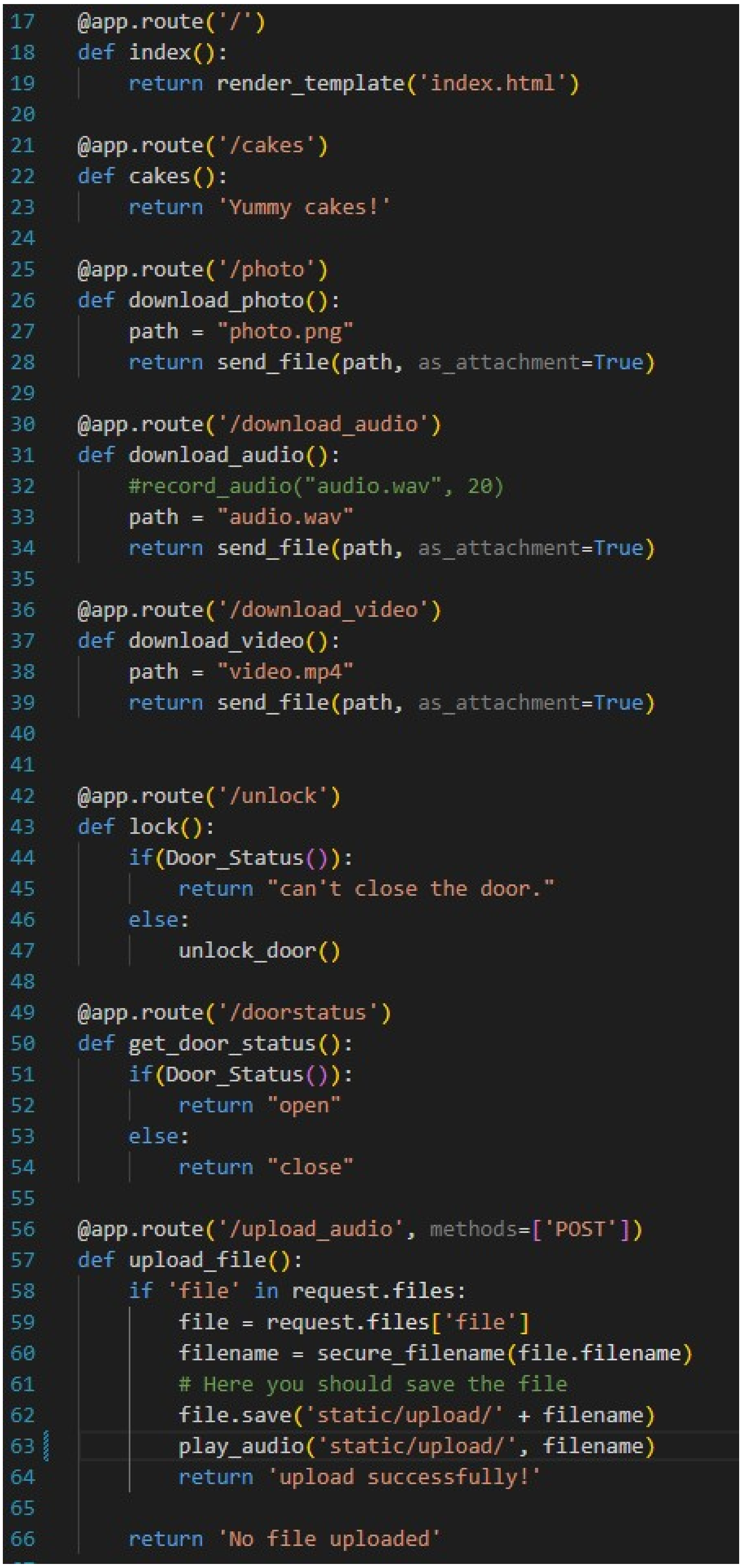
flask

Initially, we try to use database as a bridge to transmit message between pi and android. However, it fail for some reason:

1. some data like photo, video and audio need to be encoded into other format to be stored in database and need to be decode on android side after transmission.
2. when somebody press the button, the message need to be transmit to android immediately, which is not possible through database.

