ECE 558

Anti-theft Face Recognition App

Final Project Report

Ramprakash Baskar , Abhiraj Abhaykumar Eksambek, Nikolay Nikolov

[Demo Recording Link]

Introduction

IoT devices are growing in popularity and utilized in many industries. Their small form factor and small energy consumption make them ideal for various applications.

IoT devices take part in the security, data information, and control of devices responsible for different environmental conditions.

IoT devices are extensible and can help make more innovative homes that are energy-efficient and have reduced environmental impact.

The project is an implementation of an anti-theft security system that employs machine learning and face recognition. The application system helps the user identify if a person is present at the door and send a picture of that person through an Android app.

At last, the user can control the state of the door remotely through the Android App. For example, in the case of intruder detection, the user can lock the door. What is more, the application's internal implementation allows for further easy integration of other functionalities such as controlling an alarm and calling the police.

Hardware

Raspberry Pi 4b

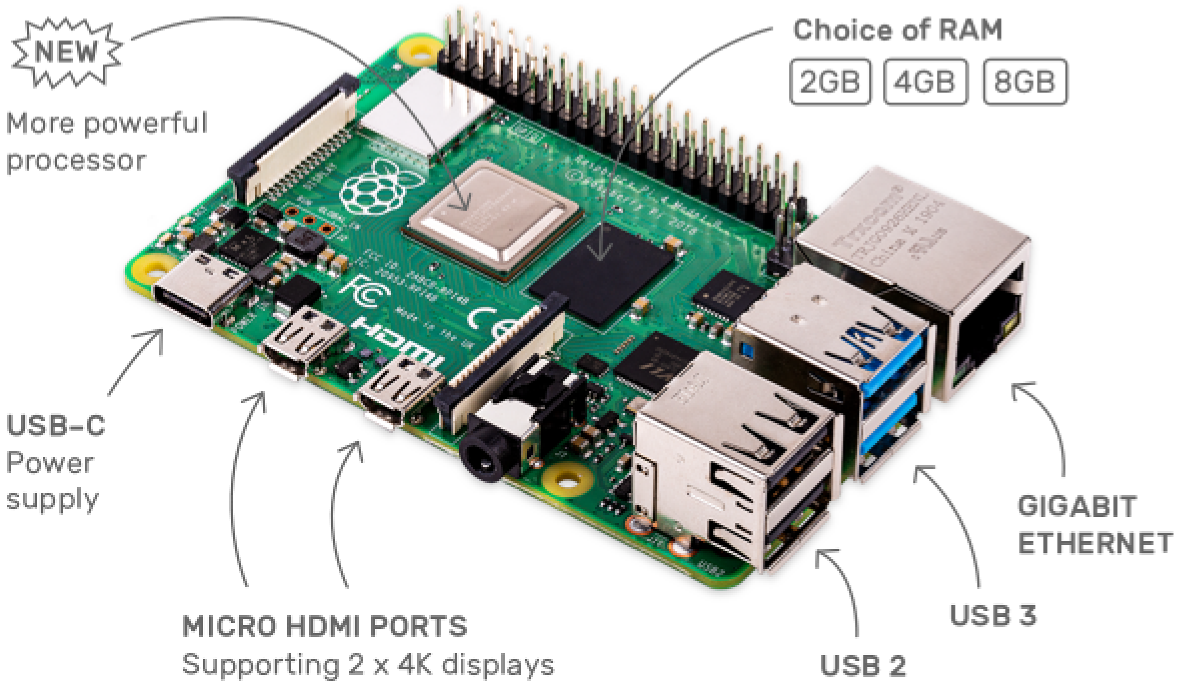


Figure 1: Using locally a raspberry pi as a backend server to process the requests for the door and face recognition.

Raspberry pi is a powerful Arm-based mini computer. We used it primarily as a local server. The local server's main functionality was to spawn a client script and the mosquitto MQTT broker, which handled the callbacks.

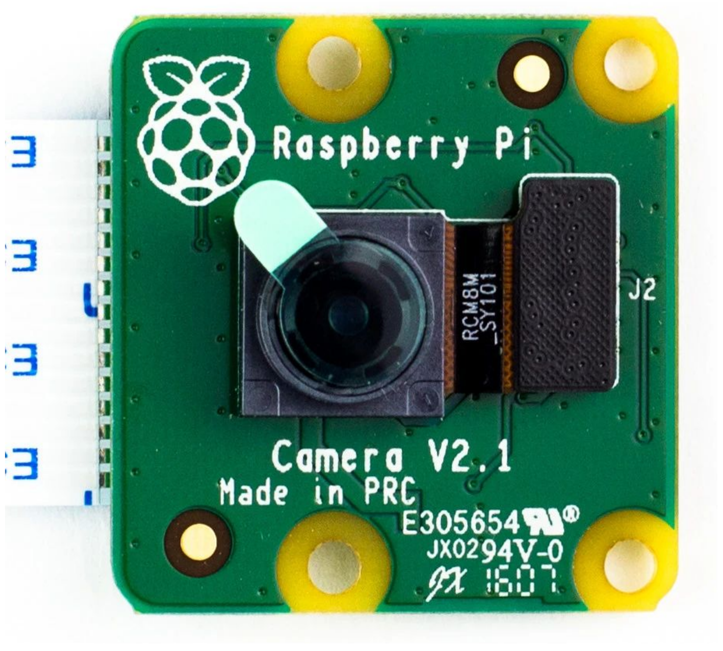
Moreover, the project's face recognition software runs on the Raspberry Pi instead of on a cloud service. That proved to be advantageous and cost-effective, and far more secure.

Besides being a competent computer, the Pi also supports GPIO pins enabling us to add button input, which simulated the doorbell.

Also, we used a GPIO pin with an LED attached that simulated an open/closed door.

Raspberry pi has WiFi support and, therefore, can communicate with cloud platforms.

Raspberry Cam V2



Our team used the RPi Cam V2 in order to gather the live video feed. The video feed then was processed for the presence of a person. We chose this particular model because of the good support and integration with the Raspberry pi.

