

SECURITY SYSTEM USING FACE RECOGNITION FOR DIU Hall

BY

MD. AL-AMIN

ID: 181-15-1951

MD. ASHIK MAHMUD

ID: 181-15-1728

MRIDUL BARAL

ID: 181-15-1845

This Report Presented in Partial Fulfillment of the Requirements
for the Degree of Bachelor of Science in Computer Science and Engineering

Supervised By

Dr. S. M. Aminul Haque

Associate Professor

Department of CSE

Daffodil International University

Co-Supervised By

Tania Khatun

Sr. Lecturer

Department of CSE

Daffodil International University



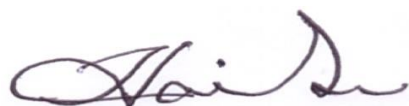
DAFFODIL INTERNATIONAL UNIVERSITY DHAKA, BANGLADESH

DECEMBER 2021

APPROVAL

This Project/internship titled “Security System using Face Recognition for DIU Hall”, submitted by *Md. Al-Amin*, ID No: 181-15-1951, *Md. Ashik Mahmud*, ID No: 181-15-1728, *Mridul Baral*, ID No: 181-15-1845 to the Department of Computer Science and Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Engineering and approved as to its style and contents. The presentation has been held on 18.01.2022.

BOARD OF EXAMINERS

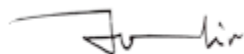


Sheak Rashed Haider Noori

Associate Professor

Department of Computer Science and Engineering
Daffodil International University

Internal Examiner



Ohidujjaman

Assistant Professor

Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University

Internal Examiner



Dr. Mohammad Shorif Uddin

Professor

Department of Computer Science and Engineering
Jahangirnagar University

External Examiner

DECLARATION

We hereby declare that this project has been done by us under the supervision of **Dr. S. M. Aminul Haque** Associate Professor Department of CSE Daffodil International University. We also declare that neither this project nor any part of this project has been submitted elsewhere for award of any degree or diploma.

Supervised by:



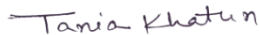
Dr. S. M. Aminul Haque

Associate Professor

Department of CSE

Daffodil International University

Co-Supervised by:



Tania Khatun

Sr. Lecturer

Department of CSE

Daffodil International University

Submitted by:



MD. AL-AMIN

ID: 181-15-1951

Department of CSE

Daffodil International University



MD. ASHIK MAHMUD

ID: 181-15-1728

Department of CSE

Daffodil International University



MRIDUL BARAL

ID: 181-15-1845

Department of CSE

Daffodil International University

© Daffodil International University

ACKNOWLEDGEMENT

First, we express our heartiest gratefulness to the almighty God for His divine blessings make us possible to complete the final year project work successfully.

We really grateful and wish our profound our indebtedness to **Dr. S. M. Aminul Haque** Associate Professor Department of CSE Daffodil International University, Dhaka. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stage have made it possible to complete this project.

We would like to express our heartiest gratitude to **Dr. S. M. Aminul Haque** Associate Professor Department of CSE Daffodil International University, for his kind help to finish our project and also to other faculty member and the staff of CSE department of Daffodil International University. We would like to thank our entire course mate in Daffodil International University, who took part in this discuss while completing the course work.

Finally, we must acknowledge with due respect the constant support and patients of our parents.

ABSTRACT

Face recognition technology is gaining popularity around the world these days for delivering highly safe and trustworthy security technology. Because of its high level of security and reliability, it is gaining major importance and attention from hundreds of private and institutional organizations. This type of security system will be more beneficial if it is connected on CCTV. Our proposed approach is to develop a face recognition security system for organizations such as university halls. By using our system we can track unknown and known people .We can also track unknown people activity. Our software will examine the information and compare it to a database of trusted individuals. When someone walks in front of the camera, our system logs the time and date on an excel sheet.

TABLE OF CONTENTS

CONTENTS	PAGE
Board of Examiners	ii
DECLARATION	iii
ACKNOWLEDGEMENT	iv
ABSTRACT	v
CHAPTER 1	1
INTRODUCTION	1
1.1 Introduction.....	1
1.2 Motivation.....	2
1.3 Objectives	2
1.4 Expected Outcome	2
1.5 Report layout.....	3
CHAPTER 2	4
BACKGROUND	4
2.1 Related Works.....	4
2.2 Comparative Analysis.....	7
2.3 Scope of the Problem	7
2.4 Challenges.....	8
CHAPTER 3	9
Requirement Specification.....	9
3.1 Business Process Modeling.....	9
3.2 Requirement Collection and Analysis.....	11
3.3 Use Case Modeling and Description.....	13
3.4 Logical Data Model	16
3.5 Design Requirements	16
Chapter 4.....	17
Design Specification	17
4.1 Frontend Design.....	17
4.2 Backend Design	36
4.3 Interaction Design and UX	47
4.4 Implementation Requirements	47
Chapter 5.....	48
Implementation and Testing	49

5.1 Implementation of Database	49
5.2 Testing Implementation of Reports.....	49
CHAPTER 6	50
Conclusion	50
6.1 Impact on Society	50
6.2 Conclusion	50
6.3 Scope for Further Developments	51
References:.....	52

List of Figures

FIGURES	PAGE
3.1: Business Process Modeling(Face Recognition System)	9
3.2: Business Process Modeling (Guard App).....	10
3.3: Business Process Modeling(Admin App)	11
3.4: Use case Modeling (Face recognition system).....	13
3.5: Use case Modeling(Guard App)	14
3.6: Use case Modeling(Admin App)	15
4.1: Detecting known faces.....	17
4.2: Detecting unknown faces.....	18
4.3: Output of face info.....	18
4.4: Face info into text format.....	19
4.5: Face info into Excel Sheet.....	19
4.6: SplashScreen of the application	20
4.7: Login screen for guard app	21
4.8: Home page of guard app	22
4.9: Getting storage permission from the user	23
4.10: Gallery image chooser view.....	24
4.11: Recognizing image after choosing.....	25
4.12: Details screen of the student	26

4.13: Report Screen for guard app	27
4.14: Getting camera permission from the user	28
4.15: Live recognition screen.....	29
4.16: Login screen (Admin App)	30
4.17: Home Screen (Admin App)	31
4.18: Add Guard Info Screen (Admin App)	32
4.19: Guard List Screen (Admin App).....	33
4.20: Add Face Data Screen (Admin App).....	34
4.21: Reports Screen (Admin App).....	35
4.22: Face data of hall student.....	36
4.23: Comparing images using OpenCV.....	36
4.24: Encoding image.....	37
4.25: Code for recognize face from video.....	37
4.26: Importing required libraries	38
4.27: Importing required libraries	38
4.28: Then we clear ram memory and then check the array	39
4.29: Converting image scale.....	40
4.30: Split dataset and create a model using TensorFlow.....	40
4.31: Model summary.....	41
4.32: Create Checkpoint for best accuracy.....	42
4.33: The training model.....	42
4.34: Loading train model.....	43
4.35: Predict value and trained value.....	43
4.36: Output model after train model.....	44
4.37: Backend Code structure Guard App.....	44
4.38: Backend Code Face Recognition Class.....	45
4.39: Backend code structure (Admin App).....	45
4.40: Backend code Update Guard Activity (Admin App).....	46
4.41: Dependency of Admin App (Admin App).....	47
5.1: Firebase database implementation.....	48
5.2: Testing face recognition.....	49

List of Tables

Tables	pages
Table 3.1 Requirement Collection and Analysis.....	12

CHAPTER 1

INTRODUCTION

1.1 Introduction:

The field of biometrics includes real-time face recognition. Biometrics refers to a computer's ability to distinguish a person based on a unique bodily characteristic. Face recognition is the capacity for a computer to recognize a person based on their facial features. Biometrics is one of the most rapidly increasing sectors in advanced technology today. Biometrics are expected to explode in the next century to authenticate identities and prevent illegal access to networks, databases, and facilities, according to predictions.

A facial recognition device is a device that matches an image or video of a human face to other human faces stored in a database. During the face identification processes, the structure, shape, and proportions of the faces are compared. Furthermore, the distance between the eyes, nose, mouth, and jaw, the top contours of the eye sockets, the sides of the mouth, the placement of the nose and eyes, and the area surrounding the cheekbones are all compared[1].

Several images of the subject must be taken at different angles and with varied facial expressions when utilizing a facial recognition program[1]. The subject stands in front of the camera for a few seconds during verification and identification, and then the image is compared to those previously recorded.

Because of its advantages, facial recognition is widely utilized. The benefits of facial recognition include the fact that it is non-intrusive and may be performed from afar without the subject being aware that he or she is being scanned[1]. Face recognition systems are superior to other biometric techniques in that they may be utilized for surveillance reasons, which is why they are used at banks, educational institutions, student halls, and government offices.

1.2 Motivation

Our primary motivation is to make a better security system for DIU Hall. We know previously a few incidents happened by unknown people who are not authorized to enter DIU Hall. To prevent future problems like that, we are making this security system.

For this reason, we have made this system that can recognize face and show his/her information using CCTV. We choose this platform because it is easy to maintain. This face recognition system will make our DIU hall more secure than before.

1.3 Objective

1. To make a better security system for DIU Hall.
2. To prevent future problems that can be occurred by unknown people.
3. This system is helpful and low in cost.
4. This system also can use to get exact information about a student just taking an image.

1.4 Expected Outcomes

We have designed the project to work on CCTV/Webcam/IP Camera. Also we will work on android based system so that it can more user friendly. Our expected outcomes will be:

1. Any institution's security system, such as banks, educational institutions, student halls, and government offices, will be enhanced.
2. The incidents occurred by unauthorized people will be reduced.
3. Authenticating a trusted individual will become much easier and faster.
4. Using this system, administrator can deal with any undesirable situation that may arise.
5. Companies and Institutions that need security service at a low price and more reliable can use our system.

1.5 Report Layout

We addressed our purpose, objective, and expectations from this application in the first chapter of this project report. The second chapter will focus on the background of this project, and we have enlisted many other studies of this field, and we talked about the challenges that come with this application.

Following that, we have chapter three, discussing. The requirement specifications, use case modeling, description, and design requirements.

The fourth chapter covers Design Specification, where we discuss Front-end, Back-end, Interaction Design and User Experience (UX), Implementation Requirements. Then in chapter five, we talk about implementing and testing the application.

After that, chapter six discusses our application's impact on society and we analyze the application's future scope, discuss other areas where we may enhance, and summarize our project and reach a conclusion.

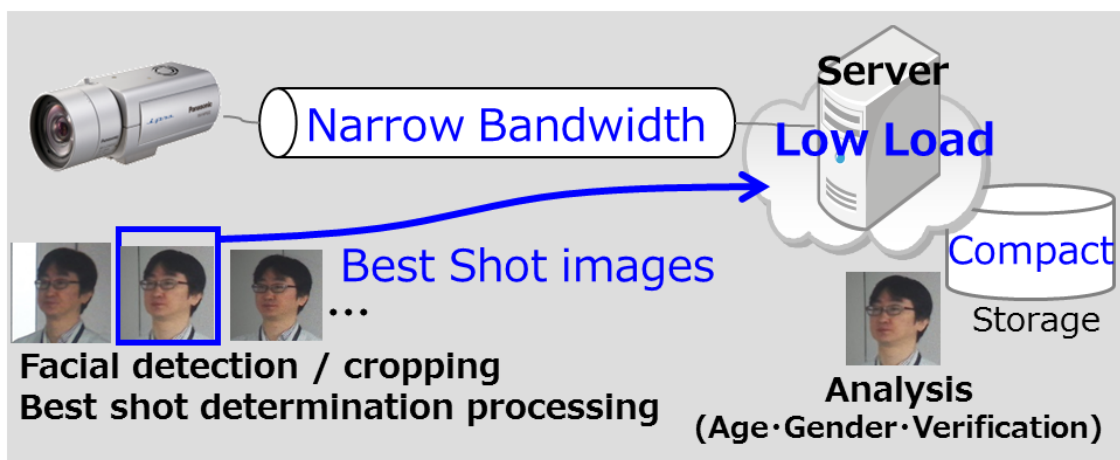
CHAPTER 2

BACKGROUND

2.1 Related Works

FacePRO™ Facial Recognition [2] Solution automatically compares a person's face to a database of enrolled faces using live or recorded footage from Panasonic i-PRO cameras, and conducts face match notification and alerting. It can analyse up to 20 cameras in real time per server and do high-speed searches of up to 30,000 registered reference faces.

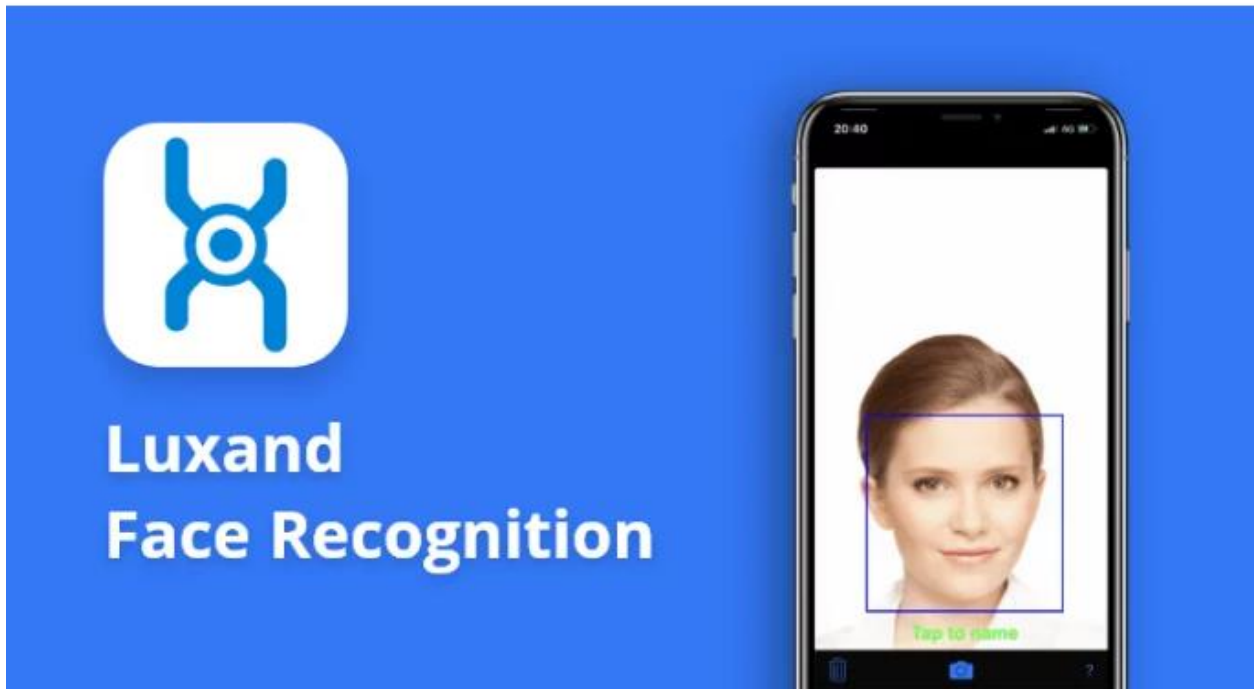
Panasonic face authentication system



The digital world has seen some ground-breaking developments in the form of AI-based facial recognition software since the introduction of biometrics. In 2019, the global facial recognition market was worth \$3.2 billion, and it is predicted to increase at a CAGR of 16.6% by 2024[3].

A few years ago, the concept of mobile face recognition seemed more like science fiction. However, the facial recognition program now aids in the prevention of wrongful arrests and the reduction of cybercrime and malware attacks.

Some applications in the market use facial recognition. One of them is Luxand Face Recognition Application. Luxand is more than simply a single app. It's a whole hi-tech corporation that was founded in 2005 and now offers a variety of services and apps [4].



It can memorize faces to recognize them for future references. It can confirm the profile identity by recognizing faces in live video broadcasts or videos. With privacy concerns and scrutiny for various face identification APIs, it makes the Face SDK functionality publicly available.

There's another app called Face Recognition Application, It can detect and recognize a user's face. It has three main modules. One is to train a person by face detection and save the user name. The second module recognizes trained users' faces and displays names of persons with matches for face detection. The third module is the face recognition gallery consists of all trained faces by face detection and face recognition. Users can delete faces as well. All images are saved in user mobile so your images are saved to train as many faces as you can [5].



