|  |  |  |
| --- | --- | --- |
|  | F:\school\fourth year\second term\graduation-project\docs\SCE Logo 20.png |  |

Control Home Devices using Android Application 2022

**Graduation Project Documentation**

****

**Control Home Devices using Android Application**

**A Project Submitted in partial fulfilment of the requirements for the Degree of Bachelor of Science in Systems and Computers Engineering**

**Submitted By**

**Youssef Mohamed Agagg 404139**

**Youssef Omar Abd-Elshafy 404137**

**Supervised by**

**Dr. Nagdy Mahmoud**

**2022**

|  |  |  |
| --- | --- | --- |
| **Examiner Committee** | | |
| **Name** | **Rule** | **Signature** |
| Prof. | President |  |
| Dr. | Supervisor |  |
| Dr. | Member |  |

ABSTRACT

This is the era of advanced computing technology. Almost all of the works we are doing by the help of automation system or computer controlled. Now peoples are moving towards smart system. As the technology is upgrading day by day the Internet has become an important part of life, and IoT is the newest and emerging internet technology. Internet of things plays an important role in human life as well as in the educational field because they are able to provide information and complete the given tasks while we are busy doing some other work. In this paper, a prototype and implementation of Control Home Devices with Wi-Fi technology are demonstrated. ESP8266 is used as a Wi-Fi technology. The proposed system consist of a hardware interface and software interface. In the hardware interface, the integration of ESP8266 Wi-Fi technology for controlling home appliances is manifested, and an application is provided for controlling to multiple users of home, with smart phones, tablets. This system is one of the best methods for controlling home devices with ease with multiple users. In this project user can easily turn on/off the lights of his/her home by using the android application and he/she can also check the current status of lights and other devices.

KEYWORDS Android; IOT; NodeMCU; Spring Boot

**ACKNOWLEDGEMENTS**

Firstly we are thankful to ALLAH ALMIGHTY who blessed us with knowledge to complete this project. We truly acknowledge the cooperation and help make by Name of Dr. Nagdy Mahmoud, for his help and guidance throughout this project. He has been a constant source of guidance throughout the course of this project. He helped in requirement gathering and specification.

TABLE OF CONTENTS

[ABSTRACT ii](#_Toc109503524)

[ACKNOWLEDGEMENTS iii](#_Toc109503525)

[TABLE OF CONTENTS iv](#_Toc109503526)

[LIST OF TABLES vi](#_Toc109503527)

[LIST OF FIGURES vii](#_Toc109503528)

[CHAPTER 1 INTRODUCTION 1](#_Toc109503529)

[1.1 Overview 1](#_Toc109503530)

[1.2 Purpose 1](#_Toc109503531)

[Chapter 2 Software Requirement Specification 2](#_Toc109503532)

[2.1 Functional Requirements 2](#_Toc109503533)

[2.2 Description of each Requirement: 2](#_Toc109503534)

[2.2.1 Authentication 2](#_Toc109503535)

[2.2.2 Room Selection 3](#_Toc109503536)

[2.2.3 Monitoring home devices 3](#_Toc109503537)

[2.2.4 Turn off/on lights 4](#_Toc109503538)

[2.2.5 Open/closed garage door 5](#_Toc109503539)

[2.3 System Requirements: 5](#_Toc109503540)

[2.3.1 Hardware: 5](#_Toc109503541)

[2.3.1.1 Hardware Component: 5](#_Toc109503542)

[2.3.1.2 Tools used to program the ESP3266 NodeMCU: 7](#_Toc109503543)

[2.3.2 Software: 7](#_Toc109503544)

[2.3.2.1 Server: 8](#_Toc109503545)

[2.3.2.2 Android application 8](#_Toc109503546)

[CHAPTER 3 SYSTEM DESIGN 9](#_Toc109503547)

[3.1 Use case diagram 9](#_Toc109503548)

[3.2 Use Case Description 10](#_Toc109503549)

[3.2.1 Authentication 10](#_Toc109503550)

[3.2.2 Room selection 10](#_Toc109503551)

[3.2.3 Monitoring home devices 10](#_Toc109503552)

[3.2.4 Control Devices 11](#_Toc109503553)

[3.3 Sequence diagram 12](#_Toc109503554)

[3.4 Flow Chart 13](#_Toc109503555)

[3.4.1 Flow Chart of the Android app 13](#_Toc109503556)

[3.4.2 Flow chart of the server 14](#_Toc109503557)

[3.5 Hardware Circuit diagram 15](#_Toc109503558)

[CHAPTER 4 Implementation 16](#_Toc109503559)

[4.1 Hardware 16](#_Toc109503560)

[4.2 Software 18](#_Toc109503561)

[4.2.1 Server 18](#_Toc109503562)

[4.2.1.1 Server code 18](#_Toc109503563)

[4.2.1.2 ESP8266 code 22](#_Toc109503564)

[4.2.1.2.1 Code for ESP8266 that control the living room devices 22](#_Toc109503565)

[4.2.1.2.2 Code for ESP8266 that control the bed room devices and the garage door 26](#_Toc109503566)

[4.2.1.3 Android application code 31](#_Toc109503567)

[CHAPTER 5 Results and discussion 36](#_Toc109503568)

[5.1 Results 36](#_Toc109503569)

[5.1.1 Home model 36](#_Toc109503570)

[5.1.2 Android application 37](#_Toc109503571)

[5.2 Discussion 42](#_Toc109503572)

[5.3 Conclusion 42](#_Toc109503573)

LIST OF TABLES

[2. 1 Define Authentication requirement 2](#_Toc109503594)

[2. 2 Room Selection Requirement 3](#_Toc109503595)

[2. 3 Monitoring home devices Requirement 4](#_Toc109503596)

[2. 4 Turn off/of Lights Requirement 4](#_Toc109503597)

[2. 5 Open/closed garage door Requirement 5](#_Toc109503598)

[3. 1 “Authentication” Use Case detail 10](#_Toc109503601)

[3. 2 “Room selection” Use Case detail 10](#_Toc109503602)

[3. 3 “monitoring home devices” Use Case detail 11](#_Toc109503603)

[3. 4 “control devices” Use Case detail 11](#_Toc109503604)

LIST OF FIGURES

[3. 1 Use case diagram 9](#_Toc109503616)

[3. 2 Sequence diagram 12](#_Toc109503617)

[3. 3 Flow Chart for the Android app 13](#_Toc109503618)

[3. 4 Flow chart of the server 14](#_Toc109503619)

[3. 5 Circuit diagram for living room 15](#_Toc109503620)

[3. 6 Circuit diagram for bed room and garage door 15](#_Toc109503621)

[4. 1 Final Hardware Enclosure 17](#_Toc109503622)

[4. 2 web Server application class diagram 18](#_Toc109503623)

[5. 1 home model 36](#_Toc109503625)

[5. 2 living room and bed room leds and buttons 36](#_Toc109503626)

[5. 3 garage door 37](#_Toc109503627)

[5. 4 android app login screen 37](#_Toc109503628)

[5. 5 android app main screen 38](#_Toc109503629)

[5. 6 android app living room screen 39](#_Toc109503630)

[5. 7 android app bed room screen 40](#_Toc109503631)

[5. 8 android app garage screen 41](#_Toc109503632)

# CHAPTER 1 INTRODUCTION

## Overview

The title of project is "Control Home Devices using Android Application". It is an IOT project that make you control home devices through Android Application. In this project a user can control his/her home devices by using his android mobile phone. This project is a combination of a software and hardware. In hardware there is an ESP8266 board, the ESP8266 chip is a powerful and cheap microcontroller with an on-board Wi-Fi connection. It is also very easy to use, thanks to the compatibility with the Arduino IDE. Therefore, it's just the perfect chip to build the Internet of Things (IoT) projects. In software there is a web server implemented by java Spring Boot. Spring Boot makes it easy to create stand-alone, production-grade Spring based web Applications that you can "just run", it provides Embedded Tomcat server and automatically configure Spring and 3rd party libraries whenever possible therefore, it's just the perfect tool to build the server .the android application implemented by dart and flutter. The android application can communicate with the ESP8266 to control the devices in the home through the web server. The application can monitor the current status of the home devices.

## Purpose

The main objective of the project is to improve the quality of life and convenience in the home. By letting the user control the home devices from anywhere. It is an energy efficient to user, he/she can check the status of the devices and turn the devices on or off.

# Chapter 2 Software Requirement Specification

## Functional Requirements

* Authentication
* Room selection
* Room devices status
* Monitoring home devices
* Turn off/on lights
* Open/closed garage door

## Description of each Requirement:

### Authentication

|  |  |
| --- | --- |
| **Requirement no. :** | Req. 01 |
| **Requirement Type:** | Functional |
| **Description:** | Authentication of user |
| **Rationale:** | The Requirement is important because the user has to authenticate himself/herself before using application. |
| **Source Person:** | User |
| **Fit Criteria:** | Any registered android mobile user can login. |
| **Customer Satisfaction:** | High |
| **Customer Dissatisfaction:** | Low |
| **Dependencies:** | N/A |

Table 2. 1 Define Authentication requirement

### Room Selection

|  |  |
| --- | --- |
| **Requirement no. :** | Req. 02 |
| **Requirement Type:** | Functional |
| **Description:** | After successful login user navigate through list of rooms. |
| **Rationale:** | User can select the room from a give list of rooms on main screan. |
| **Source Person:** | User |
| **Fit Criteria:** | Only logein user can select the room. |
| **Customer Satisfaction:** | N/A |
| **Customer Dissatisfaction:** | N/A |
| **Dependencies:** | N/A |

Table 2. 2 Room Selection Requirement

### Monitoring home devices

|  |  |
| --- | --- |
| **Requirement no. :** | Req. 03 |
| **Requirement Type:** | Functional |
| **Description:** | Automatically show user the devices status on selecting room. |
| **Rationale:** | The Requirement is important because the user has to be aware of the current status of devices. |
| **Source Person:** | Any user |
| **Fit Criteria:** | User has to tap on the room for Monitoring the devices. |
| **Customer Satisfaction:** | N/A |
| **Customer Dissatisfaction:** | N/A |
| **Dependencies:** | N/A |

Table 2. 3 Monitoring home devices Requirement

### Turn off/on lights

|  |  |
| --- | --- |
| **Requirement no. :** | Req. 04 |
| **Requirement Type:** | Functional |
| **Description:** | User has to tap on the buttons to turn on/off the lights. |
| **Rationale:** | The Requirement is important because user only can turn on/off by tapping on the button. |
| **Source Person:** | Any user |
| **Fit Criteria:** | When web server is connected with android application. |
| **Customer Satisfaction:** | N/A |
| **Customer Dissatisfaction:** | N/A |
| **Dependencies:** | N/A |

Table 2. 4 Turn off/of Lights Requirement

### Open/closed garage door

|  |  |
| --- | --- |
| **Requirement no. :** | Req. 05 |
| **Requirement Type:** | Functional |
| **Description:** | User has to tap on the a button to open or close the the door. |
| **Rationale:** | The Requirement is important because user only can open or close the door by tapping on the button. |
| **Source Person:** | Any user |
| **Fit Criteria:** | When web server is connected with android application. |
| **Customer Satisfaction:** | N/A |
| **Customer Dissatisfaction:** | N/A |
| **Dependencies:** | N/A |

Table 2. 5 Open/closed garage door Requirement

## System Requirements:

System based on two basic things The Hardware and The Software.