Pimoroni Explorer Hat Pro

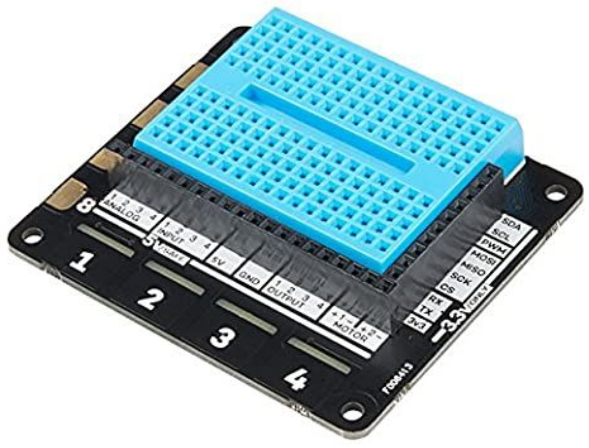


Figure2: The HAT we used to connect the button and the led

Explorer HAT includes 8 touch pads which act just like buttons.

This module acts as a shield on Raspberry Pi GPIO. Each touch pad has a number of functions for both reading its state and binding events to certain condition

It provides dedicated input and output pinouts to connect buttons, gpios and buzzers, etc.

Android Device

Our team used Nokia 6.1 with Android 10 to test the application. Android 10 (codenamed Android Q during development) is the tenth major release and the 17th version of the Android mobile operating system. It was first released as a developer preview on March 13, 2019, and was released publicly on September 3, 2019.

Machine Learning

We used Machine Learning models to determine if there are any human faces present in the images during our project. Haar Cascade's machine learning model was the main model we used in this project.

This is an object detection method using haar-like features. These features are simple, edge, line-like features. The training contains a huge amount of positive and negative datasets.

The positive dataset has images of the required object and the negative dataset containing everything else. The model is trained to extract features from a positive dataset. The extracted features from the cascade file. This cascade file is used for the detection of faces in the image.

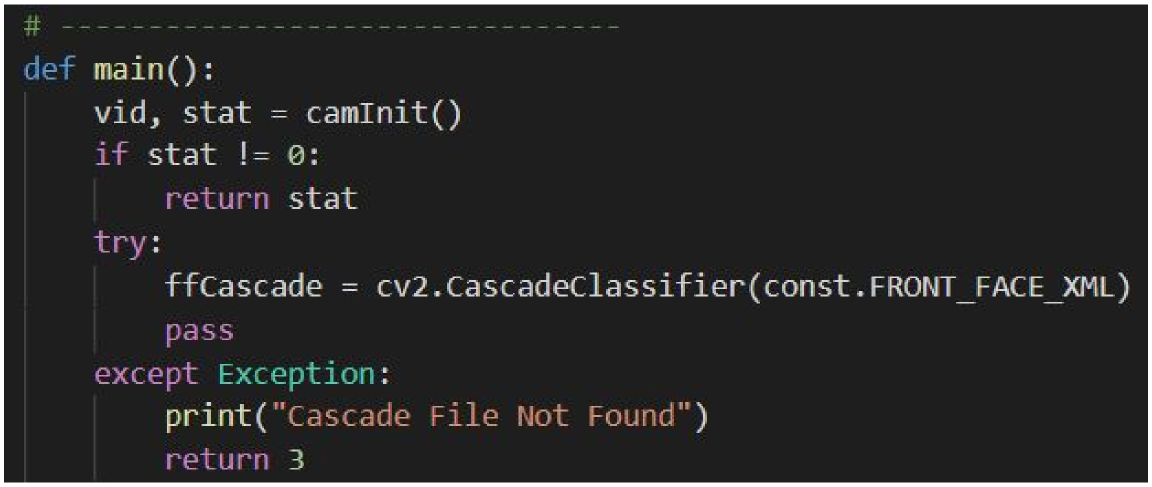
Prediction Model

OpenCV provides support for these haar-cascade models.

After collecting the image, the model analyzes a series of frames and extrapolates if there is a face present. Once it has identified a face, it passes it to the python package's function applying a scaling factor and a minimum number of frames where the algorithm must detect a face to consider the detection correct.

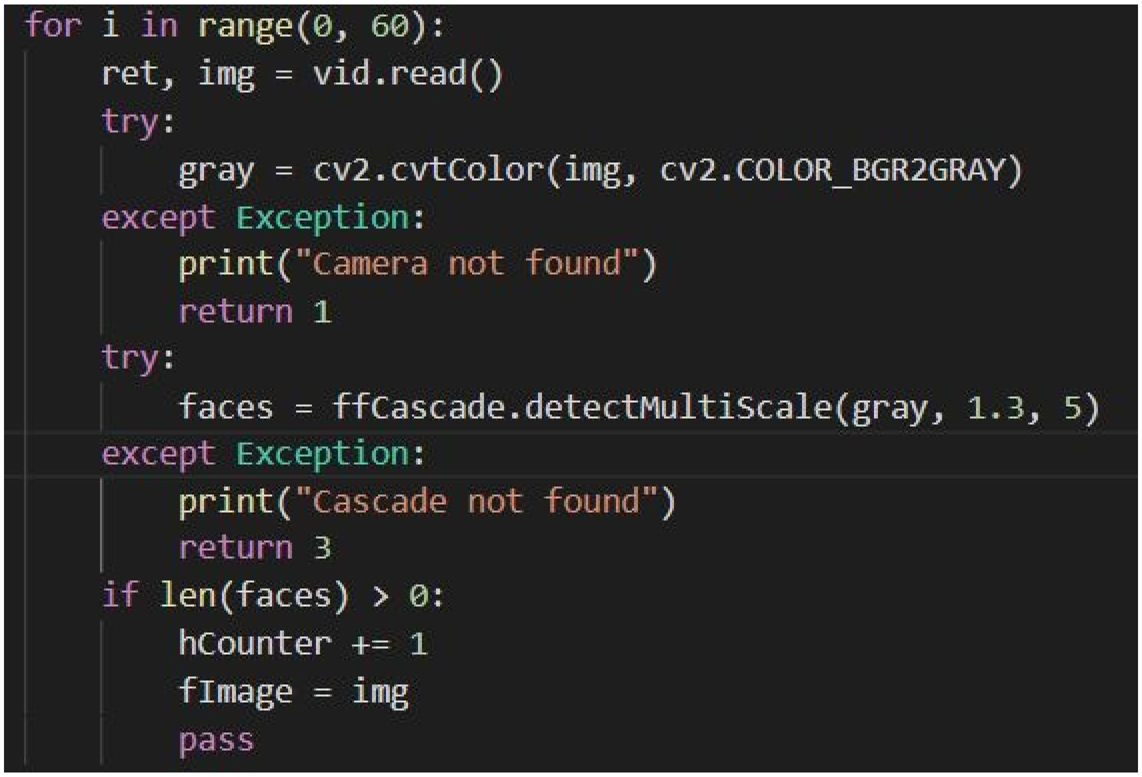
We were able to modify the accuracy of the results applying an 80% success margin.