Therefore, in order to achieve real time message transmission, we use another way to communicate between android and pi. We use a light-weight python web frame work call flask to set up a http server on raspberry pi and provide several api for the android app to get access to data on pi as a http client. The android and pi need to be in the same LAN.

Here is all the api that we provided:

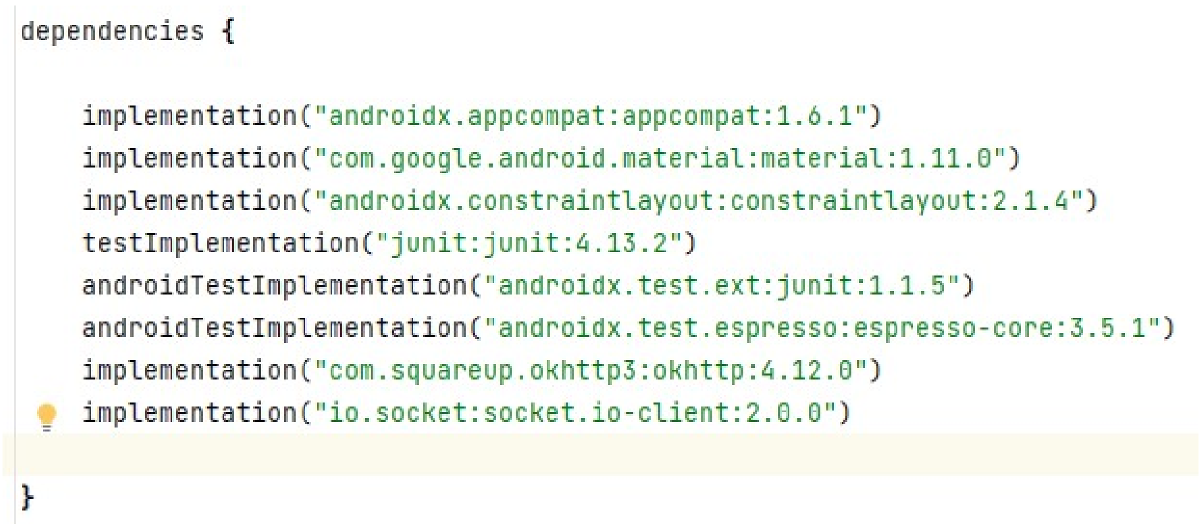


```python
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/cakes')
def cakes():
    return 'Yummy cakes!'

@app.route('/photo')
def download_photo():
    path = "photo.png"
    return send_file(path, as_attachment=True)

@app.route('/download_audio')
def download_audio():
    #record_audio("audio.wav", 20)
    path = "audio.wav"
    return send_file(path, as_attachment=True)

@app.route('/download_video')
def download_video():
    path = "video.mp4"
    return send_file(path, as_attachment=True)


@app.route('/unlock')
def lock():
    if(Door_Status()):
        return "can't close the door."
    else:
        unlock_door()

@app.route('/doorstatus')
def get_door_status():
    if(Door_Status()):
        return "open"
    else:
        return "close"

@app.route('/upload_audio', methods=['POST'])
def upload_file():
    if 'file' in request.files:
        file = request.files['file']
        filename = secure_filename(file.filename)
        # Here you should save the file
        file.save('static/upload/' + filename)
        play_audio('static/upload/', filename)
        return 'upload successfully!'

    return 'No file uploaded'
```