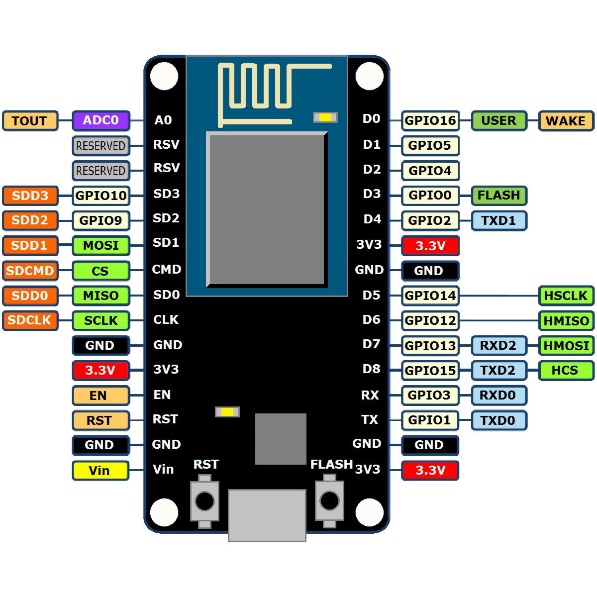
### Hardware:

#### Hardware Component:

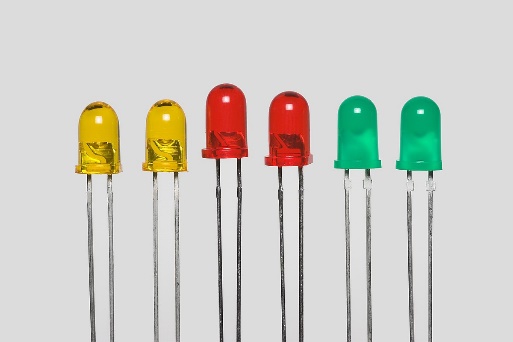
1. ESP8266 NodeMCU**:**

It is an open source firmware for which open source prototyping board designs are available. The name "NodeMCU" combines "node" and "MCU" (micro-controller unit).The firmware uses the Lua scripting language. The firmware is based on the eLua project, and built on the Espressif Non-OS SDK for ESP8266. It uses many open source projects, such as lua-cjson and SPIFFS. Due to resource constraints, users need to select the modules relevant for their project and build a firmware tailored to their needs. Support for the 32-bit ESP32 has also been implemented. The prototyping hardware typically used is a circuit board functioning as a dual in-line package (DIP) which integrates a USB controller with a smaller surface-mounted board containing the MCU and antenna. The choice of the DIP format allows for easy prototyping on breadboards. The design was initially based on the ESP-12 module of the ESP8266, which is a Wi-Fi SoC integrated with a Tensilica Xtensa LX106 core, widely used in IoT applications.

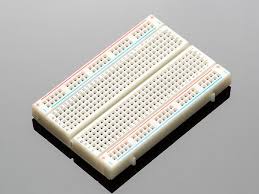
* Node MCU PINS:



1. Leds:



1. BreadBoard**:**

****

1. Jumper Wires**:**



1. On/off switches:



1. Resistors:

****

1. 4 channel Relays :



1. Servo motor sg90:



#### Tools used to program the ESP3266 NodeMCU:

1. Arduino IDE
2. Language: C/C++

### Software:

The software architecture consists of:

#### Server:

In order to make the ESP8266 send data to the mobile app and receive data from the mobile app, a server is required. The purpose of the server is to relay information between devices connected to the server. The server needs to be capable of connecting to a variety of ESP8266 and mobile devices securely and efficiently.

1. **Tools and Technologies used to build the server:**
2. Java 11
3. Spring Boot

#### Android application

The application program is developed with flutter using Android Studio and it provides a user interface for home Devices that will be controled.

1. **Tools and Technologies used to build the Android app:**
2. Latest Android Studio
3. Emulator and Android actual Device
4. Language: dart

# CHAPTER 3 SYSTEM DESIGN

## Use case diagram

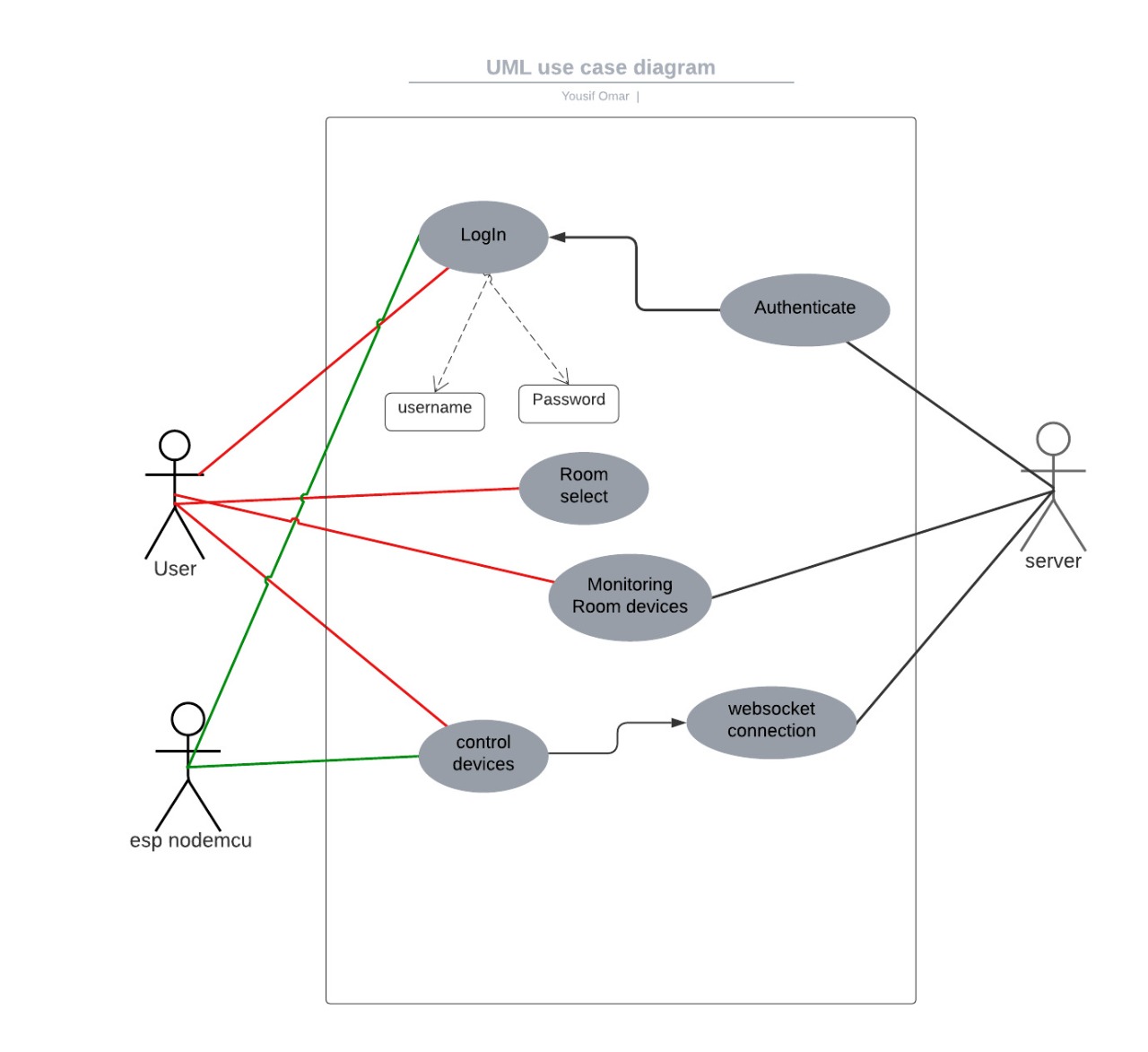


Figure 3. 1 Use case diagram

## Use Case Description

### Authentication

|  |  |
| --- | --- |
| **Name** | Authentication |
| **Actor** | User/esp |
| **Description** | Describe the proses that user can login with username and password |
| **Successful Completion** | users can access into the system |
| **Alternative** | none |
| **Precondition** | Username and password |
| **Post condition** | System is accessible to the users |

Table 3. 1 “Authentication” Use Case detail

### Room selection

|  |  |
| --- | --- |
| **Name** | Room select |
| **Actor** | User |
| **Description** | Only logein user can select room . |
| **Successful Completion** | The user will successfully enter the room |
| **Alternative** | none |
| **Precondition** | The user must be logged in |
| **Post condition** | Successful room select |

Table 3. 2 “Room selection” Use Case detail

### Monitoring home devices

|  |  |
| --- | --- |
| **Name** | Monitoring home devices |
| **Actor** | User/server |
| **Description** | The user can monitoring the status of the devices in any room by getting the condition from the server |
| **Successful Completion** | users can see the condition of the devices in any room |
| **Alternative** | none |
| **Precondition** | The user must be logged in , select a room and Network Connectivity |
| **Post condition** | Successful monitoring devices |

Table 3. 3 “monitoring home devices” Use Case detail

### Control Devices

|  |  |
| --- | --- |
| **Name** | Control devices |
| **Actor** | User/esp |
| **Description** | The user can turn on or off any device |
| **Successful Completion** | When the user tap on button light will turn on/off. |
| **Alternative** | none |
| **Precondition** | The user must be logged in , select a room and Network Connectivity |
| **Post condition** | Successful turn on/off devices |