Recognition Script- Snippets



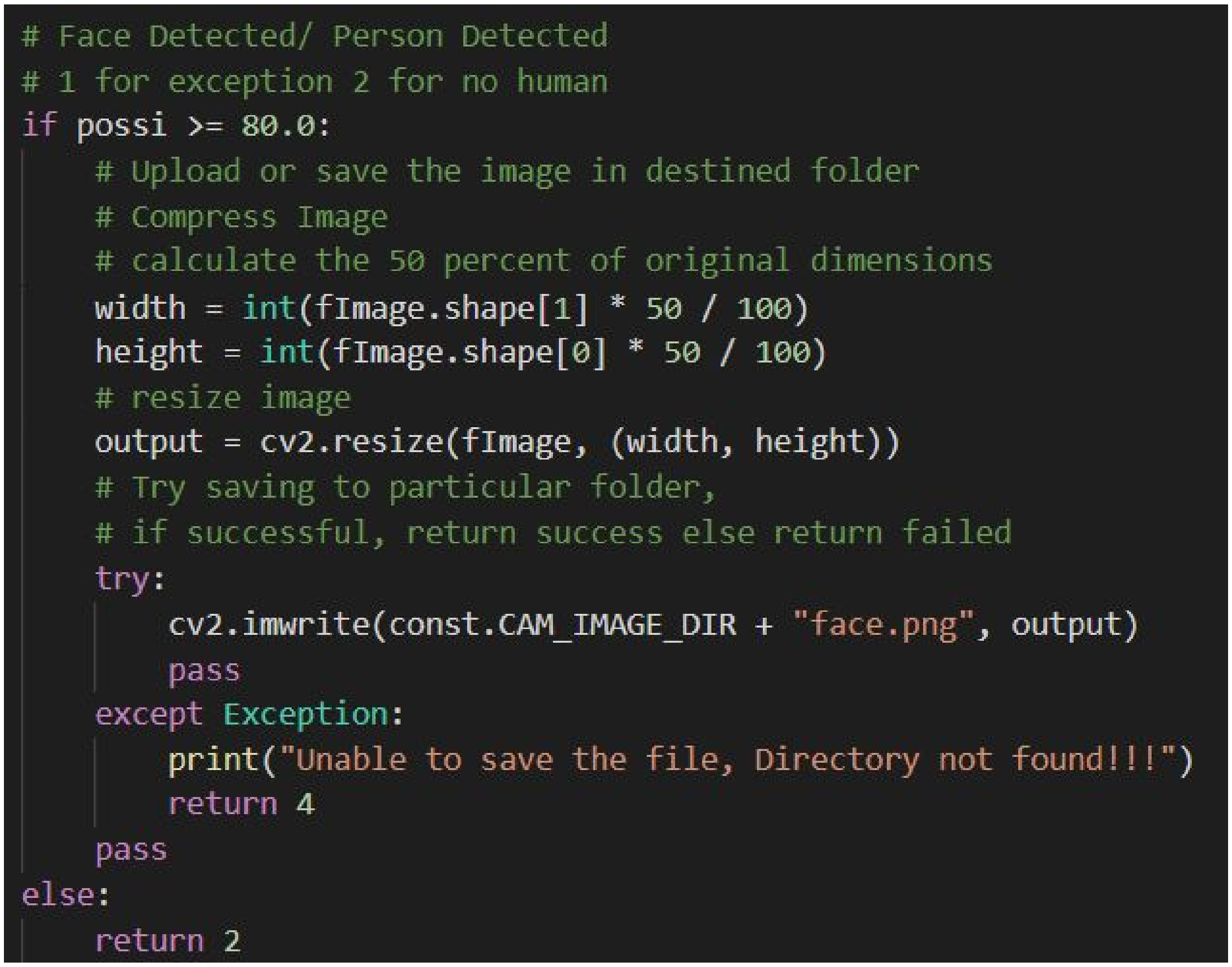
```
# --------------------------------
def main():
    vid, stat = camInit()
    if stat != 0:
        return stat
    try:
        ffCascade = cv2.CascadeClassifier(const.FRONT_FACE_XML)
        pass
    except Exception:
        print("Cascade File Not Found")
        return 3
```

The script begins with initializing the video camera and cascade file. If any of this fails to initialize, the script exits notifying the user.



```python
for i in range(0, 60):
    ret, img = vid.read()
    try:
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    except Exception:
        print("Camera not found")
        return 1
    try:
        faces = ffCascade.detectMultiScale(gray, 1.3, 5)
    except Exception:
        print("Cascade not found")
        return 3
    if len(faces) > 0:
        hCounter += 1
        fImage = img
        pass
```

