, right: 15.0, top: 20.0),
//     child: ClipRRect(
//       borderRadius: BorderRadius.circular(20.0),
//       child: Container(
//         height: 200,
//         width: MediaQuery.of(context).size.width / 2,
//         color: Colors.grey[200],
//         child: Padding(
//           padding: const EdgeInsets.only(
//             top: 15.0, right: 15.0, bottom: 15.0, left: 5.0),
//           child: Column(
//             crossAxisAlignment: CrossAxisAlignment.start,
//             children: [
//               Row(
//                 mainAxisAlignment: MainAxisAlignment.spaceBetween,
//                 mainAxisSize: MainAxisSize.max,
//                 children: [
//                   Container(
//                     height: 40,

```



```

//      width: MediaQuery.of(context).size.width / 3,
//      padding: const EdgeInsets.all(10),
//      decoration: BoxDecoration(
//        color: Colors.black,
//        borderRadius: BorderRadius.circular(20)),
//      child: Center(
//        child: Text(
//          '${document["number_plate"]}',
//          // '${document["place"]}',
//          style: TextStyle(color: Colors.white),
//          overflow: TextOverflow.ellipsis,
//          maxLines: 1,
//        ),
//      ),
//    ),
//    _notParked
//    ? Row(
//      children: [
//        Text(
//          '₹',
//          style: TextStyle(
//            color: Colors.green,
//            fontSize: 15,
//            fontWeight: FontWeight.bold,
//          ),
//        ),
//        Text(
//          "$calChg",
//          style: TextStyle(
//            color: Colors.green,
//            fontSize: 15,
//            fontWeight: FontWeight.bold,
//          ),
//        ),
//      ],
//    )
//    : Container(
//      height: 0.0,
//      width: 0.0,
//    ),
//  ],
// ),
// SizedBox(
//   height: 5,
// ),
// Padding(
//   padding: const EdgeInsets.only(left: 5.0),
//   child: SizedBox(
//     width: MediaQuery.of(context).size.width / 1.2,
//     child: Text(
//       'Place: ${document["place"]}',
//       style: TextStyle(

```



```

//      ),
//      Padding(
//        padding: const EdgeInsets.only(left: 5.0),
//        child: Text(
//          'Exit Timestamp:',
//          style: TextStyle(
//            fontSize: 15,
//            color: Colors.black,
//            fontWeight: FontWeight.bold,
//          ),
//          overflow: TextOverflow.ellipsis,
//          softWrap: false,
//          maxLines: 1,
//        ),
//      ),
//      SizedBox(
//        height: 3,
//      ),
//      _notParked
//      ? Padding(
//        padding: const EdgeInsets.only(left: 5.0),
//        child: SizedBox(
//          width: MediaQuery.of(context).size.width / 1.2,
//          child: Text(
//            "Date: ${exit?.substring(0, 10)}   Time: ${exit?.substring(11, 19)}",
//            style: TextStyle(
//              fontSize: 15,
//              color: Colors.black,
//              fontWeight: FontWeight.bold,
//            ),
//            maxLines: 1,
//            softWrap: false,
//            overflow: TextOverflow.ellipsis,
//          ),
//        ),
//      )
//      : Padding(
//        padding: const EdgeInsets.only(left: 5.0),
//        child: SizedBox(
//          width: MediaQuery.of(context).size.width / 1.2,
//          child: Text(
//            exit,
//            style: TextStyle(
//              fontSize: 15,
//              color: Colors.black,
//              fontWeight: FontWeight.bold,
//            ),
//            maxLines: 1,
//            softWrap: false,
//            overflow: TextOverflow.ellipsis,
//          ),
//        ),
//      ),

```

```

//      ),
//      _notParked
//      ? Padding(
//          padding: const EdgeInsets.only(left: 5.0),
//          child: Text(
//              "Parked for: ${exitHr - entryHr} hr. and ${exitMn - entryMn} min.",
//              style: TextStyle(
//                  color: Colors.green.shade900,
//                  fontSize: 15,
//                  fontWeight: FontWeight.bold,
//              ),
//          ),
//      ),
//      ),
//      ),
//      ),
//      ),
//      ),
//      ),
//      );
// }

```

@override

```

Widget build(BuildContext context) {
  return Scaffold(
    body: StreamBuilder(
      stream: logsRef.orderBy('entry_timestamp', descending: true)?.snapshots(),
      builder: (context, snapshot) {
        print(
          "*****entered snapshot $snapshot");
        if (!snapshot.hasData) {
          return Center(
            child: Text("Loading Logs....."),
          );
        }
        return ListView.builder(
          itemCount: snapshot.data.documents.length,
          scrollDirection: Axis.vertical,
          itemBuilder: (context, index) {
            DocumentSnapshot document = snapshot.data.documents[index];
            return LogTile(
              entryTimestamp: document["entry_timestamp"],
              exitTimestamp: document['exit_timestamp'],
              numberPlate: document["number_plate"],
              place: document['place'],
              paid: document['paid'] ?? "",
              recent: false,
            );
          }
        );
      }
    );
  }
}

```

```

    ));
  },
));
}
}

parking.dart
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:minor_project/Pages/AuthenticationPage.dart';
import 'package:minor_project/Pages/addNumberPlate.dart';
import 'package:minor_project/Pages/detailedNumberPlate.dart';

```

