A. Introduction

The problem

The client owns a house with multiple gates that need to be open either manually or by a radio remote controller appropriate for each gate. The issue is that he is forced to have multiple remote controllers, which is both impractical and uncomfortable. He wants a solution that would allow him to control all the gates remotely from only one device and with an option to add or remove gates. Furthermore, he is concerned with the safety of his property because of the robbery that took place in his neighbourhood. Thus, he wants to be able to check if the garage doors are opened or closed. The client emphasizes that above all he wants to ensure that the solution will be secure, and no one without his permission will gain access to the control system. The transcript of the consultations can be found in appendix 1.

The solution to the problem

Having listened to the client's problem, I proposed to him a solution. Since each gate has an electrical opener, it can be bypassed by another device with an internet connection. The additional devices will be connected to a phone app and will be openable via a phone app. To be specific, an application for the Android OS. The user will be able to choose which gate he wants to open. However, before a user obtains access to the application's functionality, he will have to give a password and a login. This step ensures that even if the phone is lost, no one unwanted will have access to the property behind the gates. Besides, the information about the states of gates gate is open will be shown in the GUI. The gates will be monitored by magnetic switches and stored in a file. For implementation, my languages of choice are java and python.

The rationale for solution

The given solution satisfies the major requirements of the client. It obsoletes the need for having multiple controllers and allows user to monitor the gates remotely. Connecting all the gates to one remote controller might be plausible for some time. However, such a solution does not allow to check the state of gates, which is one of the client's requirements. Besides, in the case of losing the remote controller connected to many gates, the only solution to ensure the security of gates is to replace all radio units. Such a case is both dangerous and expensive. On the contrary, if the phone is stolen and a criminal obtains credentials needed, the system can be deactivated and credentials changed. I implemented my application with use of two higher level languages; Python and Java. The choice of Python was justified by use of Raspberry pi because I intended to use its GPIO pins, which can be utilised by Python code. Java code is implemented because of its JSCH library, which is crucial in my solution. Further explanation is provided in the section C.

Success criteria

Criterion
The phone application runs
The correct credentials are required to establish a connection between a phone and gates
The states of gates are correctly identified
The gates can be both opened and closed
The credentials can be changed
Application keeps track of gates' state and registers clicks of open/close buttons