The project requires recognition of faces from the live video feed. Video is nothing but a collection of continuous images or frames. The script takes around 60 frames in a time span of 2 seconds. These frames need pre-processing and therefore are converted to gray scale to reduce noise and unwanted features. These frames are passed to the machine learning algorithm to find faces from the image.



```python
# Face Detected/ Person Detected
# 1 for exception 2 for no human
if possi >= 80.0:
    # Upload or save the image in destined folder
    # Compress Image
    # calculate the 50 percent of original dimensions
    width = int(fImage.shape[1] * 50 / 100)
    height = int(fImage.shape[0] * 50 / 100)
    # resize image
    output = cv2.resize(fImage, (width, height))
    # Try saving to particular folder,
    # if successful, return success else return failed
    try:
        cv2.imwrite(const.CAM_IMAGE_DIR + "face.png", output)
        pass
    except Exception:
        print("Unable to save the file, Directory not found!!!")
        return 4
    pass
else:
    return 2
```

If 80% of the captured frames contains faces, the script determines that a person is at the door and saves the latest image of that person to be uploaded to the cloud server.

The uploaded images notifies the android user for intruder alert and has the option to download this image.

Android App:

The Android app at the receiving end is developed using Android Studio. The App includes services to connect to the Firebase Realtime Database, Firebase Storage, and Firebase Cloud Functions. The app design takes its cues from the Android material design framework.

This is a single activity app with two toggle buttons to open or close the door and ring or stop the alarm. The user is provided with the notification. The app presents the notification in the view in the notification tray phone. The prime functionality to view the photograph is done using an Image view and a button to trigger it.

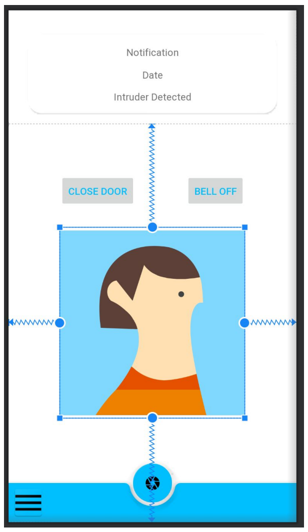


Figure3: App layout

User Interface:

The components of the UI are as follows:

- The UI consists of a coordinator layout in accordance with the material layout framework.
- The notification panel is provided in a Card View.

· The Notification, date and the Alert Message are provided as a text view.

· The door status and the bell status are changed by using a toggle button.

· The Download picture button is provided as a Floating action button in the bottom.

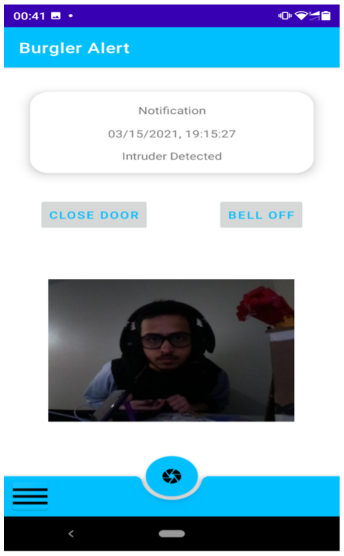


Figure4: The app has received a notification and it has downloaded the image

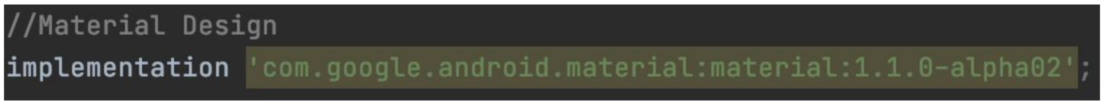
UI for Notification layout:

· The notification panel consists of a title, body displayed using text view.

· It also consists of an icon displayed using image view.

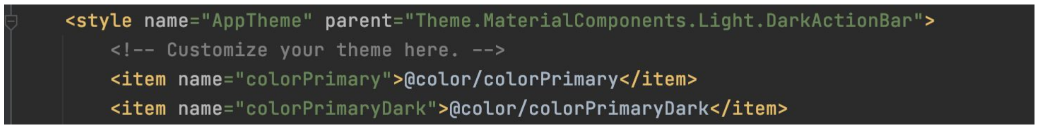
Style Theme:

The app was built using the material design framework on android with the colors, icons and the shape defined accordingly. We added the require dependency of the material package In order to implement the material theme



```
//Material Design
implementation 'com.google.android.material:material:1.1.0-alpha02';
```

Figure5: Adding material design in build.gradle



```
    <style name="AppTheme" parent="Theme.MaterialComponents.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
```

Figure6: Color/themes.xml adding the material theme

Icons:

Launcher Icon:

Opening the app from the app tray instead of the default ic launcher from android . During the view inflation a new logo is created using the clipart available in the android studio.

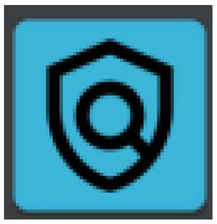


Figure7: Launcher Icon at the app tray

Notification Icon:

Similarly, every time the notification arrives to the android app the notification icon gets displayed which also was created using the clip art in android studio.

Figure8: Notification icon , when an intruder alert has been received

Camera Icon:

A new logo for the Floating app bar is defined to display this new icon.

Figure9: The button that trigger downloading the image

Activity Logic

The activity logic involves communication with the Firebase Realtime database server, Firebase Storage and Firebase Cloud Communication Tool.

In order to carry out these activities we need to add the necessary dependencies to the build grade file for the app. In order to download the photo and view it we need a glide library.