```

class Parking extends StatefulWidget {
  @override
  _ParkingState createState() => _ParkingState();
}

```

```

class _ParkingState extends State<Parking> {
  final numberPlateRef = FirebaseFirestore.instance
    .collection('number_plate')
    .doc(user.id)
    .collection("number_plate");
  String numberPlate;

  Widget listItem(BuildContext context, DocumentSnapshot document) {
    return Padding(
      padding: EdgeInsets.only(left: 15.0, right: 15.0, top: 20.0),
      child: ClipRRect(
        borderRadius: BorderRadius.circular(20.0),
        child: Container(
          height: 130,
          width: MediaQuery.of(context).size.width / 2,
          color: Colors.grey[200],
          child: InkWell(
            onTap: () => Navigator.push(
              context,
              MaterialPageRoute(
                builder: (context) => DetailedNumPltSrn(document.data()),
              ),
            child: Row(
              children: [
                Padding(
                  padding: EdgeInsets.only(
                    top: 15.0, right: 15.0, bottom: 15.0),
                  child: Column(
                    crossAxisAlignment: CrossAxisAlignment.start,
                    children: [
                      Padding(
                        padding: EdgeInsets.only(left: 5.0),
                        child: Container(
                          height: 40,

```

[illegible]

```

        ),
      ),
    ],
  ),
),
),
),
);
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    floatingActionButton: FloatingActionButton(
      onPressed: () async {
        numberPlate = await Navigator.push(context,
          MaterialPageRoute(builder: (context) => AddNumberPlate()));
      },
      child: Icon(Icons.add),
      backgroundColor: Color(0XFF003051),
    ),
    body: StreamBuilder(
      stream: numberPlateRef
        .orderBy('timestamp', descending: false)
        .snapshots(),
      builder: (context, snapshot) {
        if (!snapshot.hasData) {
          return Center(
            child: Text("Loading....."),
          );
        }
        return ListView.builder(
          itemCount: snapshot.data.documents.length,
          scrollDirection: Axis.vertical,
          itemBuilder: (context, index) {
            return listItem(context, snapshot.data.documents[index]);
          });
      },
    ));
}
}

```

```

payment.dart
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:flutter_masked_text/flutter_masked_text.dart';
import 'package:font_awesome_flutter/font_awesome_flutter.dart';
import 'package:minor_project/Models/credit_card.dart';
import 'package:minor_project/Models/ui_data.dart';
import 'package:minor_project/Pages/paymentDone.dart';
import 'package:minor_project/widgets/profile_tile.dart';

```

```

import 'AuthenticationPage.dart';

class CreditCardPage extends StatefulWidget {
  CreditCardPage({this.amountCharge, this.numberPlate});

  final double amountCharge;
  final String numberPlate;

  @override
  _CreditCardPageState createState() => _CreditCardPageState();
}

class _CreditCardPageState extends State<CreditCardPage> {
  BuildContext _context;

  CreditCardBloc cardBloc;

  MaskedTextController ccMask =
    MaskedTextController(mask: "0000 0000 0000 0000");

  MaskedTextController expMask = MaskedTextController(mask: "00/00");

  Widget bodyData() => SingleChildScrollView(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.start,
      children: <Widget>[creditCardWidget(), fillEntries()],
    ),
  );

  Widget creditCardWidget() {
    var deviceSize = MediaQuery.of(_context).size;
    return Container(
      height: deviceSize.height * 0.3,
      color: Colors.grey.shade300,
      child: Padding(
        padding: const EdgeInsets.all(8.0),
        child: Card(
          clipBehavior: Clip.antiAlias,
          elevation: 3.0,
          child: Stack(
            fit: StackFit.expand,
            children: <Widget>[
              Container(
                decoration: BoxDecoration(
                  gradient: LinearGradient(colors: UIData.kitGradients)),
              ),
              Opacity(
                opacity: 0.1,
                child: Image.asset(
                  "assets/images/map.png",
                  fit: BoxFit.cover,
                ),
              ),
            ],
          ),
        ),
      ),
    );
  }
}

```

```

),
MediaQuery.of(_context).orientation == Orientation.portrait
? cardEntries()
: FittedBox(
  child: cardEntries(),
),
Positioned(
  right: 10.0,
  top: 10.0,
  child: Icon(
    FontAwesomeIcons.ccVisa,
    size: 30.0,
    color: Colors.white,
  ),
),
Positioned(
  right: 10.0,
  bottom: 10.0,
  child: StreamBuilder<String>(
    stream: cardBloc.nameOutputStream,
    initialData: user.displayName.toUpperCase(),
    builder: (context, snapshot) => Text(
      snapshot.data.length > 0 ? snapshot.data : "Your Name",
      style: TextStyle(
        color: Colors.white,
        fontFamily: UIData.ralewayFont,
        fontSize: 20.0),
    ),
  ),
),
],
),
),
);
}

```

```

Widget cardEntries() => Padding(
  padding: const EdgeInsets.all(16.0),
  child: Column(
    mainAxisAlignment: MainAxisAlignment.spaceAround,
    crossAxisAlignment: CrossAxisAlignment.start,
    children: <Widget>[
      StreamBuilder<String>(
        stream: cardBloc.ccOutputStream,
        initialData: "**** *",
        builder: (context, snapshot) {
          snapshot.data.length > 0
            ? ccMask.updateText(snapshot.data)
            : null;
          return Text(
            snapshot.data.length > 0

```