Table 3. 4 “control devices” Use Case detail

## Sequence diagram

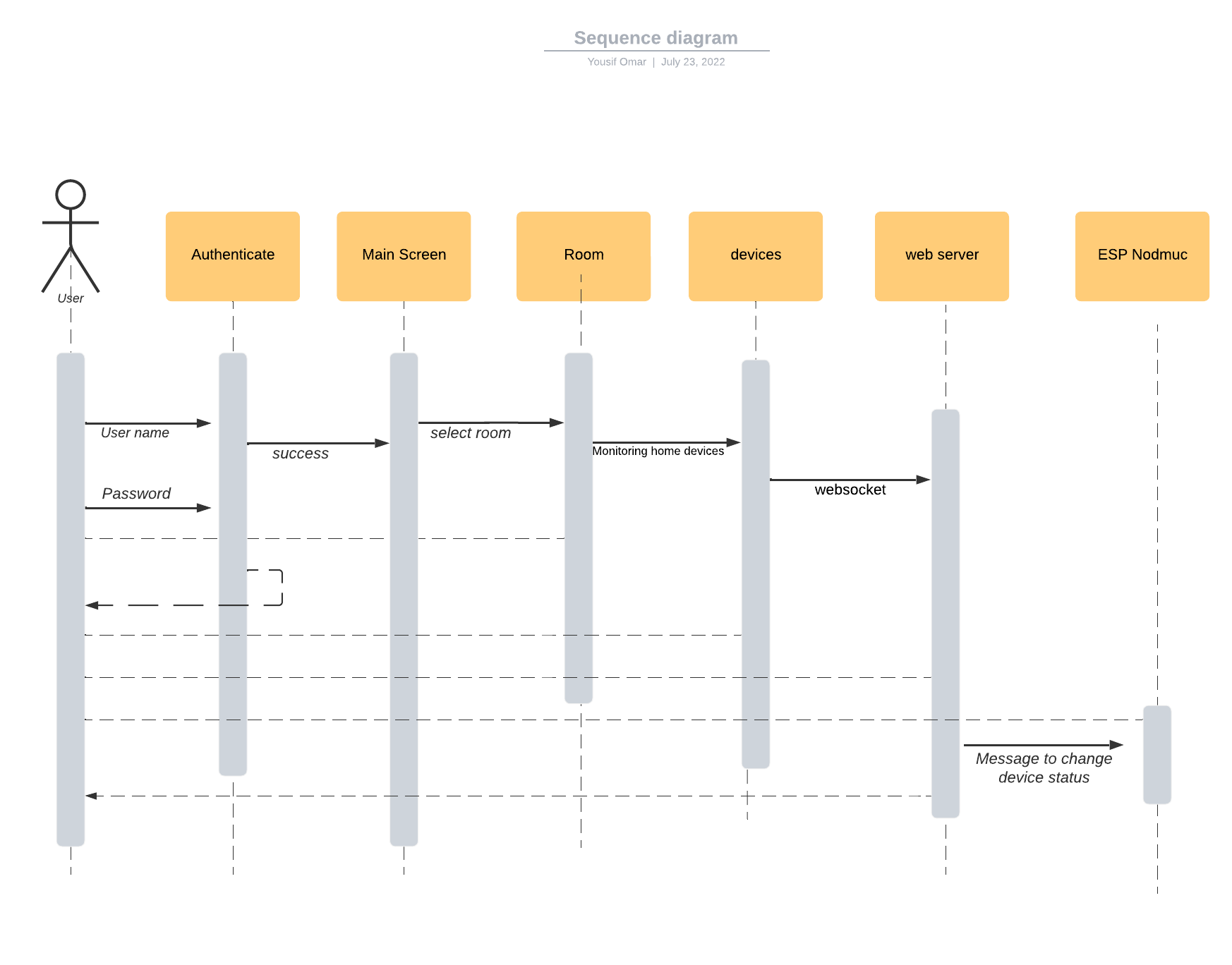


Figure 3. 2 Sequence diagram

## Flow Chart

### Flow Chart of the Android app

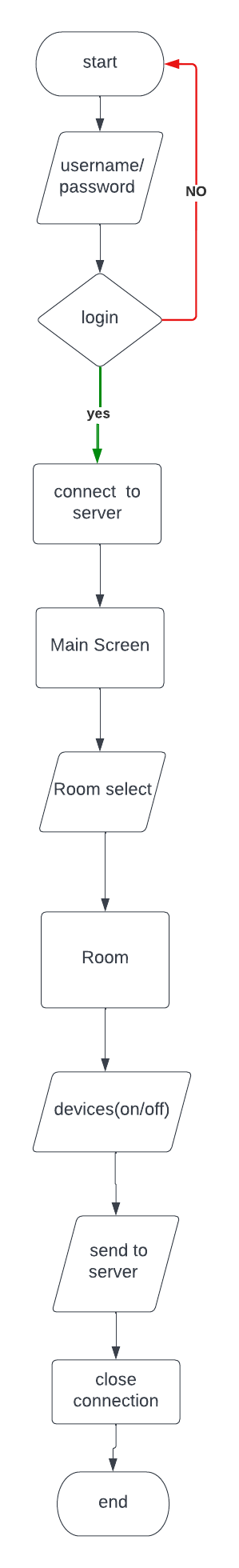


Figure 3. 3 Flow Chart for the Android app

**How it works**:

User will access the Main screen of application after login the user redirect to the main screen. The Main screen contains the list of rooms of the house. For this project only living room, bed room and garage door is functional due to hardware cost. When living room is accessed by user then there are some devices. And when the user tap on the button of any device then a message request goes to the Web server and device turn on or off according to the value.

### Flow chart of the server

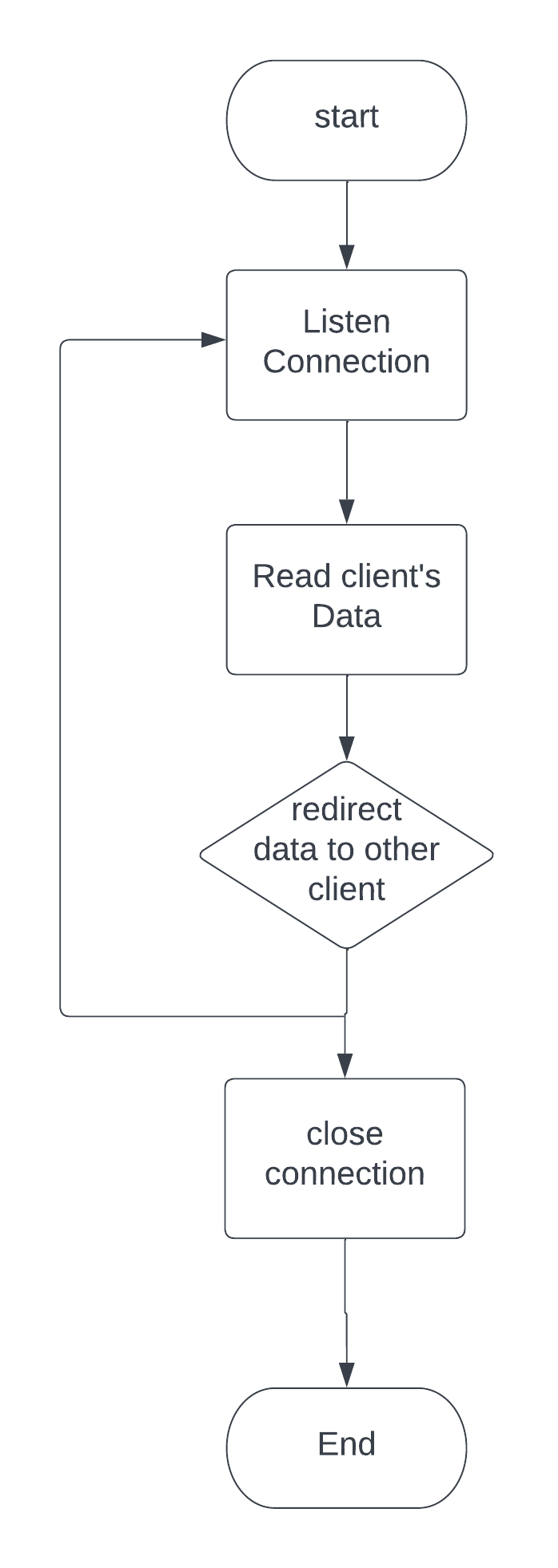


Figure 3. 4 Flow chart of the server

**How it works:**

The server is started and listen to any request connection and wait to receive data from any client and then redirect this data to other client.

## Hardware Circuit diagram

We will connect the circuit as in the blow diagram to control the living room lights

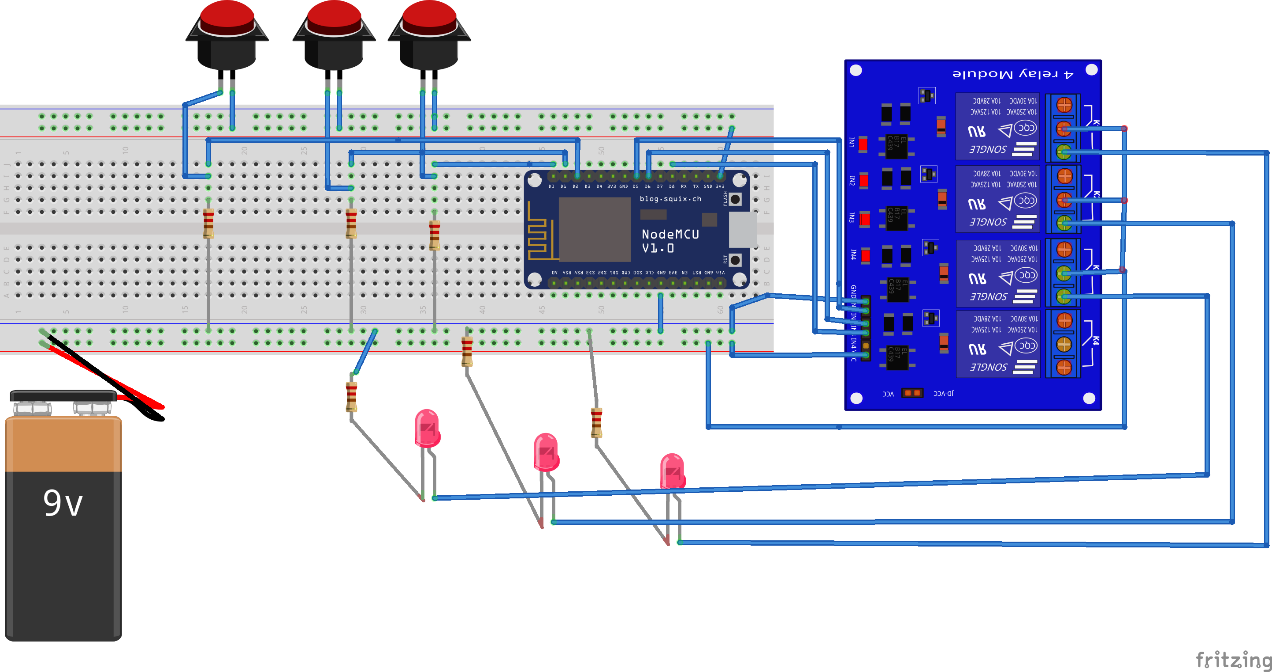


Figure 3. 5 Circuit diagram for living room

We will connect the circuit as in the blow diagram to control the bed room lights and garage door