B. Design/Solution overview

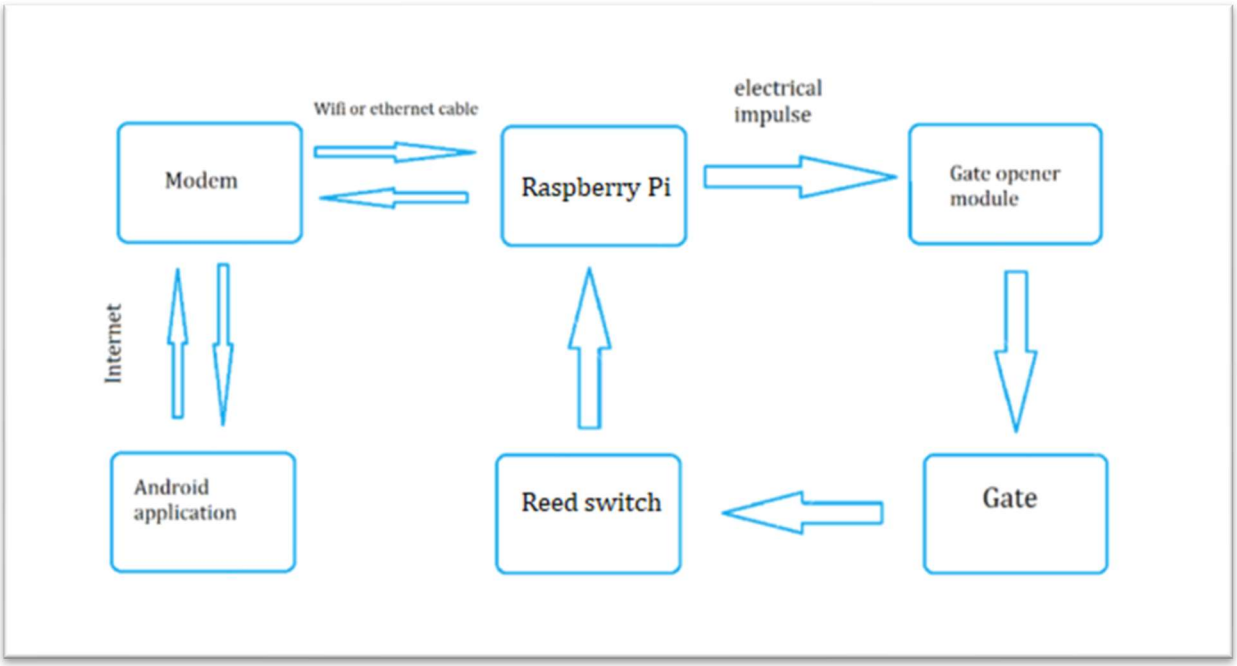

FIGURE 1 THE SYSTEM OUTLINE

The main issue that must be solved in the design part is the connection between a phone and a gate opener module. The opener module does not have internet connectivity or any other connectivity module excluding the radio receiver. The internet is the most convenient and flexible way of connecting a phone with any other device. Thus, internet connectivity must be added to the gate opener module. It can be solved with the use of a computer, preferably a single board one since high performance is not required. Raspberry Pi is such a computer and it will serve as a bridge between a phone and gate opener module. Moreover, other external devices can be connected, like magnetic switches, which are necessary to monitor a state of a gate. The magnetic switches can be connected to the Raspberry Pi as well. Thus, allowing to monitor the state of the gates remotely.