```

        ? snapshot.data
        : "**** *
        style: TextStyle(color: Colors.white, fontSize: 22.0),
    );
  )),
  Row(
    mainAxisAlignment: MainAxisAlignment.start,
    children: <Widget>[
      StreamBuilder<String>(
        stream: cardBloc.expOutputStream,
        initialData: "MM/YY",
        builder: (context, snapshot) {
          snapshot.data.length > 0
            ? expMask.updateText(snapshot.data)
            : null;
          return ProfileTile(
            textColor: Colors.white,
            title: "Expiry",
            subtitle:
              snapshot.data.length > 0 ? snapshot.data : "MM/YY",
          );
        )),
      SizedBox(
        width: 30.0,
      ),
      StreamBuilder<String>(
        stream: cardBloc.cvvOutputStream,
        initialData: "****",
        builder: (context, snapshot) => ProfileTile(
          textColor: Colors.white,
          title: "CVV",
          subtitle:
            snapshot.data.length > 0 ? snapshot.data : "****",
        )),
    ],
  ),
],
);

Widget fillEntries() => Padding(
  padding: const EdgeInsets.all(16.0),
  child: Column(
    mainAxisAlignment: MainAxisAlignment.min,
    children: <Widget>[
      TextField(
        controller: ccMask,
        keyboardType: TextInputType.number,
        maxLength: 19,
        style: TextStyle(
          fontFamily: UIData.ralewayFont, color: Colors.black),
        onChanged: (out) => cardBloc.ccInputSink.add(ccMask.text),

```

```

decoration: InputDecoration(
  labelText: "Credit Card Number",
  labelStyle: TextStyle(fontWeight: FontWeight.bold),
  border: OutlineInputBorder()),
),
TextField(
  controller: expMask,
  keyboardType: TextInputType.number,
  maxLength: 5,
  style: TextStyle(
    fontFamily: UIData.ralewayFont, color: Colors.black),
  onChanged: (out) => cardBloc.expInputSink.add(expMask.text),
  decoration: InputDecoration(
    labelStyle: TextStyle(
      fontWeight: FontWeight.bold,
    ),
    labelText: "MM/YY",
    border: OutlineInputBorder()),
),
TextField(
  keyboardType: TextInputType.number,
  maxLength: 3,
  style: TextStyle(
    fontFamily: UIData.ralewayFont, color: Colors.black),
  onChanged: (out) => cardBloc.cvvInputSink.add(out),
  decoration: InputDecoration(
    labelStyle: TextStyle(fontWeight: FontWeight.bold),
    labelText: "CVV",
    border: OutlineInputBorder()),
),
TextField(
  keyboardType: TextInputType.text,
  maxLength: 20,
  style: TextStyle(
    fontFamily: UIData.ralewayFont, color: Colors.black),
  onChanged: (out) => cardBloc.nameInputSink.add(out),
  decoration: InputDecoration(
    labelStyle: TextStyle(
      fontWeight: FontWeight.bold,
    ),
    labelText: "Name on card",
    border: OutlineInputBorder()),
),
],
),
);

```

```

Widget floatingBar() => Ink(
  decoration: ShapeDecoration(
    shape: StadiumBorder(),
    gradient: LinearGradient(colors: UIData.kitGradients)),
  child: FloatingActionButton.extended(

```

```

onPressed: () async {
  FirebaseFirestore.instance
    .collection('logs')
    .doc(user.id)
    .collection("logs")
    .doc('Q12w' + widget.numberPlate)
    .update({'paid': 'paid'});
  Navigator.push(context,
    MaterialPageRoute(builder: (context) => PaymentDone()));
},
backgroundColor: Colors.transparent,
icon: Icon(
  FontAwesomeIcons.amazonPay,
  color: Colors.white,
),
label: Text(
  "Continue ₹ ₹{widget.amountCharge}",
  // "Continue",
  style: TextStyle(color: Colors.white),
),
),
);

@override
Widget build(BuildContext context) {
  _context = context;
  cardBloc = CreditCardBloc();
  return Scaffold(
    resizeToAvoidBottomPadding: true,
    appBar: AppBar(
      backgroundColor: Color(0XFF003051),
      centerTitle: true,
      title: Text("Credit Card"),
    ),
    body: bodyData(),
    floatingActionButton: floatingBar(),
    floatingActionButtonLocation: FloatingActionButtonLocation.centerFloat,
  );
}
}

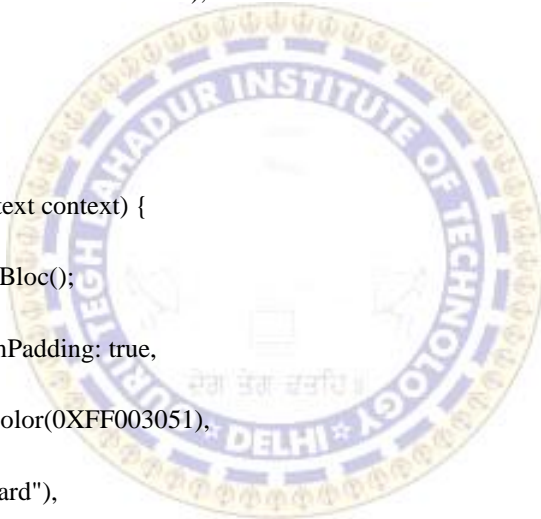
paymentDone.dart
import 'dart:async';
import 'package:flutter/material.dart';
import 'package:flutter_spinkit/flutter_spinkit.dart';

class PaymentDone extends StatefulWidget {
  PaymentDone({this.numberPlate});

  final String numberPlate;