Face Recognition

MMM developers Libraries & Demo

★★★★★ 212

E Everyone

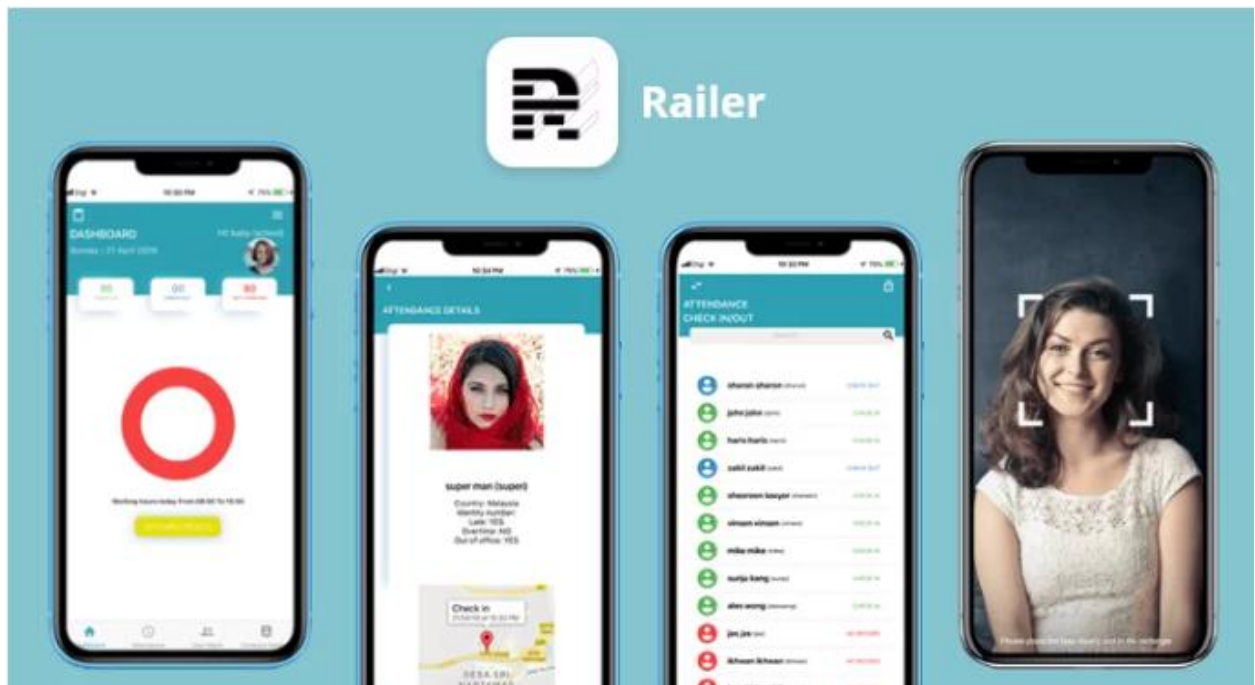
Contains Ads

i This app is available for all of your devices

+ Add to Wishlist

Install

Railer is a mobile attendance system and face recognition attendance software. It's one of the most effective face recognition applications for keeping track of staff attendance [6]. From attendance to leaves, all aspects of employee administration are smoothly integrated. For keeping track of staff movements, the app delivers detailed analytics and data.



2.2 Comparative analysis

There are some applications available for mobile in today's market. But they have some limitations in terms of detecting face and recognition.

FacePRO™ is only made for Panasonic cameras. Using Panasonic cameras it can recognize faces and this is highly cost for any company. Face Recognition app, for example, can detect and recognize the user's face. However, because all photographs are saved locally, there is no way to view the trained faces from a different device.

The Luxand Face Recognition Application memorizes faces so that it can recognize them in the future. It can confirm the profile identity by recognizing faces in live video broadcasts or videos. However, there is no camera on the back. It is solely equipped with a selfie camera. Identical twins are not recognized by the app. At any given time, it only identifies one view of your face.

Railer delivers precise analytics and data for keeping track of staff movements, and employee administration is handled smoothly, with everything from attendance to leaves effortlessly integrated. However, due to other facial recognition technologies created expressly for it, the attendance app does not engage the crowd. For a facial recognition app, the requirements are excessive.

Our application has a separate database that is not accessible locally on the phone. Our application can be used by any school, college, or university that offers a dormitory option, even at the entrance. It can recognize faces by comparing them to those in the database and provide extra information about them. Not only that, our application guard can report any occurrence that happens in front of them. Those data will directly go to the administrator application.

2.3 Scope of the problem

The purpose of our project is to improve the security system in our DIU Hall. Security guards are present in our hall to check for any difficulties and to keep an eye on the area at all times to ensure that no strangers enter. However, it is a source of regret that all security officers do not know all of the students in the hall, therefore many students or unknown people can access the hall just by claiming to be staying there.

This poses a significant risk to our DIU hall because anything can happen at any time, involving unknown persons. So we're building a system with a database of all DIU students, and we're using machine learning to recognize them.

2.4 Challenges

The challenge of our hall security system was to implement the face recognition system into the android platform and in CCTV. We face many problem while developing our app from android. For recognizing faces, we need a better CPU & GPU, so we have to do something so that we first make a model using python language then using Tensorflow lite we convert that model into TensorFlow lite model, and TensorFlow has a dependency that we can use this into the android app. We use OpenCV to recognize faces using that TensorFlow Lite model. OpenCV and Tensorflow are also in under-development. They don't make a complete system that anyone can use for face recognition on Android.

We also face problems while training our model. If we want to update our face recognition accuracy more, our app frame rate will drop, so using this app will be tricky because of lag or screen hang. So we needed more faces of one person. That was also challenging because we needed more than thousands of photos of one person to train the model. To collect that many photos were challenging for us too. Also, we need to make our guard app totally in Bangla language because we have to think about whether our users can adequately use our app or not, so we decided to make this app totally in Bangla language.

CHAPTER 3

Requirement Specification

3.1 Business Process Modeling

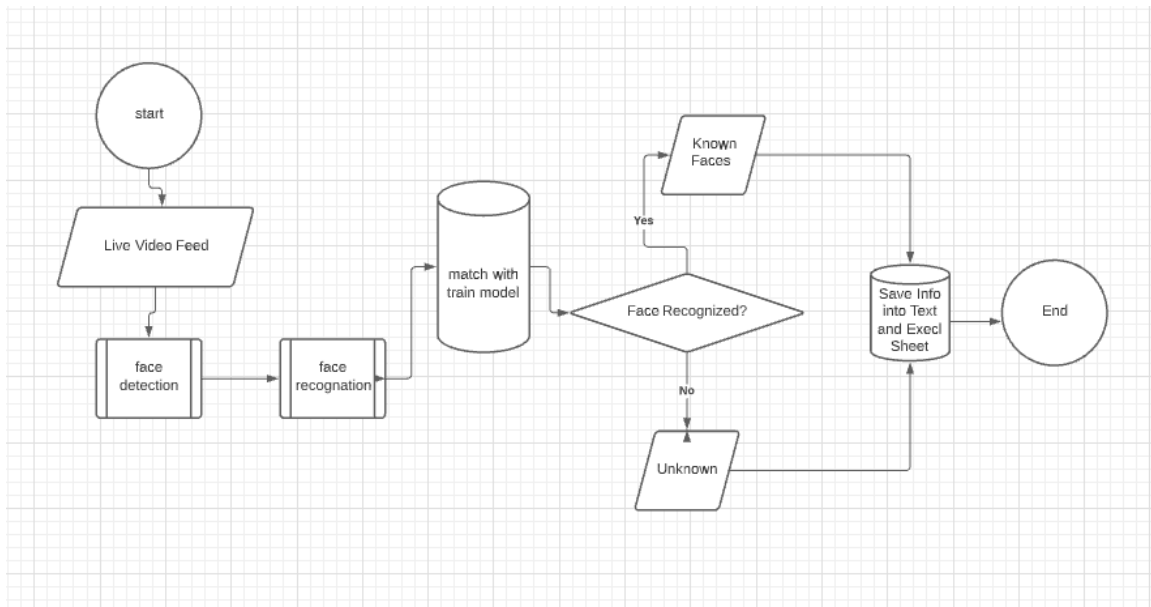


Figure 3.1: Business Process Modeling (Face Recognition System)

Every project needs a design study in order to make it more user-friendly, and ours is no exception. We simply tried to manage the scenario in an orderly and well-designed manner with minimal complexity. We chose a high-end Android version that is compatible with the majority of devices.

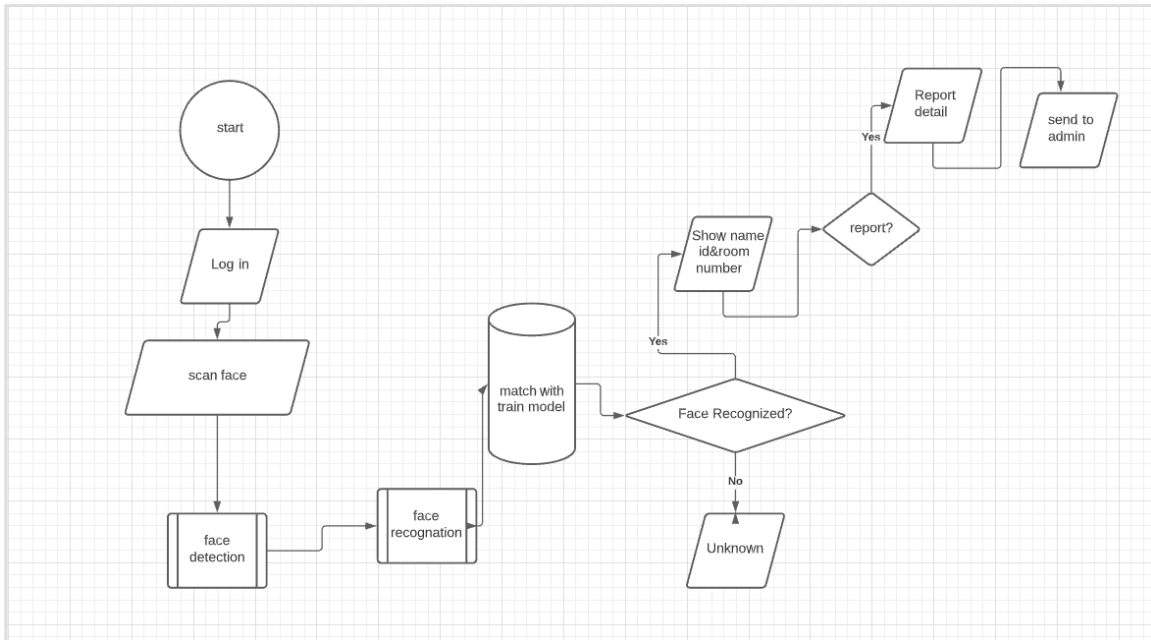


Figure 3.2: Business Process Modeling (Guard App)

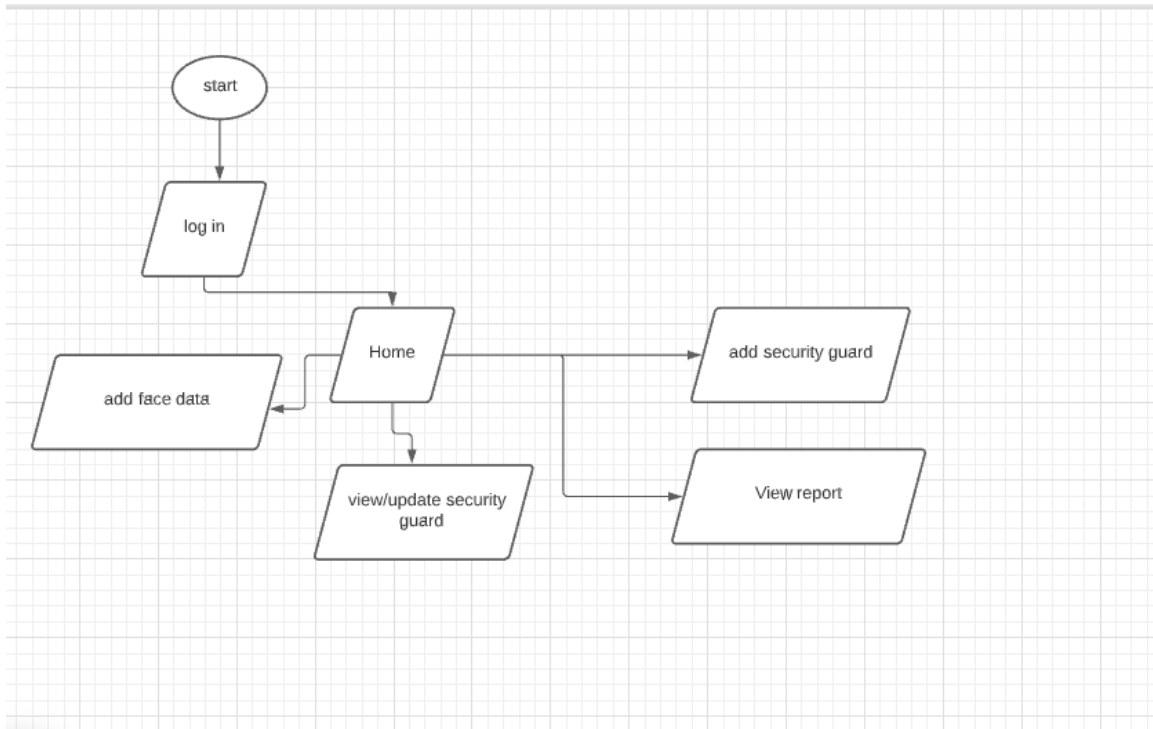


Figure 3.3: Business Process Modeling (Admin App)

3.2 Requirement Collection and Analysis

There are some basic requirements that are required for using the system.

Some of them are given below in the table:

Serial no	Requirement name	Requirement analysis
01	Admin	We need the admin to maintain the system and collect necessary data for improve the system.
02	Student Information	We need to collect student information like name, id, and room number

Table 3.1 Requirement Collection and Analysis

3.3 Use case modeling and description

The needs of a system, including internal and external factors, are gathered via use case diagrams. Our system's use case diagram is presented below.

Actors: - Admin, Security guard

The corresponding use cases for these actors are: -

System: Detect face, verify face, and gather student info

Admin: Gather student info, add student info and delete student info

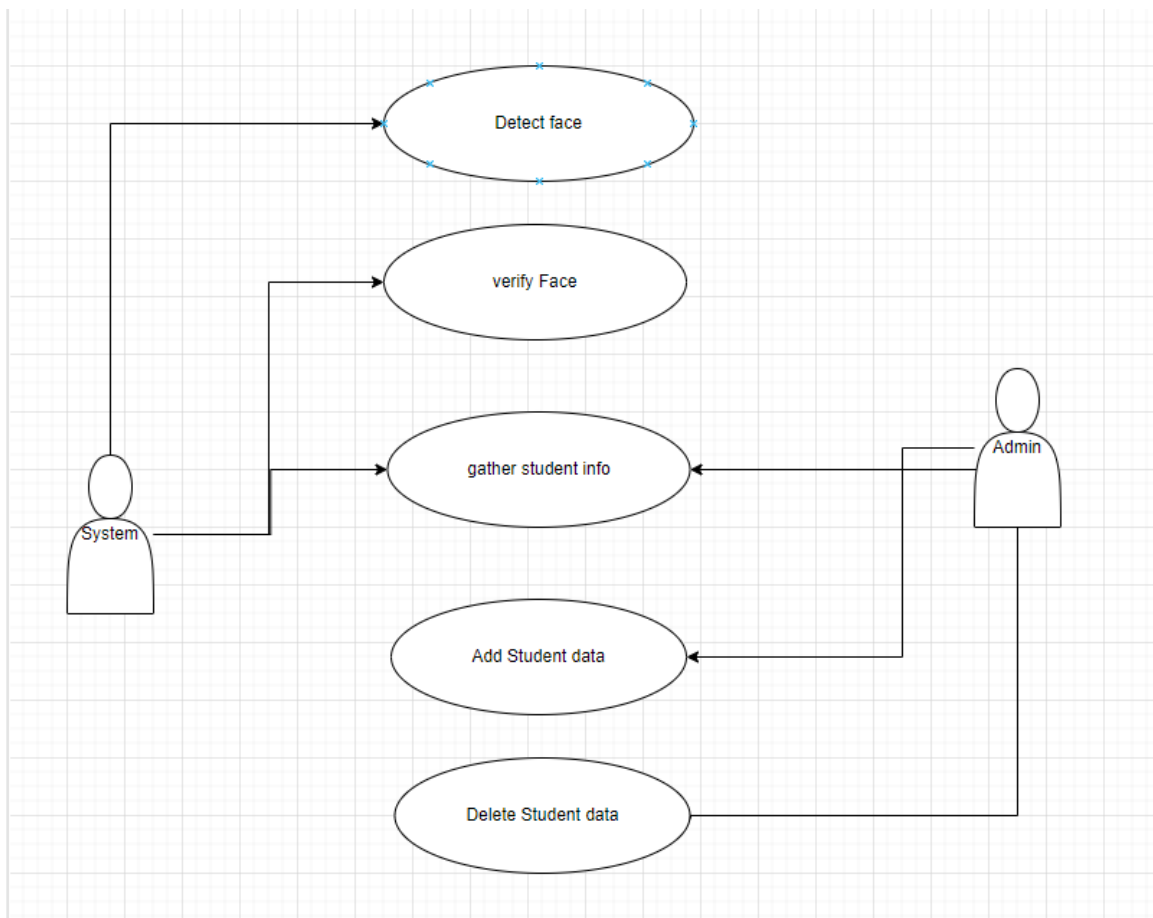


Figure 3.4: Use case Modeling (Face recognition system)

Admin: Maintain database, gather student information, add student information, delete student information, and Check reports about unauthorized people.

Security guard: check students and unknown people. The security guards can report to the authorities through this app if a guest enters the hall.