Now, I will demonstrate the design of the application

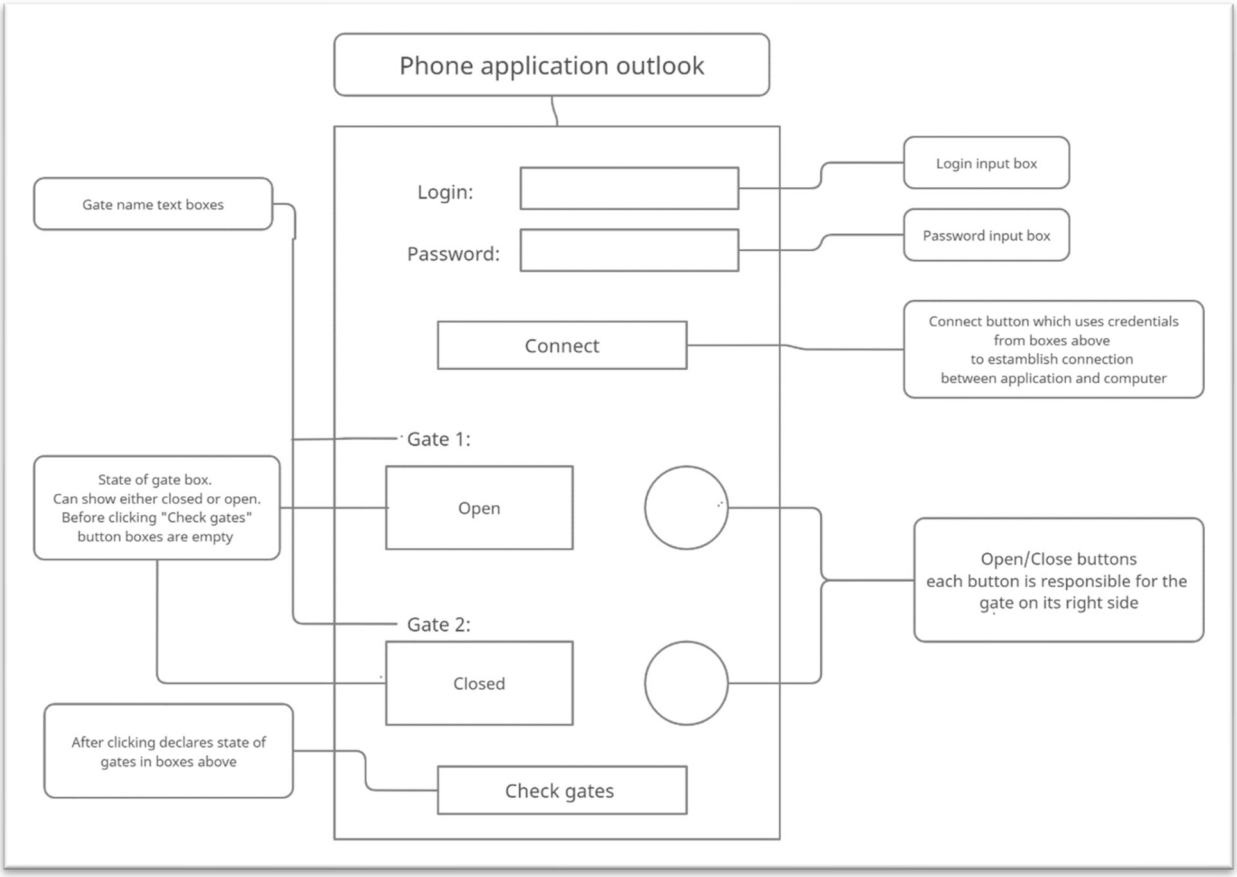

FIGURE 2 PHONE APPLICATION GUI DESIGN

The Figure (2) shows the desired final look of the phone application, which will serve as a GUI in this system.

Test plan:

Action name	Description
Check if the application runs	Transfer application file to customer's phone and open it
Verify credentials	See if the system reacts for other inputs for "login" and "password"
Check the connection between the application and raspberry pi and the state of gates is correctly identified	Click the check button and see if the states of gates are declared. Moreover, see if the states of gate are correctly identified
Open the gates	Click "open/close" button for individual gates and observe if the correct gate opens
Close the gates	Click "open/close" button for individual gates and observe if the correct gate closes
Change credentials	Change credentials and verify if the change has been successful via inputting previous credentials and new credentials. The system should not accept previous credentials and accept new ones
Open log file	Open the log file and see if the states of gates are saved and the button clicks registered.

C. Development

Despite the simple design of the solution, it consists of many independent elements and steps. The whole implementation procedure can be divided into two main parts.

- Hardware
- Software

The hardware refers to all physical and electronic elements, like a magnetic switch, whilst software refers to the code and network connectivity. Thus, this segment is divided into two parts.

Hardware

The first step in the implementation must be the placement of the magnetic switches. Their place influences further code implementation and serves as a detector for the gate's state. Figure (1) shows their placement in the case of the client's garage.