  @override

```



```

    _PaymentDoneState createState() => _PaymentDoneState();
}

class _PaymentDoneState extends State<PaymentDone> {
    // Timer _timer;
    // int _start = 4;
    // bool _isOk = false;
    //
    // void startTimer() {
    //   const oneSec = const Duration(seconds: 1);
    //   _timer = new Timer.periodic(oneSec, (Timer timer) {
    //     if (this.mounted)
    //       setState(
    //         () {
    //           if (_start < 1) {
    //             timer.cancel();
    //             _isOk = true;
    //           } else {
    //             _start = _start - 1;
    //           }
    //         },
    //       );
    //   });
    // }
    //
    // @override
    // void dispose() {
    //   _timer?.cancel();
    //   super.dispose();
    // }
    //
    // @override
    // void initState() {
    //   super.initState();
    // }

    @override
    Widget build(BuildContext context) {
      // startTimer();

      // if (_start == 0) {
      //   _timer?.cancel();
      //   // Timer(Duration(seconds: 1), () => Navigator.pop(context));
      // }
      // if (_isOk) {
      //   print("xCV");
      //   Timer(Duration(seconds: 1), () {
      //     Navigator.pop(context);
      //   });
      // }
      Timer(Duration(seconds: 2), () => Navigator.pop(context));
      double height = MediaQuery.of(context).size.height / 2;

```



```

return Scaffold(
  appBar: AppBar(
    backgroundColor: Color(0XFF003051),
    automaticallyImplyLeading: false),
  body: Column(
    children: [
      Container(
        height: height,
        child: Padding(
          padding: const EdgeInsets.all(10.0),
          child: SpinKitWave(
            color: Color(0XFF003051), type: SpinKitWaveType.start),
        ),
      ),
      // Text("Processing your Payment in $_start seconds."),
      Text("Processing your Payment..."),
    ],
  ));
}
}

```

```

profile.dart
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:minor_project/Pages/AuthenticationPage.dart';
import 'package:minor_project/Pages/detailedNumberPlate.dart';
import 'package:shimmer/shimmer.dart';

```

```

class Profile extends StatefulWidget {
  @override
  _ProfileState createState() => _ProfileState();
}

```

```

class _ProfileState extends State<Profile> {
  // Widget shimmeringLoadingCards() {
  //   return SingleChildScrollView(
  //     child: Shimmer.fromColors(
  //       highlightColor: Colors.white,
  //       baseColor: Colors.grey[300],
  //       child: Padding(
  //         padding: const EdgeInsets.all(20.0),
  //         child: Column(
  //           crossAxisAlignment: CrossAxisAlignment.start,
  //           children: [
  //             Container(
  //               height: 20,
  //               width: 200,
  //               color: Colors.grey,
  //             ),
  //             SizedBox(
  //               height: 20.0,
  //             ),
  //           ],
  //         ),
  //       ),
  //     ),
  //   ),
  // );
}

```

```

//      ClipRect(
//          borderRadius: BorderRadius.circular(20.0),
//          child: Container(
//              height: MediaQuery.of(context).size.height / 4,
//              width: MediaQuery.of(context).size.width,
//              color: Colors.grey,
//          ),
//      ),
//      SizedBox(
//          height: 20.0,
//      ),
//      Container(
//          height: 20,
//          width: 275,
//          color: Colors.grey,
//      ),
//      SizedBox(
//          height: 10.0,
//      ),
//      Container(
//          height: 20,
//          width: 300,
//          color: Colors.grey,
//      ),
//  ],
// ),
// ),
// );
// }

```



```

final numberPlateRef = FirebaseFirestore.instance
    .collection('number_plate')
    .doc(user.id)
    .collection("number_plate");

```

```

List<DocumentSnapshot> documents = [];

```

```

int counter = 0;

```

```

@override
void initState() {
    // TODO: implement initState
    getNumberPlateDocs();
    super.initState();
}

```

```

getNumberPlateDocs() async {
    final QuerySnapshot result = await FirebaseFirestore.instance
        .collection('number_plate')
        .doc(user.id)
        .collection("number_plate")

```

```

    .getDocuments();
documents = result.documents;
setState() {
  documents.forEach((data) => counter++);
});
print("cccccoouunnnntteerrrrrrr $counter");
}

```

```

Widget listItem(BuildContext context, DocumentSnapshot document) {
  return Padding(
    padding: EdgeInsets.only(left: 15.0, right: 15.0, top: 20.0),
    child: ClipRRect(
      borderRadius: BorderRadius.circular(20.0),
      child: Container(
        height: 50,
        width: MediaQuery.of(context).size.width / 2,
        color: Colors.grey[200],
        child: InkWell(
          onTap: () => Navigator.push(
            context,
            MaterialPageRoute(
              builder: (context) => DetailedNumPltSrn(document.data()),
            ),
          child: Padding(
            padding: const EdgeInsets.only(left: 5.0),
            child: Container(
              height: 40,
              width: MediaQuery.of(context).size.width / 3,
              padding: const EdgeInsets.all(10),
              child: Padding(
                padding: const EdgeInsets.only(top: 5),
                child: Text(
                  '${document.id}',
                  style: TextStyle(
                    color: Colors.black,
                    fontWeight: FontWeight.bold,
                    fontSize: 15),
                overflow: TextOverflow.ellipsis,
                maxLines: 1,
              ),
            ),
          ),
        ),
      ),
    );
}

```