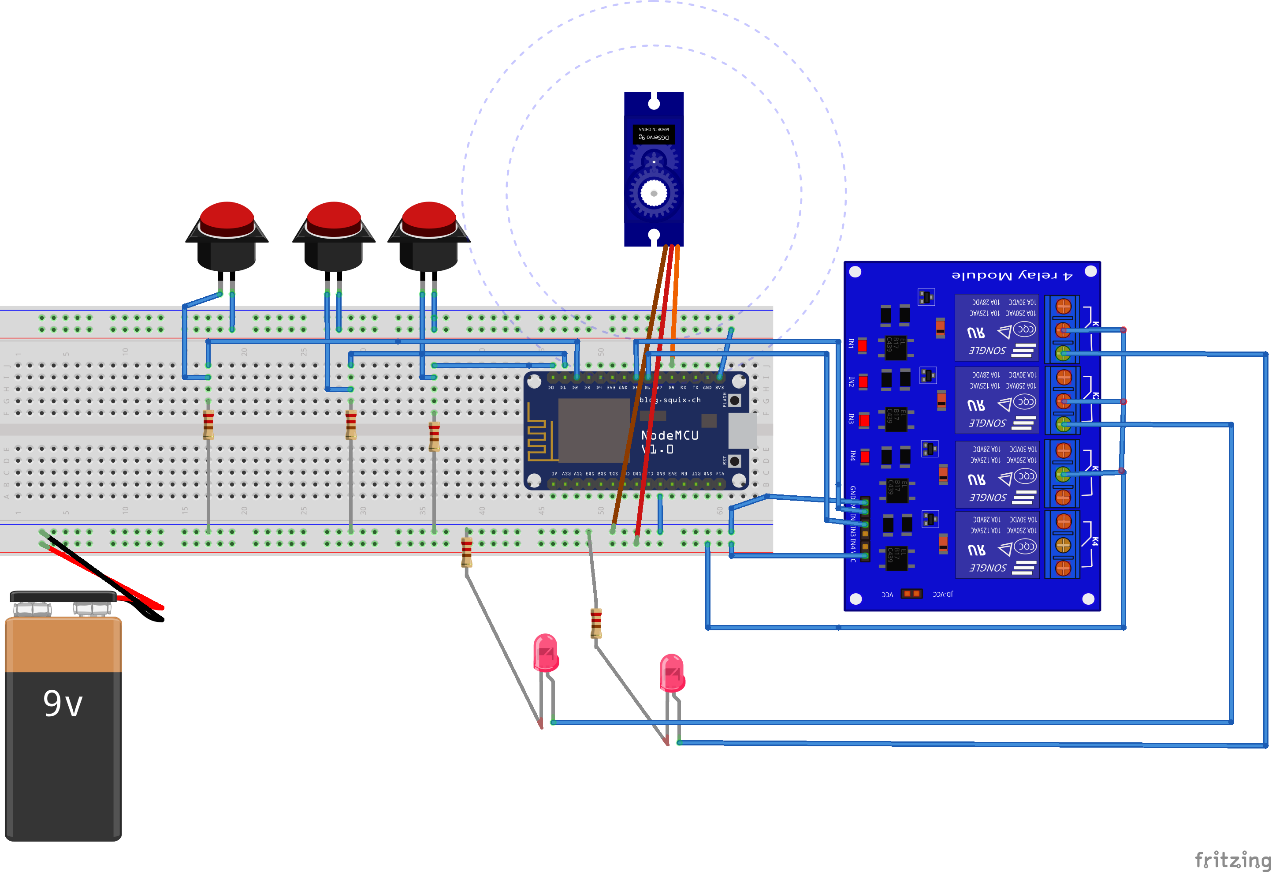
****

Figure 3. 6 Circuit diagram for bed room and garage door

### 

# CHAPTER 4 Implementation

After the completion of Logical Design and Physical Design the implementation phase was completed. For the implementation the software requires a sufficient time as different resources have to be managed for implementation. When the detailed design of the proposed system is available, the coding phase starts. In the coding phase the design is converted into actual code. This is accomplished by using programming tools. The selection of programming tools is a critical step. Each tool has its own merits and demerits but the selection of the tool for a particular project depends purely on the specific requirements of the proposed system.

## Hardware

The different components are connected to each other as describe in the circuit diagram.

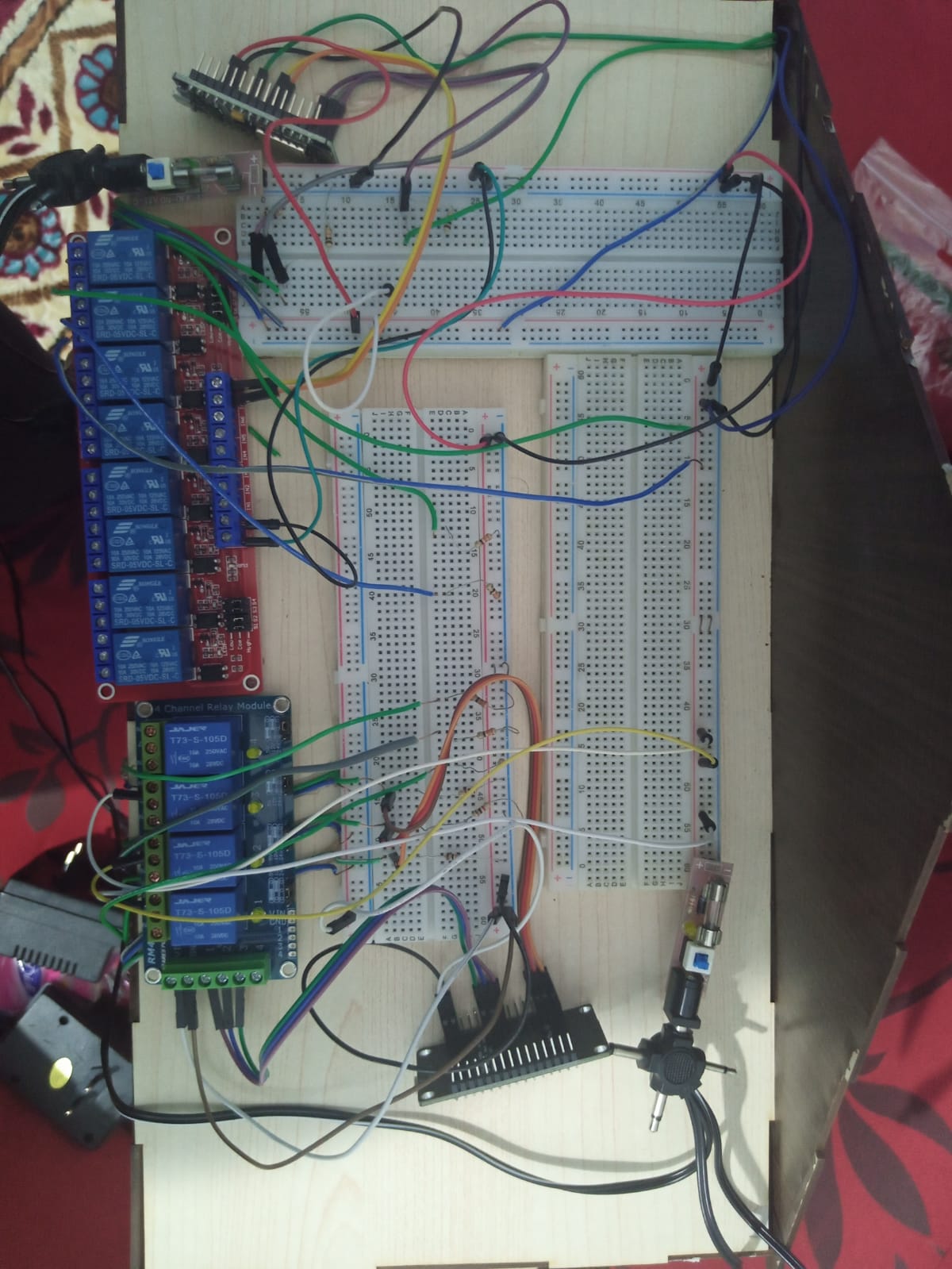


Figure 4. 1 Final Hardware Enclosure

## Software

The software consists of three components: the client application, the server and the application on the ESP8266. This section will describe the communication between the different components and how the components are built.

### Server

The server handles communication to and from both the android app and the ESP8266.

The android app and the ESP8266 communicate with the server by sending and receiving messages throw a websocket connection . These messages are retrieved as an text. The message should be in specific format so that the server knows how to process the message, and fields for other information needed.

The format used that the android app and the ESP8266 can understand is like : “roomName:deviceName:status” ex:”livingRoom:lamp1:1”

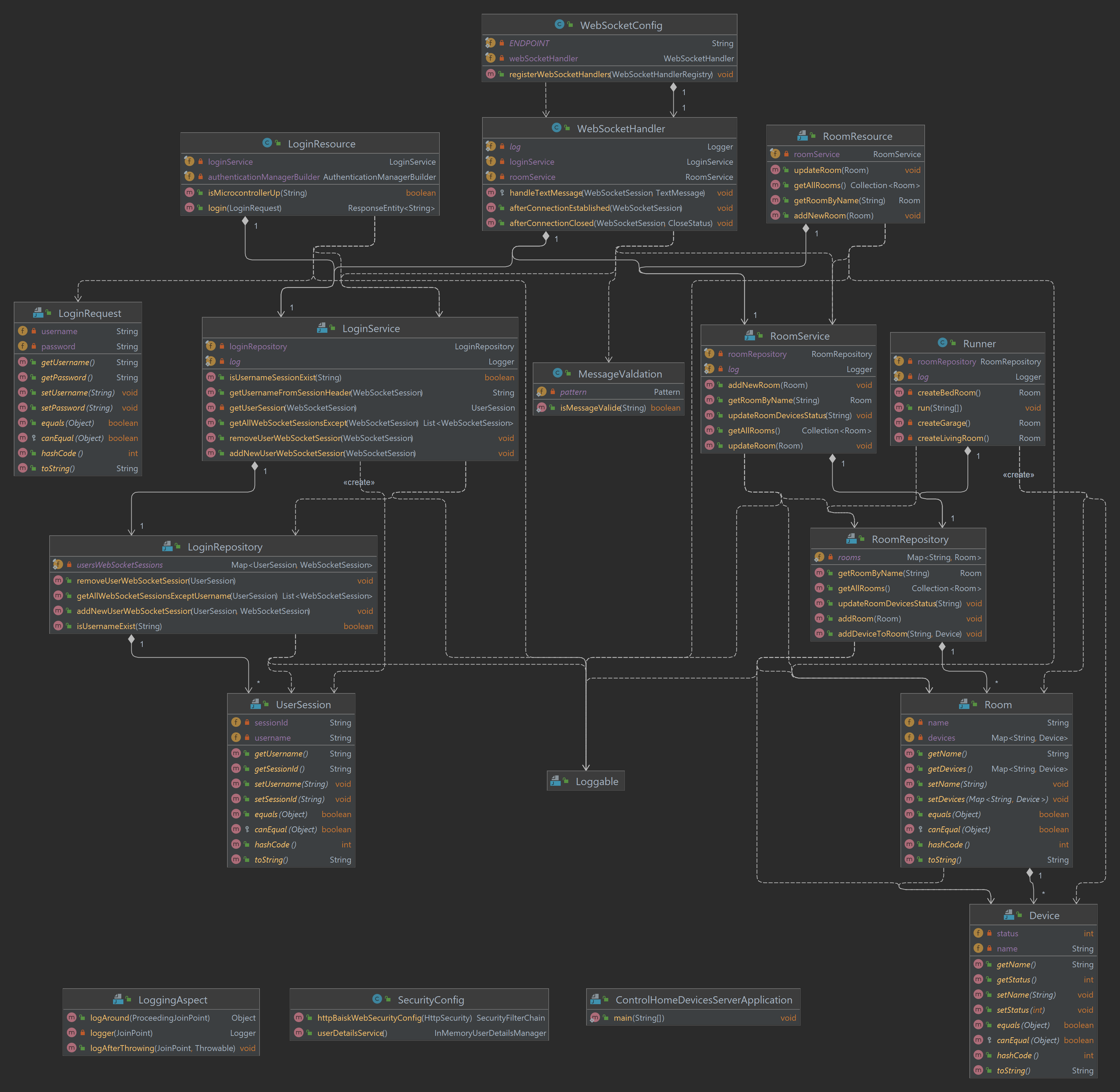


Figure 4. 2 web Server application class diagram

#### Server code

**WebSocketConfig.java**

Configure the websocket and the endpoint that the android app and the ESP8266 will connect to