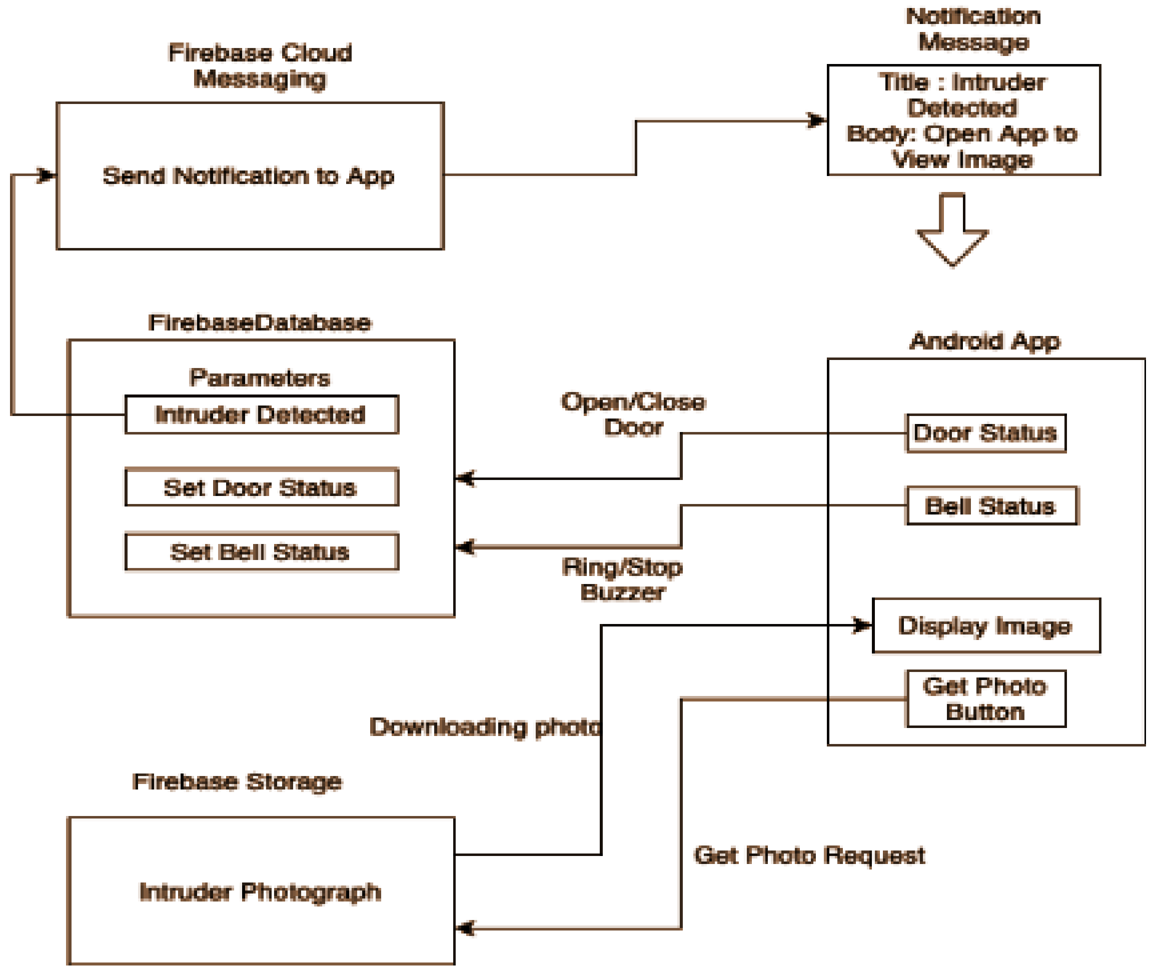
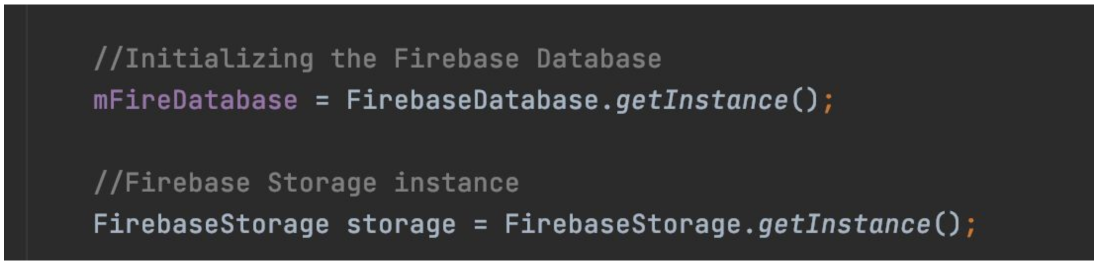


Figure10: Communication flow at the front end

Communication with Firebase

In order to communicate with Firebase an instance of the firebase realtime database, firebase storage is created by defining them as objects.

Once the reference is created then the child of the key value is used to either read the value from the server or write to it. This instance of the firebase servers is set in the listeners of the button or toggle button that triggers these firebase instances to perform the necessary action in the Realtime database.



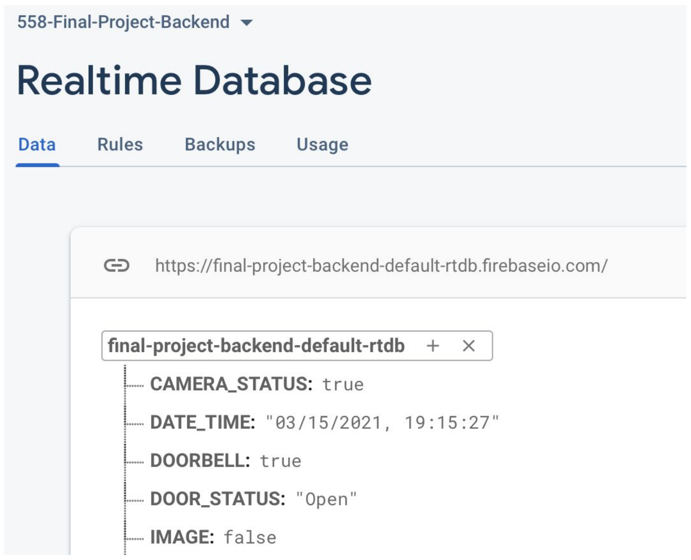
```
//Initializing the Firebase Database
mFireDatabase = FirebaseDatabase.getInstance();

//Firebase Storage instance
FirebaseStorage storage = FirebaseStorage.getInstance();
```

Figure11: initialization of the firebase Api for Android within the App

Updating the Date using Database Listener

The date value for the intruder alert needs to be acquired from the database every time the app is opened.