```

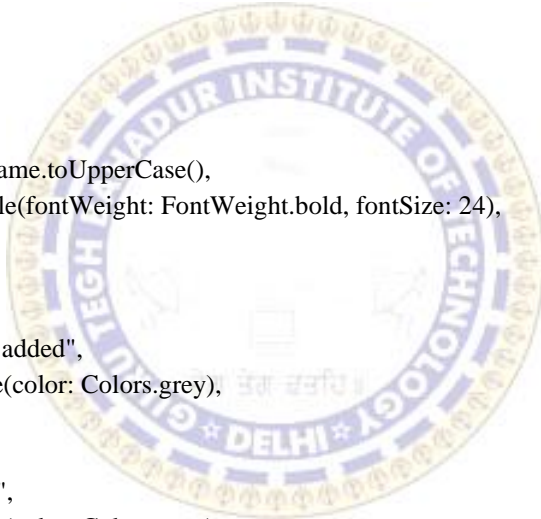
@override
Widget build(BuildContext context) {
  return Scaffold(

```

```

body: SingleChildScrollView(
  child: Container(
    child: Column(
      children: [
        SizedBox(
          height: 30,
        ),
        Container(
          height: 100,
          child: Center(
            child: ClipOval(
              child: Image.network(
                user.photoUrl,
              ),
            ),
          ),
        ),
        SizedBox(
          height: 10,
        ),
        InkWell(
          onTap: () {},
          child: Text(
            user.displayName.toUpperCase(),
            style: TextStyle(fontWeight: FontWeight.bold, fontSize: 24),
          ),
        ),
        Text(
          "$counter Cars added",
          style: TextStyle(color: Colors.grey),
        ),
        Text(
          "${user.email}",
          style: TextStyle(color: Colors.grey),
        ),
        // shimmeringLoadingCards(),
        SizedBox(
          height: 20.0,
        ),
        Text(
          "Added Number plates:",
          style: TextStyle(color: Colors.grey),
        ),
        SizedBox(
          height: 10.0,
        ),
        StreamBuilder(
          stream: numberPlateRef
            .orderBy('timestamp', descending: false)
            .snapshots(),
          builder: (context, snapshot) {
            if (!snapshot.hasData) {

```




```

        return Center(
          child: Text("Loading....."),
        );
      }
    return ListView.builder(
      itemCount: snapshot.data.documents.length,
      scrollDirection: Axis.vertical,
      shrinkWrap: true,
      itemBuilder: (context, index) {
        return listItem(
          context, snapshot.data.documents[index]);
      });
    },
  ),
],
),
),
),
);
}
}

```

recentLogs.dart

```

import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:minor_project/widgets/log_tile.dart';
import 'AuthenticationPage.dart';

```

```

class RecentLogs extends StatefulWidget {
  RecentLogs({this.numberPlate});

  final String numberPlate;

  @override
  _RecentLogsState createState() => _RecentLogsState();
}

```

```

class _RecentLogsState extends State<RecentLogs> {
  Widget listItem(BuildContext context, DocumentSnapshot document) {
    String entry = document["entry_timestamp"];
    String exit = document["exit_timestamp"];
    return Padding(
      padding: EdgeInsets.only(left: 15.0, right: 15.0, top: 20.0),
      child: ClipRRect(
        borderRadius: BorderRadius.circular(20.0),
        child: Container(
          height: 190,
          width: MediaQuery.of(context).size.width / 2,
          color: Colors.grey[200],
          child: Padding(
            padding: const EdgeInsets.only(
              top: 15.0, right: 15.0, bottom: 15.0, left: 5.0),

```

[illegible]


```

        height: 3,
      ),
      Padding(
        padding: const EdgeInsets.only(left: 5.0),
        child: SizedBox(
          width: MediaQuery.of(context).size.width / 1.2,
          child: Text(
            "Date: ${exit?.substring(0, 10)}   Time: ${exit?.substring(11, 19)}",
            style: TextStyle(
              fontSize: 15,
              color: Colors.black,
              fontWeight: FontWeight.bold,
            ),
            maxLines: 1,
            softWrap: false,
            overflow: TextOverflow.ellipsis,
          ),
        ),
      ),
    ],
  ),
),
);
}

```



```

@override
Widget build(BuildContext context) {
  final logsRef = FirebaseFirestore.instance
    .collection('logs')
    .doc(user.id)
    .collection(widget.numberPlate);
  return Scaffold(
    appBar: AppBar(
      title: Text('Recent Logs'),
      backgroundColor: Color(0XFF003051),
      centerTitle: true,
    ),
    body: StreamBuilder(
      stream:
        logsRef.orderBy('entry_timestamp', descending: true)?.snapshots(),
      builder: (context, snapshot) {
        print(
          "*****entered snapshot $snapshot");
        if (!snapshot.hasData) {
          return Center(
            child: Text("Loading Recent Logs....."),
          );
        }
      },
    ),
  );
}

```