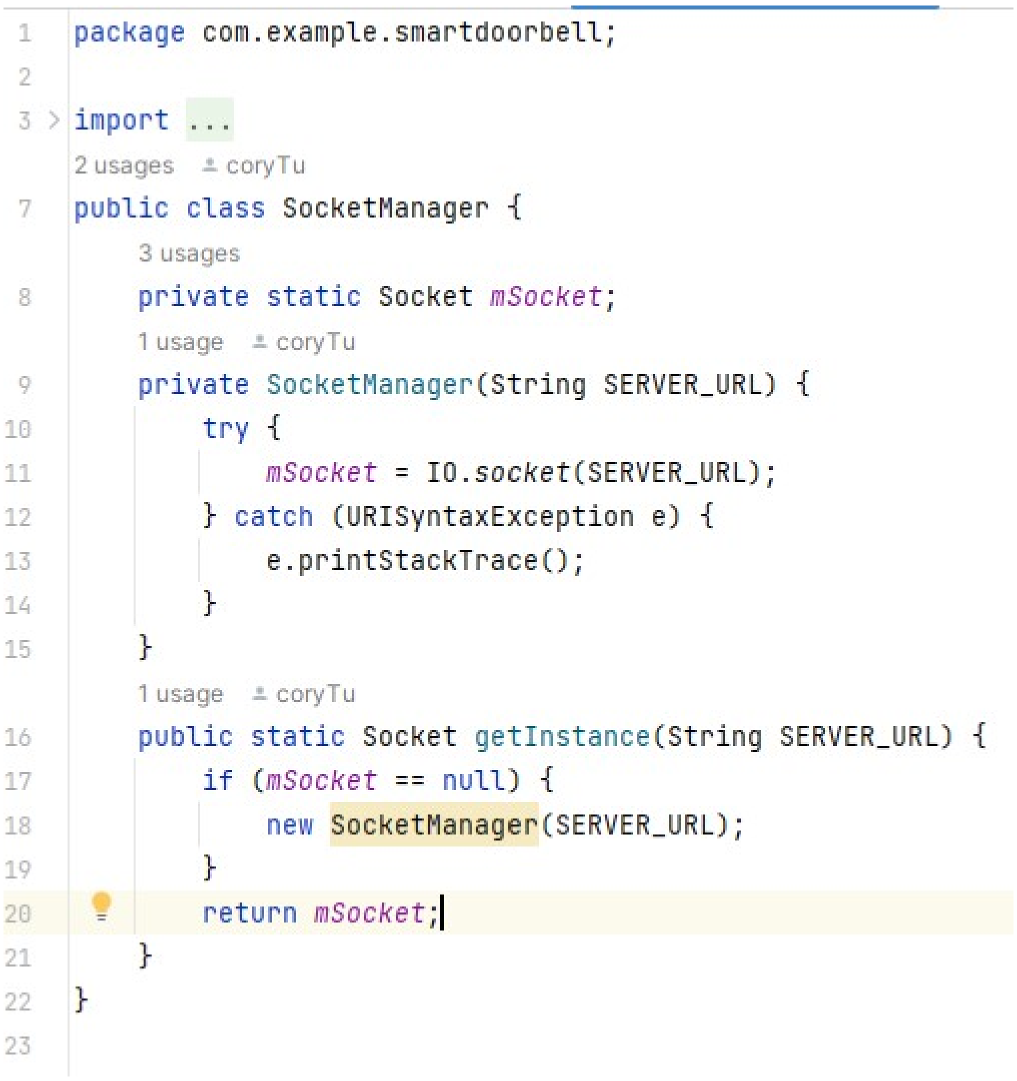
flask-socketio

With the help of flask web framework, the android app can trigger action on raspberry pi, like open the lock or record an audio, by sending a http get request to pi server. However, under some situation, we want the server to initiate connection with android client and send message to it, which is quite difficult because once one communication is over, the connection is cut off. Therefore, we use a Socket.IO integration for Flask applications call flask-socketio to set up long-life, bi-directional connection between pi and android. SocketIO is an event-driven library for real-time web applications. It enables real-time, bi-directional communication between web clients and servers.

On the android side, we use the java implementation of SocketIO to set up a socket client to receive message. Here we can see that the SocketIO dependency is added to android project and a SocketManager java class is written to handle with socket object.