The database listener is set to change the value whenever a new date is set in the Realtime database.



https://final-project-backend-default-rtdb.firebaseio.com/

```
final-project-backend-default-rtdb
  CAMERA_STATUS: true
  DATE_TIME: "03/15/2021, 19:15:27"
  DOORBELL: true
  DOOR_STATUS: "Open"
  IMAGE: false
```

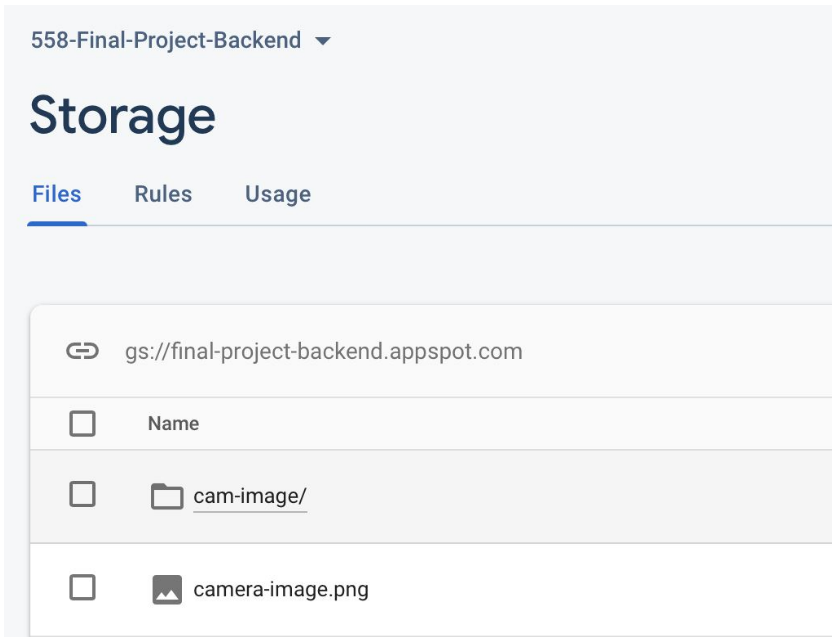
Figure12: Firebase Realtime database

Interfacing with the Firebase (Google Cloud) Storage

Firebase (Google) storage instances are created similar to database instances.

The mage file name is provided as the child of the instance to receive it. In order to get the photo a button listener is set which triggers the progress bar which gives the rotation downloading animation until the image is downloaded.

Once the image is downloaded Glide tool is used to view the image into the Image view defined in the XML file.


558-Final-Project-Backend

Storage

Files Rules Usage

gs://final-project-backend.appspot.com

Name
cam-image/
camera-image.png

Figure13: Firebase Google storage where the intruder image is pushed from the RPi server to the cloud and later retrieved and opened within the app

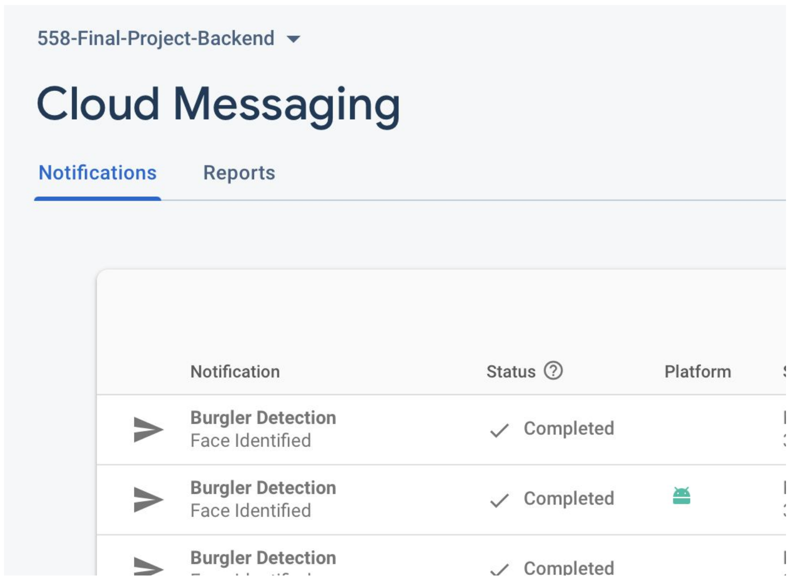
Interfacing with Firebase Cloud Messaging Service (FCM)

The cloud messaging service offered by Firebase is used to trigger push notification to the user.

In order to use this feature, we had to use the Firebase cloud messaging packages that are added to Gradle and then the Firebase Messaging class inherits it as mentioned below.

 This cloud messaging is triggered whenever there is a change in the image detected parameter in the firebase database. The script connecting the firebase database with the cloud messaging is handled by the cloud function explained below.

Once the FCM is triggered then the android app is notified with a Title, Body and an Image which are defined in the app as explained below.


558-Final-Project-Backend

Cloud Messaging

Notification	Status	Platform
Burgler Detection Face Identified	✓ Completed	
Burgler Detection Face Identified	✓ Completed	
Burgler Detection	✓ Completed	

Figure14: Firebase cloud messaging service

Notification Logic

The notification logic which alerts the user is defined by inheriting the Firebase Messaging Service class.

Under this class the methods like on Message Received, show notification, get custom design are overridden. The activities overridden and the purpose are detailed below:

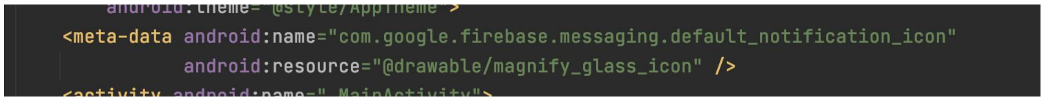

```
    <meta-data android:name="com.google.firebase.messaging.default_notification_icon"
               android:resource="@drawable/magnify_glass_icon" />
```

Figure15: Adding notification logic in build.gradle

· OnMessageReceived: This activity is called whenever a new message is received and then uses the user defined show notification method to display the notification in the notification tray to the user.