```

body: ListView.builder(
  itemCount: snapshot.data.documents.length,
  scrollDirection: Axis.vertical,
  itemBuilder: (context, index) {
    // return ListItem(context, snapshot.data.documents[index]);
    DocumentSnapshot document = snapshot.data.documents[index];

    return LogTile(
      entryTimestamp: document["entry_timestamp"],
      exitTimestamp: document['exit_timestamp'],
      numberPlate: document["number_plate"],
      place: document['place'],
      paid: "",
      recent: true,
    );
  },
);
},
));
}
}

```

Widgets

```

drawer.dart
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:minor_project/Pages/AuthenticationPage.dart';

```

```

class NavigationDrawer extends StatelessWidget {

  @override
  Widget build(BuildContext context) {
    return Drawer(
      child: Padding(
        padding: const EdgeInsets.only(left: 20.0, right: 15.0, top: 50),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: <Widget>[
            InkWell(
              onTap: () {},
              child: Row(
                mainAxisAlignment: MainAxisAlignment.spaceBetween,
                children: <Widget>[
                  Container(
                    width: 80,
                    height: 80,
                    decoration: BoxDecoration(
                      shape: BoxShape.circle, color: Colors.grey),
                  ClipOval(
                    child: Image.network(user.photoUrl),

```

[illegible]

```

InkWell(
  onTap: () {},
  child: Row(
    children: <Widget>[
      Icon(
        Icons.bookmark,
        color: Colors.black,
        size: 22,
      ),
      SizedBox(
        width: 8,
      ),
      Text(
        "Saved Locations",
        style: TextStyle(fontSize: 16),
      )
    ],
  ),
),
SizedBox(
  height: 15,
),
Row(
  children: <Widget>[
    Icon(
      Icons.money,
      color: Colors.black,
      size: 22,
    ),
    SizedBox(
      width: 8,
    ),
    Text(
      "Pending Payments",
      style: TextStyle(fontSize: 16),
    )
  ],
),
SizedBox(
  height: 15,
),
InkWell(
  onTap: () {},
  child: Row(
    children: <Widget>[
      Container(
        width: 20,
        height: 20,
        child: Icon(Icons.support_agent,color: Colors.black,)
      ),
      SizedBox(
        width: 8,

```



[illegible]


```

        )
      ],
    ),
  ),
);
}
}

```

log_tile.dart

```

import 'package:flutter/material.dart';
import 'package:minor_project/Pages/payment.dart';

```

```

class LogTile extends StatefulWidget {
  LogTile({this.numberPlate,
    this.place,
    this.exitTimestamp,
    this.entryTimestamp,
    this.paid,
    this.recent});

```

```

  final String numberPlate;
  final String place;
  final String exitTimestamp;
  final String entryTimestamp;
  String paid;
  final bool recent;

```

```

  @override
  _LogTileState createState() => _LogTileState();
}

```

```

class _LogTileState extends State<LogTile> {
  int _payRate = 30;
  double calChg = 0;
  bool _notParked = true;

```

```

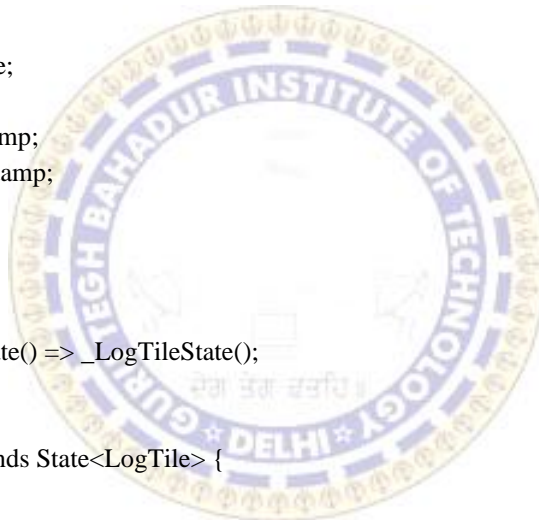
  @override
  Widget build(BuildContext context) {
    if (widget.recent) widget.paid = 'paid';
    String entry = widget.entryTimestamp;
    print("*****$entry");

```

```

    int entryHr = int.parse(entry.substring(11, 13));
    int entryMn = int.parse(entry.substring(14, 16));
    String exit = "";
    int exitHr = 0;
    int exitMn = 0;
    try {
      exit = widget.exitTimestamp;
      exitHr = int.parse(exit.substring(11, 13));
      exitMn = int.parse(exit.substring(14, 16));
      calChg = (exitHr - entryHr) * _payRate +

