Project Report

Elevator control panel application

Haitham Chabayta 1065457

Supervised By: Dr. Mohammed Ghazal

جامعة أبوظبي
ABU DHABI UNIVERSITY

Submitted: June 19, 2020

Contents

List of Figures

List of Tables

Abstract

In this project I have designed a contact-less system to press on elevator buttons through just your mobile phone. After the recent spread of COVID-19, people have started to fear touching anything in public. To go up and down building people are forced to use an elevator and to use an elevator you need to physical press on a button. The system proposed elements this risk of having to touch an elevator button and helps you to stay safe during this pandemic. To use the system all you need is to download the android application and scan the QR code beside the elevator you want to use and the elevator control panel will pop on your screen. Once you press on any button on your phone it will physical push that elevator button with the help of a raspberry pi. The results of our system were as expected and it worked success-full in providing a contact-less pushing mechanism with the help of a raspberry pi. Improvements to the system can be done through providing a better pushing mechanism and more convenient for real world application.

1 System Introduction and overview

The system overview is provided in figure 1 below.

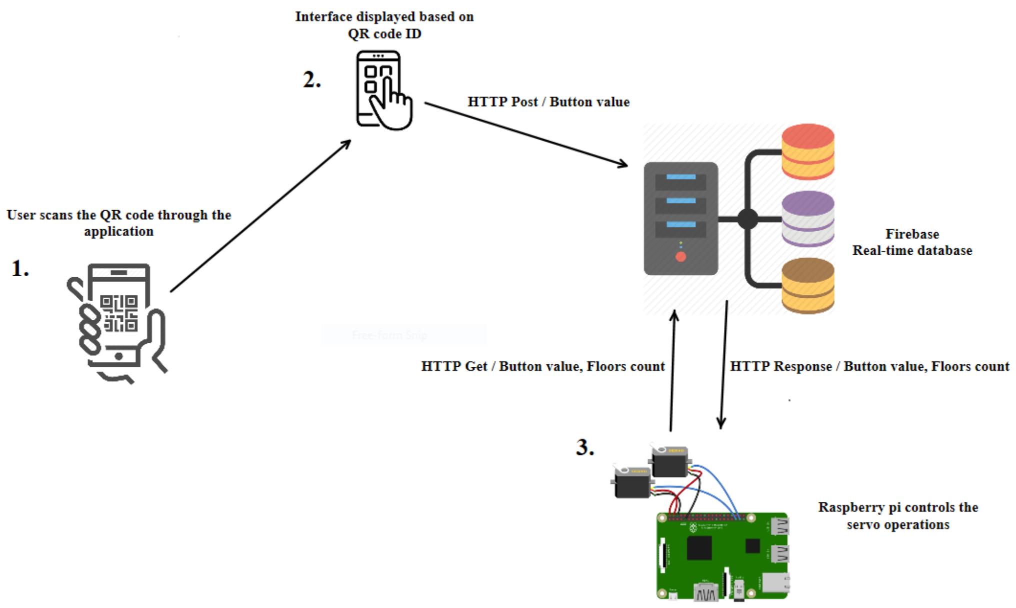

Interface displayed based on QR code ID
2.
HTTP Post / Button value
User scans the QR code through the application
1.
Firebase
Real-time database
HTTP Get / Button value, Floors count
HTTP Response / Button value, Floors count
3.
Raspberry pi controls the servo operations

Figure 1: System overview design.

The system functionality is highlighted by the points below:

1. The user first will have to scan QR code using the mobile application to access the control panel. If a wrong QR code is scanned it will show a message that the QR code is invalid.

2. Once a correct QR code scanned it will show the elevators control panel based on the ID of the QR code scanned. Each ID is mapped to an elevator in a hash-table and the application will fetch the control panel of the elevator mapped to the ID scanned.

3. The user then can press on a button on the screen. Once a button is pressed the value of that button will be alerted by sending an in the fire base HTTP post request.

4. A Raspberry pi is present in our system and multiple servo motors will be connected to the raspberry pi GPIO pins.

5. The raspberry pi will continuously monitor the value of the concerned buttons in the fire base by sending HTTP get requests.
6. Once a button value changes to 1 it will move a servo motor by 90 degrees.
7. The servo motor will have a wooden piece attached to it shaped like a pendulum that will press the elevator button physically.

2 FSM design

The system FSM design is separated into to main statecharts; one for the application and one for the Raspberry pi. The two statecharts will work concurrently together without any transition between the two system.

2.1 FSM design for the application

The FSM design for the application is shown below in figure 2.

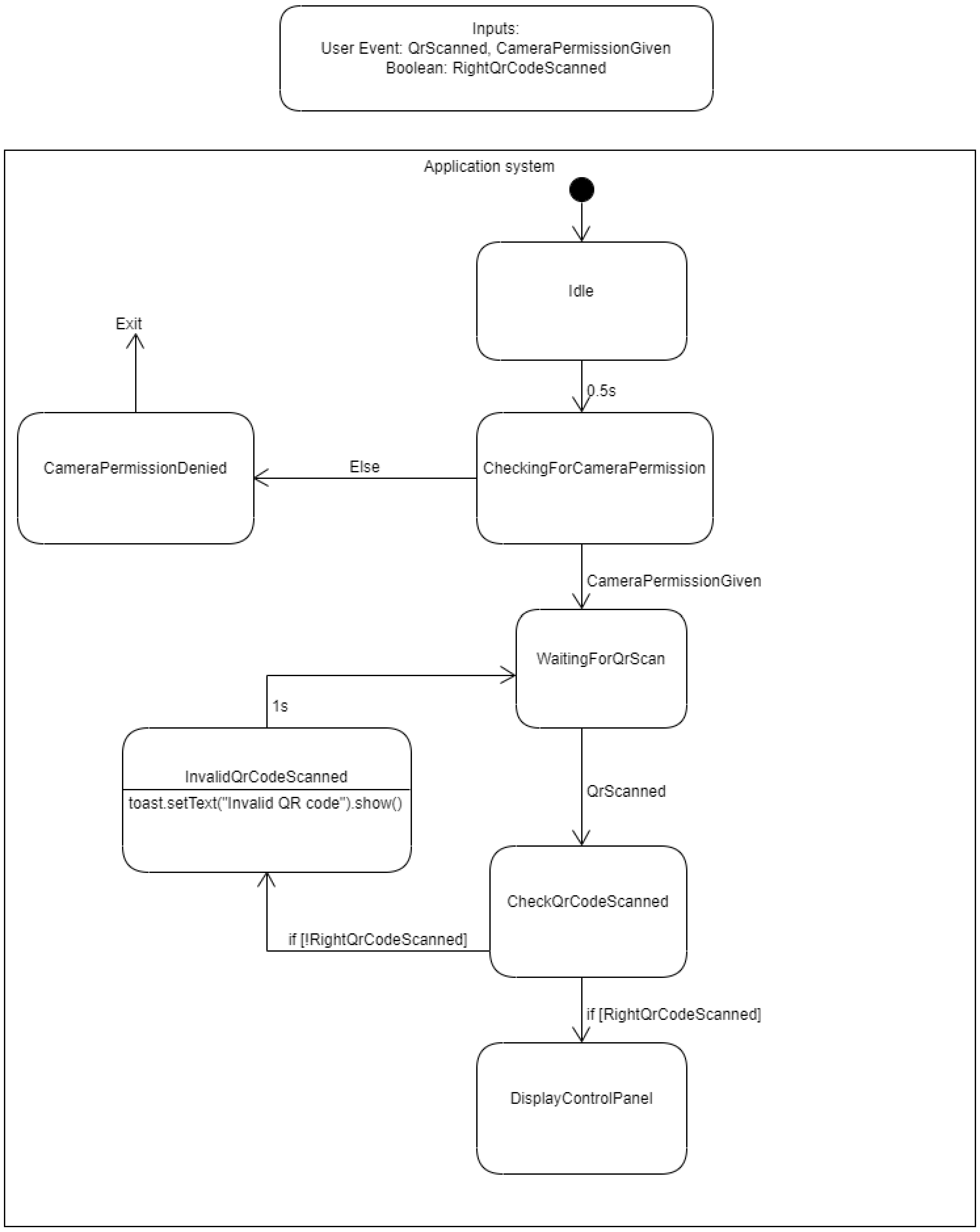


Figure 2: Application FSM design.

For the application, there is three inputs and no visible outputs. The three inputs consists of two types which are user events and a Boolean. There are two events that are triggered through the user which is one a permission is given for the application to use the mobile camera or when the user scans a QR code.

The application system will start in Idle state and will transition into a state to ask for the user's permission for the application to utilize the mobile phone's camera. If the user denies permission the app will exit and once the user opens up the application again he will be asked for his permission once again. Once the permission is given the user will be asked to scan QR code through his camera. The system will check then the value of the QR code scanned and if it is not equal to one of the IDs in the hashtable that gives each elevator an ID as a key, it will display a toast saying that the QR code is invalid. If the Qr code scanned provides a valid ID it will transition then to a state in which the application will move to another android activity, that contains the elevator's control panel, based on the ID. If the user then presses on any button, the button on click event listener will be raised and it will send an HTTP post request to alert the value of that button from 0 to 1 for 2 seconds and that it will change it back to one. While all of this is happening the Raspberry pi will also be running.