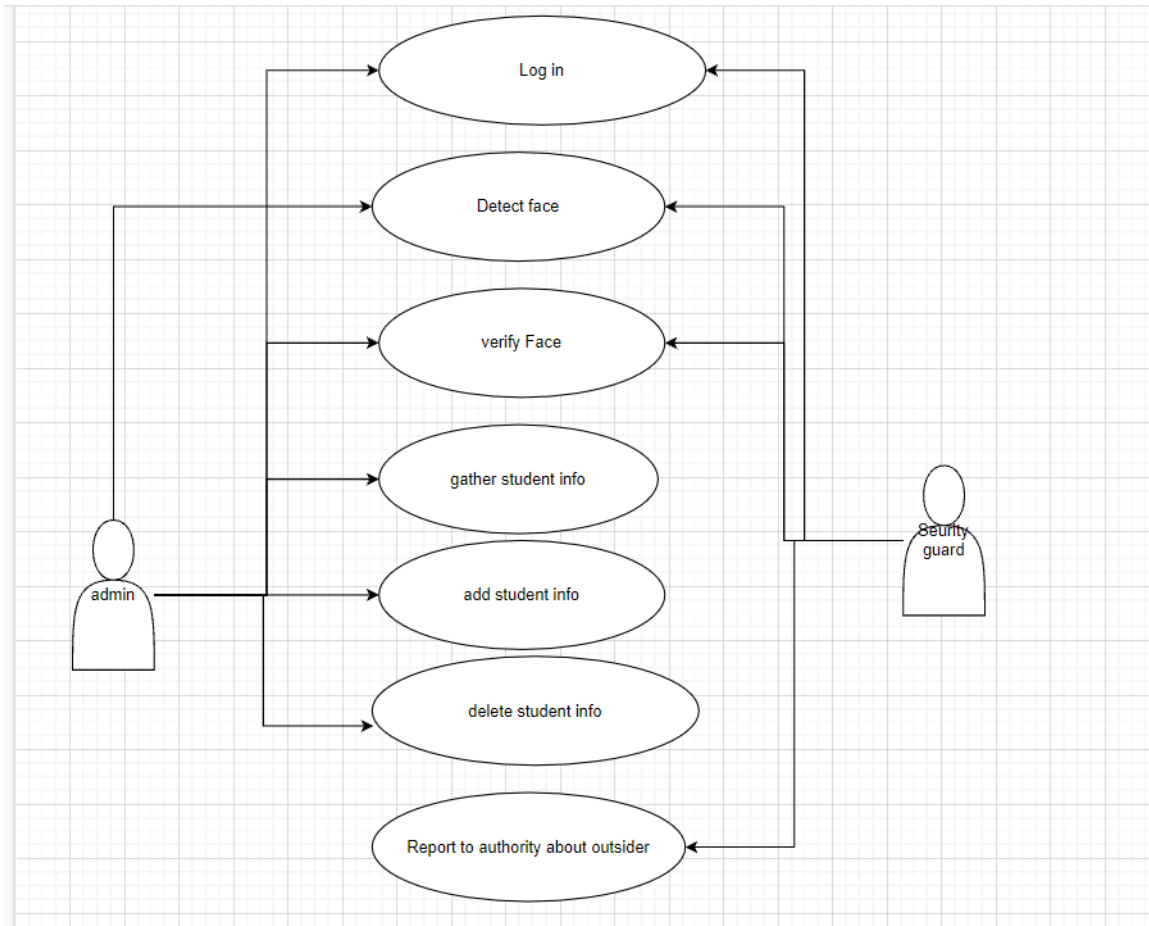


Figure 3.5: Use case Modeling (Guard App)

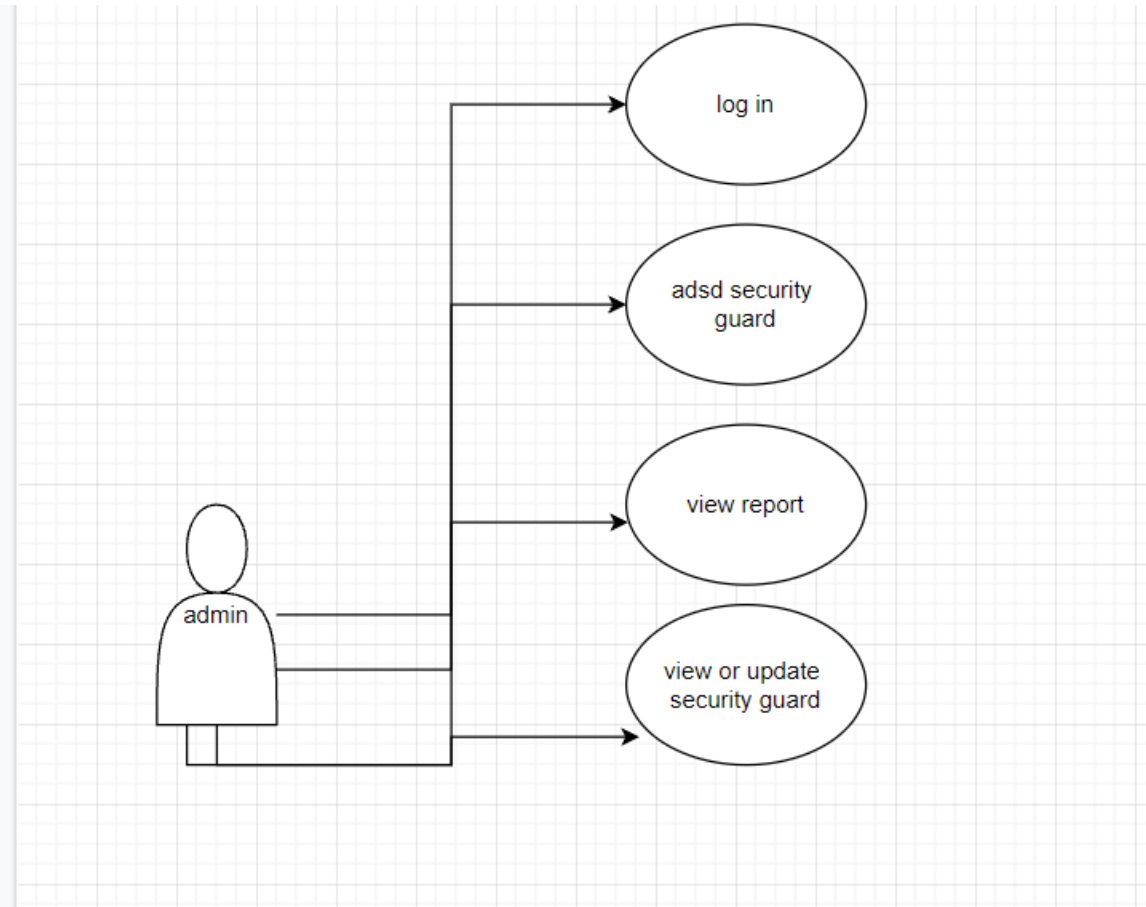


Figure 3.6: Use case Modeling (Admin App)

3.4 Logical Data Model



3.5 Design Requirements

A proper design is critical in the development of this project. Every developer attempts to do good work with a project, and the look of a good project is quite important. As a result, we focus on making our Face recognition app as user-friendly as possible. So we use XML - for Design

JAVA - Android CODE

OpenCV - (For Recognize Face in Android)

Firestore - (For Student Database)

Android Studio - Make Android App

Firestore - Online database

Tensorflow - machine learning library for train model

EfficientNet - (CNN based neural network algorithm to scale all

Dimensions of depth/width/resolution from image)