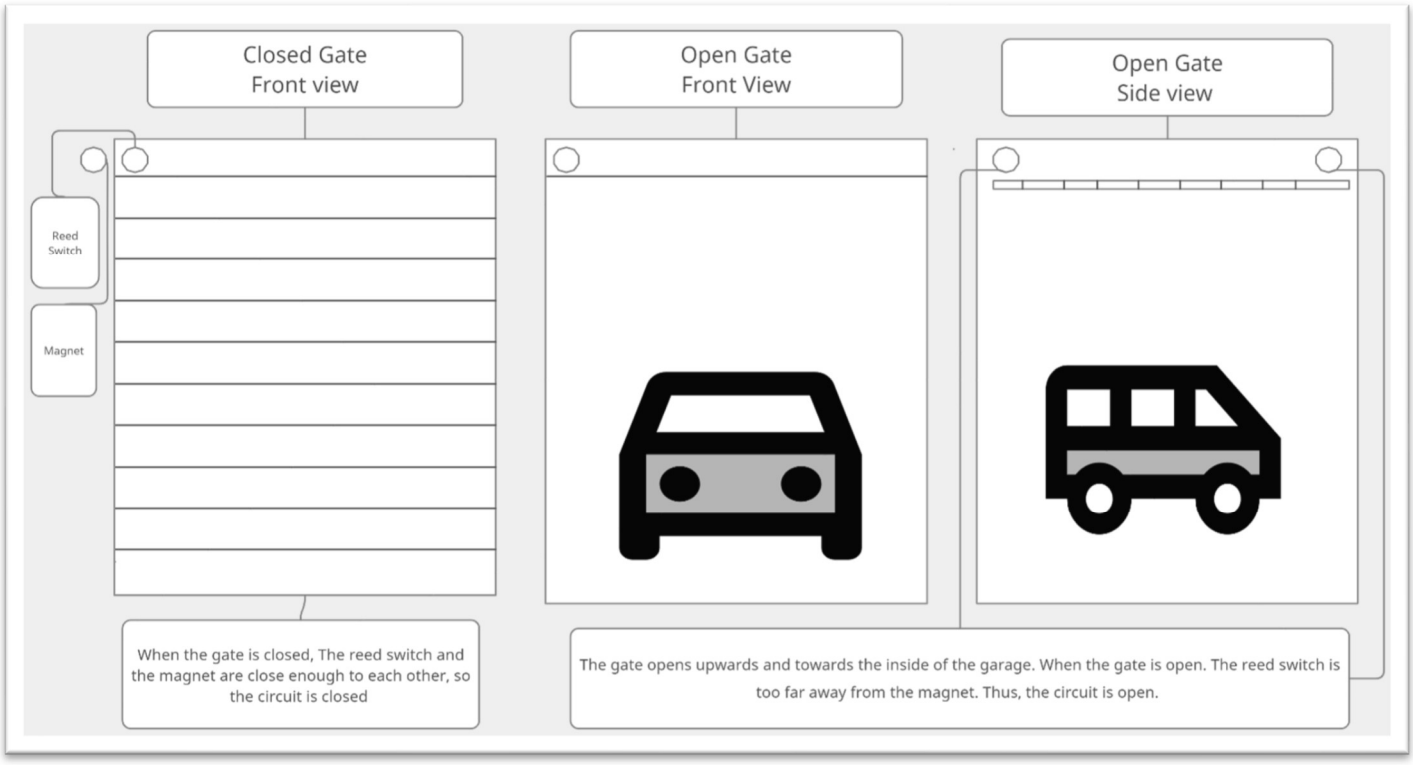

FIGURE 1 PLACEMENT OF REED SWITCH AND MAGNET IN A GARAGE

Once the switches are placed, they must be connected to the Raspberry pi, which will read information from them. They are connected to the raspberry's GPIO board with a connector jumper cable. Moreover, a relay board is connected to the board. The computer could not be directly