2.2 Raspberry pi FSM design

The FSM regarding the raspberry functionality in our system is shown in figure 3

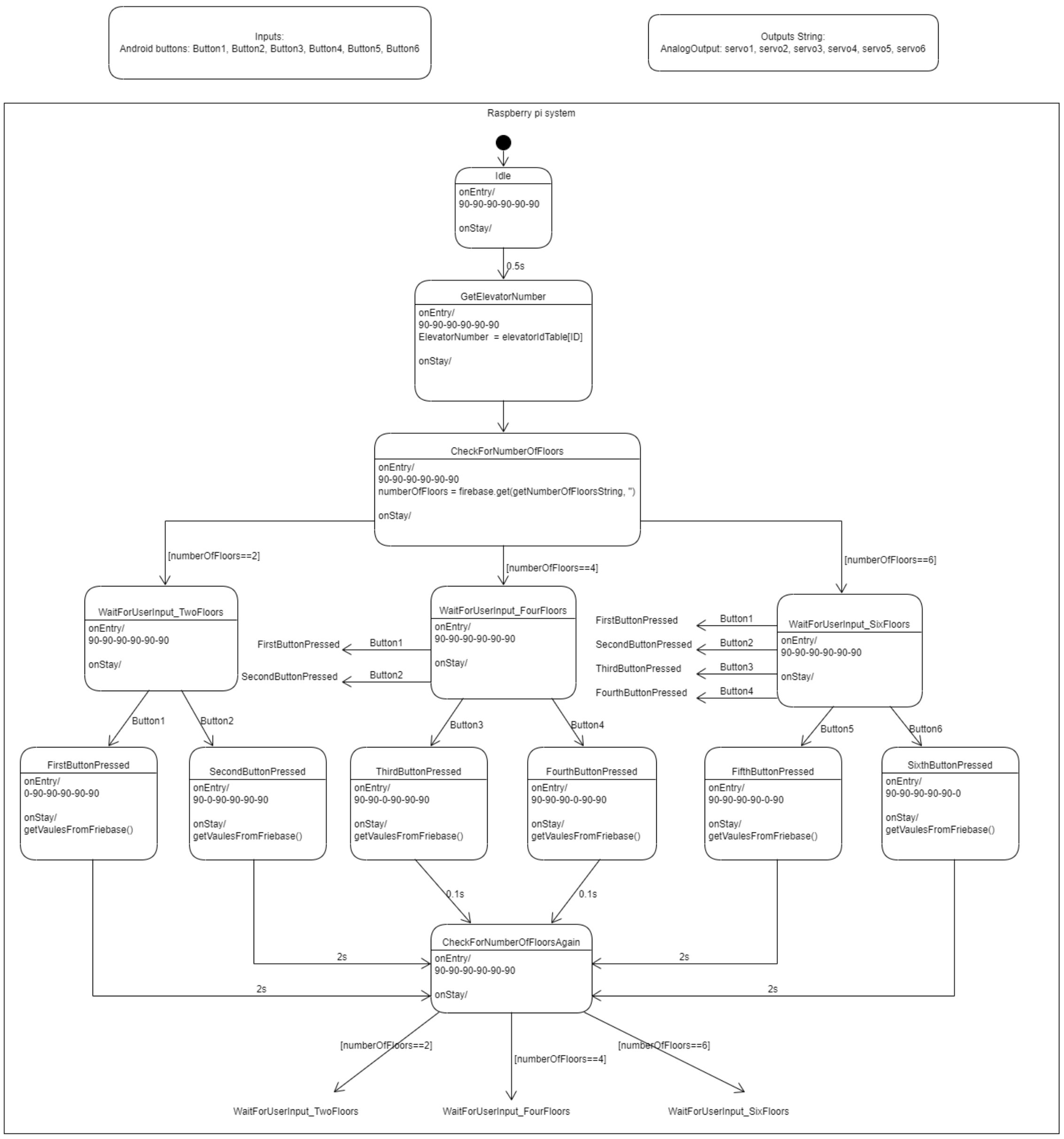


Figure 3: Raspberry pi FSM design.

For the raspberry pi there are four input that the raspberry pi is concerned with and 4 outputs that it controls; The outputs and the inputs potentially could be a lot more but for now we are concerned with just 6 as a prototype. The raspberry pi depends on the input of the android buttons in the application we described earlier and it controls the function of multiple servo motors.

The Raspberry pi will also start in an Idle state and each servo will be initialized to angle of 90 degrees. After half a second the Raspberry will check the number of the elevator it is responsible for through the ID given for the RPi. The elevators are numbered in the firebase and the each elevator will store how many floors it has and the state of each button present in its panel. After getting the number of the elevator it will transition into a state where it will send a get request to get the number of floors present in that elevator. Once the number of floors is acquired it will setup a number of servos that are equal to the number of floors and it will transition into a state to wait for the users input. The Raspberry pi will check on a number of buttons based on the number of floor and will continuously send get requests to the firebase to get the state of the button in the firebase. Once the users pushes a button on his phone the Raspberry will sense the change and will update the angle of that servo to 0 degrees to press the button and back then to 90 degrees after a short delay. Afterwards, the system will check for the number of floors found before to know which state to go back to, to wait for the user's input.

3 Code walkthorugh

A python script was developed to implement the Raspberry pi part of the system described earlier and the application was developed as a native android application using android studio.

3.1 Raspberry python script

First we need to import the enum library to define the states, the sleep function from the time library, the Raspberry pi GPIO library to able to use the GPIO pins of the RPi, and the firebase python library to connect to our application's firebase.

```python
from enum import Enum
from time import sleep
import RPi.GPIO as GPIO
from firebase import firebase
```

Afterwards we give the Raspberry pi an ID and provide the hashtable that links each QR code ID to the elevator number. By implementing the code this way the Rpi is flexible and can function on any of our systems elevators by just changing the ID when needed.

```python
ID = 2266
elevatorIdTable = {2266 : 'ElevatorNumberOne', 5544 :
    'ElevatorNumberTwo', 4433 : 'ElevatorNumberThree'}
```

Then we connect to our firebase by through the link of our real time database.The servos are also initialized to none and the states for the Raspberry pi in figure ?? are defined. [1]

```python
firebase =
    firebase.FirebaseApplication("https://elevatorcontrolpanelappli
catio.firebaseio.com/", None)

servo1=None
servo2=None
servo3=None
servo4=None
servo5=None
servo6=None

class State(Enum):
    Idle = 1
    onEntryGetElevatorNumber = 2
    onStayGetElevatorNumber = 3
    onEntryCheckNumberOfFloors = 4
    onStayCheckNumberOfFloors = 5
    onEntryWaitForUserInput_OneFloor = 6
    onStayWaitForUserInput_OneFloor = 7
    onEntryWaitForUserInput_TwoFloor = 8
    onStayWaitForUserInput_TwoFloors = 9
    onEntryWaitForUserInput_ThreeFloors = 10
    onStayWaitForUserInput_ThreeFloors = 11
    onEntryUpdateFirstButton = 12
    onStayUpdateFirstButton = 13
    onEntryUpdateSecondButton = 14
    onStayUpdateSecondButton = 15
    onEntryUpdateThirdButton = 16
    onStayUpdateThirdButton = 17
    onEntryUpdateFourthButton = 18
    onStayUpdateFourthButton = 19
    onEntryUpdateFifthButton = 20
    onStayUpdateFifthButton = 21
    onEntryUpdateSixthButton = 22
    onStayUpdateSixthButton = 23
    onEntryCheckNumberOfFloorsAgain = 24
    onStayCheckNumberOfFloorsAgain = 25

current = State.Idle

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
```

We have defined then a function to update the angle of the servo where you provide the angle you want to set for the servo and the number of the servo you want to update.

```
def update(angle, y):
    duty = float(angle) / 10.0 + 2.5
        if y == 1:
                global servo1
        servo1.ChangeDutyCycle(duty)
        elif y == 2:
                global servo2
                servo2.ChangeDutyCycle(duty)
        elif y == 3:
        global servo3
        servo3.ChangeDutyCycle(duty)
        elif y == 4:
        global servo4
        servo4.ChangeDutyCycle(duty)
        elif y == 5:
        global servo5
        servo5.ChangeDutyCycle(duty)
        else:
                global servo6
                servo6.ChangeDutyCycle(duty)
```