Python - Code for train model

Kaggle - Software for training the model

CHAPTER 4

Design Specification

4.1 Front-end Design

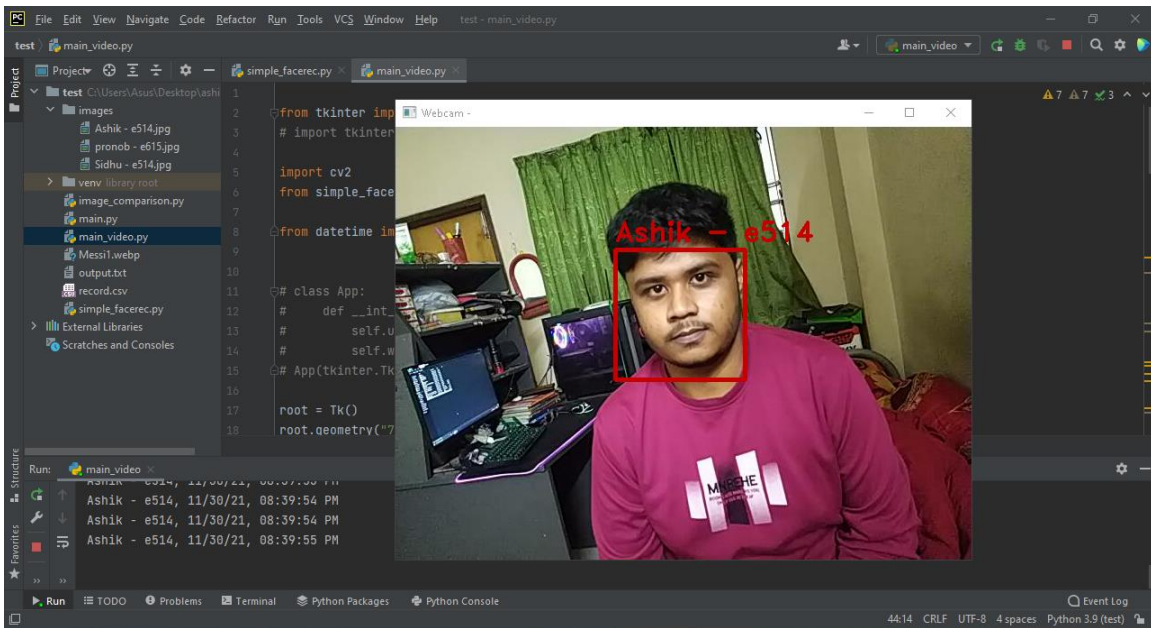


Figure 4.1: Detecting known faces

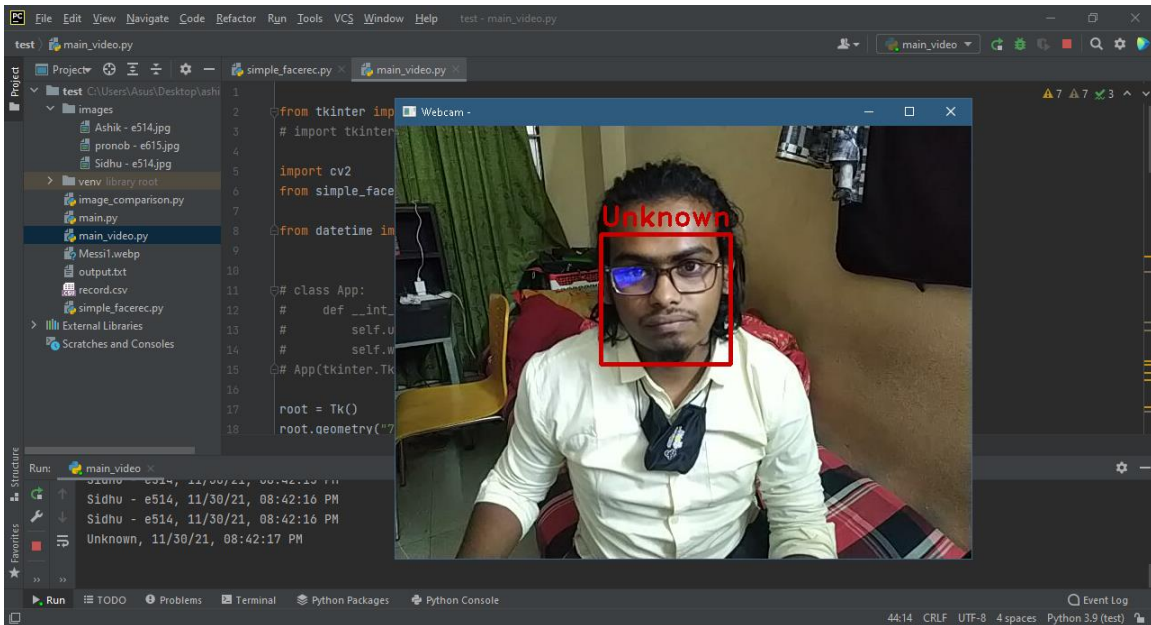


Figure 4.2: Detecting unknown faces|

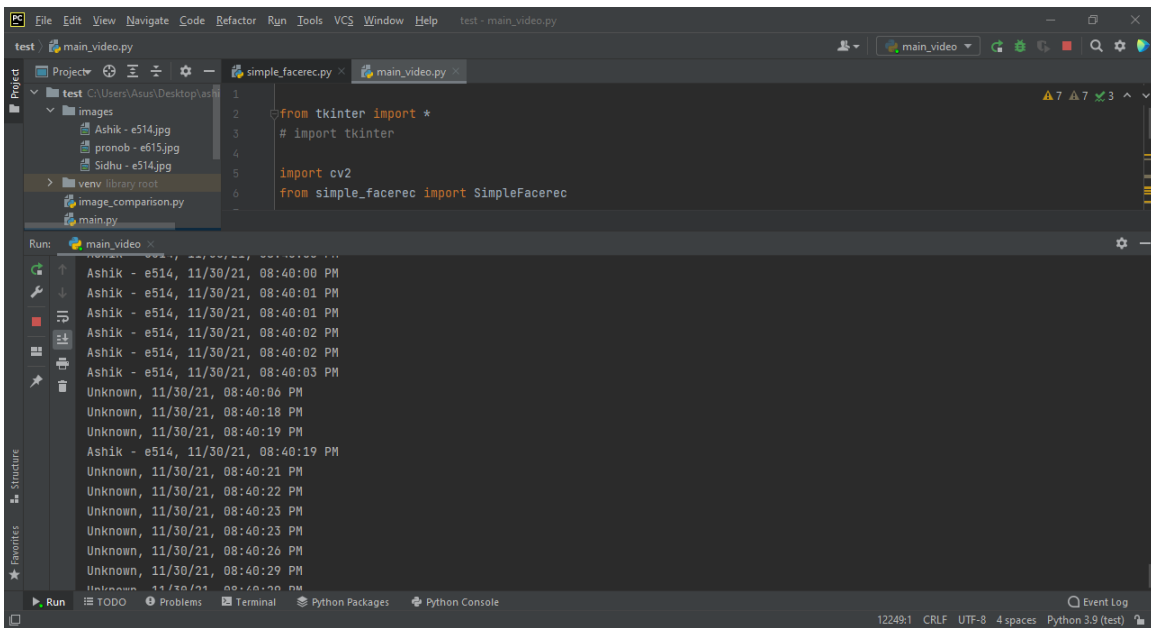


Figure 4.3: Output of face info

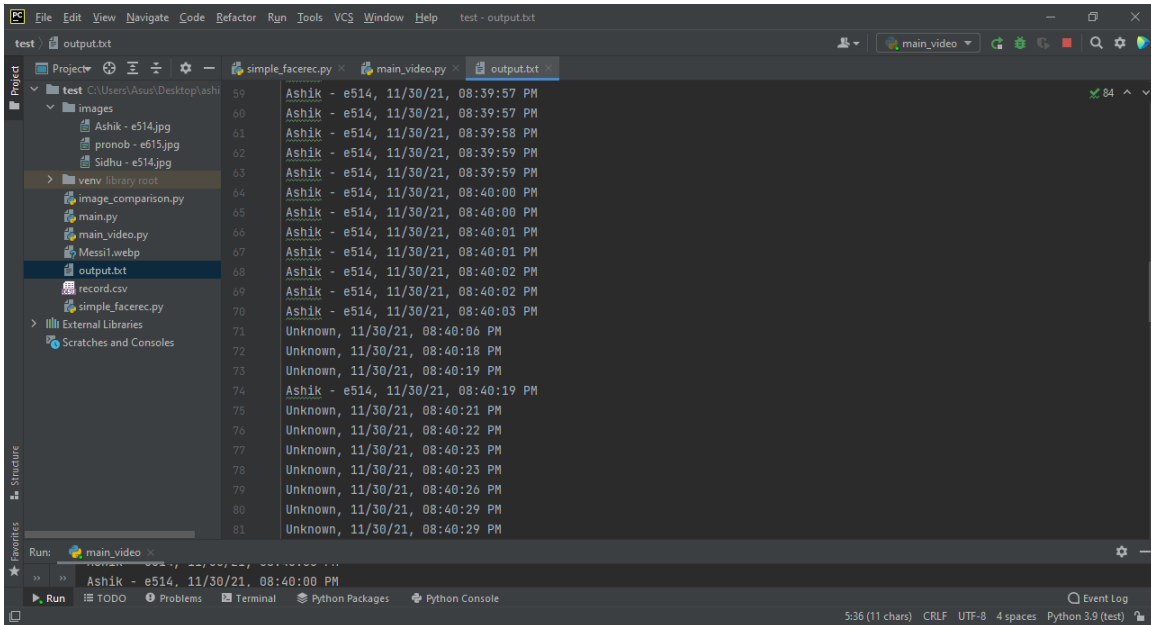


Figure 4.4: Face info into text format

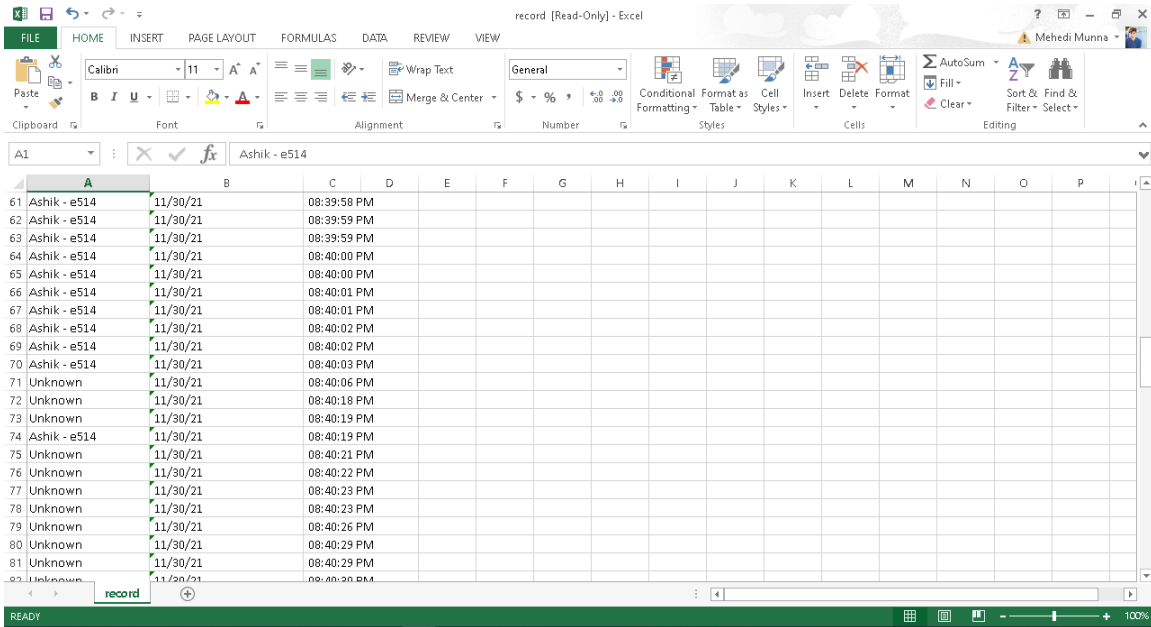


Figure 4.5: Face info into Excel Sheet

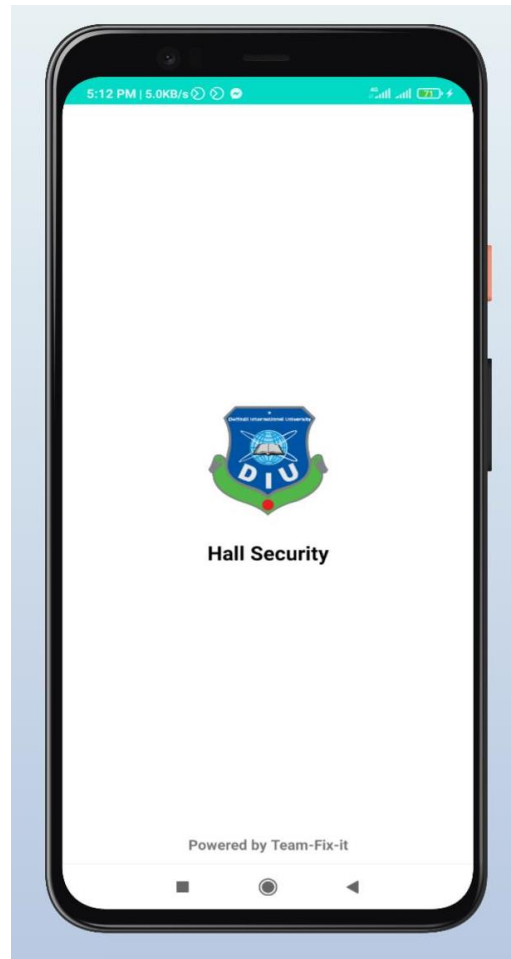


Figure 4.6: SplashScreen of the application

This is the first screen that will appear when a user clicks on the app icon. This is an animated splash screen where users can see our DIU logo along with the app name & our team name. This screen will automatically change after 2 sec.

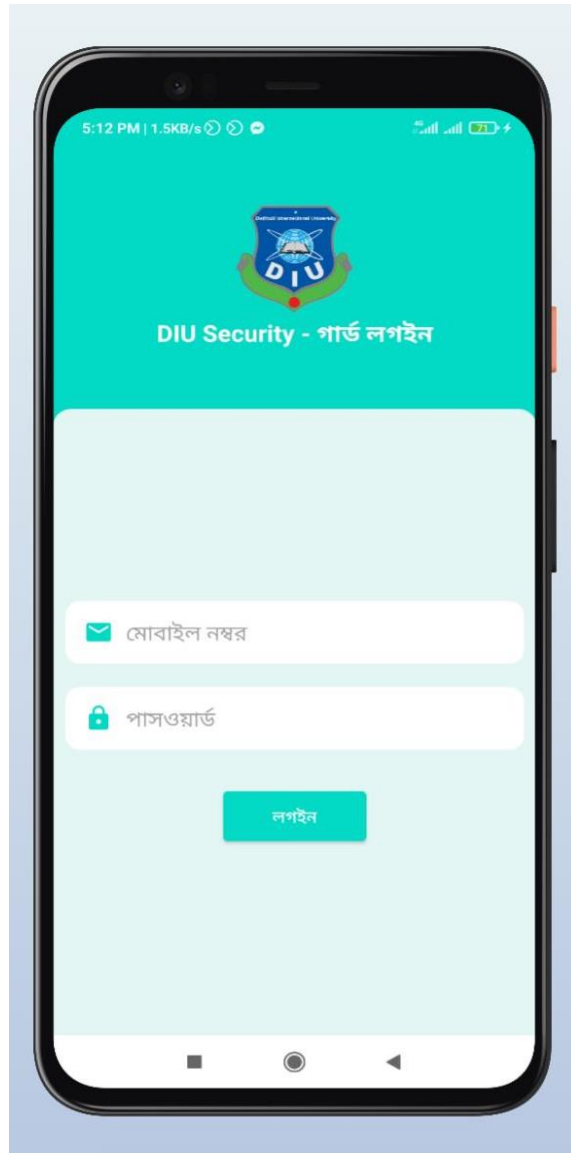


Figure 4.7: Login screen for guard app

After the splash screen, this login screen will appear for the user if it is his/her first time of the app. That user login information will be provided by the administration. Without authentication, no user can log in to our app.

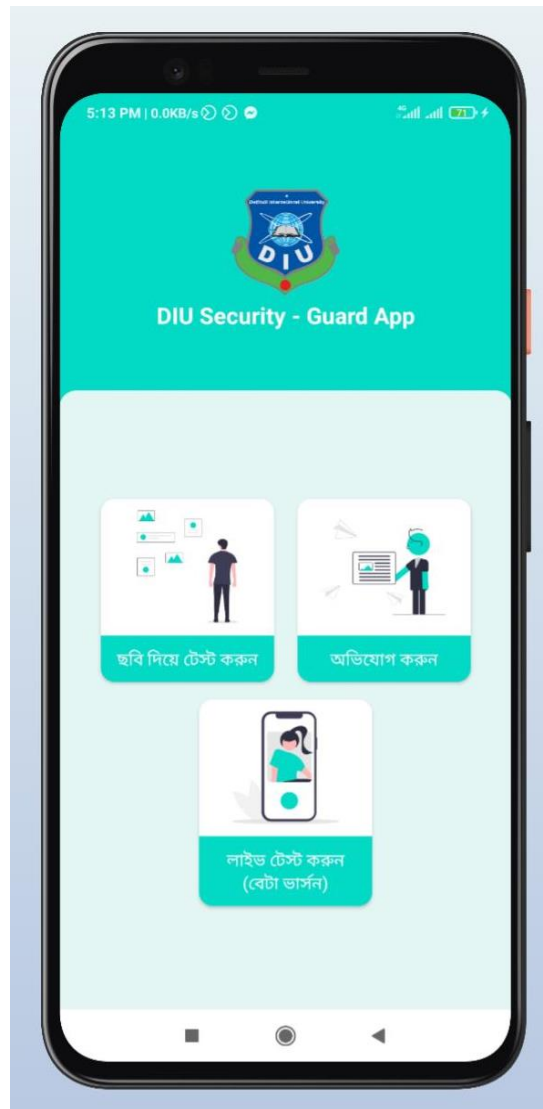


Figure 4.8: Home page of guard app

After successful login, the user will be intent into the home screen. From the home screen, users can do 3 thigs which are already implemented in this app. Users can recognize faces from gallery photos. Users can write reports about any student or any occurrence that happened. Users can detect faces using live camera features.

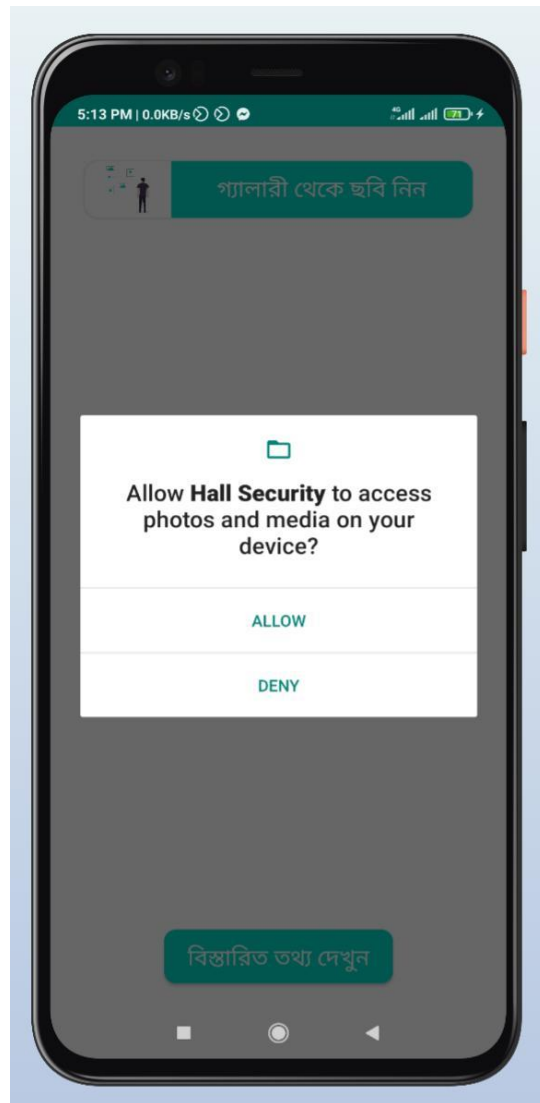


Figure 4.9: Getting storage permission from the user

When the user clicks on the test from the gallery option then this pop-up will come. To access user's storage so that users can select images from their gallery and check that faces are recognizable or not.

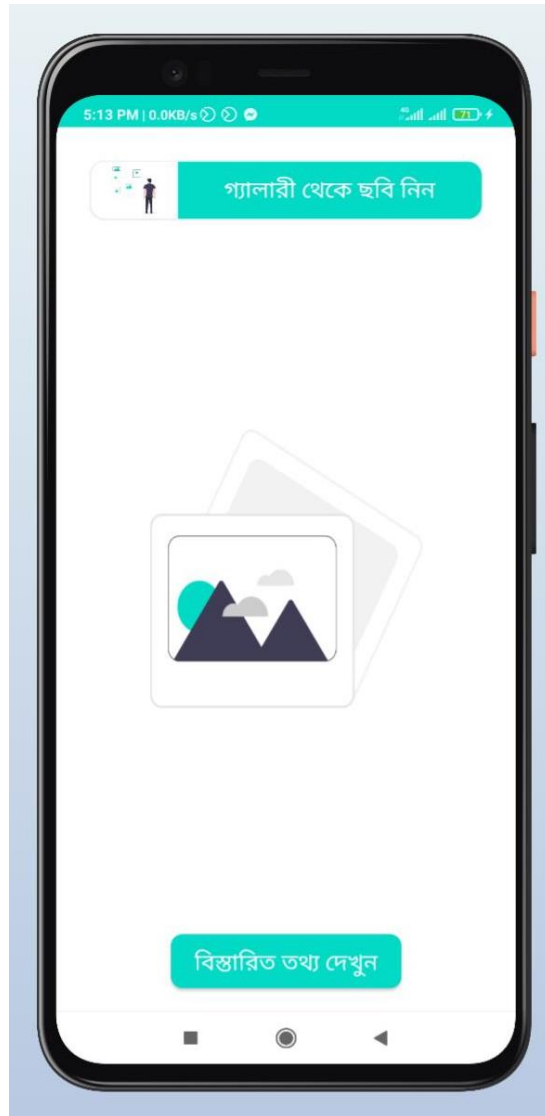


Figure 4.10: Gallery image chooser view

This is the screen for getting images from the gallery and recognizing those images. We have a dedicated button to choose an image from the gallery then this image will be chosen and detected from the gallery.

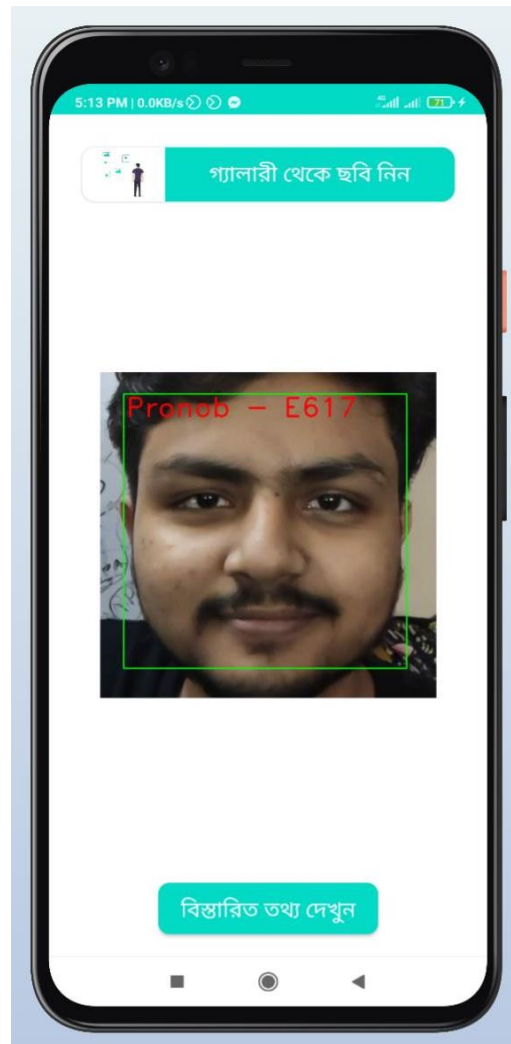


Figure 4.11: Recognizing image after choosing

After choosing a photo from the gallery the image will be detected using face detection technology we use in the background. If the image is detected then it will show the name into the image view.

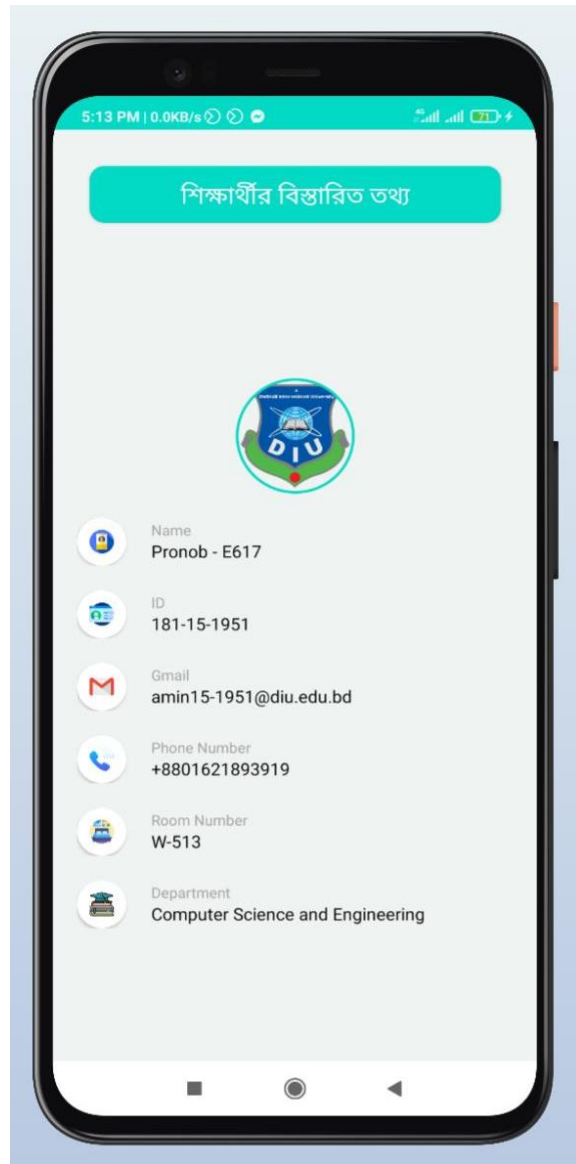


Figure 4.12: Details screen of the student

This is the detailed information screen of the student. After recognizing this feature will work, if not recognized then it will not open the details screen. From this screen, users can collect information about that student.