connected to the gate opener steering module because they operate at different voltages. If connected directly, the damaged could be caused to the circuits. The wiring is shown in figure (2).

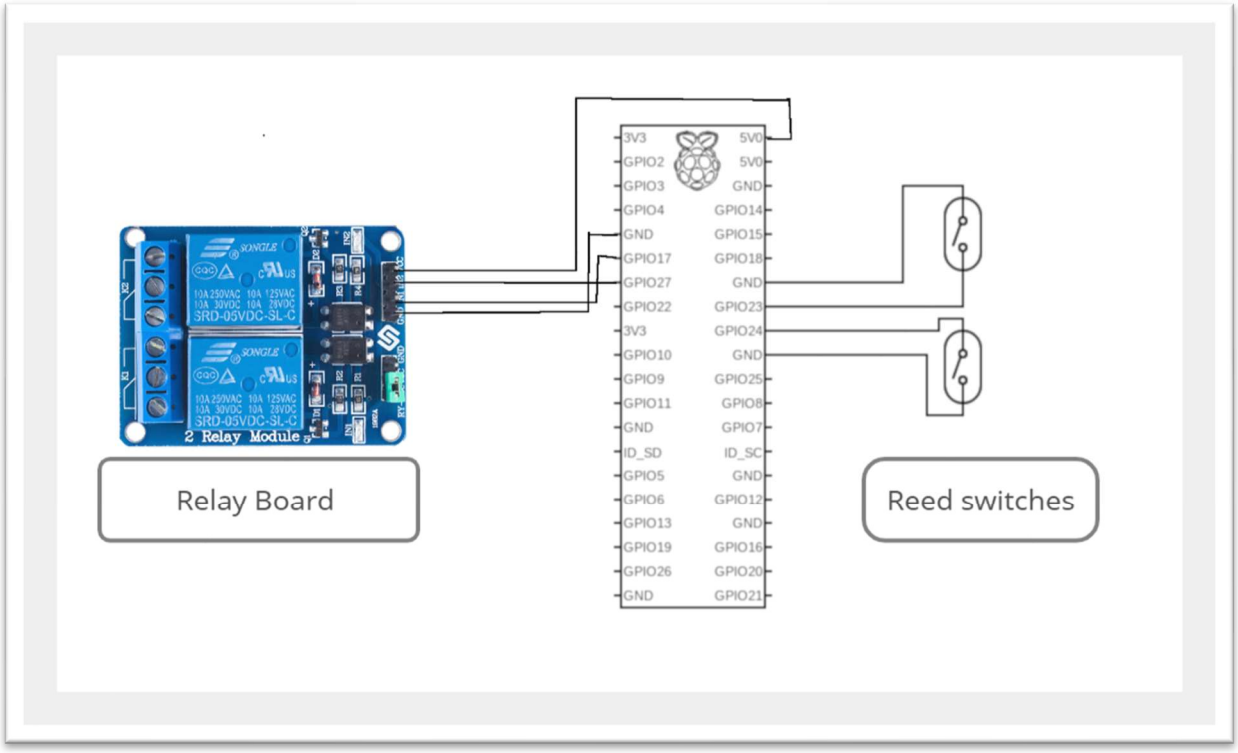

FIGURE 2 RASPBERRY PI WIRING SCHEMATICS

Figure (3) shows a connected device to the raspberry pi, which has already a mounted SD card with Raspbian OS.