We have defined also another 3 functions to setup our GPIO pins and servo motors to 90 degrees for elevators with number of floors equal to two, four and six.

```
def setupForOneFloors():
    print("Setting up for One floor")
    global servo1
    global servo2
    GPIO.setup(17, GPIO.OUT)
    GPIO.setup(27, GPIO.OUT)
    servo1 = GPIO.PWM(17, 100)
    servo1.start(5)
    servo2 = GPIO.PWM(27, 100)
    servo2.start(5)
    update(90, 1)
    update(90, 2)

def setupForTwoFloors():
    global servo1
    global servo2
    global servo3
    global servo4
    print("Setting up for Two floors")
    GPIO.setup(17, GPIO.OUT)
    GPIO.setup(22, GPIO.OUT)
    GPIO.setup(27, GPIO.OUT)
    GPIO.setup(6, GPIO.OUT)
    servo1 = GPIO.PWM(17, 100)
    servo1.start(5)
    servo2 = GPIO.PWM(22, 100)
    servo2.start(5)
    servo3 = GPIO.PWM(27, 100)
    servo3.start(5)
```

```python
    servo4 = GPIO.PWM(6, 100)
    servo4.start(5)
    update(90, 1)
    update(90, 2)
    update(90, 3)
    update(90, 4)

def setupForThreeFloors():
    global servo1
    global servo2
    global servo3
    global servo4
    global servo5
    global servo6
    print("Setting up for Three floors")
    GPIO.setup(17, GPIO.OUT)
    GPIO.setup(27, GPIO.OUT)
    GPIO.setup(22, GPIO.OUT)
    GPIO.setup(23, GPIO.OUT)
    GPIO.setup(24, GPIO.OUT)
    GPIO.setup(25, GPIO.OUT)
    servo1 = GPIO.PWM(17, 100)
    servo1.start(5)
    servo2 = GPIO.PWM(27, 100)
    servo2.start(5)
    servo3 = GPIO.PWM(22, 100)
    servo3.start(5)
    servo4 = GPIO.PWM(23, 100)
    servo4.start(5)
    servo5 = GPIO.PWM(24, 100)
    servo5.start(5)
    servo6 = GPIO.PWM(25, 100)
    servo6.start(5)
    update(90, 1)
    update(90, 2)
    update(90, 3)
    update(90, 4)
    update(90, 5)
    update(90, 6)
```

Now we'll start defining what we will happen inside each state and the transition inside a infinite while loop. We'll start from the Idle state and move into a state to get the number of the elevator. Upon entry to that state we will acquire the elevator's number of the ID and define the string to send with the get request to get the button value inside our real time database.

```python
while True:
    if current == State.Idle:
        sleep(0.5)
        current = State.GetElevatorNumber

    elif current == State.onEntryGetElevatorNumber:
        ElevatorNumber = elevatorIdTable[ID]
        getButton1StateString = '/Elevator/' + ElevatorNumber +
            '/button1'
        getButton2StateString = '/Elevator/' + ElevatorNumber +
            '/button2'
        getButton3StateString = '/Elevator/' + ElevatorNumber +
            '/button3'
        getButton4StateString = '/Elevator/' + ElevatorNumber +
            '/button4'
        getButton5StateString = '/Elevator/' + ElevatorNumber +
            '/button5'
        getButton6StateString = '/Elevator/' + ElevatorNumber +
            '/button6'
        current = State.onStayGetElevatorNumber

    elif current == State.onStayGetElevatorNumber:
        current = State.onEntryCheckNumberOfFloors
```

After getting the number of the elevator the Raspberry pi is responsible for, the code will move into a state that will send a get request to the firebase to get the number of floors of that elevator and convert it into an integer. Based on the number of floors it will setup a number of servos that are equal to the floor count using the setup functions defined earlier for each floor count, and move into a state to wait for an input from the user that is in line with the floor count.

```python
    elif current == State.onEntryCheckNumberOfFloors:
        getNumberOfFloorsString = '/Elevator/' + ElevatorNumber +
            '/numberOfFloors'
        numberOfFloors = firebase.get(getNumberOfFloorsString, '')
        numberOfFloorsInteger = int(numberOfFloors)
        current = State.onStayCheckNumberOfFloors

    elif current == State.onStayCheckNumberOfFloors:
        if numberOfFloorsInteger == 1:
            setupForOneFloor()
            current = State.WaitForUserInput_OneFloor

        elif numberOfFloorsInteger == 2:
            setupForTwoFloors()
            current = State.WaitForUserInput_FourFloors

        elif numberOfFloorsInteger == 3:
            setupForThreeFloors()
            current = State.WaitForUserInput_ThreeFloors
```

For each of the states that waits for the user input, it will get the value of a number of buttons that are equal to the number of floors, so that no extra delay is caused for getting the value of buttons that don't exits. Once the value is changed due to the user pressing a button on the application the code will transition to update the angle of the servo motor that represents the button pressed.

```
        elif current == State.onEntryWaitForUserInput_OneFloor:
                button1State =
                    int(firebase.get(getButton1StateString, ''))
                button2State =
                    int(firebase.get(getButton2StateString, ''))
                current = State.onStayWaitForUserInput_TwoFloors

        elif current == State.onStayWaitForUserInput_OneFloor:
                if button1State == 1:
                current = State.UpdateFirstButton
                elif button2State == 1:
                current = State.UpdateSecondButton

        elif current == State.onEntryWaitForUserInput_TwoFloors:
                button1State =
                    int(firebase.get(getButton1StateString, ''))
                button2State =
                    int(firebase.get(getButton2StateString, ''))
                button3State =
                    int(firebase.get(getButton3StateString, ''))
                button4State =
                    int(firebase.get(getButton4StateString, ''))
                current = State.onStayWaitForUserInput_FourFloors

        elif current == State.onStayWaitForUserInput_TwoFloors:
                if button1State == 1:
                current = State.UpdateFirstButton
                elif button2State == 1:
                current = State.UpdateSecondButton
                elif button3State == 1:
                current = State.UpdateThirdButton
                elif button4State == 1:
                current = State.UpdateFourthButton

        elif current == State.onEntryWaitForUserInput_ThreeFloors:
                button1State =
                    int(firebase.get(getButton1StateString, ''))
                button2State =
                    int(firebase.get(getButton2StateString, ''))
                button3State =
                    int(firebase.get(getButton3StateString, ''))
                button4State =
                    int(firebase.get(getButton4StateString, ''))
                button5State =
                    int(firebase.get(getButton5StateString, ''))
                button6State =
                    int(firebase.get(getButton6StateString, ''))
                current = State.onStayWaitForUserInput_SixFloors

        elif current == State.onStayWaitForUserInput_ThreeFloors:
                if button1State == 1:
                current = State.UpdateFirstButton
                elif button2State == 1:
                        current = State.UpdateSecondButton
                elif button3State == 1:
```

```
                current = State.UpdateThirdButton
                elif button4State == 1:
                current = State.UpdateFourthButton
                elif button5State == 1:
                current = State.UpdateFifthButton
                elif button6State == 1:
                current = State.UpdateSixthButton
```

Inside each state that updates a button, it will change the angle of the servo to 0 degrees from the initial 90 degrees and then back to 90 degrees after 2 seconds. Then all the update buttons states transition to a common state that will check the floor count again.

```

        elif current == State.onEntryUpdateFirstButton:
                    print("updating button1")
                update(0,1)
                    sleep(2)
                    sleep(90,1)
                current = State.onStayUpdateFirstButton

        elif current == State.onStayUpdateFirstButton:
            current = State.onEntryCheckNumberOfFloorsAgain

        elif current == State.onEntryUpdateSecondButton:
                    print("updating button2")
                update(0,2)
                    sleep(2)
                    update(90,2)
                current = State.onStayUpdateSecondButton

        elif current == State.onStayUpdateSecondButton:
            current = State.onEntryCheckNumberOfFloorsAgain

        elif current == State.onEntryUpdateThirdButton:
                    print("updating button3")
                update(0,3)
                    sleep(2)
                    update(90,3)
                current = State.onStayUpdateThirdButton

        elif current == State.onStayUpdateThirdButton:
            current = State.onEntryCheckNumberOfFloorsAgain

        elif current == State.onEntryUpdateFourthButton:
                    print("updating button4")
                update(0,4)
                    sleep(2)
                    update(90,4)
                current = State.onStayUpdateFourthButton

        elif current == State.onStayUpdateFourthButton:
            current = State.onEntryCheckNumberOfFloorsAgain

        elif current == State.onEntryUpdateFifthButton:
                    print("updating button5")
                update(0,5)
                    sleep(2)
                    update(90,5)
                current = State.onStayUpdateFifthButton

        elif current == State.onStayUpdateFifthButton:
            current = State.onEntryCheckNumberOfFloorsAgain

        elif current == State.onEntryUpdateSixthButton:
                    print("updating button6")
                update(0,6)
                    sleep(2)
                    update(90,6)
                current = State.onStayUpdateSixthButton

        elif current == State.onStayUpdateSixButton:
            current = State.onEntryCheckNumberOfFloorsAgain
```

At the end, the CheckNumberOfFloorsAgain State will transition back to the state of waiting for a user input that is in line with the floor count.

```
    elif current == State.onEntryCheckNumberOfFloorsAgain:
        current = State.onStayCheckNumberOfFloorsAgain

elif current == State.onStayCheckNumberOfFloorsAgain:
        if numberOfFloorsInteger == 1:
        current = State.WaitForUserInput_OneFloor

    elif numberOfFloorsInteger == 2:
        current = State.WaitForUserInput_TwoFloors

    elif numberOfFloorsInteger == 3:
        current = State.WaitForUserInput_ThreeFloors

sleep(0.1)
```

3.2 Application code

The system application has been designed as a android native mobile application using android studio. The application will use a firebase and will scan Qr codes so we want to add the dependencies first for google firebase and google vision inside the gradle.