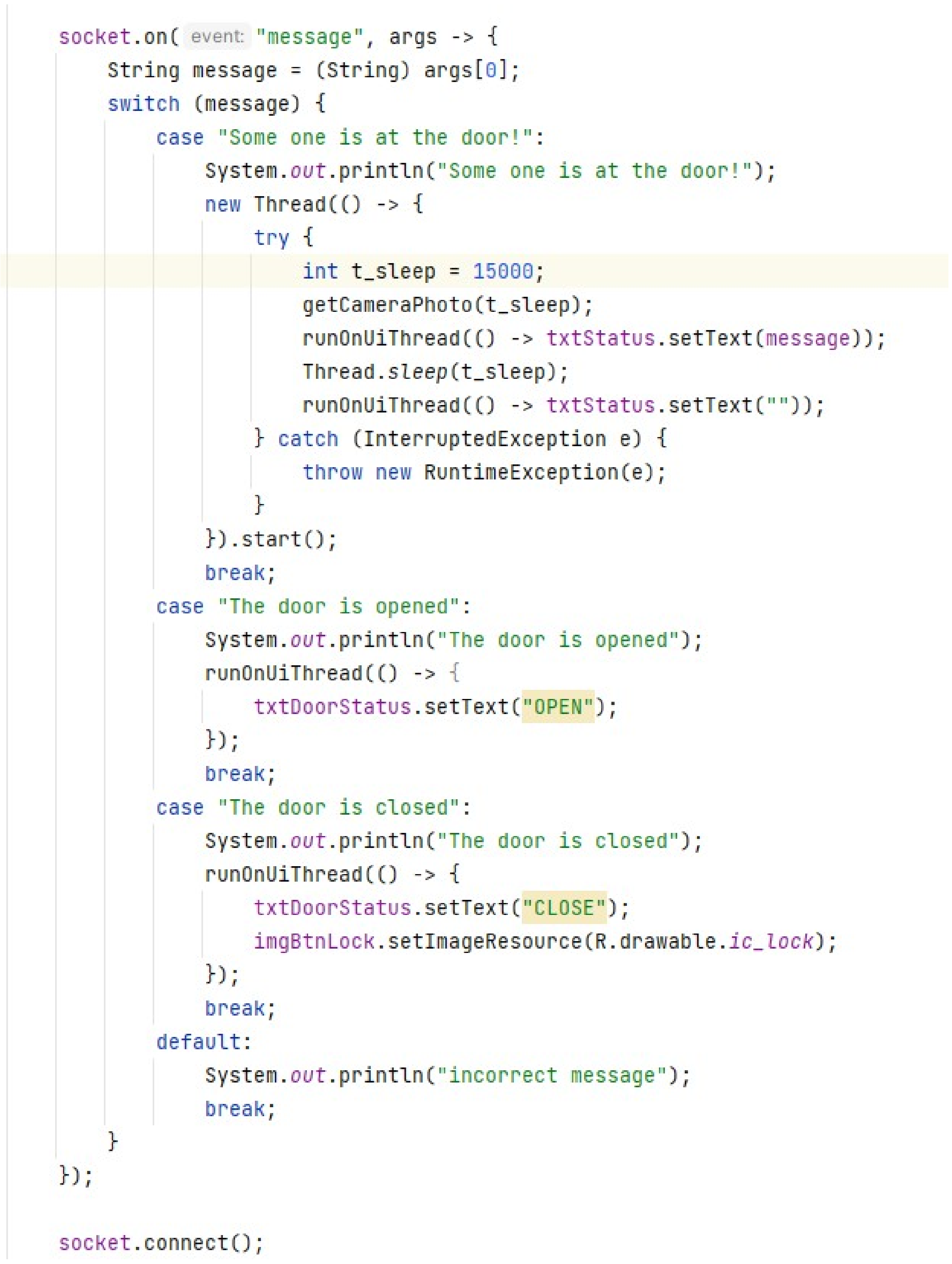


```
dependencies {

    implementation("androidx.appcompat:appcompat:1.6.1")
    implementation("com.google.android.material:material:1.11.0")
    implementation("androidx.constraintlayout:constraintlayout:2.1.4")
    testImplementation("junit:junit:4.13.2")
    androidTestImplementation("androidx.test.ext:junit:1.1.5")
    androidTestImplementation("androidx.test.espresso:espresso-core:3.5.1")
    implementation("com.squareup.okhttp3:okhttp:4.12.0")
    implementation("io.socket:socket.io-client:2.0.0")

}
```



```
package com.example.smartdoorbell;

import ...
2 usages  coryTu
public class SocketManager {
    3 usages
    private static Socket mSocket;
    1 usage  coryTu
    private SocketManager(String SERVER_URL) {
        try {
            mSocket = IO.socket(SERVER_URL);
        } catch (URISyntaxException e) {
            e.printStackTrace();
        }
    }
    1 usage  coryTu
    public static Socket getInstance(String SERVER_URL) {
        if (mSocket == null) {
            new SocketManager(SERVER_URL);
        }
        return mSocket;
    }
}