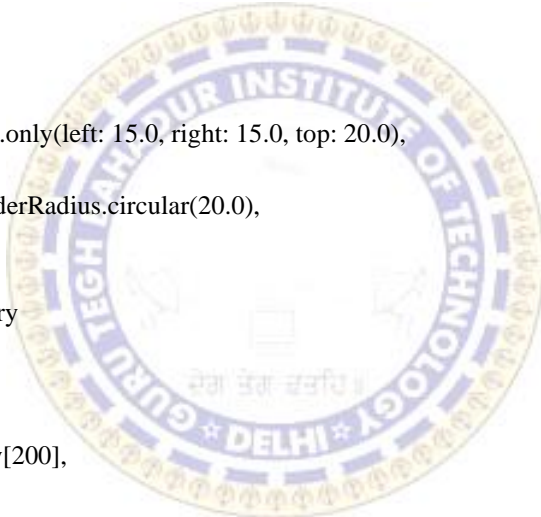
```



```

        ((exitMn - entryMn) * (_payRate / 60));
    print("Charge : $calChg");
    _notParked = true;
} catch (e) {
    print(e);
    exit = "Vehicle in Parking";
    _notParked = false;
}
return GestureDetector(
  onTap: () {
    if (_notParked && widget.paid == "" && widget.paid.isEmpty) {
      Navigator.push(
        context,
        MaterialPageRoute(
          builder: (context) =>
            CreditCardPage(
              amountCharge: calChg,
              numberPlate: widget.numberPlate,
            )),
      );
    }
  },
  child: Padding(
    padding: EdgeInsets.only(left: 15.0, right: 15.0, top: 20.0),
    child: ClipRRect(
      borderRadius: BorderRadius.circular(20.0),
      child: Container(
        height: 200,
        width: MediaQuery
          .of(context)
          .size
          .width / 2,
        color: Colors.grey[200],
        child: Padding(
          padding: const EdgeInsets.only(
            top: 15.0, right: 15.0, bottom: 15.0, left: 5.0),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Row(
                mainAxisAlignment: MainAxisAlignment.spaceBetween,
                mainAxisSize: MainAxisSize.max,
                children: [
                  Row(
                    children: [
                      Container(
                        height: 40,
                        width: MediaQuery
                          .of(context)
                          .size
                          .width / 3,
                        padding: const EdgeInsets.all(10),
                        decoration: BoxDecoration(

```



```

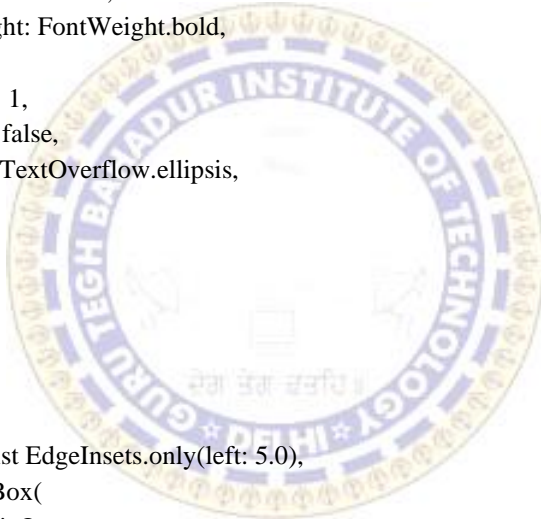
        color: Colors.black,
        borderRadius: BorderRadius.circular(20)),
child: Center(
  child: Text(
    '${widget.numberPlate}',
    // '${document["place"]}',
    style: TextStyle(color: Colors.white),
    overflow: TextOverflow.ellipsis,
    maxLines: 1,
  ),
),
),
Container(
  height: 40,
  width: MediaQuery
    .of(context)
    .size
    .width / 5,
  decoration: BoxDecoration(
    color: (widget.paid.isNotEmpty &&
      widget.paid != "")
      ? Colors.green
      : Colors.redAccent,
    borderRadius: BorderRadius.circular(20)),
  child: Center(
    child: Text(
      (widget.paid.isNotEmpty && widget.paid != "")
        ? "Paid"
        : "Not Paid",
      style: TextStyle(color: Colors.white),
      overflow: TextOverflow.ellipsis,
      maxLines: 1,
    ),
  ),
),
],
),
_notParked
? Row(
  children: [
    Text(
      '₹',
      style: TextStyle(color: Colors.green),
    ),
    Text(
      "$calChg",
      style: TextStyle(color: Colors.green),
    )
  ],
)
: Container(
  height: 0.0,

```

```

        width: 0.0,
      ),
    ],
  ),
  SizedBox(
    height: 5,
  ),
  Padding(
    padding: const EdgeInsets.only(left: 5.0),
    child: SizedBox(
      width: MediaQuery
        .of(context)
        .size
        .width / 1.2,
      child: Text(
        'Place: ${widget.place}',
        style: TextStyle(
          fontSize: 15,
          color: Colors.black,
          fontWeight: FontWeight.bold,
        ),
        maxLines: 1,
        softWrap: false,
        overflow: TextOverflow.ellipsis,
      ),
    ),
  ),
  SizedBox(
    height: 5,
  ),
  Padding(
    padding: const EdgeInsets.only(left: 5.0),
    child: SizedBox(
      width: MediaQuery
        .of(context)
        .size
        .width / 1.2,
      child: Text(
        "Entry Timestamp:",
        style: TextStyle(
          fontSize: 15,
          color: Colors.black,
          fontWeight: FontWeight.bold,
        ),
        maxLines: 1,
        softWrap: false,
        overflow: TextOverflow.ellipsis,
      ),
    ),
  ),
  SizedBox(
    height: 3,