@Configuration  
@EnableWebSocket  
@RequiredArgsConstructor  
public class WebSocketConfig implements WebSocketConfigurer {  
 private final static String *ENDPOINT* = "/connect";  
 private final WebSocketHandler webSocketHandler;  
  
 @Override  
 public void registerWebSocketHandlers(WebSocketHandlerRegistry webSocketHandlerRegistry) {  
 webSocketHandlerRegistry.addHandler(webSocketHandler, *ENDPOINT*)  
 .setAllowedOrigins("\*");  
 }  
  
  
  
}

**WebSocketHandler.java**

Handling the connection with tha android app and the ESP8266 and send and receive messages from the clients

@RequiredArgsConstructor  
@Slf4j  
@Component  
public class WebSocketHandler extends TextWebSocketHandler {  
  
 private final LoginService loginService;  
 private final RoomService roomService;  
  
 @Override  
 public void afterConnectionEstablished(WebSocketSession session){  
 loginService.addNewUserWebSocketSession(session);  
 *log*.debug("connected : {}" , session);  
 *log*.debug("connected : {}",loginService.getUsernameFromSessionHeader(session));  
 }  
  
 @Override  
 protected void handleTextMessage(WebSocketSession session, TextMessage message) throws Exception {  
 String username = loginService.getUsernameFromSessionHeader(session);  
 *log*.debug("Send from: {}, message {}", username, message);  
 if (MessageValdation.*isMessageValide*(message.getPayload())) {  
 roomService.updateRoomDevicesStatus(message.getPayload());  
 *log*.debug("rooms {}", roomService.getAllRooms());  
 for (WebSocketSession webSocketSession : loginService.getAllWebSocketSessionsExcept(  
 session)) {  
 webSocketSession.sendMessage(message);  
 }  
 }  
 }  
  
 @Override  
 public void afterConnectionClosed(WebSocketSession session, CloseStatus status) {  
 *log*.debug("remove : {}", session);  
 loginService.removeUserWebSocketSession(session);  
 }  
}

**SecurityConfig.java**

Configure spring security to secure the endpoints of the server and to authenticate the user

EnableWebSecurity  
public class SecurityConfig {  
  
 @Bean  
 public SecurityFilterChain httpBaiskWebSecurityConfig(HttpSecurity http) throws Exception {  
 http.csrf().disable();  
 http.httpBasic().and()  
 .authorizeRequests()  
 .antMatchers("/login").permitAll()  
 .anyRequest()  
 .authenticated();  
 return http.build();  
  
 }  
  
 @Bean  
 @SuppressWarnings( "deprecation" )  
 public InMemoryUserDetailsManager userDetailsService() {  
 UserDetails mobileClient = User.*withDefaultPasswordEncoder*()  
 .username("android")  
 .password("password")  
 .roles("USER")  
 .build();  
 UserDetails espClient1 = User.*withDefaultPasswordEncoder*()  
 .username("esp1")  
 .password("password")  
 .roles("USER")  
 .build();  
 UserDetails espClient2 = User.*withDefaultPasswordEncoder*()  
 .username("esp2")  
 .password("password")  
 .roles("USER")  
 .build();  
 return new InMemoryUserDetailsManager(List.*of*(mobileClient,espClient1,espClient2));  
 }  
  
}

**LoginResource.java**

A rest endpoint to Authenticate the user by receiving the username and password

@RestController  
@RequiredArgsConstructor  
public class LoginResource {  
 private final LoginService loginService;  
  
 private final AuthenticationManagerBuilder authenticationManagerBuilder;  
 @Loggable  
 @GetMapping("/isEspUp")  
 public boolean isMicrocontrollerUp(@RequestParam String esp){  
 return loginService.isUsernameSessionExist(esp);  
 }  
 @Loggable  
 @PostMapping("/login")  
 public ResponseEntity<String> login(@RequestBody LoginRequest login){  
 var auth=new UsernamePasswordAuthenticationToken(login.getUsername(),login.getPassword());  
 authenticationManagerBuilder.getObject().authenticate(auth);  
 return ResponseEntity.*ok*("loged in");  
 }  
  
}

**RoomResource.java**

A rest endpoint to get the status of the room and to create, update and delete a room.

@RestController  
@RequiredArgsConstructor  
public class RoomResource {  
 private final RoomService roomService;  
  
 @Loggable  
 @GetMapping("/rooms")  
 public Collection<Room> getAllRooms(){  
 return roomService.getAllRooms();  
 }  
  
  
 @Loggable  
 @GetMapping("/rooms/{roomName}")  
 public Room getRoomByName(@PathVariable String roomName){  
 return roomService.getRoomByName(roomName);  
 }  
  
 @Loggable  
 @PostMapping("rooms")  
 public void addNewRoom(@RequestBody Room room){  
 roomService.addNewRoom(room);  
 }  
  
 @Loggable  
 @PutMapping ("rooms")  
 public void updateRoom(@RequestBody Room room){  
 roomService.updateRoom(room);  
 }  
}

The complete code of the server in github :

<https://github.com/YoussefAgagg/graduation-project/tree/master/control-home-devices-server>

#### ESP8266 code

##### Code for ESP8266 that control the living room devices

**ESP8266-control-livingroom-devices.ino**

Connect the NodeMCU to wifi and then connect to the web server through websocket connection so that it can communicate with the android app through the server

#include <Arduino.h>  
#include <ESP8266WiFi.h>  
#include <WebSocketsClient.h>  
  
#define DEBUG\_SERIAL Serial  
  
WebSocketsClient webSocket;  
  
const char \*ssid = "Jo";  
const char \*password = "Jomo@1210";  
  
unsigned long messageInterval = 5000;  
bool connected = *false*;  
  
void setup()  
{  
 setupLivingRoomInputOutpt();  
  
 DEBUG\_SERIAL.begin(9600);  
 DEBUG\_SERIAL.printf("try to connect to the wifi");  
  
 WiFi.begin(ssid, password);  
  
 if (WiFi.status() == WL\_CONNECTED)  
 {  
 DEBUG\_SERIAL.print("Local IP: ");  
 DEBUG\_SERIAL.println(WiFi.localIP());  
 }else{  
 DEBUG\_SERIAL.println("not connected to wifi");  
 }  
 // server address, port and URL  
 webSocket.begin("192.168.1.6", 8080, "/connect");  
 webSocket.setAuthorization("esp1", "password");  
 // event handler  
 webSocket.onEvent(webSocketEvent);  
}  
  
void loop()  
{  
 if (WiFi.status() == WL\_CONNECTED)  
 {  
 webSocket.loop();  
 }  
  
 livingRoom();  
}

**ESP8266WebsocketEvent.ino**

Handle the receiving messages from the server

void webSocketEvent(WStype\_t type, uint8\_t \*payload, size\_t length)  
{  
 switch (type)  
 {  
 case WStype\_DISCONNECTED:  
 DEBUG\_SERIAL.printf("[WSc] Disconnected!*\n*");  
 break;  
  
 case WStype\_CONNECTED:  
 DEBUG\_SERIAL.printf("[WSc] Connected to url: *%s\n*", payload);  
 break;  
  
 case WStype\_TEXT:  
 {  
 String str = (char \*)payload;  
 Serial.println(str);  
 int index = str.indexOf(":");  
 String room = str.substring(0, index);  
 if (room.equals("livingRoom"))  
 {  
 String device = str.substring(index + 1, str.length());  
 livingRoom(device);  
 }  
  
  
 DEBUG\_SERIAL.printf("[WSc] RESPONSE: *%s\n*", str);  
 }  
 break;  
 }  
}

**livingRoom.ino**

control the devices connected to the NodeMCU through the physical on/off buttons or the message received from the server to on/off any device.

// GPIO2 input pins  
uint8\_t livingRoomLampButton1 = D0;  
uint8\_t livingRoomLampButton2 = D1;   
uint8\_t livingRoomLampButton3 = D2;   
  
  
// GPIO2 output pins  
uint8\_t relayPin1 = D5;   
uint8\_t relayPin2 = D6;   
uint8\_t relayPin3 = D7;   
  
  
int previuseStateLivingRoomButton1 = 0;  
int previuseStateLivingRoomButton2 = 0;  
int previuseStateLivingRoomButton3 = 0;  
  
void setupLivingRoomInputOutpt()  
{  
 DEBUG\_SERIAL.printf("setup livingroom input ,outpit pins*\n*");  
 // Set output pins  
 pinMode(relayPin1, OUTPUT);  
 pinMode(relayPin2, OUTPUT);  
 pinMode(relayPin3, OUTPUT);  
 // Set input pins  
 pinMode(livingRoomLampButton1, INPUT);  
 pinMode(livingRoomLampButton2, INPUT);  
 pinMode(livingRoomLampButton3, INPUT);  
}  
void livingRoom()  
{  
 livingRoomButton1();  
 livingRoomButton2();  
 livingRoomButton3();  
}  
  
/\*  
 \* manage livingroom button 1  
 \*/  
void livingRoomButton1()  
{  
 int currentState = digitalRead(livingRoomLampButton1);  
 if (currentState == 0)  
 {  
 if (previuseStateLivingRoomButton1 != currentState)  
 {  
 Serial.println("button 1 not clicked");  
 webSocket.sendTXT("livingRoom:lamp1:0");  
 digitalWrite(relayPin1, LOW);  
 }  
 }  
 else  
 {  
  
 if (previuseStateLivingRoomButton1 != currentState)  
 {  
 Serial.println("button 1 clicked");  
 webSocket.sendTXT("livingRoom:lamp1:1");  
 digitalWrite(relayPin1, HIGH);  
 }  
 }  
 previuseStateLivingRoomButton1 = currentState;  
}  
/\*  
 \* manage livingroom button 2  
 \*/  
void livingRoomButton2()  
{  
 int currentState = digitalRead(livingRoomLampButton2);  
 if (currentState == 0)  
 {  
  
 if (previuseStateLivingRoomButton2 != currentState)  
 {  
 Serial.println("button 2 not clicked");  
 webSocket.sendTXT("livingRoom:lamp2:0");  
 digitalWrite(relayPin2, LOW);  
 }  
 }  
 else  
 {  
  
 if (previuseStateLivingRoomButton2 != currentState)  
 {  
 Serial.println("button 2 clicked");  
 webSocket.sendTXT("livingRoom:lamp2:1");  
 digitalWrite(relayPin2, HIGH);  
 }  
 }  
 previuseStateLivingRoomButton2 = currentState;  
}  
/\*  
 \* manage livingroom button 3  
 \*/  
void livingRoomButton3()  
{  
 int currentState = digitalRead(livingRoomLampButton3);  
 if (currentState == 0)  
 {  
   
  
 if (previuseStateLivingRoomButton3 != currentState)  
 {  
 Serial.println("button 3 not clicked");  
 webSocket.sendTXT("livingRoom:lamp3:0");  
 digitalWrite(relayPin3, LOW);  
 }  
 }  
 else  
 {  
   
  
 if (previuseStateLivingRoomButton3 != currentState)  
 {  
 Serial.println("button 3 clicked");  
 webSocket.sendTXT("livingRoom:lamp3:1");  
 digitalWrite(relayPin3, HIGH);  
 }  
 }  
 previuseStateLivingRoomButton3 = currentState;  
}  
  
  
void livingRoom(String str)  
{  
 int index = str.indexOf(":");  
 String led = str.substring(0, index);  
 int outputStatus = str.substring(index + 1, str.length()).toInt();  
  
 if (led.equals("lamp1"))  
 {  
 DEBUG\_SERIAL.printf("[lamp1] RESPONSE: *%s\n*", led);  
 if (outputStatus == 0)  
 digitalWrite(relayPin1, LOW);  
 else  
 digitalWrite(relayPin1, HIGH);  
 }  
 else if (led.equals("lamp2"))  
 {  
 DEBUG\_SERIAL.printf("[lamp1] RESPONSE: *%s\n*", led);  
 if (outputStatus == 0)  
 digitalWrite(relayPin2, LOW);  
 else  
 digitalWrite(relayPin2, HIGH);  
 }  
 else if (led.equals("lamp3"))  
 {  
 DEBUG\_SERIAL.printf("[lamp1] RESPONSE: *%s\n*", led);  
 if (outputStatus == 0)  
 digitalWrite(relayPin3, LOW);  
 else  
 digitalWrite(relayPin3, HIGH);  
 }  
}