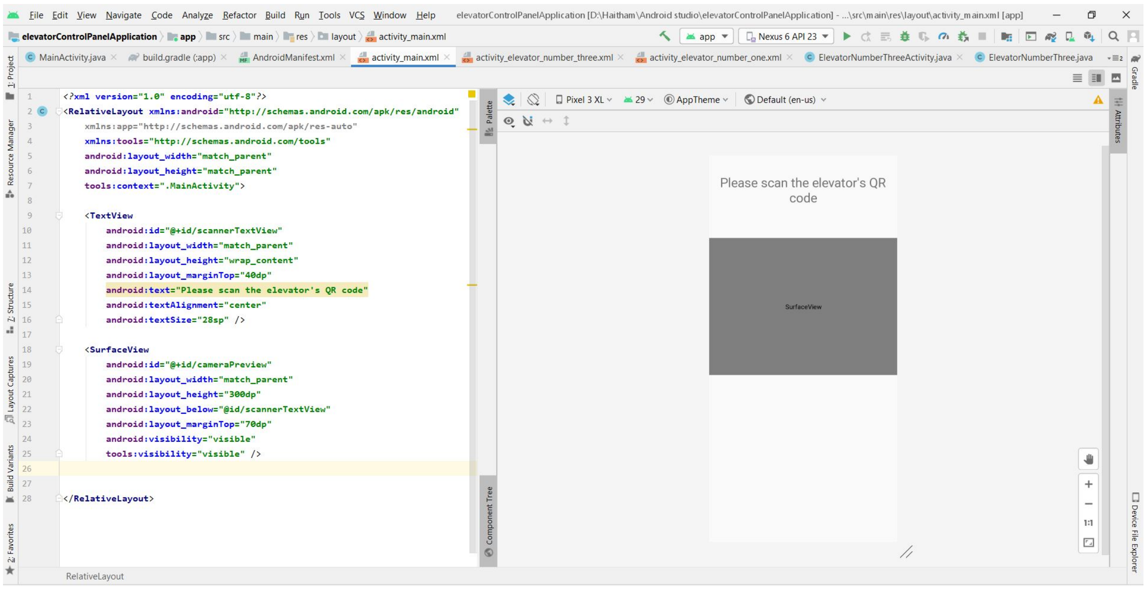
```
dependencies {
implementation 'com.google.firebase:firebase-database:19.3.0'
implementation 'com.google.android.gms:play-services-vision:15.0.0'
}
```

After that, we need to add in the AndroidManifest xml file that the application needs permission for the mobile's camera and vibrator, and we will define the google vision dependencies by a value equal to barcode.

```
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.VIBRATE" />
    <meta-data
            android:name="com.google.android.gms.vision.DEPENDENCIES"
            android:value="barcode" />
```

3.2.1 Main activity

Now we can start building the main activity. The main activity will be used to scan a QR code through the user's mobile camera and transition into another activity based on the Qr code scanned. So first we will design the xml layout for the main activity. It will only contain a surface view to use the camera to scan and a text view on top asking the user to scan a QR code. The layout is displayed in figure 4 below.



```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/scannerTextView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="40dp"
        android:text="Please scan the elevator's QR code"
        android:textAlignment="center"
        android:textSize="28sp" />

    <SurfaceView
        android:id="@+id/cameraPreview"
        android:layout_width="match_parent"
        android:layout_height="300dp"
        android:layout_below="@id/scannerTextView"
        android:layout_marginTop="70dp"
        android:visibility="visible"
        tools:visibility="visible" />


</RelativeLayout>
```

Figure 4: Main activity layout.

Afterwards, we can start building the java class for the main activity which will be the default activity that will pop when you run the application as defined in the manifest file. In the java class first, we are going to define the views we have, the barcode detector, and a hashtable that will link an ID to each elevator activity. We need to also define that upon creation of the activity to display the layout of the main activity described earlier on.

```java
public class MainActivity extends AppCompatActivity {

    SurfaceView surfaceView;
    CameraSource cameraSource;
    BarcodeDetector barcodeDetector;
    TextView textView;
    Hashtable<String, Intent> qrCodesTable = new Hashtable<String,
        Intent>();
    Intent elevatorOneActivity;
    Intent elevatorTwoActivity;
    Intent elevatorThreeActivity;

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

Then we create intents for each elevator we have and insert the intent in the hashtable with the ID as key and the intent as a value.

```java
        elevatorOneActivity = new Intent(this,
            ElevatorNumberOneActivity.class);
        elevatorTwoActivity = new Intent(this,
            ElevatorNumberTwoActivity.class);
        elevatorThreeActivity = new Intent(this,
            ElevatorNumberThreeActivity.class);

        qrCodesTable.put("2266", elevatorOneActivity);
        qrCodesTable.put("5544", elevatorTwoActivity);
        qrCodesTable.put("4433", elevatorThreeActivity);
```

Surface view then needs to linked to the layout's surface view through the ID provided to it in the layout, and the same thing is done for the text view.

We need to also build the barcode detector and build the camera source and command the camera source to use the barcode detector.

```
        surfaceView = (SurfaceView)findViewById(R.id.cameraPreview);
        textView = (TextView)findViewById(R.id.scannerTextView);

        barcodeDetector = new BarcodeDetector.Builder(this)
                    .setBarcodeFormats(Barcode.QR_CODE).build();

        cameraSource = new CameraSource.Builder(this,
            barcodeDetector)
                    .setRequestedPreviewSize(640,480).build();
```

Now we can add a call back for the surface view and define what happens when the surface is created, destroyed or changed. When the surface is created we will check if the application has been granted permission to use the mobile's camera, if not then it will ask the user for permission and once permission is given it will start the camera to be displayed in the surface view. If the surface changes nothing will occur and if it is destroyed we need to stop the camera source.

```
surfaceView.getHolder().addCallback(new SurfaceHolder.Callback() {
            @Override
            public void surfaceCreated(SurfaceHolder holder) {
                if(ActivityCompat.checkSelfPermission(getApplicationContext(),
                    Manifest.permission.CAMERA) !=
                    PackageManager.PERMISSION_GRANTED){
                    requestPermission();
                }

                try {
                    Thread.sleep(2000);
                    cameraSource.start(holder);
                } catch (IOException | InterruptedException e) {
                    e.printStackTrace();
                }
            }

            @Override
            public void surfaceChanged(SurfaceHolder holder, int
                format, int width, int height) {

            }

            @Override
            public void surfaceDestroyed(SurfaceHolder holder) {
                    cameraSource.stop();
            }
        });

    public void requestPermission(){

        ActivityCompat.requestPermissions(this,
                new String[]{Manifest.permission.CAMERA}, 1);
    }
```

Then we need to set processor for the barcode were we will define what happens when we receive detection of a QR code. Once we receive detection well define a sparse to store the values of the QR code. If the array size is not equal to zero means we have detected a QR code and the phone will vibrate for a split second. The value of the QR code second or the ID in this case is represented by the value of the zero index of the array. We will get the value and see if it gives a null value in the hashtable. If it is equal to null we will display a toast that says Invalid QR code scanned. If it is not equal to null then we will get the value of that key which is an intent of an elevator activity and we start that activity represented by the value. [2]

```
barcodeDetector.setProcessor(new Detector.Processor<Barcode>() {
    @Override
    public void release() {

    }

    @Override
    public void receiveDetections(Detector.Detections<Barcode>
        detections) {
        final SparseArray<Barcode> qrCodes =
            detections.getDetectedItems();
```

```java

        if(qrCodes.size()!=0)
        {
                Vibrator vibrator = (Vibrator)getApplicationContext()
                .getSystemService(Context.VIBRATOR_SERVICE);
                String qrCodeString =
                    qrCodes.valueAt(0).displayValue;
                if(qrCodesTable.get(qrCodeString)==null){
                    vibrator.vibrate(100);
                    textView.post(new Runnable() {
                        @Override
                        public void run() {
                            Toast toast =
                                Toast.makeText(getApplicationContext(),
                                "Invalid QR code scanned",
                                Toast.LENGTH_LONG);
                            toast.show();
                        }
                    });
                    try {
                        Thread.sleep(3000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
                else {
                    vibrator.vibrate(100);
                    startActivity(qrCodesTable.get(qrCodeString));
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }

                }

        }

    }
});
```

```

}
```

3.2.2 Elevator class

After we are done from creating the main activity we need to create a java class for each Elevator. Inside each elevator class we will give an attribute for each button we have in the elevator as an Integer and we will also define a number of floors attribute as an integer to define the number of floors we have in the elevator.