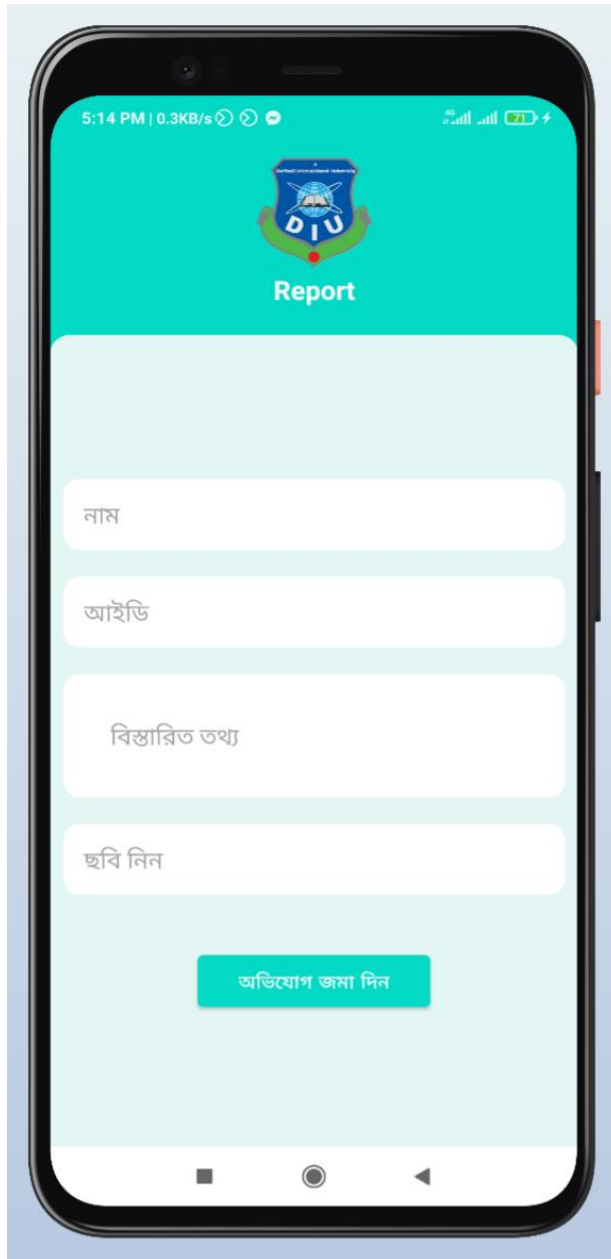


Figure 4.13: Report Screen for guard app

This is the report screen where users can report about a student if he/she does anything wrong. This report will automatically go to the administration app.

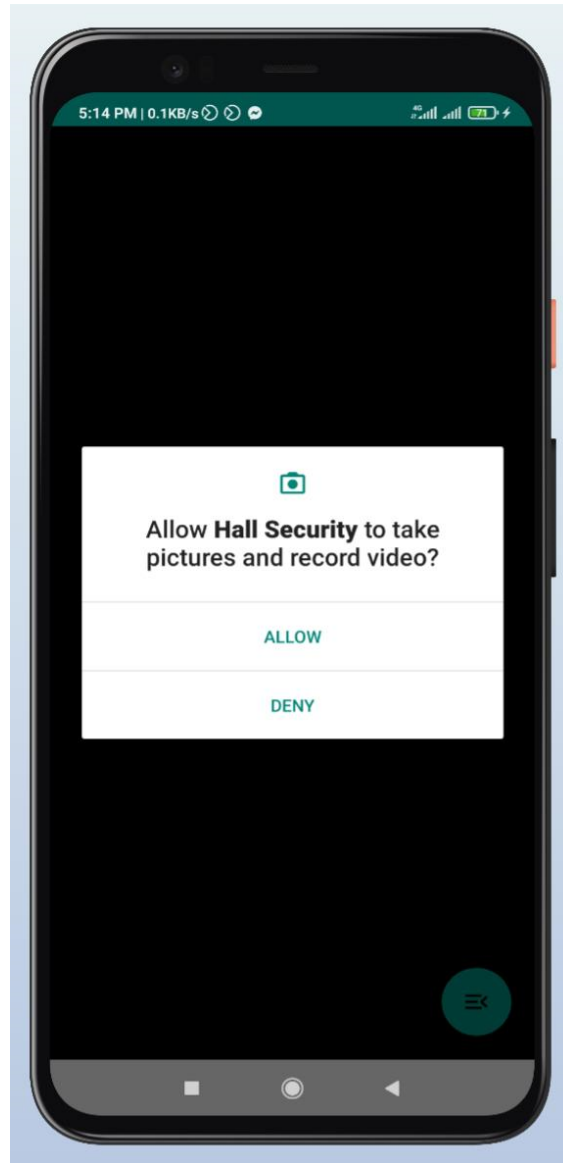


Figure 4.14: Getting camera permission from the user

When the user will try to open live recognition, the first time it will ask for camera permission and if the user gives the camera permission then it will start recognizing images using the backend code that we will use.

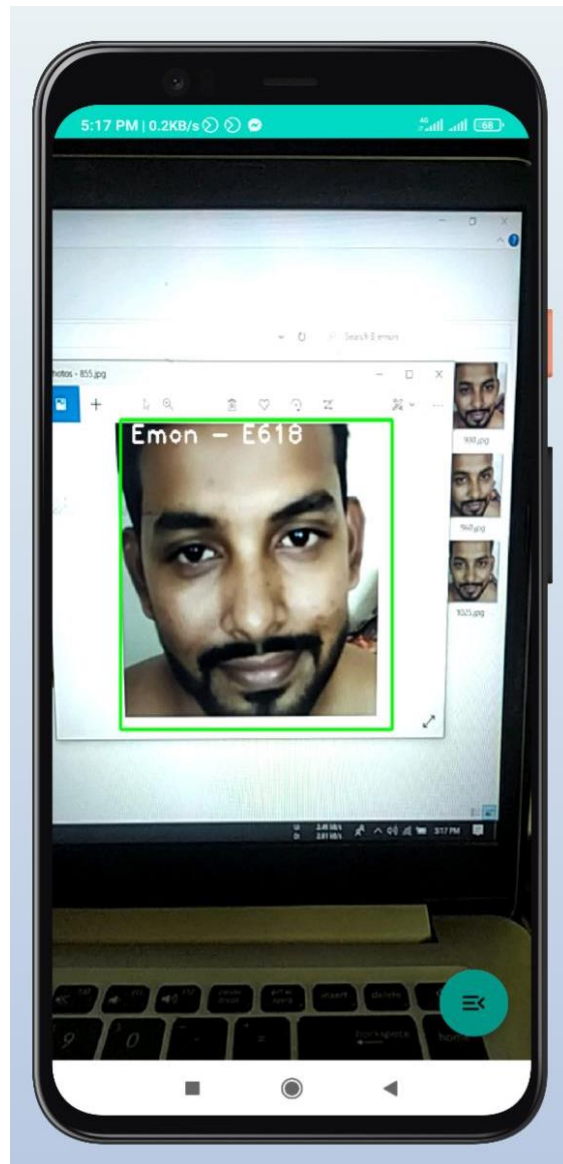


Figure 4.15: Live recognition screen

This is the live recognition screen for the guard app where guards can detect faces using a live camera. In this screen, we have a button below from where the user can get detailed information about that student.

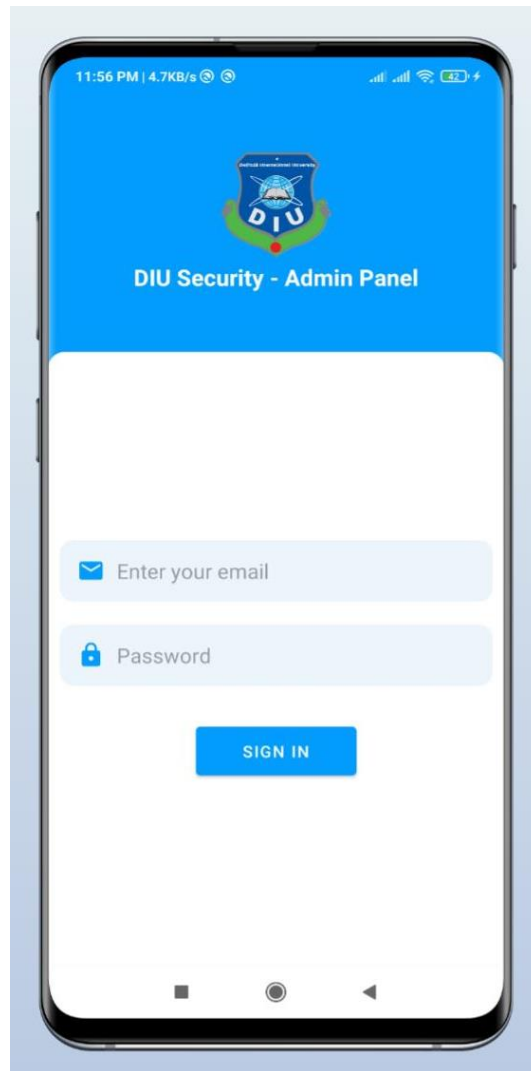


Figure 4.16: Login screen (Admin App)

This is our Admin App Login screen where admin can log in to their account.

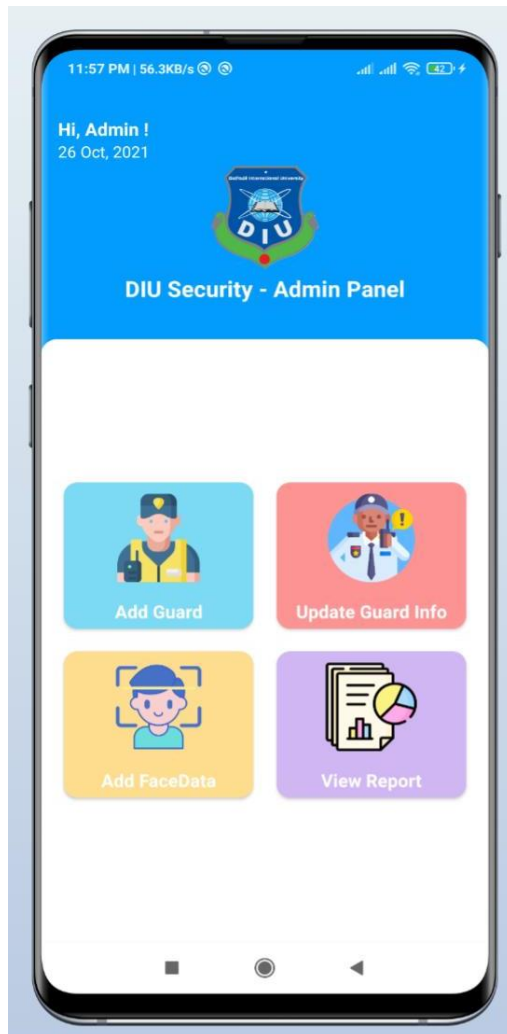


Figure 4.17: Home Screen (Admin App)

This is the home screen of our admin app where an admin can add a guard, update guard information, add trained face information, and also can view reports

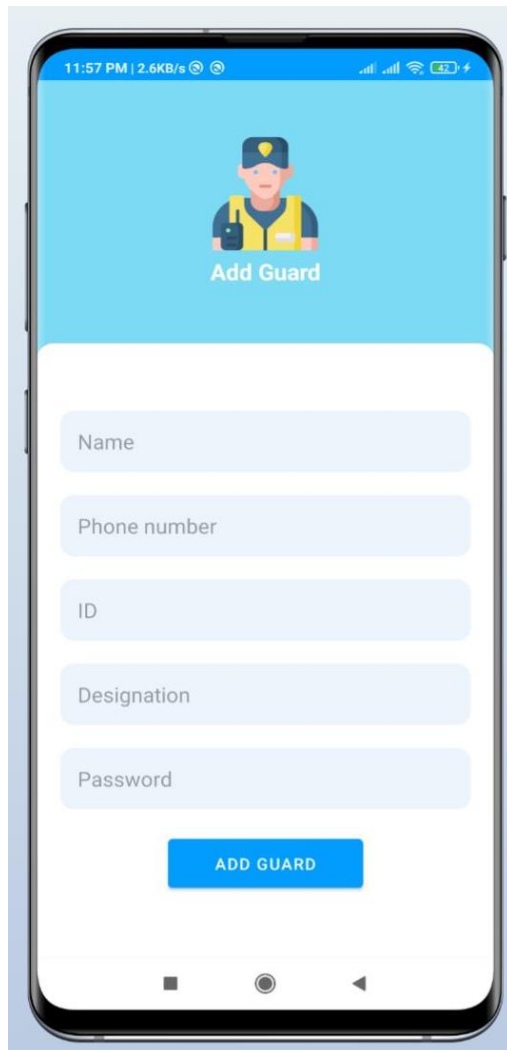


Figure 4.18: Add Guard Info Screen (Admin App)

From this screen, we are watching in figure 4.18 it is the add guard info screen where an admin can add guard information. Using this information a guard only can log into their account. So before giving guard this app admin need to create their account from this app.

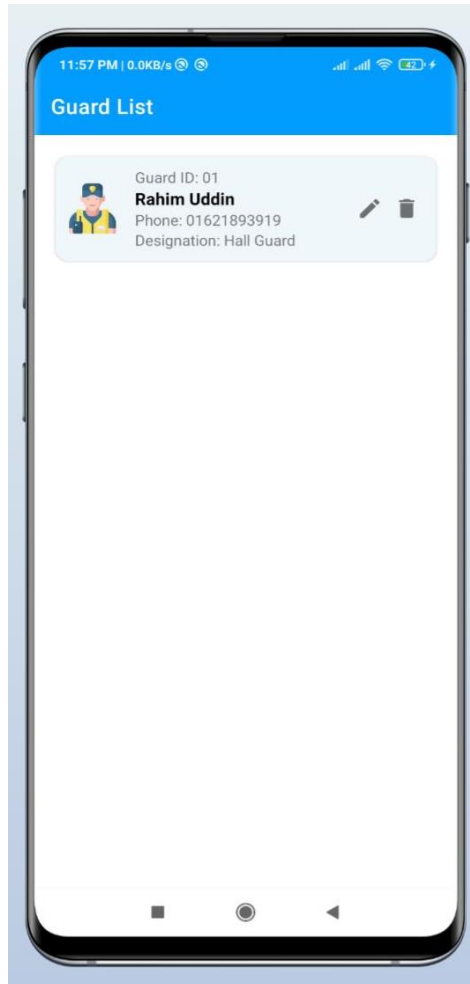


Figure 4.19: Guard List Screen (Admin App)

This is the guard list screen where an admin can check how many guard account they created and where they can edit/update guard information as they want also can delete guard information if an admin wants.

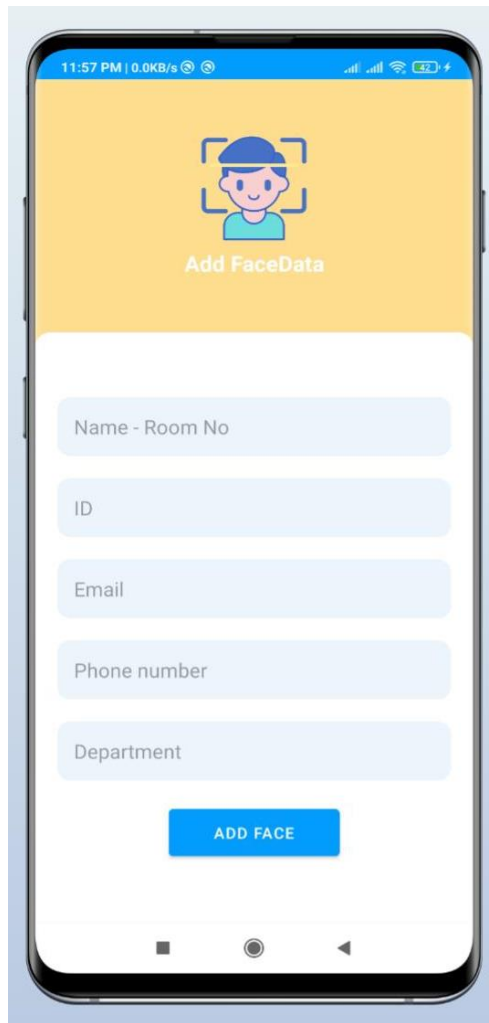


Figure 4.20: Add Face Data Screen (Admin App)

This is the screen where the admin can add face information that was trained before in the Kaggle model. This is mainly stored in the firebase database and from the guard app if a guard checks a user and if recognizes faces this face info will be taken from this database that the admin is adding.

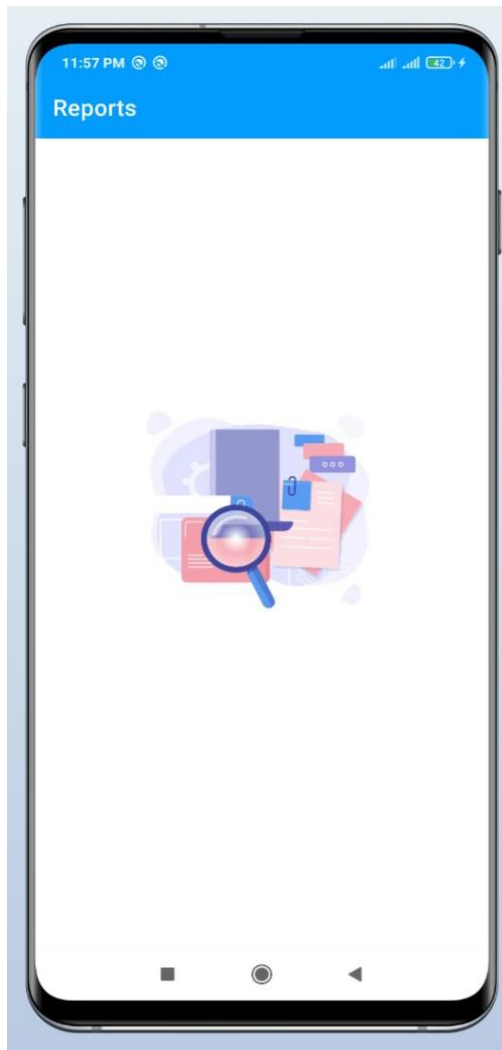


Figure 4.21: Reports Screen (Admin App)

In figure 4.21 we see a nice design. This is mainly an animated screen if there are no reports then it will be animated automatically. And If there is any report by the guard then it will appear here.