##### Code for ESP8266 that control the bed room devices and the garage door

**ESP8266-control-garage-bedroom-devices.ino**

Connect the second NodeMCU to wifi and then connect to the web server through websocket connection so that it can communicate with the android app through the server

#include <Arduino.h>  
#include <ESP8266WiFi.h>  
#include <WebSocketsClient.h>  
  
#define DEBUG\_SERIAL Serial  
  
WebSocketsClient webSocket;  
  
  
const char \*ssid = "Jo";  
const char \*password = "Jomo@1210";  
  
unsigned long messageInterval = 5000;  
bool connected = *false*;  
  
void setup()  
{  
 setupBedRoomInputOutpt();  
 setupGarageInputOutpt();  
 DEBUG\_SERIAL.begin(115200);  
 DEBUG\_SERIAL.printf("try to connect to the wifi");  
  
 WiFi.begin(ssid, password);  
  
 if (WiFi.status() == WL\_CONNECTED)  
 {  
 DEBUG\_SERIAL.print("Local IP: ");  
 DEBUG\_SERIAL.println(WiFi.localIP());  
 }else{  
 DEBUG\_SERIAL.println("not connected to wifi");  
 }  
   
 // server address, port and URL  
 webSocket.begin("192.168.1.6", 8080, "/connect");  
 webSocket.setAuthorization("esp2", "password");  
 // event handler  
 webSocket.onEvent(webSocketEvent);  
}  
  
void loop()  
{  
 if (WiFi.status() == WL\_CONNECTED)  
 {  
 webSocket.loop();  
 }  
   
 bedRoom();  
 garage();  
}

**ESP8266WebsocketEvent.ino**

void webSocketEvent(WStype\_t type, uint8\_t \*payload, size\_t length)  
{  
 switch (type)  
 {  
 case WStype\_DISCONNECTED:  
 DEBUG\_SERIAL.printf("[WSc] Disconnected!*\n*");  
 break;  
  
 case WStype\_CONNECTED:  
 DEBUG\_SERIAL.printf("[WSc] Connected to url: *%s\n*", payload);  
 break;  
  
 case WStype\_TEXT:  
 {  
 String str = (char \*)payload;  
 Serial.println(str);  
 int index = str.indexOf(":");  
 String room = str.substring(0, index);  
 if (room.equals("bedRoom"))  
 {  
 String device = str.substring(index + 1, str.length());  
 bedRoom(device);  
 }  
 else if (room.equals("garage"))  
 {  
 String device = str.substring(index + 1, str.length());  
 garage(device);  
 }  
  
 DEBUG\_SERIAL.printf("[WSc] RESPONSE: *%s\n*", str);  
 }  
 break;  
 }  
}

**garage.ino**

control the door through the physical on/off button or the message received from the server to open/close any device.

We used Servo.h library to control the servo motor

#include<Servo.h>  
  
Servo motor;  
// GPIO2 input pins  
uint8\_t garageDoorButton = D2;   
  
// GPIO2 output pins  
uint8\_t door = D8;   
  
void setupGarageInputOutpt()  
{  
 DEBUG\_SERIAL.printf("setup garage input ,outpit pins*\n*");  
   
 motor.attach(door);  
  
 // Set input pins  
 pinMode(garageDoorButton, INPUT);  
}  
  
bool isOpened = *false*;  
int previuseStateGarageButton = 0;  
void garage()  
{  
 int currentState = digitalRead(garageDoorButton);  
 if (currentState == 0 && isOpened)  
 {  
 if (previuseStateGarageButton != currentState)  
 {  
 isOpened = *false*;  
 webSocket.sendTXT("garage:door:0");  
 closeDoor();  
 }  
 }  
 else if (currentState == 1 && !isOpened)  
 {  
 if (previuseStateGarageButton != currentState)  
 {  
 isOpened = *true*;  
 webSocket.sendTXT("garage:door:1");  
 openDoor();  
 }  
   
 }  
 previuseStateGarageButton= currentState;  
   
}  
  
void garage(String str)  
{  
 int index = str.indexOf(":");  
 String led = str.substring(0, index);  
 int outputStatus = str.substring(index + 1, str.length()).toInt();  
  
 if (led.equals("door"))  
 {  
 DEBUG\_SERIAL.printf("[garage] RESPONSE: *%s\n*", led);  
 if (outputStatus == 0 && isOpened){  
 isOpened = *false*;  
 closeDoor();  
 }  
 else if (outputStatus == 1 && !isOpened){  
 isOpened = *true*;  
 openDoor();  
 }  
 }  
}  
void openDoor(){  
 motor.write(180);  
 delay(50);  
 motor.write(10);  
 delay(50);  
   
  
  
 }  
void closeDoor(){  
 motor.write(10);  
 delay(50);  
 motor.write(180);  
  
 }

**bedroom.ino**

control the devices connected to the NodeMCU through the physical on/off buttons or the message received from the server to on/off any lights.

// GPIO2 input pins  
uint8\_t bedRoomLampButton1 =D0;   
uint8\_t bedRoomLampButton2 =D1;   
  
// GPIO2 output pins  
  
uint8\_t relayPin5 = D5;   
uint8\_t relayPin6 = D6;  
  
  
int previuseStateBedRoomButton1 = 0;  
int previuseStateBedRoomButton2 = 0;  
  
  
void setupBedRoomInputOutpt()  
{  
 DEBUG\_SERIAL.printf("setup bedroom input ,outpit pins*\n*");  
   
 // Set output pins  
 pinMode(relayPin5, OUTPUT);  
 pinMode(relayPin6, OUTPUT);  
  
 // Set input pins  
 pinMode(bedRoomLampButton1, INPUT);  
 pinMode(bedRoomLampButton2, INPUT);  
   
}  
void bedRoom()  
{  
 bedRoomButton1();  
 bedRoomButton2();  
  
   
}  
  
/\*  
 \* manage bedroom button 1  
 \*/  
void bedRoomButton1()  
{  
 int currentState = digitalRead(bedRoomLampButton1);  
 if (currentState == 0)  
 {  
 if (previuseStateBedRoomButton1 != currentState)  
 {  
 webSocket.sendTXT("bedRoom:lamp1:0");  
 digitalWrite(relayPin5, LOW);  
 }  
 }  
 else  
 {  
  
 if (previuseStateBedRoomButton1 != currentState)  
 {  
 webSocket.sendTXT("bedRoom:lamp1:1");  
 digitalWrite(relayPin5, HIGH);  
 }  
 }  
 previuseStateBedRoomButton1 = currentState;  
}  
/\*  
 \* manage bedroomroom button 2  
 \*/  
void bedRoomButton2()  
{  
 int currentState = digitalRead(bedRoomLampButton2);  
 if (currentState == 0)  
 {  
  
 if (previuseStateBedRoomButton2 != currentState)  
 {  
 webSocket.sendTXT("bedRoom:lamp2:0");  
 digitalWrite(relayPin6, LOW);  
 }  
 }  
 else  
 {  
  
 if (previuseStateBedRoomButton2 != currentState)  
 {  
 webSocket.sendTXT("bedRoom:lamp2:1");  
 digitalWrite(relayPin6, HIGH);  
 }  
 }  
 previuseStateBedRoomButton2 = currentState;  
}  
  
  
void bedRoom(String str)  
{  
 DEBUG\_SERIAL.printf("[bedroom] RESPONSE: *%s\n*", str);  
  
 int index = str.indexOf(":");  
 String led = str.substring(0, index);  
 int outputStatus = str.substring(index + 1, str.length()).toInt();  
  
 if (led.equals("lamp1"))  
 {  
 DEBUG\_SERIAL.printf("[bedroom] RESPONSE: *%s\n*", led);  
 if (outputStatus == 0)  
 digitalWrite(relayPin5, LOW);  
 else  
 digitalWrite(relayPin5, HIGH);  
 }  
 else if (led.equals("lamp2"))  
 {  
 DEBUG\_SERIAL.printf("[bedroom] RESPONSE: *%s\n*", led);  
 if (outputStatus == 0)  
 digitalWrite(relayPin6, LOW);  
 else  
 digitalWrite(relayPin6, HIGH);  
 }  
   
  
}

#### Android application code

**Server connection:**

void connect() async{  
 String basicAuth =  
 'Basic ' + base64.encode(utf8.encode('$*username*:$*password*'));  
 socket = IOWebSocketChannel.connect(  
 endpoint ,  
 headers: {  
 'Authorization': basicAuth ,  
 },);  
 print(basicAuth);

sureConnect();  
 connected = true;  
 emit(ConnectServer());  
 debugPrint('Web socket is connect');  
 socket!.stream.listen((message) {  
 // \_recipientCtrl.add(event);  
 debugPrint(message);  
 if(message == "livingRoom:lamp1:1"){  
 livingRoomMainLightIsOn = true;  
 }else if(message == "livingRoom:lamp1:0"){  
 livingRoomMainLightIsOn = false;  
 } else if(message == "livingRoom:lamp2:1"){  
 livingRoomTableLightIsOn = true;  
 }else if(message == "livingRoom:lamp2:0"){  
 livingRoomTableLightIsOn = false;  
 } else if(message == "livingRoom:lamp3:1"){  
 livingRoomFloorLightIsOn = true;  
 } else if(message == "livingRoom:lamp3:0"){  
 livingRoomFloorLightIsOn = false;  
 } else if(message == "bedRoom:lamp1:1"){  
 bedRoomMainLightIsOn = true;  
 } else if(message == "bedRoom:lamp1:0"){  
 bedRoomMainLightIsOn = false;  
 }else if(message == "bedRoom:lamp2:1"){  
 bedRoomTableLightIsOn = true;  
 } else if(message == "bedRoom:lamp2:0"){  
 bedRoomTableLightIsOn = false;  
 }  
 else if(message == "garage:door:1"){  
 garageDoorIsOn = true;  
 } else if(message == "garage:door:0"){  
 garageDoorIsOn = false;  
 }  
  
 emit(ListenMessage());  
 }  
 , onError: (e) async {  
 \_recipientCtrl.addError(e);  
 debugPrint(e);  
 emit(ErrorConnectServer());  
 await Future.delayed(Duration(seconds: delay!));  
 connect();  
 }, onDone: () async {  
 await Future.delayed(const Duration(seconds: 5));  
 debugPrint('close sn reconnect');  
 emit(OnDonConnectServer());  
 connect();  
 // AutoReconnectWebSocket(endpoint);  
 },  
 cancelOnError: true);}