· onNewToken: This activity is used whenever a new token is generated. Tokens used to establish authentication between the app and the Firebase

Communication Server.

This token is intrun set in the Firebase Realtime database for the cloud functions to read and publish the notification to the app.

- getCustomDesign:

Every time a notification arrives it displays according to the design which we have defined here by inflating the corresponding layout.

- showNotification:

The notification needs to be shown in the app track based on the time of arrival. That is the recent notification should be shown at the top of the notification tray. And on click of the notification the app even if it currently paused it should be called to resume. This is done by using Intent to call the activity to the top of the stack.

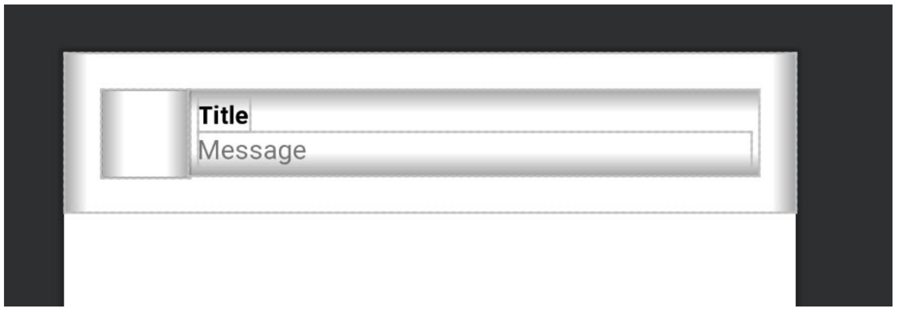


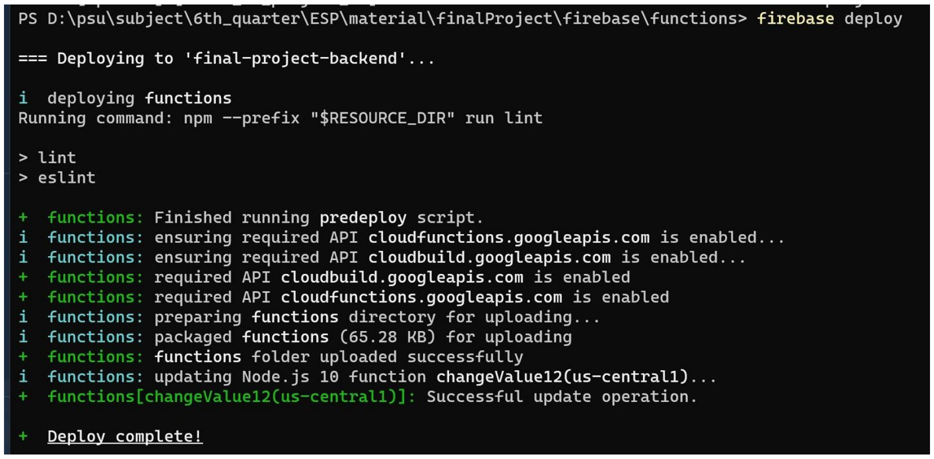
Figure16: Notification tray logic

Cloud Functions:

The cloud functions are the backend daemons that connect the Firebase Realtime database with the Firebase Cloud Messaging service.

This is facilitated by the Firebase CLI (Command Line Interface) powered by npm (Node js Package Manager). The firebase cloud functions are programmed using JavaScript or Typescript.

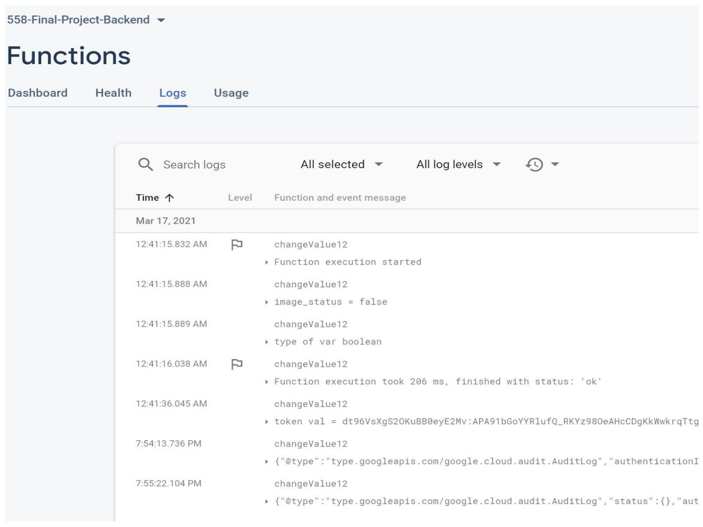
The cloud functions create an instance of the database and the cloud messaging service and then check the parameter Image Detected from the database. If that parameter is true, then it triggers a cloud message to the android app.



```
PS D:\psu\subject\6th_quarter\ESP\material\finalProject\firebase\functions> firebase deploy

=== Deploying to 'final-project-backend'...

i  deploying functions
Running command: npm --prefix "$RESOURCE_DIR" run lint

> lint
> eslint

+  functions: Finished running predeploy script.
i  functions: ensuring required API cloudfunctions.googleapis.com is enabled...
i  functions: ensuring required API cloudbuild.googleapis.com is enabled...
+  functions: required API cloudbuild.googleapis.com is enabled
+  functions: required API cloudfunctions.googleapis.com is enabled
i  functions: preparing functions directory for uploading...
i  functions: packaged functions (65.28 KB) for uploading
+  functions: functions folder uploaded successfully
i  functions: updating Node.js 10 function changeValue12(us-central1)...
+  functions[changeValue12(us-central1)]: Successful update operation.

+  Deploy complete!
```