4.2 Back-end Design

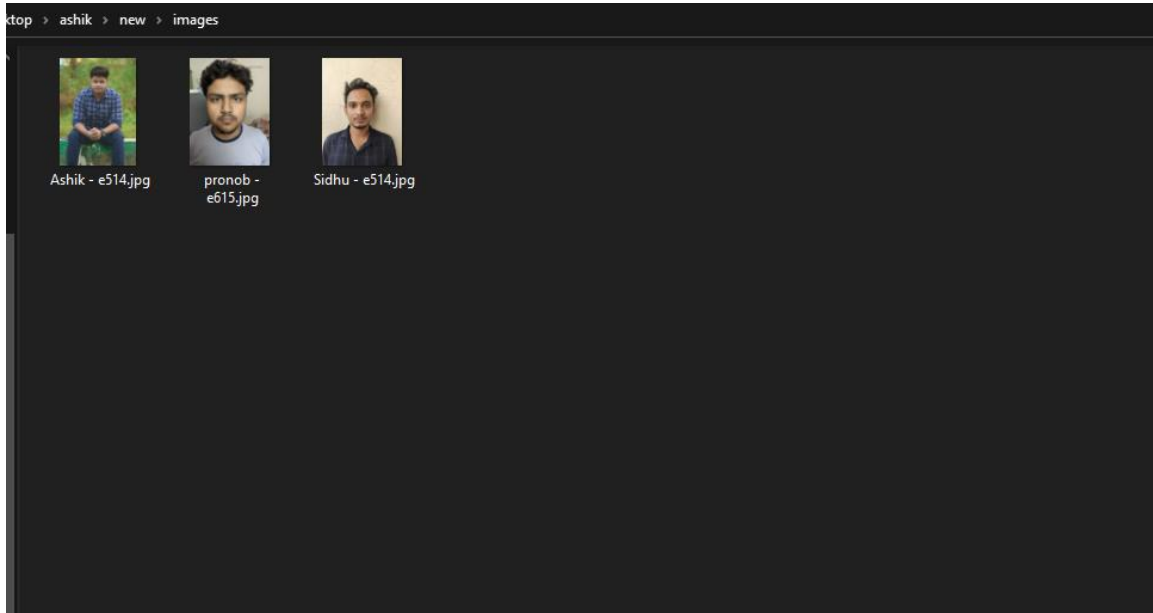


Figure 4.22: Face data of hall student

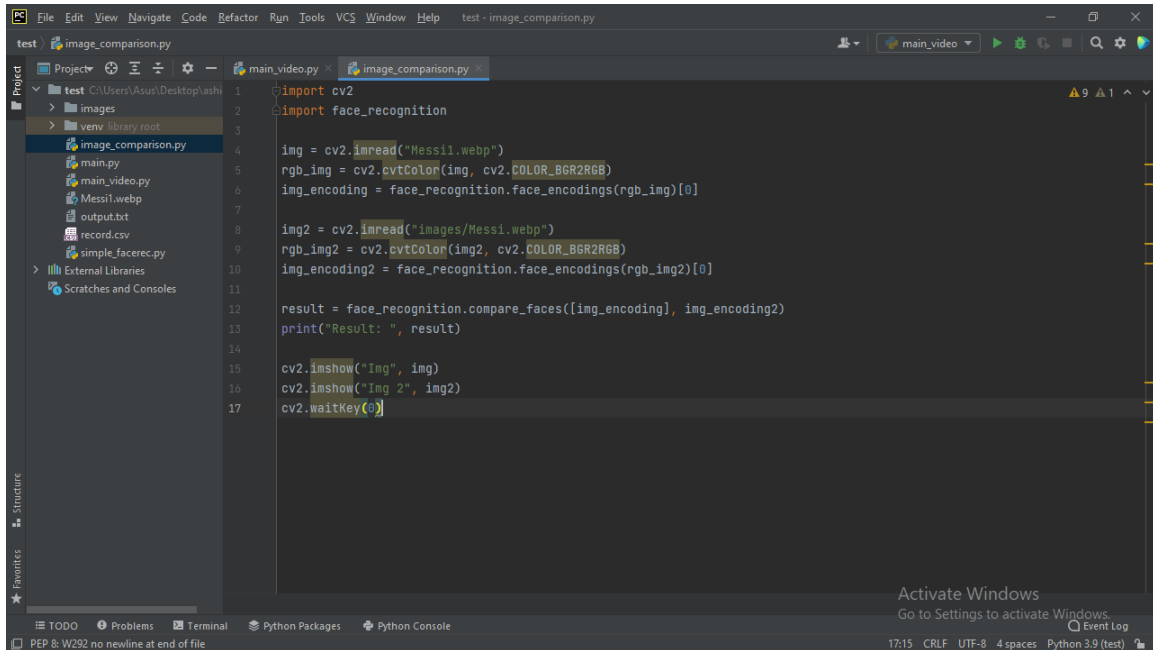


Figure 4.23: Comparing images using OpenCV

```

1 import ...
2
3
4
5
6
7 class SimpleFacerec:
8     def __init__(self):
9         self.known_face_encodings = []
10        self.known_face_names = []
11
12        # Resize frame for a faster speed
13        self.frame_resizing = 0.25
14
15    def Load_encoding_images(self, images_path):
16        """
17        Load encoding images from path
18        :param images_path:
19        :return:
20        """
21        # Load Images
22        images_path = glob.glob(os.path.join(images_path, "*.*"))
23
24        print("{} encoding images found.".format(len(images_path)))
25
26        # Store image encoding and names
27        for img_path in images_path:
28            img = cv2.imread(img_path)
29            path_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
30            self.known_face_encodings.append(dlib.get_face_encodings(path_img, self.frame_resizing))
31
32    def detect_known_faces(self, video_path):
33        """
34        Detect known faces from video
35        :param video_path:
36        :return:
37        """
38        # Load Video
39        video = cv2.VideoCapture(video_path)
40
41        # Get video frame
42        while True:
43            ret, frame = video.read()
44            if not ret:
45                break
46
47            # Detect faces
48            faces = self.detect_faces(frame)
49
50            # Draw bounding boxes
51            for (i, (x, y, x2, y2)) in enumerate(faces):
52                cv2.rectangle(frame, (x, y), (x2, y2), (0, 255, 0), 2)
53
54            # Display the frame
55            cv2.imshow("Video", frame)
56
57            # Press 'q' to quit
58            if cv2.waitKey(1) & 0xFF == ord('q'):
59                break
60
61        # Release the video
62        video.release()
63
64        # Destroy all windows
65        cv2.destroyAllWindows()
66
67    def detect_faces(self, frame):
68        """
69        Detect faces from frame
70        :param frame:
71        :return:
72        """
73        # Detect faces
74        faces = dlib.get_face_encodings(frame, self.frame_resizing)
75
76        # Get face names
77        names = []
78        for (i, (x, y, x2, y2)) in enumerate(faces):
79            name = self.known_face_names[i]
80            names.append(name)
81
82        return faces, names
83
84    def detect_faces_names(self, frame):
85        """
86        Detect faces from frame and return names
87        :param frame:
88        :return:
89        """
90        # Detect faces
91        faces, names = self.detect_faces(frame)
92
93        return names
94
95    def detect_faces_names(self, frame):
96        """
97        Detect faces from frame and return names
98        :param frame:
99        :return:
100       """
101       # Detect faces
102       faces, names = self.detect_faces(frame)
103
104       return names

```

Figure 4.24: Encoding image

```

1 from tkinter import *
2 # import tkinter
3
4 import cv2
5 from simple_facerec import SimpleFacerec
6
7 from datetime import datetime
8
9 # class App:
10 # def __init__(self, window, window_title, video_source=0):
11 #     self.update()
12 #     self.window.mainloop()
13 # App(tkinter.Tk(), "Tkinter and OpenCV", "big_buck_bunny_480p_surrround-fix.avi")
14
15 root = Tk()
16 root.geometry("700x350")
17
18 menu = Menu(root)
19 root.config(menu=menu)
20 filemenu = Menu(menu)
21 menu.add_cascade(label="File", menu=filemenu)
22 filemenu.add_command(label="Open", command=self.open)
23
24 # def open(self):
25 #     filename = askopenfilename()
26 #     if filename:
27 #         self.video_source = filename
28 #         self.update()
29 #
30 # def update(self):
31 #     self.window.title(f"Tkinter and OpenCV - {datetime.now()}")
32 #     self.window.geometry(f"700x350")
33 #     self.window.resizable(True, True)
34 #     self.window.update()
35 #
36 # def main(self):
37 #     self.open()
38 #     self.update()
39 #     self.window.mainloop()
40
41 if __name__ == "__main__":
42     app = App()
43     app.main()

```

Figure 4.25: Code for recognize face from video

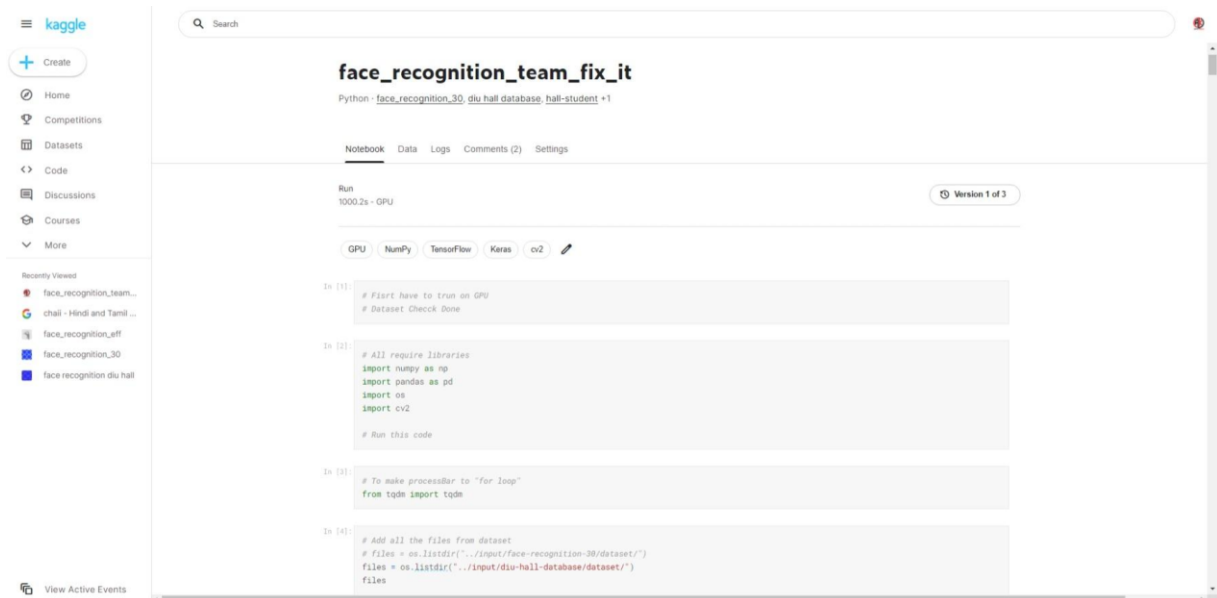


Figure 4.26: Importing required libraries

First import the python libraries for training the faces. Some of them are NumPy, pandas, os, cv2, and tqdm. Then we import the list that we used to train.

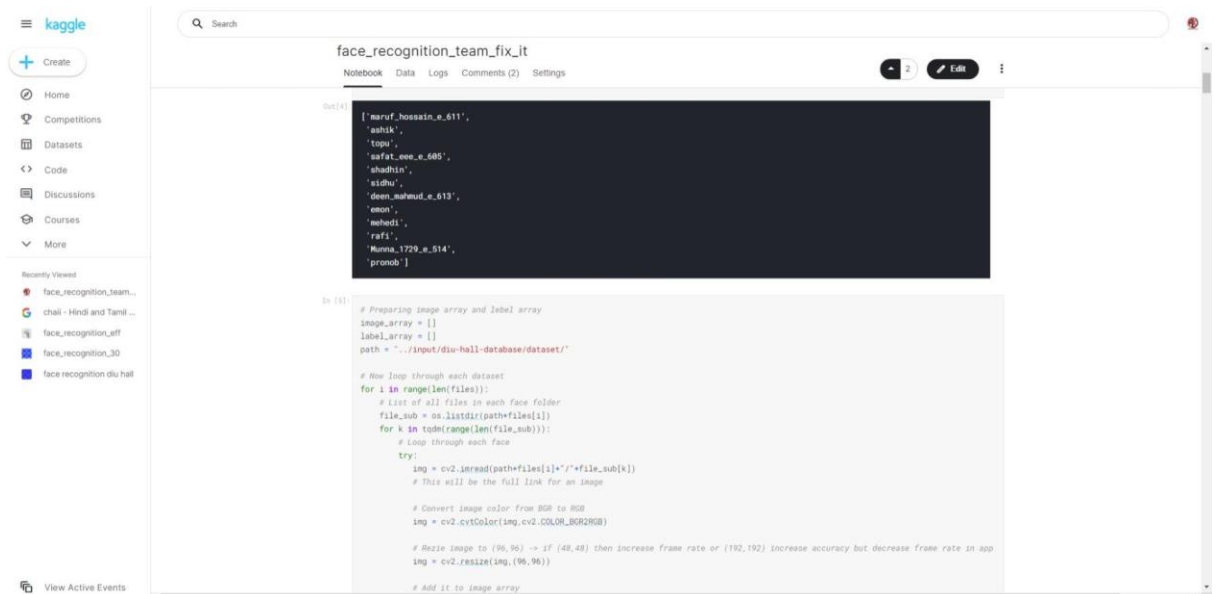
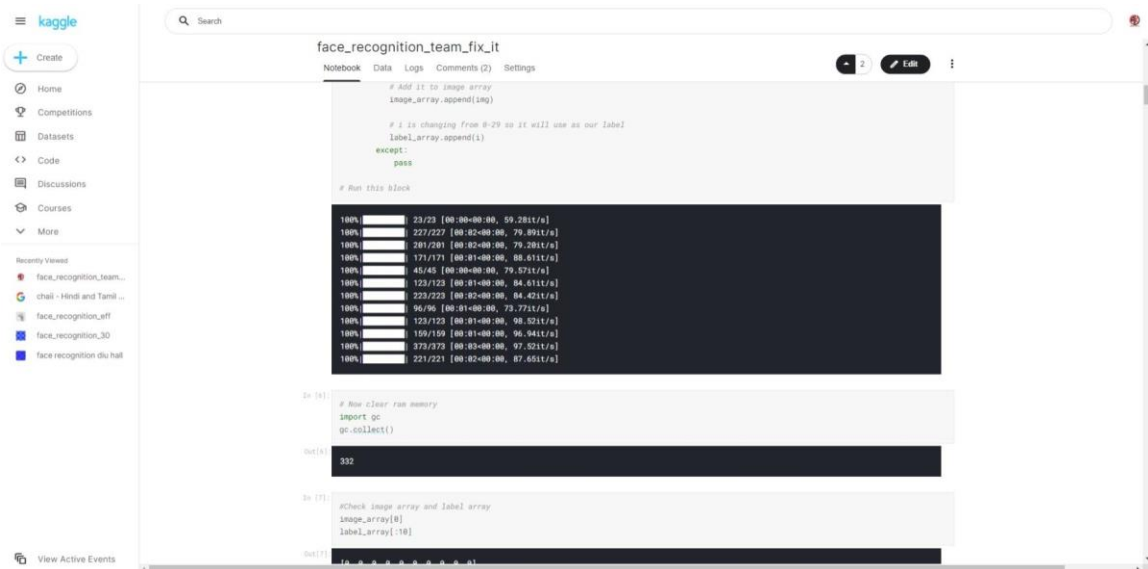


Figure 4.27: Importing required libraries

This is the output of the list that we will train faces. Then we are making two images to append those images.



```
# Add it to image array
image_array.append(img)

# I is changing from 0-29 so it will use as our label
label_array.append(i)
except:
    pass

# Run this block

100% |#####| 23/23 [00:00:00.00, 59.281t/s]
100% |#####| 227/227 [00:02:00.00, 79.891t/s]
100% |#####| 201/201 [00:02:00.00, 79.281t/s]
100% |#####| 171/171 [00:01:00.00, 85.511t/s]
100% |#####| 45/45 [00:00:00.00, 79.571t/s]
100% |#####| 123/123 [00:01:00.00, 84.611t/s]
100% |#####| 223/223 [00:02:00.00, 84.421t/s]
100% |#####| 90/90 [00:01:00.00, 73.771t/s]
100% |#####| 123/123 [00:01:00.00, 90.521t/s]
100% |#####| 199/199 [00:01:00.00, 96.941t/s]
100% |#####| 373/373 [00:03:00.00, 97.521t/s]
100% |#####| 221/221 [00:02:00.00, 87.651t/s]

In [34]: # Now clear ram memory
import gc
gc.collect()

Out[34]: 332

In [35]: #Check image array and label array
image_array[0]
label_array[10]
```