**send message(action) function :**

Future<void> sendCmd(String cmd) async {  
  
 if(connected == true){  
 socket!.sink.add(cmd);  
 emit(SendMessage());  
 }else{  
 debugPrint("Websockets is not connected.");  
 connect();  
 // emit(SendMessage());  
 }  
}

**Login function**

void login(String email ,String password) async {  
 emit(AppLoginLoadingState());  
 print ('username:'+email+' '+'password:'+password);  
 try{  
 Response response = await http.post(  
 Uri.*parse*('http://${SettingScreen.*ip*}:8080/login'),  
 body: jsonEncode({  
 "username":email,  
 "password":password  
 }),  
 headers: {  
 'Content-Type': 'application/json',  
 }, );  
 print(response.statusCode);  
 print(response.body);  
 if(response.statusCode == 200){  
 print('Login successfully');  
 emit(ShopLoginSuccessState());  
  
 }else {  
 print('failed');  
 }  
 }catch(e){  
 print(e.toString());  
 emit(AppLoginErrorState(e.toString()));  
 }  
}

**Switch build Item:**

Widget switchBuildItem({  
 required Icon icon,  
 required String title,  
 required String subtitle,  
 required bool lightIsOn,  
 required VoidCallback onTap,  
 double ? subtitleFontSize = 17,  
 double ? titleFontSize = 14,  
 Size ? size,  
 Color? unSelectedImageColor,  
})=> GestureDetector(  
 onTap: (){  
 onTap();  
 },  
 child: Container(  
 decoration: BoxDecoration(  
 borderRadius: BorderRadius.circular(15),  
 ),  
 child: Card(  
 color: ChooseColor. *defaultBackgroundColor*,  
 child: SizedBox(  
 height: size!.width \* 0.27,  
 width: size.width \* 0.30,  
 child: Center(  
 child: Column(  
 children: [  
 ListTile(  
 leading : icon ,  
 title: Text(  
 subtitle ,  
 style: TextStyle(  
 color: Colors.*grey*,  
 fontWeight: FontWeight.*bold*,  
 fontSize: size.width\*0.03  
 ),  
 ),  
 ),  
 const SizedBox(height: 10,),  
 Text(  
 title,  
 style: TextStyle(  
 color: Colors.*black*,  
 fontWeight: FontWeight.*bold*,  
 fontSize: size.width\*0.03),  
 ),  
 ]),),  
 ), ),),);

**Switch action (on/off)**

onTap: (){  
// cubit.connect();  
 if(cubit.livingRoomMainLightIsOn){ //if isLed1On is true, then turn off the led  
 //if led is on, turn off  
 cubit.sendCmd('livingRoom:lamp1:0');  
 cubit.livingRoomMainLightIsOn = false;  
 }else{ //if isLed1On is false, then turn on the led  
 //if led is off, turn on  
 cubit.sendCmd('livingRoom:lamp1:1');  
 cubit. livingRoomMainLightIsOn = true;  
 }

**Room Build Item :**

Widget roomBuildItem (  
{  
 required String image,  
 required String text,  
 required bool isSelected,  
 required VoidCallback onTap,  
 double ? fontSize = 18,  
 Color? unSelectedImageColor,  
})=> Expanded (  
 child: GestureDetector(  
 onTap: (){  
 onTap();  
 },  
 child: Container(  
 decoration: BoxDecoration(  
 borderRadius: BorderRadius.circular(15),  
 //gradient: isSelected ? appGradient : null,  
 color: !isSelected ? ChooseColor.*defaultBackgroundColor* : ChooseColor.*defaultColor*,  
  
 ),  
 child: Column(  
 children: [  
 AppSpaces.*vertical15*,  
 Center(  
 child: SizedBox(  
 width: Get.width/ 5,  
 height: Get.height / 10,  
 child: Image.asset(image,   
 ),  
 ),  
 ),  
 AppSpaces.*vertical15*,  
 Text(  
 text,  
 style: TextStyle(  
 color: isSelected ? Colors.*white* : Colors.*black*,  
 fontSize: fontSize , ),  
 ),  
 AppSpaces.*vertical15*,  
 ]),  
 ),  
 ),  
 );

**Room selects:**

onTap: (){  
 setState(() {  
 controller.setIndex(0);  
 navigateTo(context, LivingRoom());  
 });

The complete code of the server in github :

[**https://github.com/YoussefAgagg/graduation-project/tree/master/control-home-devices-app**](https://github.com/YoussefAgagg/graduation-project/tree/master/control-home-devices-app)

# CHAPTER 5 Results and discussion

## Results

This section presents the acquired results from the project. It describes the final outcome of the prototype and the android application