Figure17: Using NodeJS to created the firebase function

1. Once deployed into the firebase service, the javascript command gets updated
2. Into the console then the firebase listener can view the function along with the
3. If there is some error, then it can be displayed there.
4. Similarly, the console logs defined in the javascript code get updated here.



```
Time        Level  Function and event message
Mar 17, 2021
12:41:15.832 AM    changeValue12
                   Function execution started
12:41:15.888 AM    changeValue12
                   image_status = false
12:41:15.889 AM    changeValue12
                   type of var boolean
12:41:16.038 AM    changeValue12
                   Function execution took 206 ms, finished with status: 'ok'
12:41:36.045 AM    changeValue12
                   token val = dt96VsXgS2OKuBB0eyE2Mv:APA91bGoYYRlufQ_RKYz980eAHcCDgKkWwkrqTtg
7:54:13.736 PM     changeValue12
                   {"@type":"type.googleapis.com/google.cloud.audit.AuditLog","authenticationI
7:55:22.104 PM     changeValue12
                   {"@type":"type.googleapis.com/google.cloud.audit.AuditLog","status":{},"aut
```

Figure18: Firebase cloud functions

Working of the system

This project consists of 2 systems working independently but sharing information with each other over the internet. Failure of one system does not hamper functions of another system.

The Raspberry Pi is placed inside the house. It acts as a brain of the door control system. It is connected to a button, which acts as a doorbell and an LED which indicates the door.

When the doorbell is pressed, the raspberry pi starts the camera and records a 2 seconds video. The video is processed to find if faces are detected in this video. If a person is detected in more than 80% of the video, the system decides that person is present at the door. Then this system uploads an image of the person to the firebase cloud server and updates the realtime database with time of person detection, notification that image of that person is updated.

Another feature of this system is that all the detection and controls are independent of each other. Failure in camera or prediction does not hamper the working of door control or bell detection.

The other system is on an android device. This system gets information about the person at the door, images and control opening and closing of the door. Whenever the home system detects a person at the door and updates the database, the android system checks the database and generates a notification to be delivered to the user even if the application is closed.

The user can open the app and press on the get image button to get the image. The image is displayed on the app and the user can decide whether or not to let the person in. The user can toggle the door status to open or close. The user also has an option to sound a buzzer to alert nearby residents about a suspicious person.

Backend Server:

The backend client used MQTT client for controlling the door. And a main client that was getting the request by continually monitoring the changes in the firebase realtime database.

The backend consisted of one main Python script serving a client. During the execution of the client it creates another mosquitto MQTT client that is responsible for handling the responses for the door.

Instead of creating one large script or maybe two, we separated the concerns accordingly. Hence, the main client did not have any other functionality beyond calling the correct Python script worker.

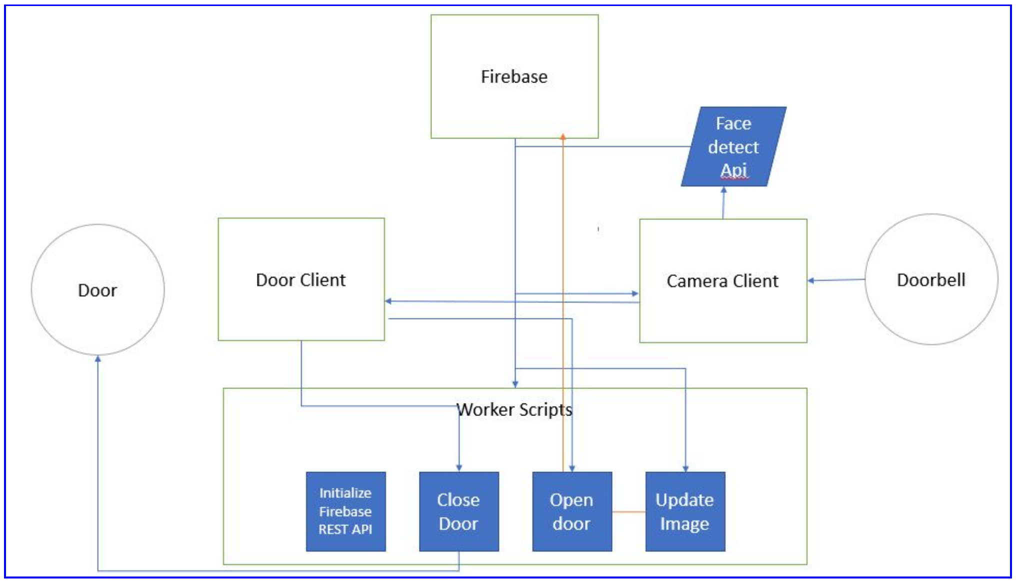


Figure19: Execution flow for the RPi local server client

Future Work:

Face recognition

Possible future work with this same method is to predict the number of humans present at the door. The detection algorithm gives locations of faces found in the image and therefore multiple locations detected indicate the number of faces present in the image. This can be also useful information for the homeowner.

This algorithm cannot identify different faces. A different machine learning algorithm can be used to store faces of homeowners and trusted humans. If the script detects any known human face, it can automatically open the door and let them in. And if it detects an unknown face then it can notify the home owner and ask whether or not to let him in.

Securely exchanging the information

The current system passes the information without any encryption. This makes the system vulnerable to attacks.

An encryption algorithm can be implemented to have decryption keys present on all the end systems, that is, raspberry pi and android devices. The information passed from raspberry pi is encrypted and updated on the server. Attackers, not having the decryption key will not be able to retrieve information from the database. And the android device with the key, can decipher the information for the user.

By implementing an encryption, decryption algorithm, even if the attacker changes the real time database to try to open the door, he cannot do that as the decrypted message at raspberry pi end will decipher a garbage message the attacker tried to hamper with.

This can also help the system to make it more autonomous that if the attacker is trying multiple times to open the door by changing the real time database, the raspberry pi will allow only a few attempts to receive garbage value, and if attempts passes set count, it can autonomously ring the buzzer or contact 911 to request an immediate help.