Figure 4.28: Then we clear ram memory and then check the array

Now we will clear ram memory then we check if the image array and label array are appended or not.


```

In [7]: #Check image array and Label array
image_array[0]
label_array[10]

Out[7]: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

In [8]: # Now we will convert image_array list to array and make 0-255 scale to 0-1 scale
image_array = np.array(image_array)/255
label_array = np.array(label_array)

In [9]: image_array[0]
# Checking the scale is converted 0-1

Out[9]: array([[0.74117647, 0.75294118, 0.7254902 ],
 [0.63021569, 0.64998039, 0.62352941],
 [0.61969704, 0.62352941, 0.59215686],
 ...,
 [0.24998039, 0.24312725, 0.24708882],
 [0.72156863, 0.71764706, 0.69889222],
 [0.8592157 , 0.79697843, 0.74901561]],
 ...,
 [0.74981961, 0.76878431, 0.7254902 ],
 [0.78186878, 0.71322549, 0.67843137],
 [0.61969704, 0.62352941, 0.59215686],
 ...,
 [0.23529412, 0.23529412, 0.22745698],
 [0.4627451 , 0.4627451 , 0.42352941],
 [0.81968627, 0.80784314, 0.78878431]],
 ...,
 [0.74599904, 0.75662725, 0.72156863],
 [0.72333333, 0.7372549 , 0.78888235],
 [0.78186878, 0.78988392, 0.67843137],
 ...,

```

Figure 4.29: Converting image scale

Now we convert the time-image array and resize it to a 0-1 scale. Then we check the image array.

```

In [10]: # Now split and shuffle our dataset for training and validation
from sklearn.model_selection import train_test_split
# images label split ratio
X_train,X_test,Y_train,Y_test=train_test_split(image_array,label_array,test_size=0.15)

In [11]: # X_train and Y_train are 85% of images and label
# X_test and Y_test are 15% of images and label

In [12]: # Now we will create our model
# We will use image classification model

In [13]: from keras import layers, callbacks, utils, applications, optimizers
from keras.models import Sequential, Model, load_model

In [14]: # We need to import tensorflow
import tensorflow as tf

In [15]: # Create Model
model = Sequential()

# We will use EfficientNet pretrained model
# B0 to B7 -> high frame low accuracy and B7 -> low frame high accuracy
pretrained_model = tf.keras.applications.EfficientNetB0(input_shape=(96, 96, 3), include_top=False, weights='imagenet')

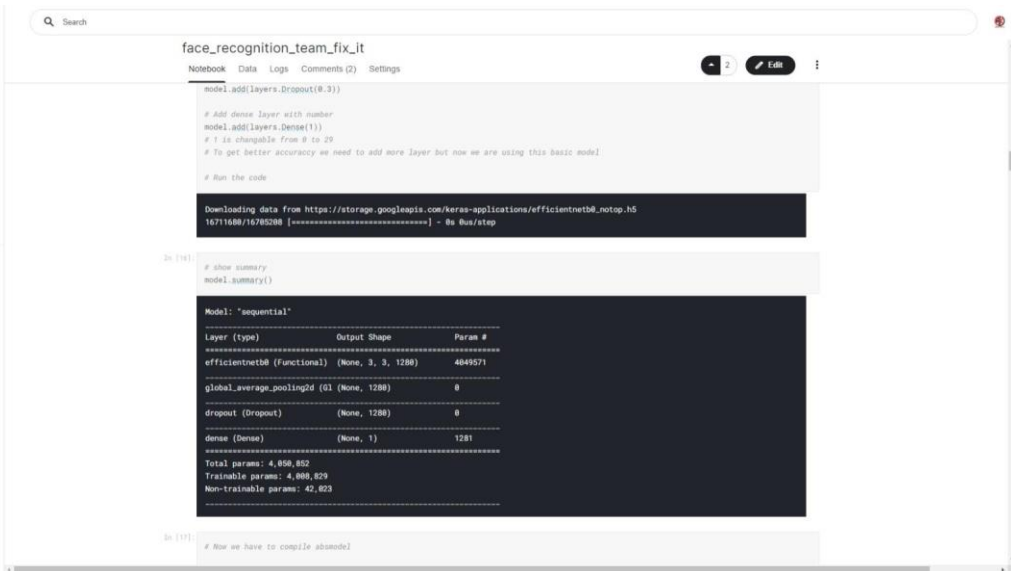
# Now add this to our model
model.add(pretrained_model)

# Now add some layer to improve accuracy
model.add(layers.GlobalAveragePooling2D())
model.add(layers.Dropout(0.3))

```

Figure 4.30: Split dataset and create a model using TensorFlow

After checking the scaling then we split the dataset for training and validation. Now create the model and using the image classification model we also import TensorFlow



```
model.add(layers.Dropout(0.3))

# Add dense layer with number
model.add(layers.Dense(1))
# It is changeable from 0 to 20
# To get better accuracy we need to add more layer but now we are using this basic model

# Run the code

Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb0_notop.h5
16711698/16786298 [=====] - 0s/0s/step

In [1]:
# Show summary
model.summary()

Model: "sequential"
Layer (type)                Output Shape              Param #
-----
efficientnetb0 (Function)    (None, 3, 3, 1280)        4649371
global_average_pooling2d (G1 (None, 1280)          0
dropout (Dropout)           (None, 1280)              0
dense (Dense)                (None, 1)                 1281

Total params: 4,650,652
Trainable params: 4,649,371
Non-trainable params: 42,823

In [1]:
# Now we have to compile absmode1
```

Figure 4.31: Model summary

Now we add some layers to improve accuracy and dense layers. After that, we show the model summary.

```

model.compile(optimizer="adam", loss="mean_squared_error", metrics=["mae"])
# optimizer: There are many optimizer to improve accuracy. Like (SGD, etc)
# loss: We can try different losses to improve accuracy

# Create checkpoint to save best accuracy model
ckpt_path = "trained_model1/model1"
model_checkpoint = tf.keras.callbacks.ModelCheckpoint(filepath=ckpt_path, monitor="val_mae", mode="auto", save_best_only=True, save_weights_only=True)

# monitor = "val_mae" : watch mae of validation set and when it decrease save the model
# mode: It is use to check for decrease or increase in mae -> (auto, min, max)
# save_weights_only: It true save only weight (matrix of number)

# Create a lr reducer which will decrease learning rate when accuracy doesn't increase
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(factor=0.9, monitor="val_mae", mode="auto", cooldown=0, patience=5, verbose=1, min_lr=1e-6)

# patience: wait for 5 epoch then decrease learning rate
# verbose: show accuracy(val_mae) every epoch
# min_lr: minimum learning abstrate

# We can use other reduce_lr to increase accuracy

# Train model
Epoch = 300
Batch_Size = 64
history = model.fit(X_train, Y_train, validation_data=(X_test, Y_test), batch_size = Batch_Size, epochs = Epoch, callbacks = [model_checkpoint, reduce_lr])

#Run the code

```

Figure 4.32: Create Checkpoint for best accuracy

Now we make a checkpoint that can save the best accuracy model and then create an LR reducer which will decrease the learning rate if the accuracy doesn't increase.

```

Batch_Size = 64
history = model.fit(X_train, Y_train, validation_data=(X_test, Y_test), batch_size = Batch_Size, epochs = Epoch, callbacks = [model_checkpoint, reduce_lr])

#Run the code

Epoch 1/300
27/27 [=====] - 19s 184ms/step - loss: 26.7787 - mae: 4.2356 - val_loss: 31.2874 - val_mae: 4.6382
Epoch 2/300
27/27 [=====] - 3s 187ms/step - loss: 1.6334 - mae: 0.9767 - val_loss: 15.7657 - val_mae: 3.4535
Epoch 3/300
27/27 [=====] - 3s 188ms/step - loss: 0.7269 - mae: 0.6418 - val_loss: 16.8747 - val_mae: 3.4883
Epoch 4/300
27/27 [=====] - 3s 182ms/step - loss: 0.9181 - mae: 0.7585 - val_loss: 16.8638 - val_mae: 3.5523
Epoch 5/300
27/27 [=====] - 3s 182ms/step - loss: 0.5518 - mae: 0.5871 - val_loss: 28.6144 - val_mae: 3.8893
Epoch 6/300
27/27 [=====] - 3s 184ms/step - loss: 0.5881 - mae: 0.6312 - val_loss: 21.8739 - val_mae: 3.8988
Epoch 7/300
27/27 [=====] - 3s 188ms/step - loss: 0.3787 - mae: 0.4599 - val_loss: 21.3454 - val_mae: 3.8548

Epoch 08/07: ReduceLROnPlateau reducing learning rate to 0.009900000427477862.
Epoch 8/300
27/27 [=====] - 3s 182ms/step - loss: 0.3259 - mae: 0.4393 - val_loss: 21.8998 - val_mae: 3.8738
Epoch 9/300
27/27 [=====] - 3s 181ms/step - loss: 0.2717 - mae: 0.4822 - val_loss: 17.8934 - val_mae: 3.5866
Epoch 10/300
27/27 [=====] - 3s 184ms/step - loss: 0.1979 - mae: 0.3573 - val_loss: 20.3291 - val_mae: 3.8837
Epoch 11/300
27/27 [=====] - 3s 182ms/step - loss: 0.2879 - mae: 0.3433 - val_loss: 12.5852 - val_mae: 3.1148
Epoch 12/300
27/27 [=====] - 3s 183ms/step - loss: 0.2118 - mae: 0.3555 - val_loss: 22.4239 - val_mae: 3.9768
Epoch 13/300
27/27 [=====] - 3s 183ms/step - loss: 0.1894 - mae: 0.3351 - val_loss: 12.4621 - val_mae: 3.8634
Epoch 14/300
27/27 [=====] - 3s 182ms/step - loss: 0.1566 - mae: 0.3879 - val_loss: 14.4887 - val_mae: 3.2571
Epoch 15/300
27/27 [=====] - 3s 183ms/step - loss: 0.1482 - mae: 0.3163 - val_loss: 14.8858 - val_mae: 3.2471

```

Figure 4.33: The training model

Now we start training models. We make our batch size 64 and epoch size 300.

The screenshot shows a Kaggle notebook titled 'face_recognition_team_fix_it'. The notebook contains several code cells. The first cell shows training progress for epochs 295, 296, 297, 298, 299, and 300. The second cell contains code to load the best model using `tf.keras.callbacks.get_model`. The third cell contains code to convert the loaded model to TensorFlow Lite format using `tf.lite.TFLiteConverter.from_keras_model`. The fourth cell contains code to save the converted model as a file.

```

Epoch 295/300
27/27 [=====] - 3s 101ms/step - loss: 0.8254 - mae: 0.1221 - val_loss: 0.0091 - val_mae: 0.0562
Epoch 296/300
27/27 [=====] - 3s 99ms/step - loss: 0.8253 - mae: 0.1245 - val_loss: 0.0079 - val_mae: 0.0465

Epoch 00296: ReduceLRonPlateau reducing learning rate to 4.638398286268781e-06.
Epoch 297/300
27/27 [=====] - 3s 182ms/step - loss: 0.8252 - mae: 0.1228 - val_loss: 0.0056 - val_mae: 0.0456
Epoch 298/300
27/27 [=====] - 3s 106ms/step - loss: 0.8263 - mae: 0.1221 - val_loss: 0.0075 - val_mae: 0.0549
Epoch 299/300
27/27 [=====] - 3s 108ms/step - loss: 0.8263 - mae: 0.1241 - val_loss: 0.0069 - val_mae: 0.0536
Epoch 300/300
27/27 [=====] - 3s 99ms/step - loss: 0.8278 - mae: 0.1267 - val_loss: 0.0088 - val_mae: 0.0555

In [121]: # val_mae will decrease
In [122]: # After finish we have to load best model
         model_load_weights(checkpoint_path)
In [123]: ~tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x77547404fe08~
In [124]: # Now convert best model to tensorflow lite format
         converter = tf.lite.TFLiteConverter.from_keras_model(model)
         tflite_model = converter.convert()
         #Now save model
         with open("model.tflite","wb") as f:
             f.write(tflite_model)
In [124]: # To watch prediction of validation set
         prediction_val = model.predict(x_test_batch_size = 64)

```

Figure 4.34: Loading train model

After 300 epochs we load the best model. Then we convert this model into TensorFlow lite format so that we can use this model in our android app.

The screenshot shows a Kaggle notebook titled 'face_recognition_team_fix_it'. The notebook contains code to predict on a validation set and print the results. The output shows a list of predicted values and a list of target values.

```

In [124]: # To watch prediction of validation set
         prediction_val = model.predict(x_test_batch_size = 64)
         prediction_val[:20] # Show first 20 value
In [124]:
array([[ 4.994712 ],
       [ 0.004251 ],
       [ 9.986352 ],
       [ 4.087719 ],
       [ 8.956309 ],
       [ 3.853592 ],
       [ 4.965643 ],
       [ 6.058873 ],
       [ 9.966191 ],
       [ 8.01268495 ],
       [ -0.0180398 ],
       [ 9.823902 ],
       [ 9.044647 ],
       [ 6.9927845 ],
       [ 3.047251 ],
       [ 7.964304 ],
       [ 11.087358 ],
       [ 2.996515 ],
       [ 4.9761233 ],
       [ 8.010445 ]], dtype=float32)
In [124]: # Original Label
         y_test[:20]
In [124]:
array([ 5, 10, 10, 4, 9, 3, 7, 6, 10, 0, 0, 9, 9, 7, 3, 8, 11,
        3, 5, 8])

```

Figure 4.35: Predict value and trained value

Here are the predicted value and trained value of the result that we train.

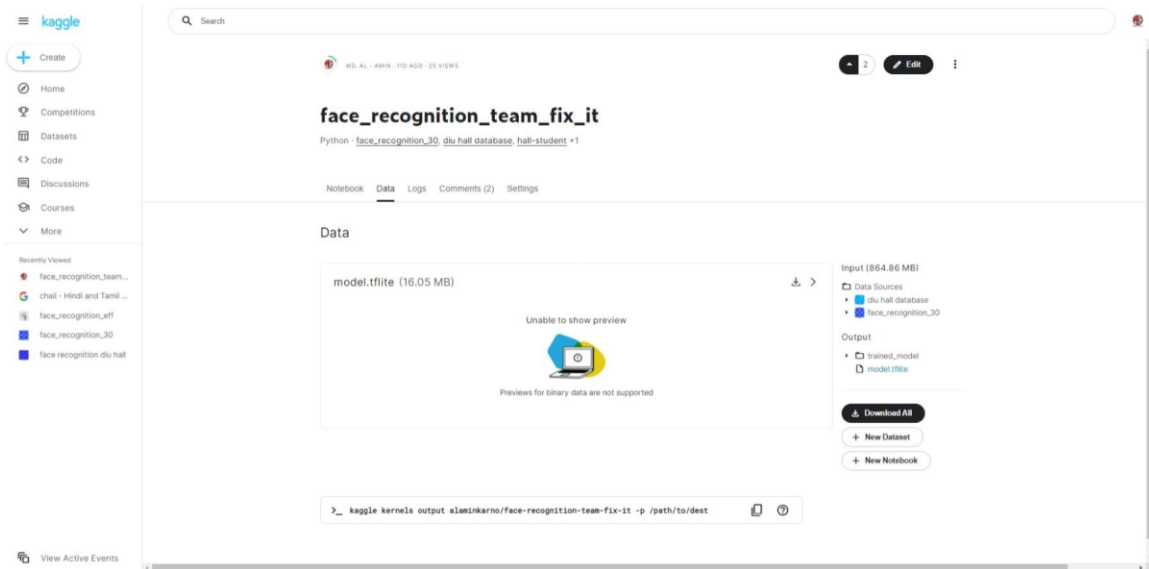


Figure 4.36: Output model after train model

After recognizing it we find the model and save it into our Kaggle account. We use this model in our android app to recognize faces.