### Home model



Figure 5. 1 home model



Figure 5. 2 living room and bed room leds and buttons



Figure 5. 3 garage door

### Android application

The user UI

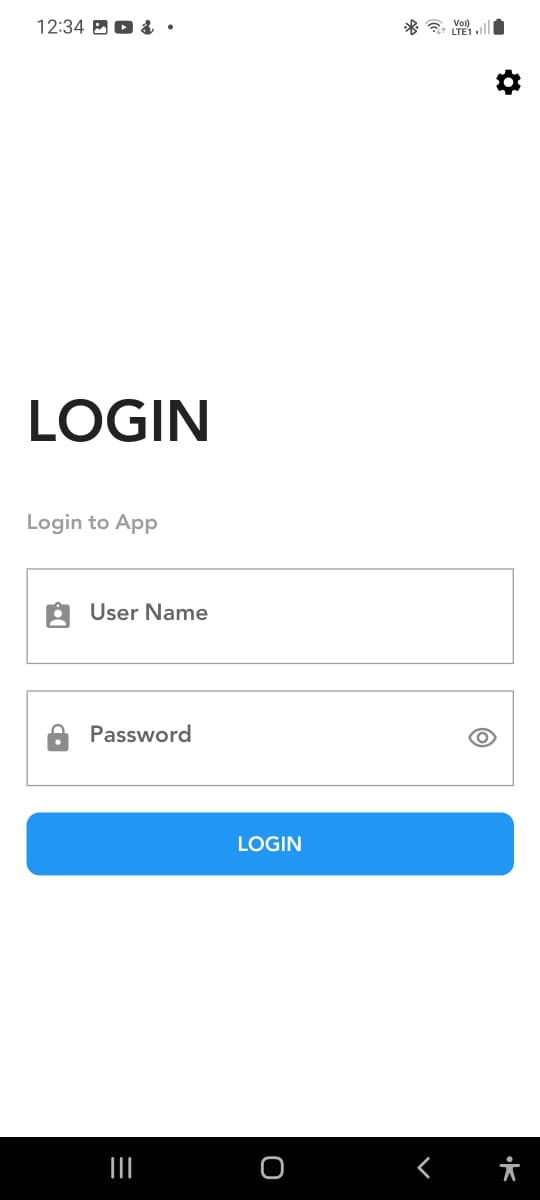


Figure 5. 4 android app login screen

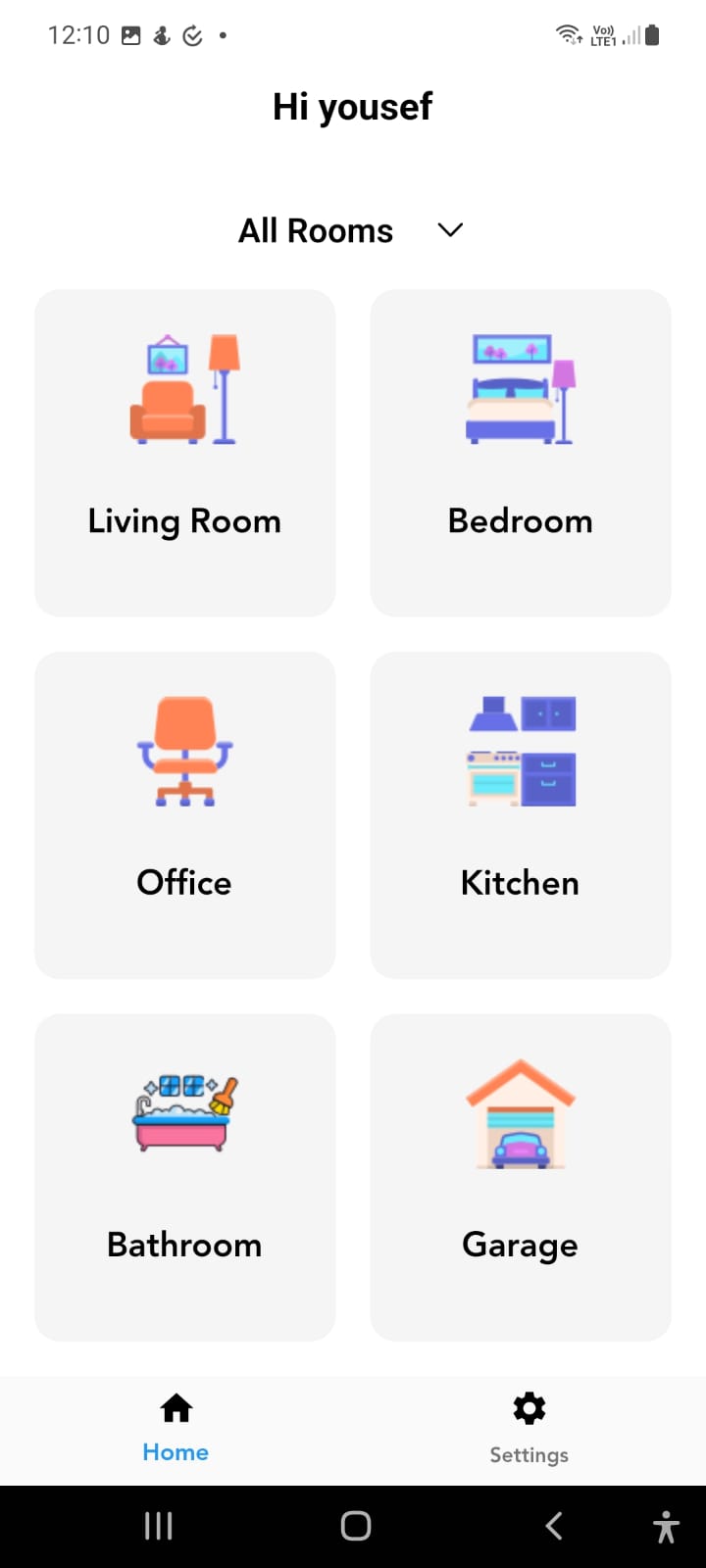


Figure 5. 5 android app main screen

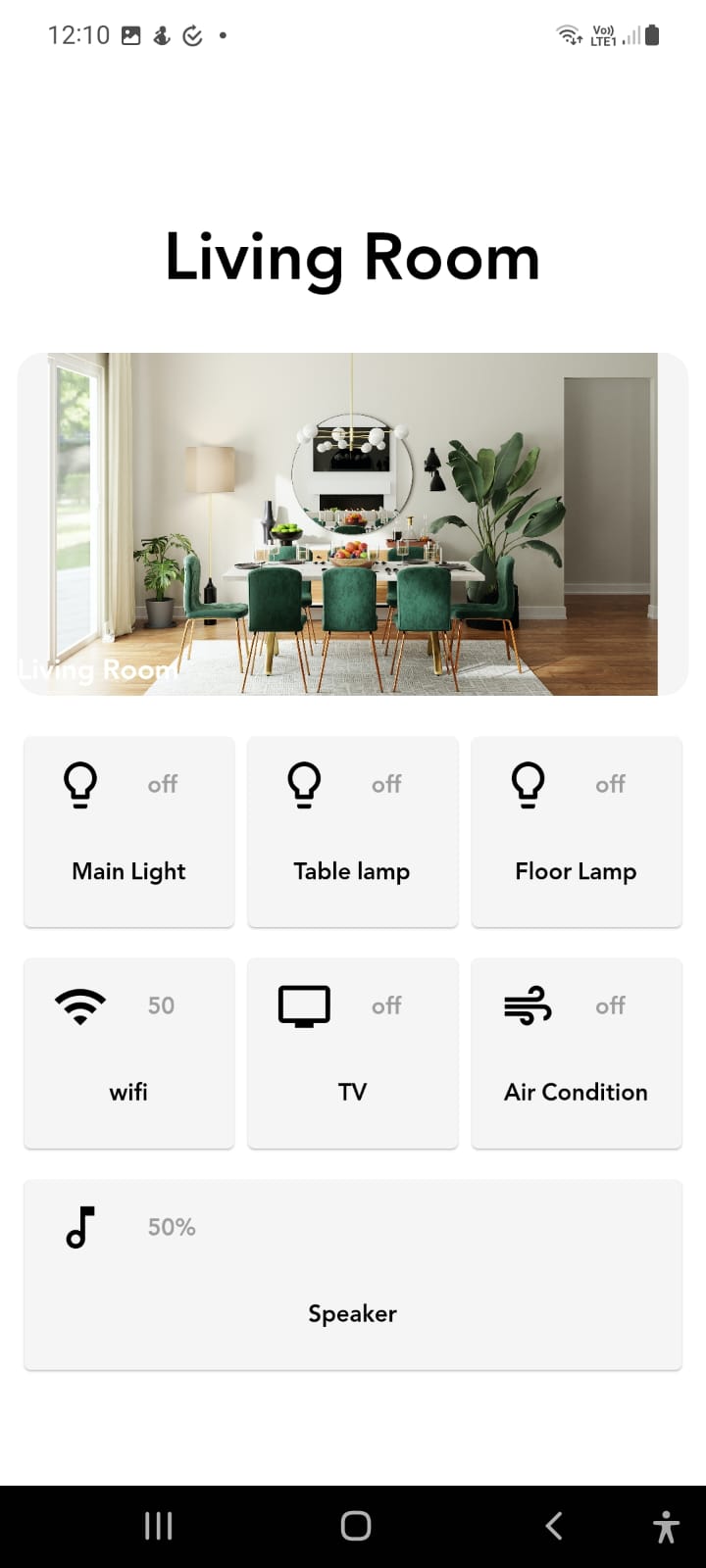


Figure 5. 6 android app living room screen

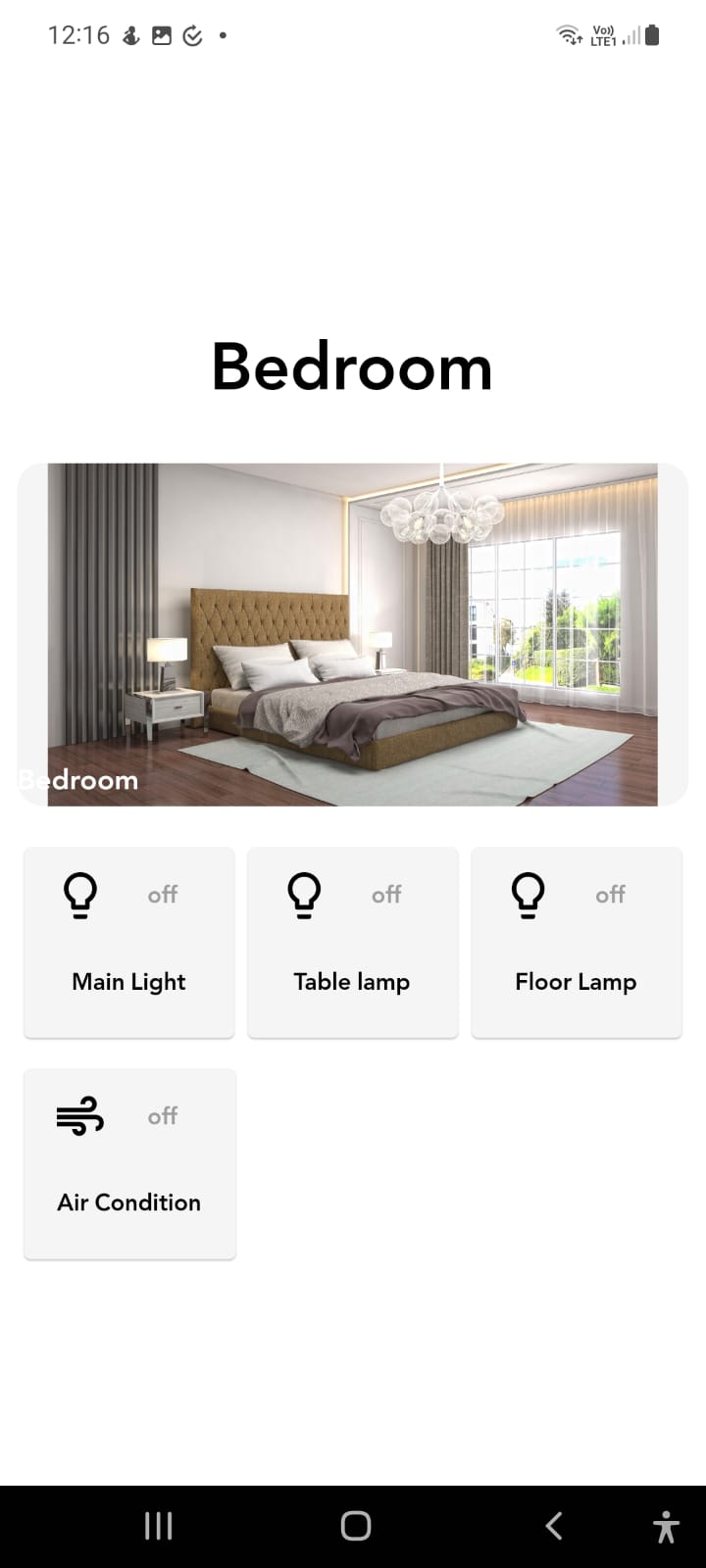


Figure 5. 7 android app bed room screen

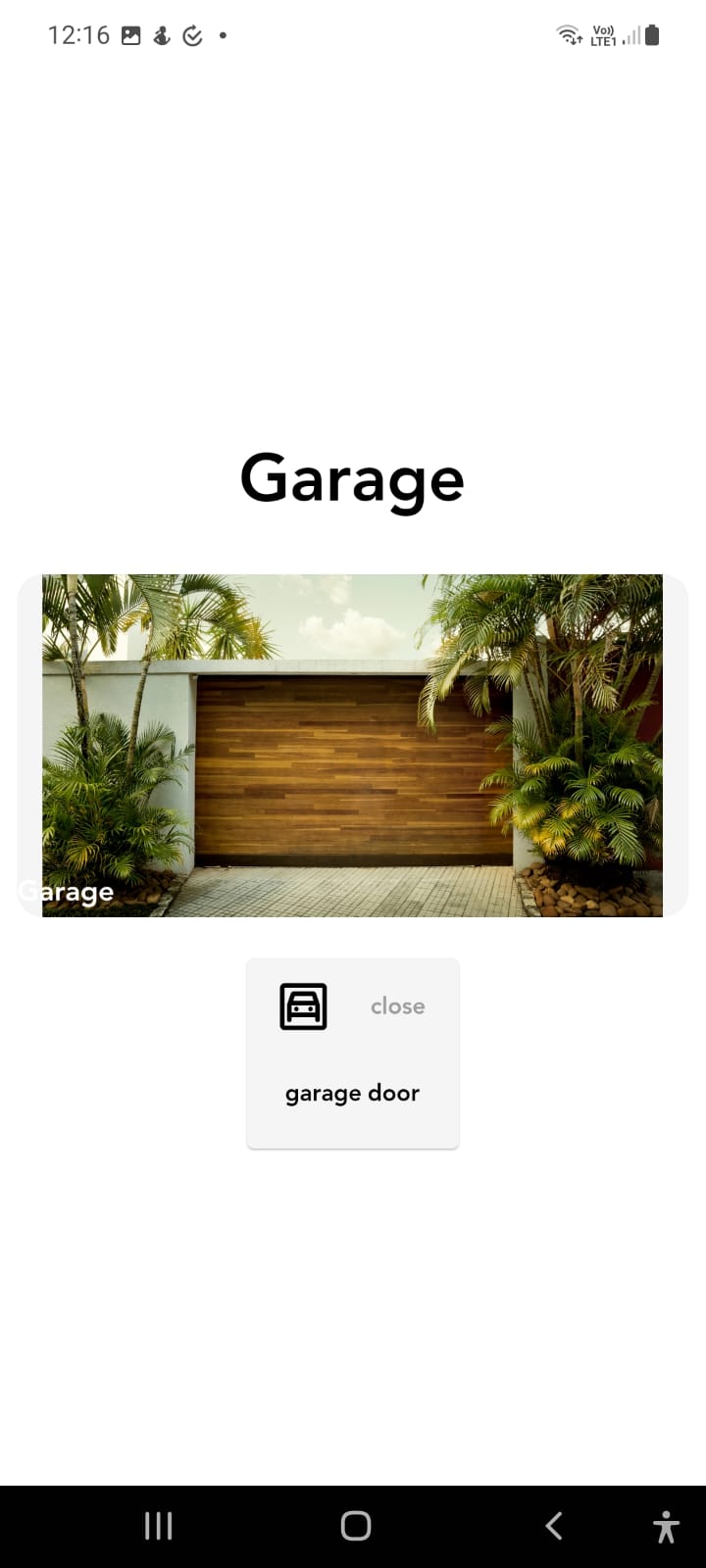


Figure 5. 8 android app garage screen

## Discussion

Over all most of the design specifications were implemented successfully. A variety of Devices are connected to the ESP8266 that allows the devices to be controlled

remotely using an android app. The server has a successful implementation of user verification that makes use of a username and password to limit unauthorized connections to the system. The android application has a user-friendly interface that can obtain information and control the Home’s devices. Thesystem is designed in such a way that it allows simple implementation of new devices into the system.

## Conclusion

Overall, the implementation of the project is a success. This

senior design project has been a valuable learning experience over the

past year, providing insight into design philosophies and the Internet of

Things. Information learned from this project will be useful moving forward

into a post-grad setting, and will lead to better engineering in the future.