```java
public class ElevatorNumberOne {
    private int button1;
    private int button2;
    private int button3;
    private int button4;

    private int numberOfFloors;
```

Then we need to initialize each button value to zero and the number of floors to the actually number of floors represented by this elevator inside the constructor.

```java
    public ElevatorNumberOne(){
        button1 = 0;
        button2 = 0;
        button3 = 0;
        button4 = 0;
        numberOfFloors = 2;
    }
```

Afterwards, we will just add setters and getters for each of the attributes defined in the class. For the number of floors we won't add a set method for it as obviously we don't need to change the number of floors it should remain the same.

```java
    public int getButton1() {
        return button1;
    }

    public void setButton1(int button1) {
        this.button1 = button1;
    }

    public int getButton2() {
        return button2;
    }

    public void setButton2(int button2) {
        this.button2 = button2;
    }

    public int getButton3() {
        return button3;
    }

    public void setButton3(int button3) {
        this.button3 = button3;
    }

    public int getButton4() {
        return button4;
    }

    public void setButton4(int button4) {
        this.button4 = button4;
    }

    public int getNumberOfFloors() {
    return numberOfFloors;
    }
}
```

The class described is for elevator number one we will also create two classes for two more elevators. The difference between each class is just the number of buttons attributes and the number of floors. Elevator number two will have six floor and elevator number three will have 4 floors. The code for both classes can be found in listing 1 and listing 2 respectively below.

Listing 1: Second elevator java class

```java
public class ElevatorNumberTwo{
    private int button1;
    private int button2;
    private int button3;
    private int button4;
    private int button5;
    private int button6;

    private int numberOfFloors;

    public ElevatorNumberTwo(){
        button1 = 0;
        button2 = 0;
        button3 = 0;
        button4 = 0;
        button5 = 0;
        button6 = 0;
        numberOfFloors = 3;
    }

    public int getButton1() {return button1;}

    public void setButton1(int button1) {this.button1 = button1;}

    public int getButton2() {return button2;}

    public void setButton2(int button2) {this.button2 = button2;}

    public int getButton3() {return button3;}

    public void setButton3(int button3) {this.button3 = button3;}

    public int getButton4() {return button4;}

    public void setButton4(int button4) {this.button4 = button4;}

    public int getButton5() { return button5; }

    public void setButton5(int button5) { this.button5 = button5; }

    public int getButton6() { return button6; }

    public void setButton6(int button6) { this.button6 = button6; }

    public int getNumberOfFloors() { return numberOfFloors; }
}
```

Listing 2: Third elevator java class

```java
package adu.ac.ae.elevatorcontrolpanelapplication;

public class ElevatorNumberThree {
    private int button1;
    private int button2;

    private int numberOfFloors;

    public ElevatorNumberThree(){
        button1 = 0;
        button2 = 0;
        numberOfFloors = 1;
    }

    public int getButton1() { return button1; }

    public void setButton1(int button1) { this.button1 = button1; }

    public int getButton2() { return button2; }

    public void setButton2(int button2) { this.button2 = button2; }

    public int getNumberOfFloors() { return numberOfFloors; }
}
```

3.2.3 Elevator Activity class

Before we starting developing the java class for the activity we need to develop the xml layout for the activity. The xml layout will be a constraint layout and will consist of a number of android buttons that are equal to the number of floors of the elevator. Each button has also been given a drawable object shaped like a circle as a background so that the application buttons are shaped like a elevator buttons. Elevator one layout is shown in figure 5 below.

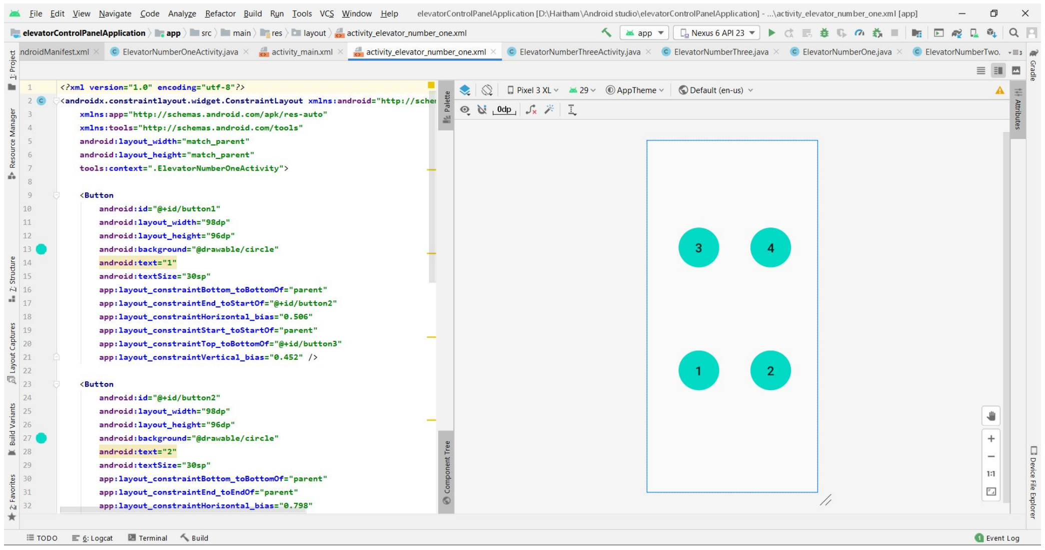
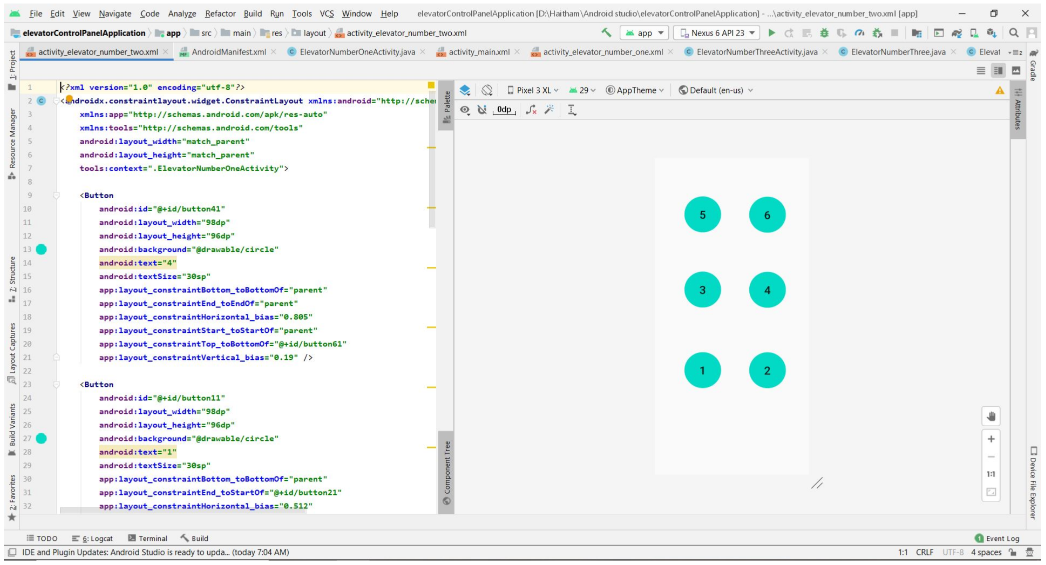

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://scher
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ElevatorNumberOneActivity">

    <Button
        android:id="@+id/button1"
        android:layout_width="98dp"
        android:layout_height="96dp"
        android:background="@drawable/circle"
        android:text="1"
        android:textSize="30sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/button2"
        app:layout_constraintHorizontal_bias="0.506"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/button3"
        app:layout_constraintVertical_bias="0.452" />

    <Button
        android:id="@+id/button2"
        android:layout_width="98dp"
        android:layout_height="96dp"
        android:background="@drawable/circle"
        android:text="2"
        android:textSize="30sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.798"
```
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
elevatorControlPanelApplication [D:\Haitham\Android studio\elevatorControlPanelApplication] - ...\activity_elevator_number_one.xml [app]
elevatorControlPanelApplication app src main res layout activity_elevator_number_one.xml
ndroidManifest.xml
ElevatorNumberOneActivity.java
activity_main.xml
activity_elevator_number_one.xml
ElevatorNumberThreeActivity.java
ElevatorNumberThree.java
ElevatorNumberOne.java
ElevatorNumberTwo.
Pixel 3 XL
29
AppTheme
Default (en-us)
0dp
3
4
1
2
TODO
6: Logcat
Terminal
Build
Event Log