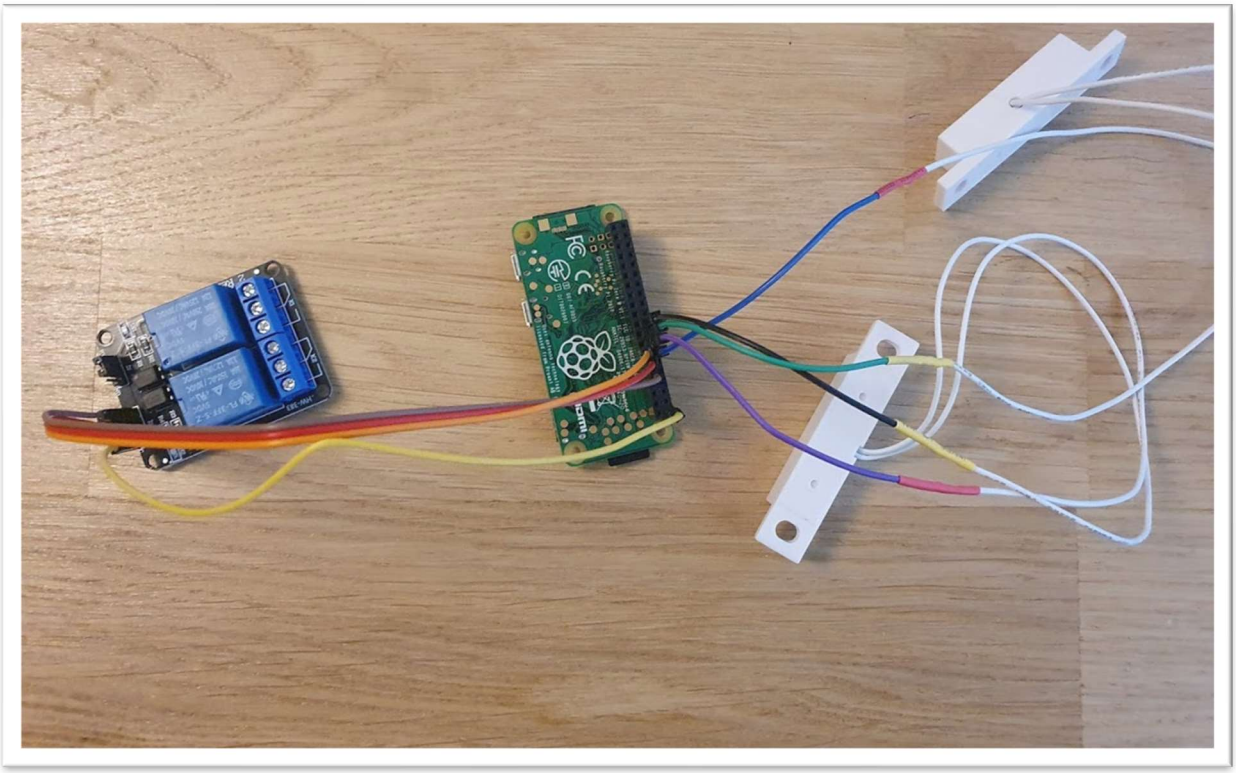

FIGURE 3 PICTURE OF RASPBERRY PI WITH CONNECTED DEVICES

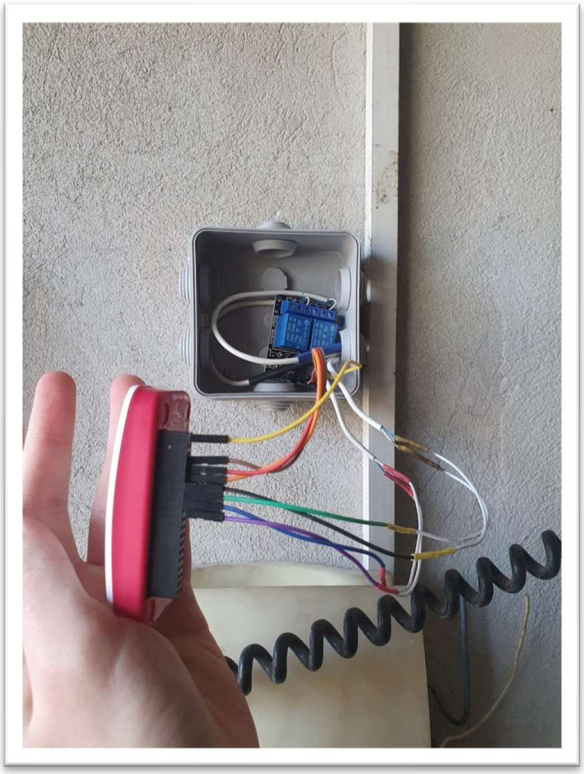

FIGURE 4 RASPBERRY PI MOUNTED IN THE GARAGE

FIGURE 5 RASPBERRY PI IN THE GARAGE AND CLOSED IN A BOX

The next step was writing the necessary code for controlling the external devices. The language I have chosen is Python. The main reason for this choice was its GPIO library which allows using GPIO pins on the Raspberry and its relative simplicity in writing in text files via code.

Software

The solution regarding solution can be divided into two independent elements:

1. The python code on the Raspberry pi
2. The android application, which can be subdivided into:
    a. XML layout code
    b. Java main code

the reason is that as a way of connecting the Raspberry pi with a phone is used SSH. The phone application does not contain any code responsible for the procedures of controlling and monitoring gates. These programmes are stored on the Raspberry Pi and only requests to execute them are sent from the application. Hence, two different languages can be implemented and developed separately. Moreover, Android studio IDE provides many XML methods, which shorten the GUI development process. Thus, three different languages are used. Nonetheless, before any attempt to perform any actions via an SSH connection, the connection's stability must be ensured.

To establish a SSH connection between two devices, three components are needed:

- Name of user
- Password or a key

- IP address

The first two elements do not change autonomously under usual circumstances, but the IP address might. In this case, the required address is the one assigned and used within a LAN. Usually, the routers assign to any connecting device one of the available addresses, which can be different for each session. Therefore, a static IP address has to be assigned to Raspberry Pi, unless the client wants to search for an address each time one uses the application.

The code responsible for controlling and monitoring codes is divided into four programmes:

1. Code that declares the state of gates (appendix 2)
2. Code that opens/closes gate 1 (appendix 3)
3. Code that opens/closes gate 2 (appendix 4)
4. Code that registers in the log file the states of gates (appendix 5)

The first three are executed upon a request, whilst the last one is always running in the background. The key component in this implementation phase is the use of RPi.GPIO library for python, which allows using external devices with python programmes. Besides RPi.GPIO library, the programs have a time library, which allows sending an electronic impulse for a precise amount of time. Another method imported is DateTime, which allows declaring the current time, which is used in the last method, which is editing text files. The logging of the gates states is performed is done by checking their state every five minutes and registering their state in a text file.

Once the way of controlling the gates is developed, the android application can be implemented.

The GUI of the app is presented on the Figure (5) below.