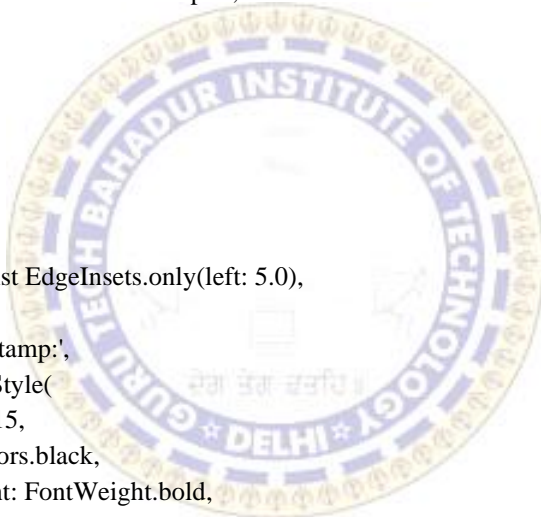
```



```

),
Padding(
  padding: const EdgeInsets.only(left: 5.0),
  child: SizedBox(
    width: MediaQuery
      .of(context)
      .size
      .width / 1.2,
    child: Text(
      "Date: ${entry.substring(0, 10)}   Time: ${entry
        .substring(11, 19)}",
      style: TextStyle(
        fontSize: 15,
        color: Colors.black,
        fontWeight: FontWeight.bold,
      ),
      maxLines: 1,
      softWrap: false,
      overflow: TextOverflow.ellipsis,
    ),
  ),
),
SizedBox(
  height: 5,
),
Padding(
  padding: const EdgeInsets.only(left: 5.0),
  child: Text(
    'Exit Timestamp:',
    style: TextStyle(
      fontSize: 15,
      color: Colors.black,
      fontWeight: FontWeight.bold,
    ),
    overflow: TextOverflow.ellipsis,
    softWrap: false,
    maxLines: 1,
  ),
),
SizedBox(
  height: 3,
),
_notParked
  ? Padding(
    padding: const EdgeInsets.only(left: 5.0),
    child: SizedBox(
      width: MediaQuery
        .of(context)
        .size
        .width / 1.2,
      child: Text(
        "Date: ${exit?.substring(0, 10)}   Time: ${exit

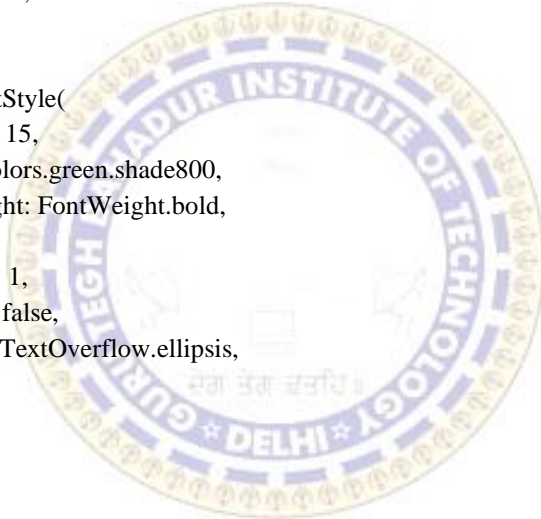
```



```

        ?.substring(11, 19)}",
        style: TextStyle(
          fontSize: 15,
          color: Colors.black,
          fontWeight: FontWeight.bold,
        ),
        maxLines: 1,
        softWrap: false,
        overflow: TextOverflow.ellipsis,
      ),
    ),
  )
  : Padding(
    padding: const EdgeInsets.only(left: 5.0),
    child: SizedBox(
      width: MediaQuery
        .of(context)
        .size
        .width / 1.2,
      child: Text(
        exit,
        style: TextStyle(
          fontSize: 15,
          color: Colors.green.shade800,
          fontWeight: FontWeight.bold,
        ),
        maxLines: 1,
        softWrap: false,
        overflow: TextOverflow.ellipsis,
      ),
    ),
  ),
  _notParked
    ? Padding(
      padding: const EdgeInsets.only(left: 5.0),
      child: Text(
        "Parked for: ${exitHr - entryHr} hr. and ${exitMn -
          entryMn} min.",
        style: TextStyle(
          color: Colors.green.shade900,
          fontSize: 15,
          fontWeight: FontWeight.bold,
        ),
      ),
    )
    : Container(
      height: 0.0,
      width: 0.0,
    ),
  ],
),
),
),

```



```

    ),
  ),
),
);
}
}

```

profile_tile.dart

```
import 'package:flutter/material.dart';
```

```

class ProfileTile extends StatelessWidget {
  final title;
  final subtitle;
  final textColor;
  ProfileTile({this.title, this.subtitle, this.textColor = Colors.black});
  @override
  Widget build(BuildContext context) {
    return Column(
      // crossAxisAlignment: CrossAxisAlignment.start,
      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
        Text(
          title,
          style: TextStyle(
            fontSize: 20.0, fontWeight: FontWeight.w700, color: textColor),
        ),
        SizedBox(
          height: 5.0,
        ),
        Text(
          subtitle,
          style: TextStyle(
            fontSize: 15.0, fontWeight: FontWeight.normal, color: textColor),
        ),
      ],
    );
  }
}

```

progress.dart

```
import 'package:flutter/material.dart';
```

```

Widget circularProgress() {
  return Center(
    child: Container(
      alignment: Alignment.center,
      padding: EdgeInsets.only(top: 10.0),
      child: CircularProgressIndicator(
        valueColor: AlwaysStoppedAnimation(Colors.purple),
      )),
  );
}

```

```
Widget linearProgress() {  
  return Scaffold(  
    body: Container(  
      padding: EdgeInsets.only(bottom: 10.0),  
      child: LinearProgressIndicator(  
        valueColor: AlwaysStoppedAnimation(Colors.purple),  
      ),  
    ),  
  );  
}
```