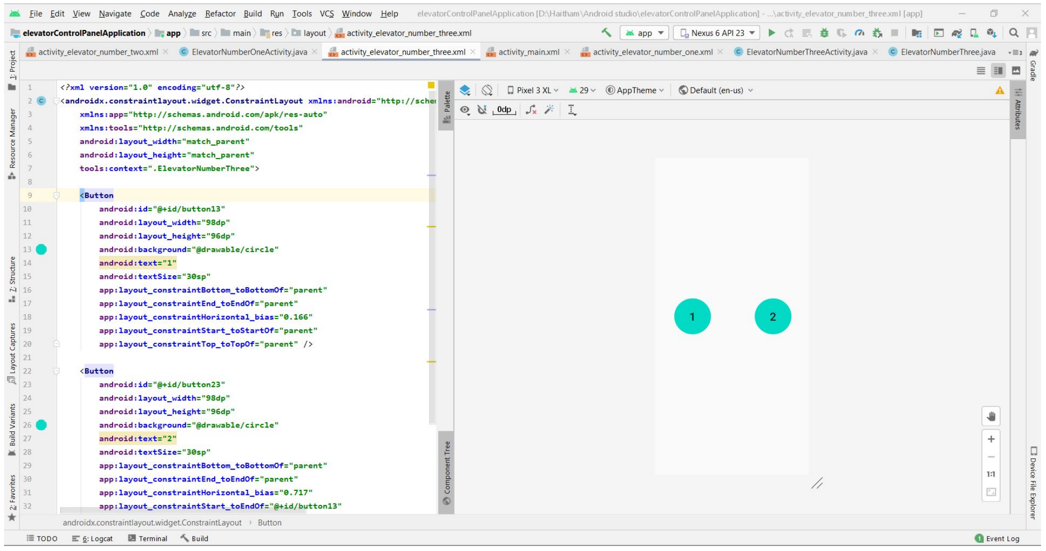
Figure 5: Elevator one activity layout.

The layouts developed for the other two elevators based on the same constraints can be seen in figure 6 and 7 respectively.



```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schem
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ElevatorNumberOneActivity">

    <Button
        android:id="@+id/button41"
        android:layout_width="98dp"
        android:layout_height="96dp"
        android:background="@drawable/circle"
        android:text="4"
        android:textSize="30sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.805"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/button61"
        app:layout_constraintVertical_bias="0.19" />

    <Button
        android:id="@+id/button11"
        android:layout_width="98dp"
        android:layout_height="96dp"
        android:background="@drawable/circle"
        android:text="1"
        android:textSize="30sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/button21"
        app:layout_constraintHorizontal_bias="0.512"
```

Figure 6: Elevator two activity layout.



```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schem
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ElevatorNumberThree">

    <Button
        android:id="@+id/button13"
        android:layout_width="98dp"
        android:layout_height="96dp"
        android:background="@drawable/circle"
        android:text="1"
        android:textSize="30sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.166"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/button23"
        android:layout_width="98dp"
        android:layout_height="96dp"
        android:background="@drawable/circle"
        android:text="2"
        android:textSize="30sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.717"
        app:layout_constraintStart_toEndOf="@+id/button13"
```

Figure 7: Elevator two activity layout.

Now we can start with developing the java classes for the activities. But just before we start you need to make sure you've connected your android project to your real time database in firebase. First we need to define the buttons in the elevator and we need to define a refrence to the database. [3]

```java
public class ElevatorNumberOneActivity extends AppCompatActivity {

    DatabaseReference reff;
    Button firstFloorButton;
    Button secondFloorButton;
    Button thirdFloorButton;
    Button fourthFloorButton;
```

Then we need to define the layout to display when running this activity and link all the buttons defined to the layout buttons using their ID's.

```java
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_elevator_number_one);
        firstFloorButton = (Button)findViewById(R.id.button1);
        secondFloorButton = (Button)findViewById(R.id.button2);
        thirdFloorButton = (Button)findViewById(R.id.button3);
        fourthFloorButton = (Button)findViewById(R.id.button4);
```

Now we will create an ElevatorNumberOne object built from the default constructor of the class. Then we will link the reff of the database to a child object we called it Elevator. Each elevator will be defined under this object and we will add to it the ElevatorNumberOne object as a child and will give it the name "ElevatorNumberOne"

```java
final ElevatorNumberOne elevatorNumberOne = new
    ElevatorNumberOne();
reff =
    FirebaseDatabase.getInstance().getReference().child("Elevator");
reff.child("ElevatorNumberOne").setValue(elevatorNumberOne);
```

And finally we will add on click listeners to the buttons defined earlier and once the event on click occurs to one of the buttons, inside the listener we will set the value of that button to 1 and update the database. We will add afterwards a sleep for 2 seconds and then set back the value of the button to 0 and update the database again so that the Raspberry pi reads the change in value just once.

```java
firstFloorButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        elevatorNumberOne.setButton1(1);
        reff.child("ElevatorNumberOne").setValue(elevatorNumberOne);
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        elevatorNumberOne.setButton1(0);
        reff.child("ElevatorNumberOne").setValue(elevatorNumberOne);
    }
});

secondFloorButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        elevatorNumberOne.setButton2(1);
        reff.child("ElevatorNumberOne").setValue(elevatorNumberOne);
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        elevatorNumberOne.setButton2(0);
        reff.child("ElevatorNumberOne").setValue(elevatorNumberOne);
    }
});

thirdFloorButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        elevatorNumberOne.setButton3(1);
        reff.child("ElevatorNumberOne").setValue(elevatorNumberOne);
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        elevatorNumberOne.setButton3(0);
        reff.child("ElevatorNumberOne").setValue(elevatorNumberOne);
    }
});

fourthFloorButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        elevatorNumberOne.setButton4(1);
        reff.child("ElevatorNumberOne").setValue(elevatorNumberOne);
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        elevatorNumberOne.setButton4(0);
        reff.child("ElevatorNumberOne").setValue(elevatorNumberOne);
    }
});
}
}
```

The java code for the other two elevators activities is identical to the class just described with changes just in the number of buttons and the number of listeners.

4 Results and analysis

4.1 Application results

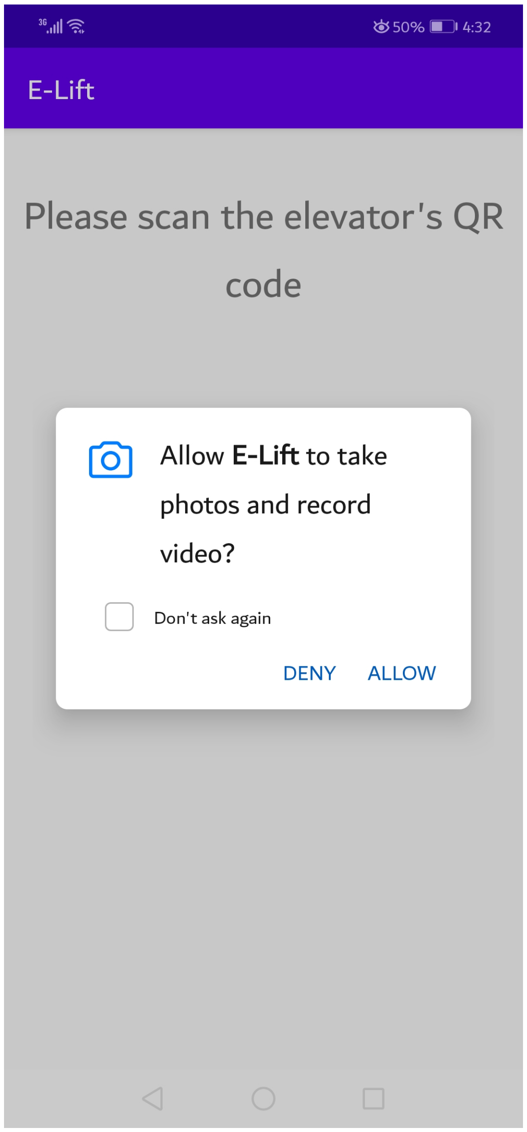
The application designed we tested it out on 3 QR codes that each represent an ID present in the hashtable that links IDs to elevator activities. To perform the test we have generated 3 QR codes with IDs 2266 , 5544, and 4433. The first QR code or the first ID will is linked to the activity of the first elevator which has 2 floors, the second one is linked to the second elevator which has 3 floors, and the final one is linked to third elevator which has 1 floor. The 3 QR codes generated can be seen in figure 8, 9, and 10.

Figure 8: QR code for ID 2266.

Figure 9: QR code for ID 5544 .

Figure 10: QR code for ID 4433.

But before we scan QR codes through the application once we start it we expect it to ask the user for permission to use the camera and it it does do that as we can see from the figure 11 below.



Please scan the elevator's QR code

Allow **E-Lift** to take photos and record video?

Figure 11: Asking the user for camera usage permission.