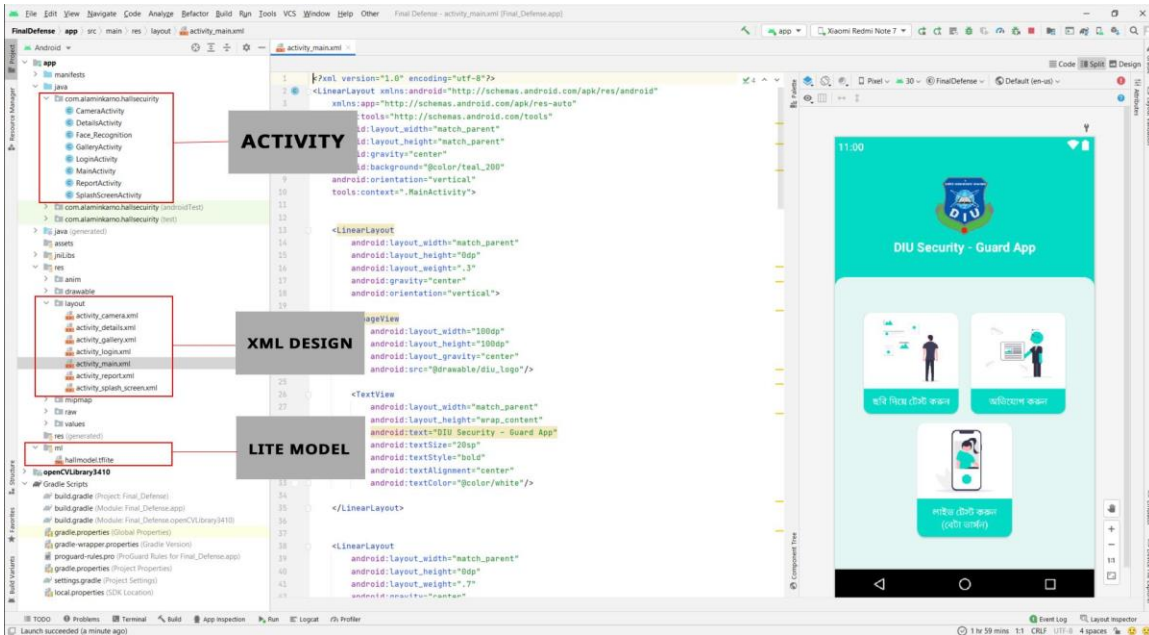


Figure 4.37: Backend Code structure Guard App

In figure 4.36 we show you the backend code structure of our Guard App. Here is our Activity, XML Design Code, and Tensorflow lite model.

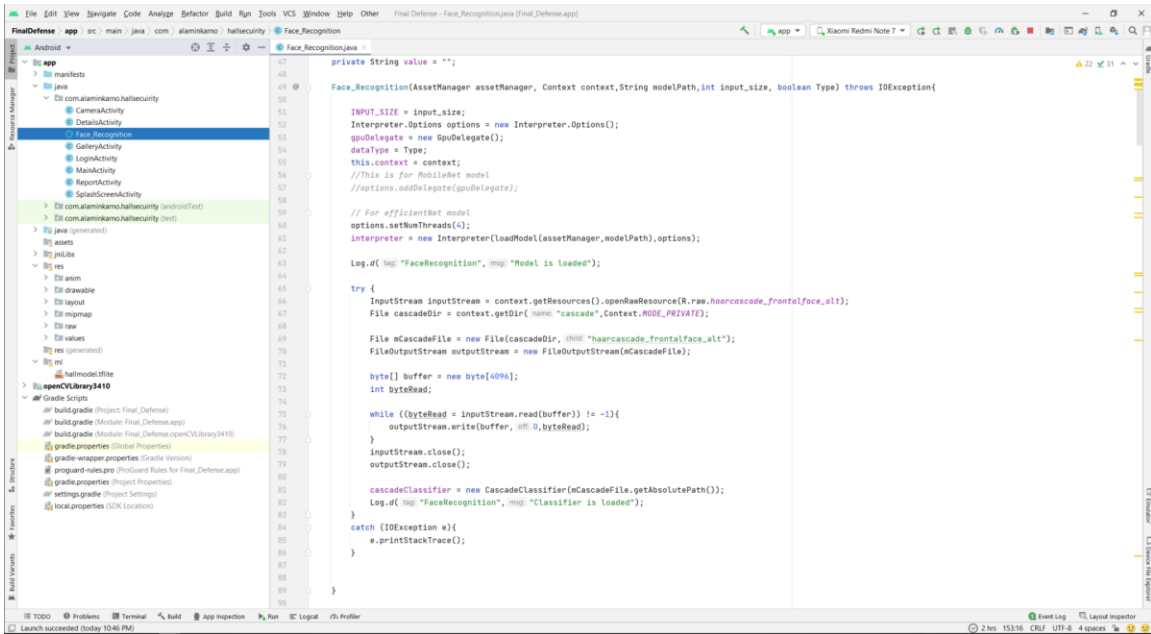


Figure 4.38: Backend Code FaceRecognition Class

This is a snapshot of the face-recognition class where we do the face detection and recognition part. Here we used haarcascade_frontalface_alt to detect faces. Use Interpreter to interpret our model with live or gallery images.

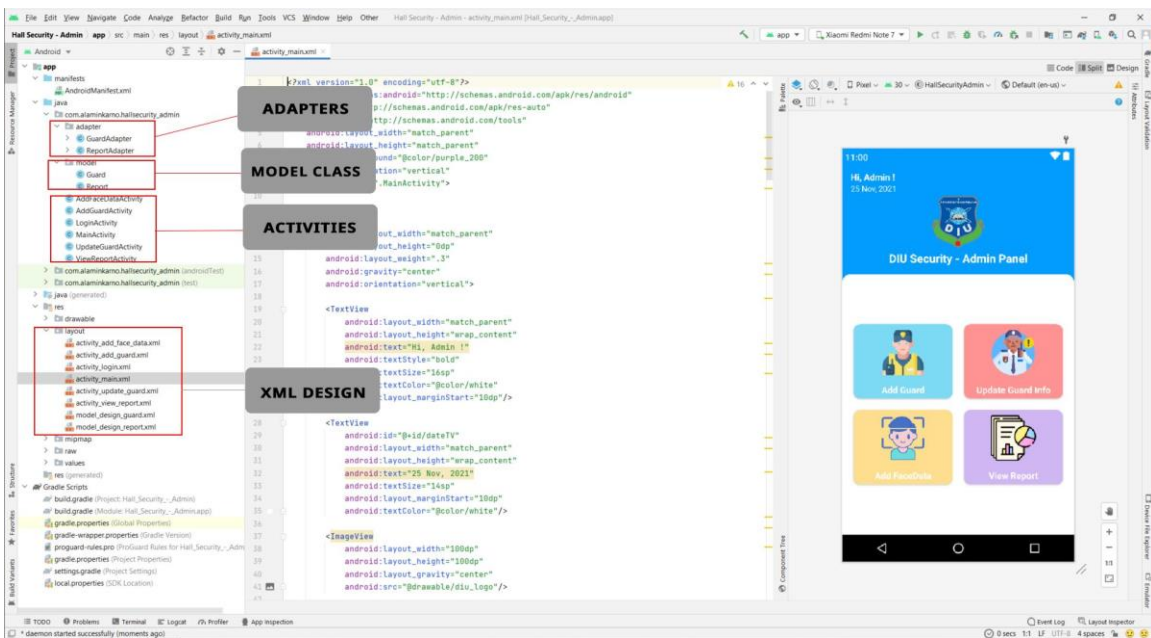


Figure 4.39: Backend code structure (Admin App)

This is the admin app structure in figure 4.38 where we have used the MVVM coding pattern. We used model class, adapters, and activities. Also, we have XML design where we have 2 different types of design one is for activity another is for creating model XML design.

```

package com.alaminkamo.hallsecurity_admin;

import androidx.appcompat.app.AppCompatActivity;

private RecyclerView recyclerView;
private GuardAdapter adapter;
private List<Guard> guardList;
private LiveData<AnimationView> animationView;
private Toolbar toolbar;

DatabaseReference databaseReference;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_update_guard);

    initialize();
    getGuardInfo();
}

private void getGuardInfo() {
    DatabaseReference guardRef = databaseReference.child("guard");
    guardRef.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            if(snapshot.exists()){
                guardList.clear();

                for(DataSnapshot dataSnapshot : snapshot.getChildren()){
                    Guard guard = dataSnapshot.getValue(Guard.class);
                    guardList.add(guard);
                }
                animationView.setVisibility(View.INVISIBLE);
                recyclerView.setVisibility(View.VISIBLE);
            }
        }
    });
}

```

Figure 4.40: Backend code Update Guard Activity (Admin App)

In figure 4.39 is just a glimpse of code about update guard activity. Here we just take a sample backend code screenshot.

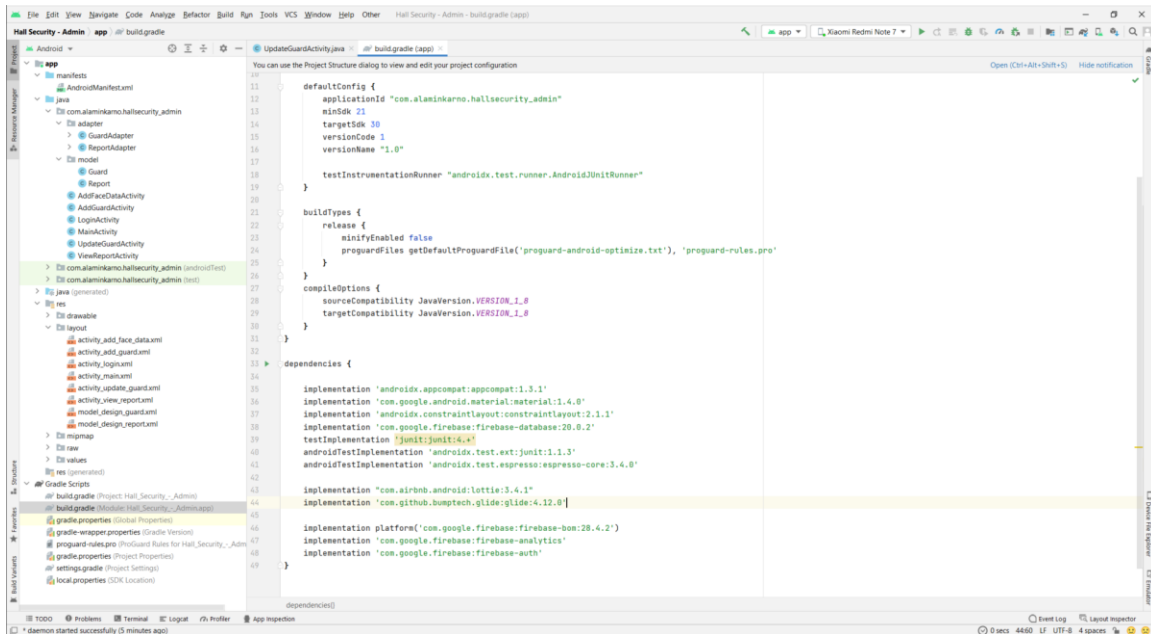


Figure 4.41: Dependency of Admin App (Admin App)

In the Admin app, we have used lots of dependency injections that help our app to make it easy. For example, we have used Lottie[8] dependency injection for a nice animation screen. We have used firebase dependency injection for using the firebase database.

4.3 Interaction Design and User Experience (UX)

Yes, we do our design in such a way that users can get the best user experience ever. We first look at our app color combination and our button design is also elegant too. We try to use animation and simple design concepts that all types of users like our design.

4.4 Implementation Requirements

For implementing these font-end designs and backend code we use PyCharm and Android Studio. Mainly those are the complete version of the software where we can make desktop software and an android application. To develop our project we used Python 3.9 and Android SDK 30 [7]. We used this latest version of SDK to get the best user experience for our users.

Chapter 5: Implementation and Testing

5.1 Implementation of Database

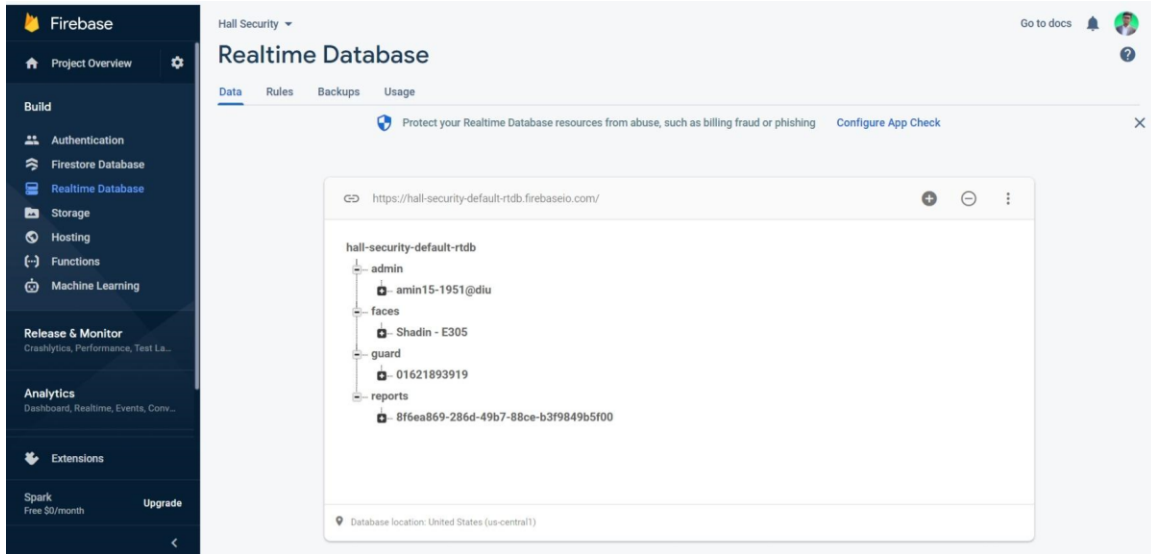


Figure 5.1: Firebase database implementation

For implementing a real-time database for our application we use the firebase database which is faster and easy to read and write from our android application. Also, we have used our trained model database for recognizing the faces.

5.12 Testing Implementation and Reports

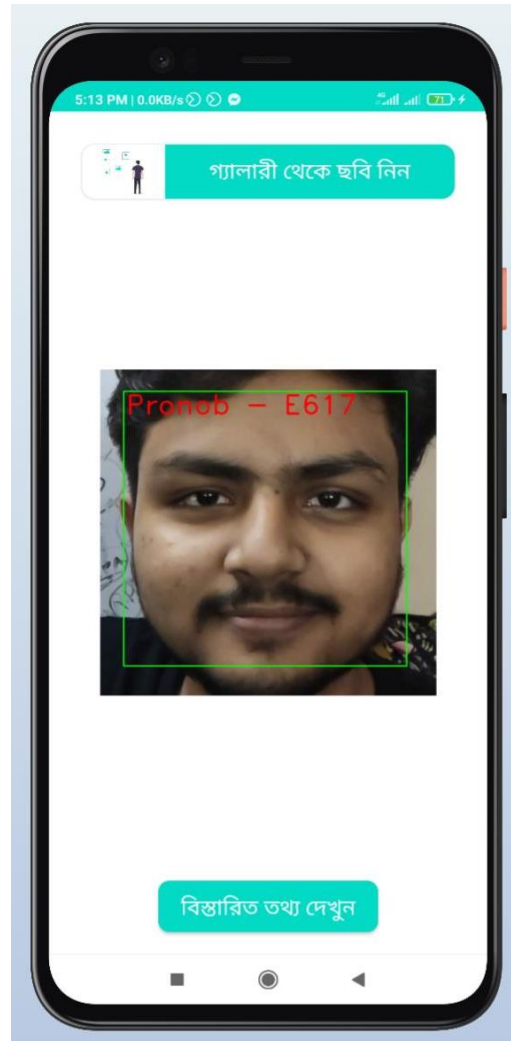


Figure 5.2: Testing face recognition

For testing our application we have used still images from the gallery and it can easily recognize faces that we trained in the Kaggle model. Yes, we have some problems with live image recognition because if we move our camera a little bit then the measure of the value change that's why it gives us a random value.

Chapter 6: Conclusion

6.1 Impact on society

The process of associating photos or data of a face with a specific individual is known as facial recognition. In today's society, as well as in university halls, crimes are on the rise. A better security system for university halls, private institutions, and other organizations will be built as a result of the creation of this software. Our system detects intruders accessing a restricted or private area of a person's home. The purpose of our security system is to protect private places from theft, burglary, property destruction, and invasions of privacy. Unknown people will not be able to cause problems in the future. It can be useful and inexpensive because it is just a system which we can apply any CCTV. This system may also be used to obtain precise information on a student by simply taking a picture of them.

6.2 Conclusion

We have seen some applications that can detect faces and some are capable of recognizing faces. But our system is more accurate and we can store the recognize face data info into text or excel sheet which can use for future. Our target is using this technology we can solve our problem and yes we are capable to do something that can use for many organizations.

6.3 Scope of further development

We tried to integrate a variety of current android technology into our system. We were unable to cover all of them due to time and resource constraints. Many features will be added in the near future to improve its efficiency. So we make our system into desktop version only and our android app is now in beta version we are working to resolve those problem. Some of them are pointed below –

1. We will add fingerprint recognition to our application.
2. As we are using TensorFlow lite, in the future, we can integrate the full version for better accuracy.
3. We will add voice recognition to our system.

Reference

- [1] Facial recognition: <http://findbiometrics.com/solutions/facial-recognition/>.
- [2] FacePro: <https://surveillance.i-pro.com/products/intelligent-analytics/facepro/>
- [3] Facial recognition market:
<https://www.marketsandmarkets.com/Market-Reports/facial-recognition-market-995.html>
- [4] Luxand face recognition application:
https://play.google.com/store/apps/details?id=com.luxand.facerecognition&hl=en_IN&showAllReviews=true
- [5] Face recognition, available:
<https://play.google.com/store/apps/details?id=org.opencv.javacv.recognition>
- [6] Railer - face recognition attendance:
https://play.google.com/store/apps/details?id=com.railer.face&hl=en_IN
- [7] Android SDK 30, available at:
[SDK Build Tools release notes | Android Developers](#)
- [8] Lottiefiles, available at,
[Free Lottie Animation Files, Tools & Plugins - LottieFiles](#)