```

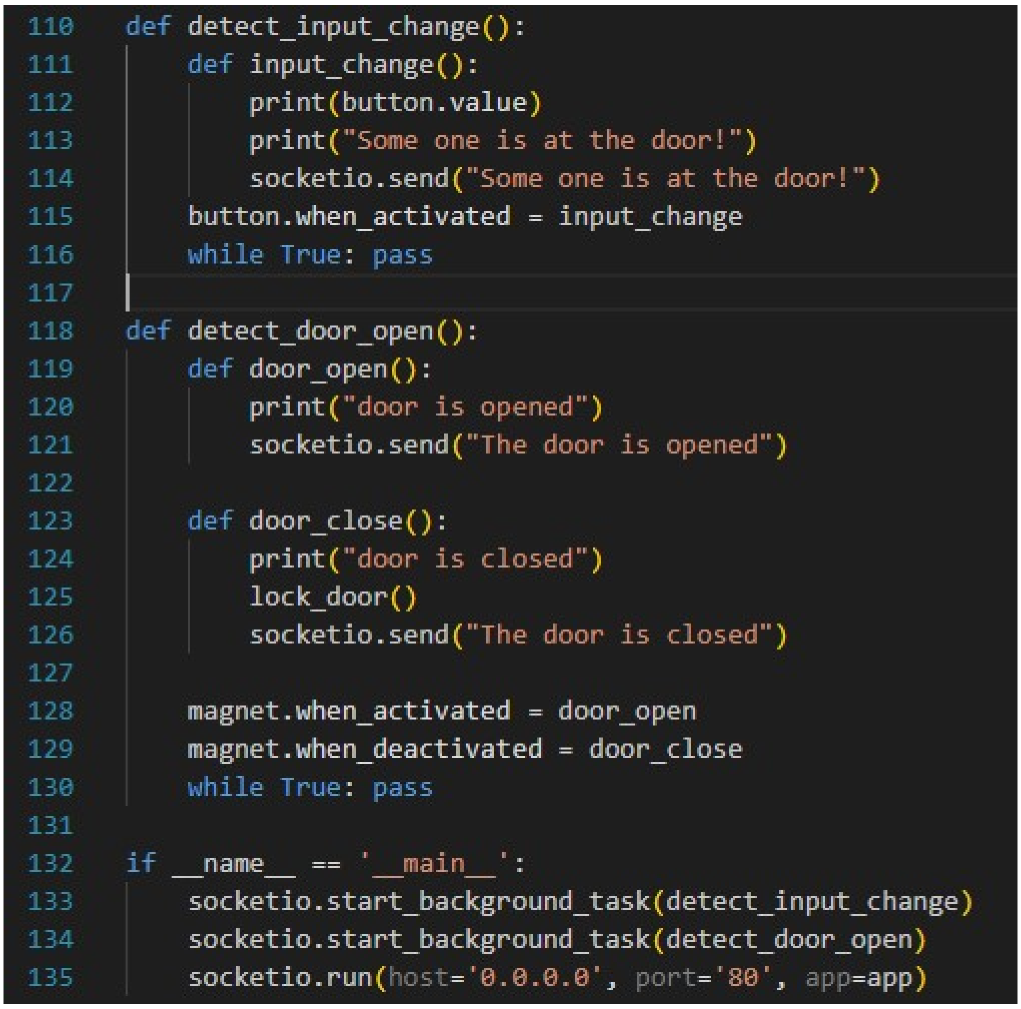
In the MainActicity, a function is written to handle with the message receiving from server.



```java
    socket.on( event: "message", args -> {
        String message = (String) args[0];
        switch (message) {
            case "Some one is at the door!":
                System.out.println("Some one is at the door!");
                new Thread(() -> {
                    try {
                        int t_sleep = 15000;
                        getCameraPhoto(t_sleep);
                        runOnUiThread(() -> txtStatus.setText(message));
                        Thread.sleep(t_sleep);
                        runOnUiThread(() -> txtStatus.setText(""));
                    } catch (InterruptedException e) {
                        throw new RuntimeException(e);
                    }
                }).start();
                break;
            case "The door is opened":
                System.out.println("The door is opened");
                runOnUiThread(() -> {
                    txtDoorStatus.setText("OPEN");
                });
                break;
            case "The door is closed":
                System.out.println("The door is closed");
                runOnUiThread(() -> {
                    txtDoorStatus.setText("CLOSE");
                    imgBtnLock.setImageResource(R.drawable.ic_lock);
                });
                break;
            default:
                System.out.println("incorrect message");
                break;
        }
    });

    socket.connect();
```