Once the user grants permission we can see the camera displayed in the surface view and a text view on top asking the user to scan an elevator QR code. If the user's scans an invalid or irrelevant QR code, as expected it will vibrate for a split second and display a toast saying invalid QR code as shown in figure 12 below.

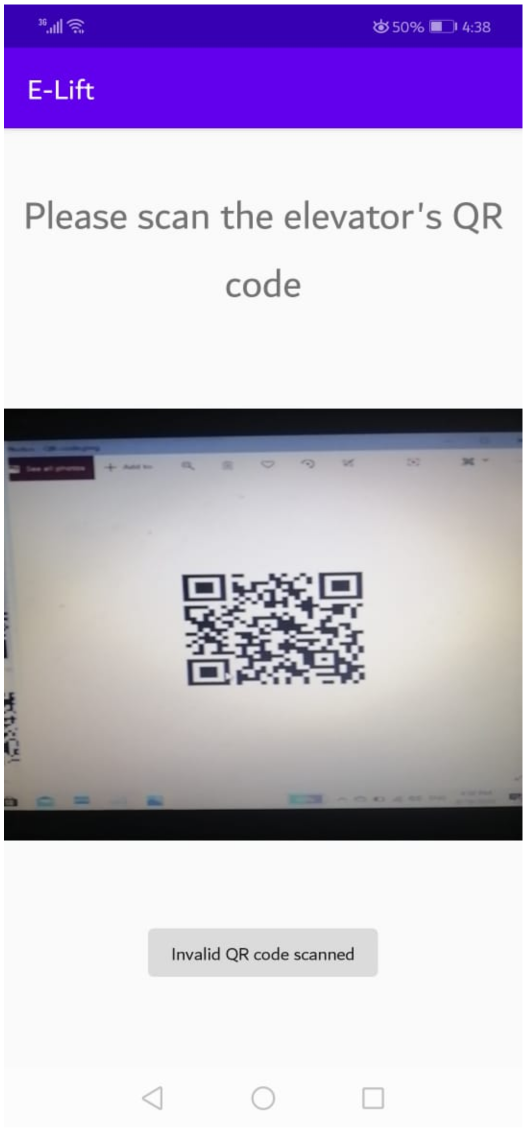


Figure 12: Invalid QR code scanned.

If we scan now one of the QR codes in figures 8, 9, and 10 the application will move into an elevator activity and will display the layout of the activity. If we scan the QR code in figure 8 it will provide us with an Integer ID of 2266 and this ID links to elevator one activity which has 2 floors so we expect to see 4 buttons in the layout and we do see that as displayed in figure 13 below.

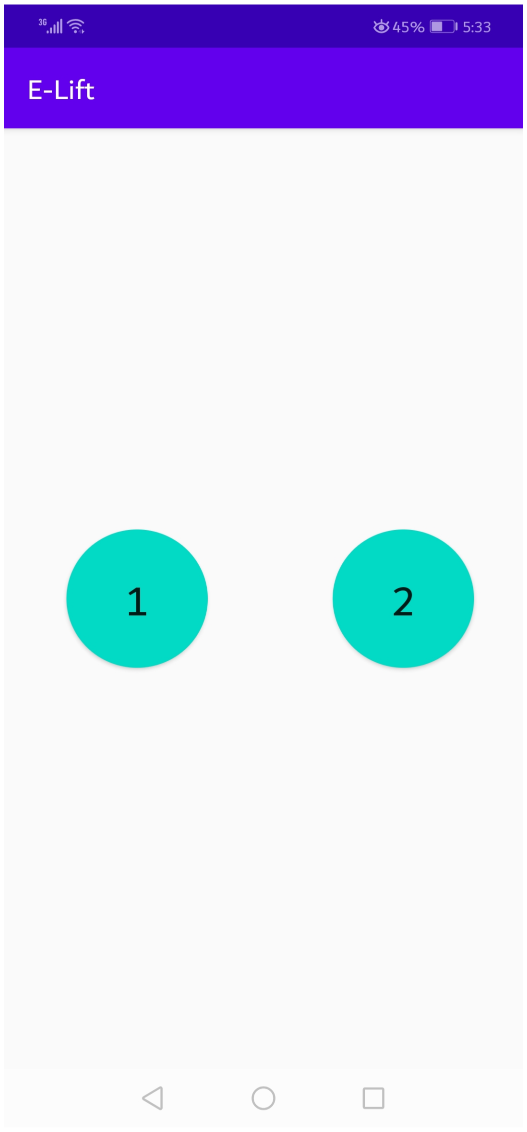


Figure 13: QR code with ID 2266 scanned result.

If we scan the QR code in figure 9 that provide us with an Integer ID of 5544. The application will transition into elevator two activity and elevator two has 3 floors so we expect a layout of 6 buttons to be displayed and the result is as expected as we can see from figure 14 below.

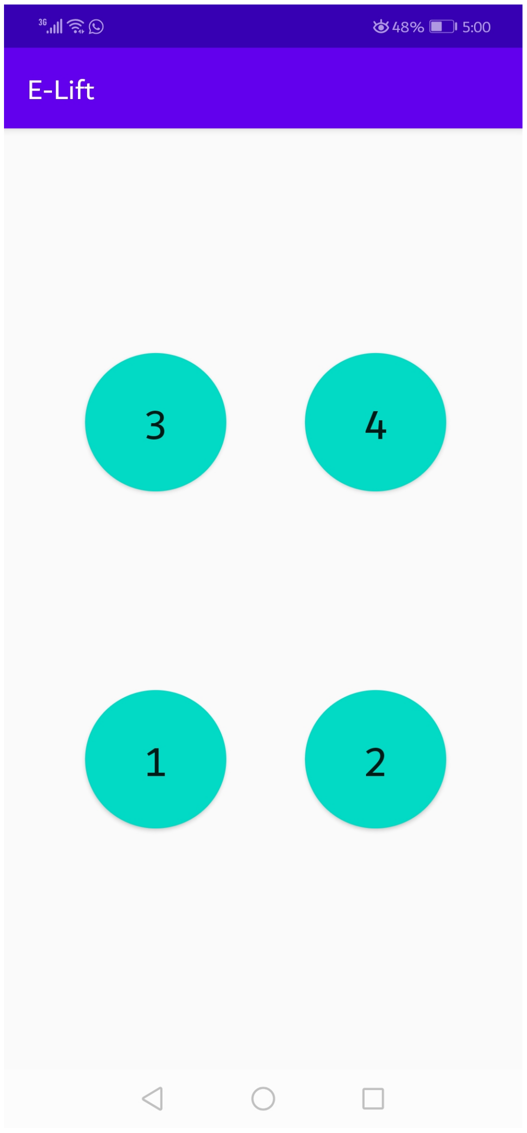


Figure 14: QR code with ID 5544 scanned result.

And finally, if we scan the QR code in figure 10 which represent an ID of 5544. The application will transition into elevator three activity which only 1 floor and 2 buttons. Result for the QR code scan is displayed in figure 15 below.