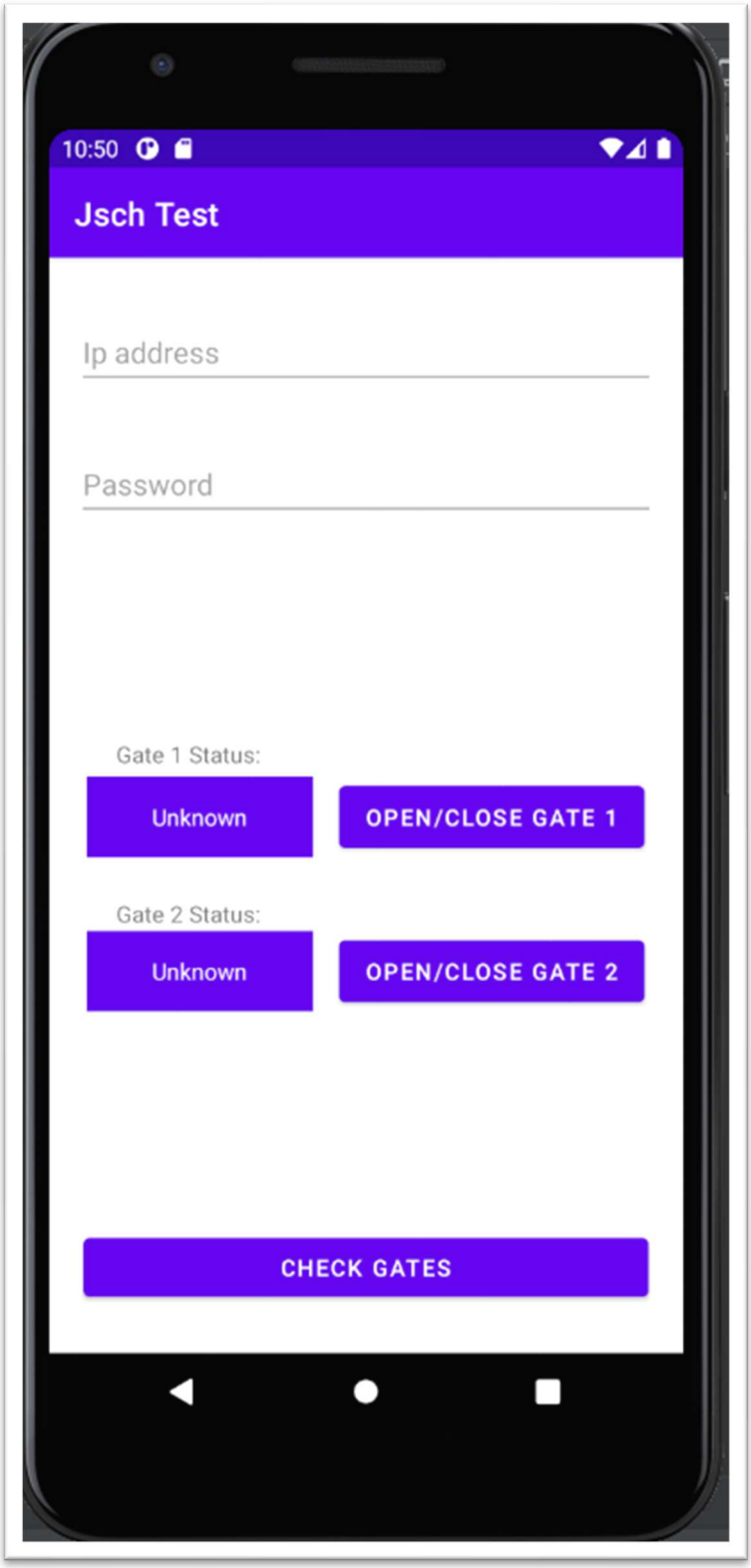

FIGURE 5 GUI OF APPLICATION

The two classes used to create it are TextInputLayout and Button. The first class creates an editable input field, where user can write data in. It serves as a security measure since even if client's phone is stolen, the gates cannot be open without credentials. In my solution, the credentials are the IP address and the password to Raspberry Pi's user's password.

```
// input box for the login (ip) of the device
<com.google.android.material.textfield.TextInputEditText
    android:id="@+id/HostInput"
    android:layout_width="359dp"
    android:layout_height="46dp"
    android:layout_marginStart="16dp"
    android:layout_marginTop="32dp"
    android:layout_marginEnd="16dp"
    android:layout_marginBottom="32dp"
    android:hint="Ip address"
    android:inputType="text"
    app:layout_constraintBottom_toTopOf="@+id/PasswordInput"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

FIGURE 6 TEXTINPUTEDITTEXT XML METHOD

The data input in these boxes are used to establish SSH connection. It is performed by clicking one of three buttons visible on the GUI. The object button can be set up to listen on a click, meaning whenever it is clicked it executes a given task. The description of all elements in the layout can be read in the appendix 6.