Depends on different message, the app do different operation. For example, if the message from raspberry pi is "some body is at the door!", then the app will send a request to ask pi to take a picture of the visitor and show the picture together with the message on the home page for 15 seconds.

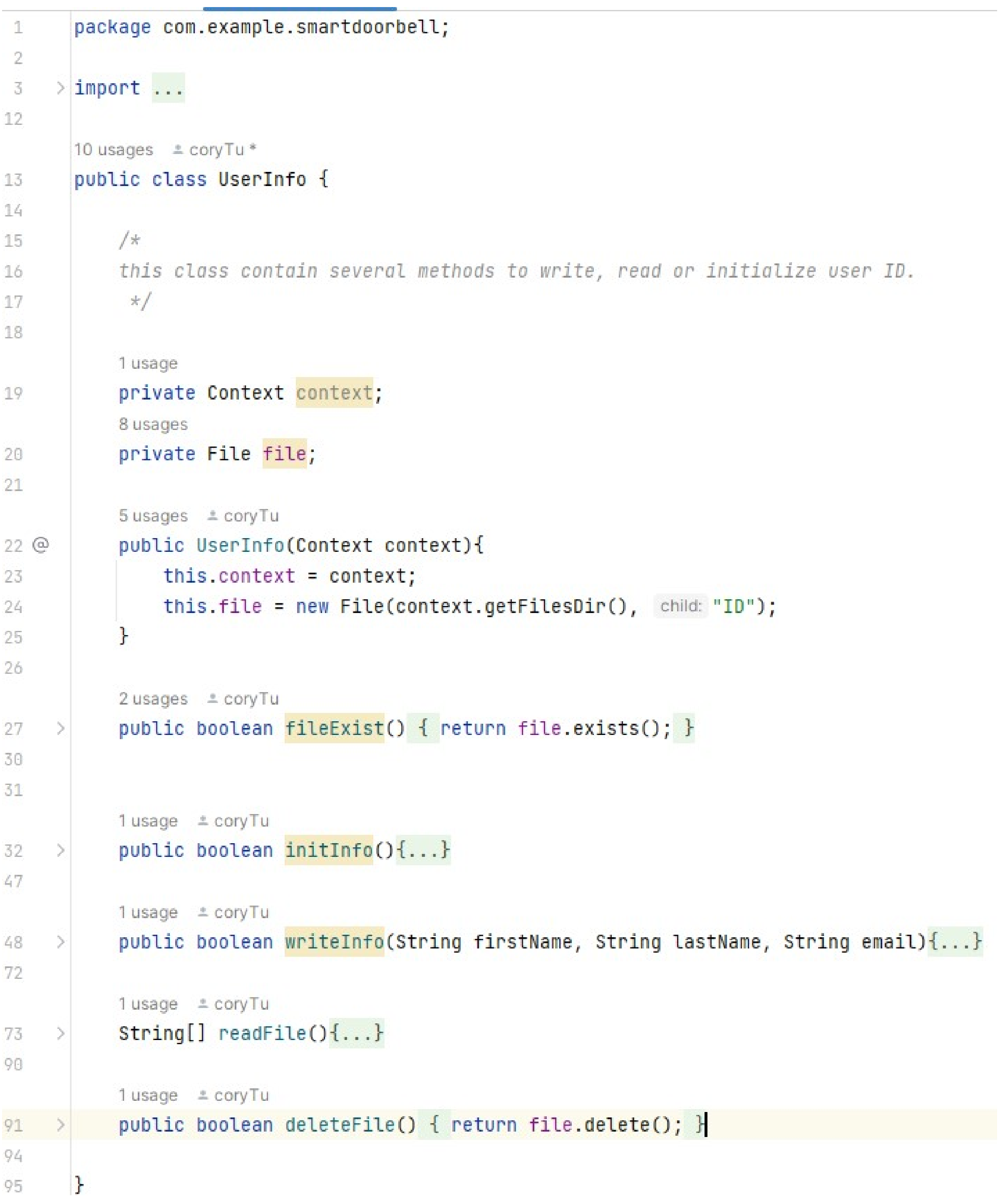
On the server side, we have two function to listen to the status of the door and button, and we let them work on another thread by adding these two function to background.



```python
def detect_input_change():
    def input_change():
        print(button.value)
        print("Some one is at the door!")
        socketio.send("Some one is at the door!")
    button.when_activated = input_change
    while True: pass

def detect_door_open():
    def door_open():
        print("door is opened")
        socketio.send("The door is opened")

    def door_close():
        print("door is closed")
        lock_door()
        socketio.send("The door is closed")

    magnet.when_activated = door_open
    magnet.when_deactivated = door_close
    while True: pass

if __name__ == '__main__':
    socketio.start_background_task(detect_input_change)
    socketio.start_background_task(detect_door_open)
    socketio.run(host='0.0.0.0', port='80', app=app)
```

User Information

On the android app, we have user management functions to check the permission of user. The android app can get access to a mySQL database, read user information or register new user. Therefore, we need a file at local to store the user information. A userInfo java class is written to manage user information. The following show all the method of a userInfo class.



```
package com.example.smartdoorbell;

import ...

10 usages  coryTu *
public class UserInfo {

    /*
    this class contain several methods to write, read or initialize user ID.
     */

    1 usage
    private Context context;
    8 usages
    private File file;

    5 usages  coryTu
    public UserInfo(Context context){
        this.context = context;
        this.file = new File(context.getFilesDir(),  child: "ID");
    }

    2 usages  coryTu
    public boolean fileExist() { return file.exists(); }


    1 usage  coryTu
    public boolean initInfo(){...}

    1 usage  coryTu
    public boolean writeInfo(String firstName, String lastName, String email){...}

    1 usage  coryTu
    String[] readFile(){...}

    1 usage  coryTu
    public boolean deleteFile() { return file.delete(); }

}
```

The UserInfo class needs to get the context of current running app, which can be get by the method getApplicationContext().