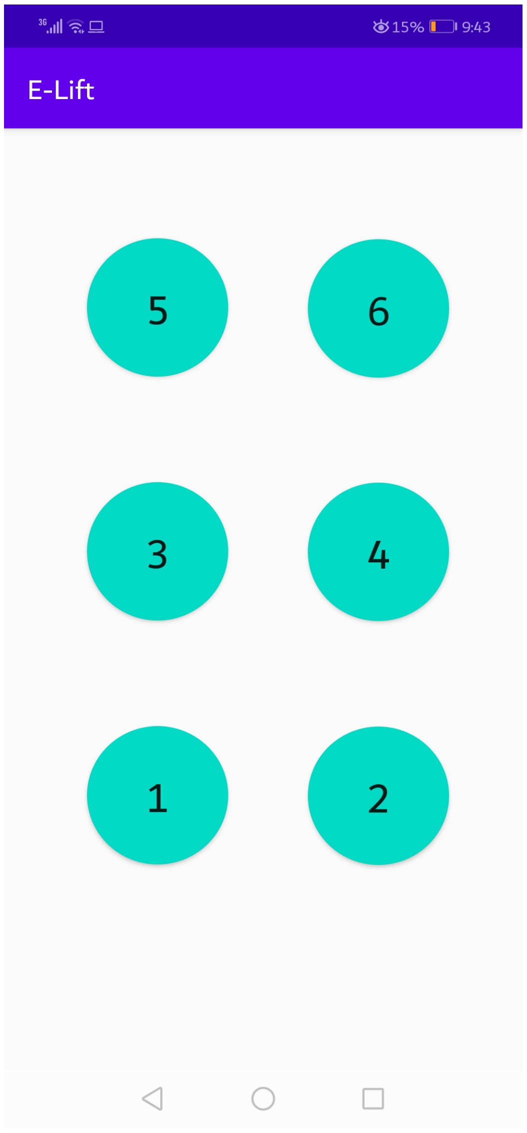
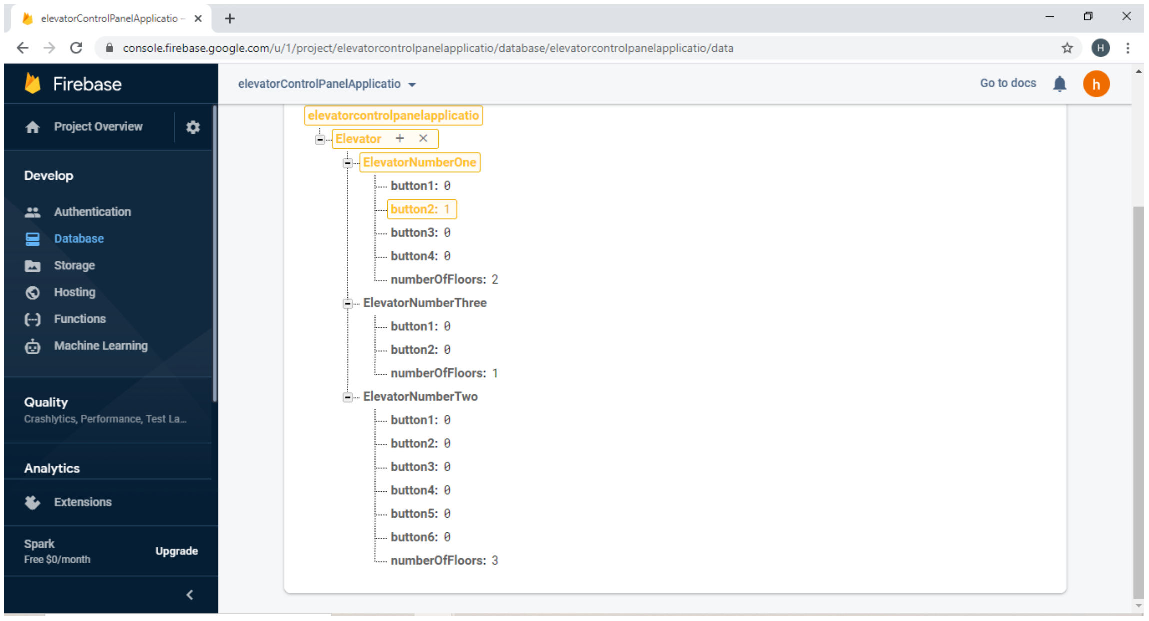


Figure 15: QR code with ID 4433 scanned result.

4.2 Overall system results

After we have tested out our application we will see if the whole system is functioning as expected with the Raspberry pi. When you press on an android button in the application the value of the button in database should be alerted from zero to 1 for few seconds and as we can see from figure 16, the value does change.


elevatorControlPanelApplicatio
console.firebase.google.com/u/1/project/elevatorcontrolpanelapplicatio/database/elevatorcontrolpanelapplicatio/data
Firebase
Project Overview
Develop
Authentication
Database
Storage
Hosting
Functions
Machine Learning
Quality
Crashlytics, Performance, Test La...
Analytics
Extensions
Spark
Free $0/month
Upgrade
elevatorControlPanelApplicatio
Go to docs

```
elevatorcontrolpanelapplicatio
  Elevator
    ElevatorNumberOne
      button1: 0
      button2: 1
      button3: 0
      button4: 0
      numberOfFloors: 2
    ElevatorNumberThree
      button1: 0
      button2: 0
      numberOfFloors: 1
    ElevatorNumberTwo
      button1: 0
      button2: 0
      button3: 0
      button4: 0
      button5: 0
      button6: 0
      numberOfFloors: 3
```

Figure 16: Value of button changed from 0 to 1 in database.

After we connected our raspberry pi, we build the prototype shown in figure 17 to stimulate servo motors attached to an elevator control panel

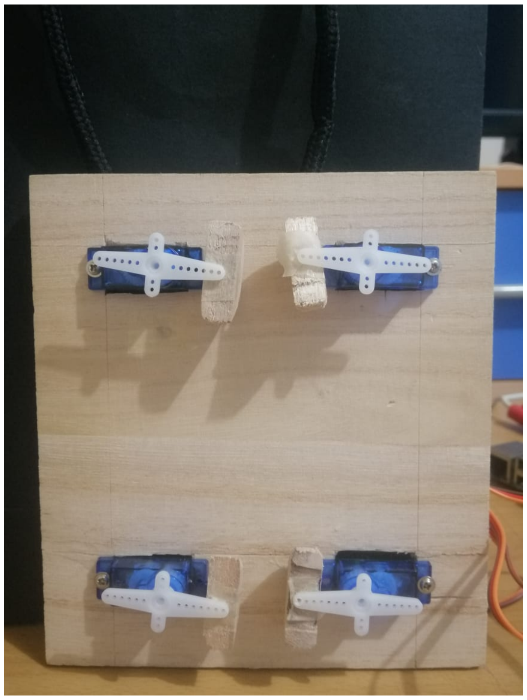


Figure 17: Servo motors placement prototype.

Each servo motor has a wooden peace attached to in a downward position and once an android button is pressed by the user on the application the servo motor will rotate by 90 degrees to press the elevator button. If the second button is pressed on the application we expect the second servo motor in our prototype to rotate 90 degrees and hit any physical button above it and as we can see from figure 18, it functions as expected.

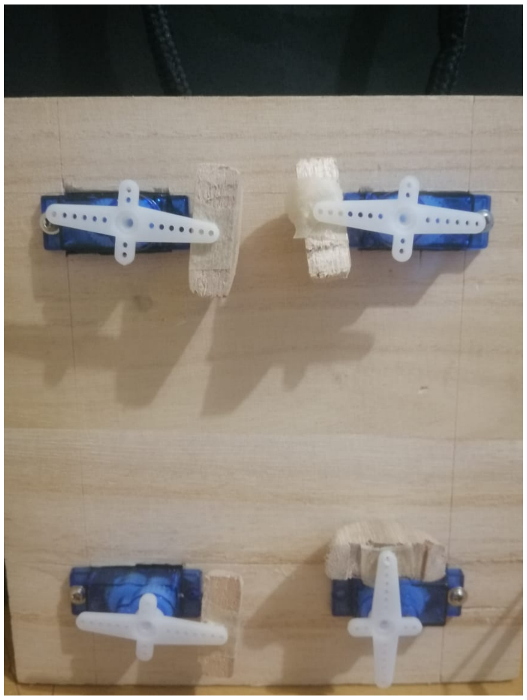


Figure 18: Second button pressed result.

We tested it also on the third button and once we press it we expect the same to happen for the servo motor; to rotate 90 degrees and hit any physical button below it and we can see that is this through figure 19 below.

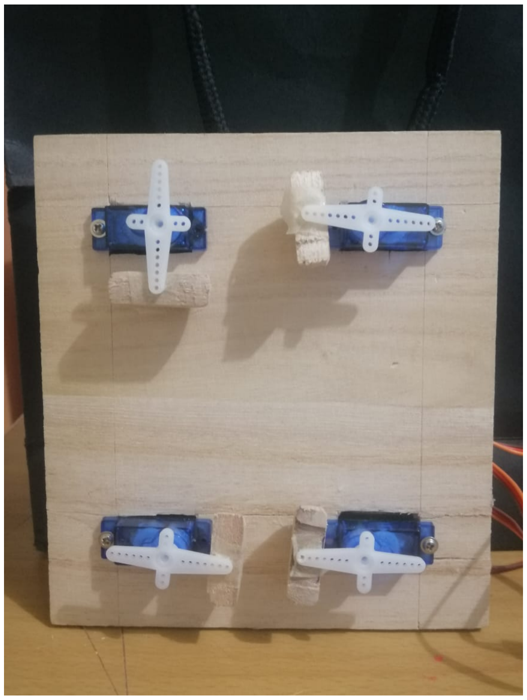


Figure 19: Third button pressed result.

5 Conclusion

After working on the project for several days, I could say that I've learned a lot through that experience. I've been able to enhance my skills in application development and learned about scanning QR codes and firebase applications through android studio. I've also learned about python firebase applications and how to use Raspberry pi in controlling GPIO's. In addition I have been able to explore with servo motors and how to create a pushing mechanism using them. To summarize the project, the system designed proofed to be a successfully in providing people a way to use an elevator without any contact through their phones. The results from testing the system were all successful and inline with what is expected. I hope that through the system designed will help many reduce the risk of contracting the virus during this pandemic and I am happy to have used technology combined with innovation to help with solving a common issue that many will probably face during this difficult period of time.

References

[1] O. Vantansever, "Pypi. 2014. python-firebase," march 2014. [Online]. Available: https://pypi.org/project/python-firebase/

[2] A. Chugh, "Qr code scanner - barcode scanner for android - journaldev." [Online]. Available: https://www.journaldev.com/18198/qr-code-barcode-scanner-android

[3] google, "Add firebase to your android project," June 2020. [Online]. Available: https://firebase.google.com/docs/android/setup