```
//button for opening/closing gate 1
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="50dp"
    android:layout_marginTop="100dp"
    android:layout_marginEnd="8dp"
    android:layout_marginBottom="24dp"
    android:text="Open/Close Gate 1"
    app:layout_constraintBottom_toTopOf="@+id/button2"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toEndOf="@+id/StateGate1"
    app:layout_constraintTop_toTopOf="@+id/guideline2" />
```

FIGURE 7 BUTTON XML METHOD

```
//After clicking Button 1 the code method is executed. The button 1 opens/closes gate 1, so
// its sets the command to execute code that opens gate 1
Button btn1 = findViewById(R.id.button1);
btn1.setOnClickListener(v -> {
    //open directory test and execute python code Relay 1
    command = "cd test && python Relay1.py";
    code(command);
    })
    ;
```

FIGURE 8 SETTING BUTTON TO LISTEN ON CLICK

In this case, button1 after a click executes a class “code” with a given string named “command”. The “code” class is a crucial component of the phone application because it is responsible for establishing SSH connection and sending a request to execute a given command. The detailed description of this method can be found in the appendix (7).

The next method created is a method Check that is to a great extent similar to the previous method. It is executed by the “Check gates” button on the GUI and changes the displayed states of gates.

```
@SuppressLint("SetTextI18n")
private void onClick(View v) {
    command = "cd test && python mswitchtest.py";
    Check(command);
    }

// method Check is very similar to the method Code. The both establish an SSH connection
// and execute a given command. However, the method Check listens on the response from other
// device and depending on it, changes the text describing states of gates in the GUI.

private void Check(String command) {
    String host = HostText.getText().toString();
    String password = PasswordText.getText().toString();
    Executor executor = newSingleThreadExecutor ();

    //Creates an empty object outputBuffer
    StringBuilder outputBuffer = new StringBuilder();
```

FIGURE 9 METHOD CHECK

```
InputStream in = channel.getInputStream();
channel.connect();

byte[] tmp = new byte[1024];
while (true) {
    while (in.available() > 0) {
        int i = in.read(tmp,  off: 0,  len: 1024);
        if (i < 0)
            break;
        // Appends an outputBuffer with string of characters recived from the input
        // stream of channel
        outputBuffer.append(new String(tmp,  offset: 0, i));
```

FIGURE 10 THE OUTPUT BUFFER CONTAINS A STRING FROM INPUT STREAM

```
// The code below is responsible for displaying states of gates in the GUI
// The displayed test is dependent on the information printed by the mswitchtest.py

if (outputBuffer.toString().contains("Garage 1 is Closed")) {
    StateGate1.setText("Closed");
}else if(outputBuffer.toString().contains("Garage 1 is Open")){
    StateGate1.setText("Open");
}

if (outputBuffer.toString().contains("Garage 2 is Closed")) {
    StateGate2.setText("Closed");
}else if(outputBuffer.toString().contains("Garage 2 is Open")){
    StateGate2.setText("Open");
}
```

FIGURE 11 SEQUENCE DISPLAYING STATES OF GATES IN GUI

Criterion E

Test plan evaluation

Action name	Description
Check if the application runs	The application installs on the phone and runs.
Verify credentials	The incorrect credentials are not accepted. It was verified by inputting an incorrect pair of credentials.
Check the connection between the application and raspberry pi and the state of gates is correctly identified	After a click the states of gates are both correctly identified and displayed. Initially the state of gates is displayed as unknown, but once the button is clicked, the states are correctly displayed.
Open the gates	The phone application opens a correct gate after pressing the button. The gates are categorised as “gate 1” and “gate 2”, The gate 1 is on the left, if one is facing the garage from outside and stands in front of it. Correspondingly, the gate 2 is no the right side.
Close the gates	The phone application closes the correct gate. As previously, the gate 1 is on the left and gate 2 on the right side, if one is facing the garage from outside and stands in front of it.

Change credentials	The credentials can be modified, but not through the GUI. They can be modified through raspberry pi itself or through SSH connection.
Open log file	The log file can be open through a terminal and the required data is stored in it. Both the time and date are saved for each individual gate. Raspberry Pi registers states of gates continuously.

The application meets for the most part client's needs. The lack of an easy way of modifying credentials is compensated for a presumably a small need to modify them.

Further suggestions

- The VPN service might be introduced to the product. It would allow a way of checking the states of the gates from outside LAN, whilst remaining a necessary safety. As mentioned by the client (appendix 1) it would make the solution more convenient.
- The implementation of the solution allows for adding other devices to the ecosystem. For example, a driveway gate could be controlled through the application, if a raspberry pi with necessary components was connected to it.
- The application might be adjusted to the android wearable OS, resulting in the possibility of having the same functionality from a smartwatch, which would be an additional convenience.
- The client in the last feedback (appendix 1) mentions that he would like to have a different colour in the application. Thus, a possibility to change layout colours could be added in another version of application.