INTERNET OF THINGS (IOT) COURSEWORK REPORT

Do not write your name on your work unless your lecturer has explicitly told you to do so.

Student ID number	Title of degree studying	**Level/Year**
2200918	Bachelor of Science (Honours) Data Science and Analytics	2

Short unit name:	M31890 - IOT			Due date: 18 December 23		Deadline: 18 Dec
Full unit name:	INTERNET OF THINGS					
Unit lecturer name:	Jason Tan				Group: *(if applicable)*	
Additional items e.g. CD/disk/USB:	Yes		No	✓	**Details:**	

All additional items should be clearly labelled with ID number and unit name and securely attached to your work.

Candidates are reminded that the following are defined as Assessment Offences and will be dealt with in accordance with the University's Code of Student Discipline:

a) Any attempt to complete an assessment by means considered to be unfair;
b) Plagiarism, which the University defines as the incorporation by a student in work for assessment of material which is not their own, in the sense that all or a substantial part of the work has been copied without any adequate attempt at attribution or has been incorporated as if it were the student's own work when in fact it is wholly or substantially the work of another person or persons.

Please note: Group **coursework** will be filed under the first Student ID number at the top of the list. Ensure you know all **group member's ID numbers.**

NB: **Coursework not collected** will be disposed of six months **after** the hand-in date.

FOR OFFICIAL USE ONLY

Date received/Office stamp	Provisional mark % / Comments
Administration Office	Academic Staff Member

Table of Contents

UNIVERSITY OF PORTSMOUTH

UOP BScSE Intake 11

M31890 Internet of Things (IOT)

IOT Main Assignment

Title: ESP32-Based Home Automation with Voice Control Integration

1. Introduction

Smart technology integration into homes is evolving from a luxury to a necessity, enhancing daily living. This report presents an ESP32-based Home Automation system, uniquely integrated with Google Assistant for voice control, offering accessibility and convenience.

1.1 Problem Statement

Home automation faces challenges like complexity, high costs, and limited accessibility. This project addresses these by creating an affordable, user-friendly system with voice control, making smart homes more accessible, especially for those with disabilities.

1.2 Application Scenario

Focused on smart living, the system targets households aiming to modernize routines through intuitive voice-controlled appliance management, benefiting users with mobility or visual challenges.

1.3 Motivation and Justification

Bridging high-tech and practicality, our system stands out in the smart home market for its simplicity and inclusiveness. Using the ESP32 for Wi-Fi capability and cost-effectiveness, alongside Google Assistant for advanced voice recognition, it delivers a seamless experience.

1.4 System Architecture

The system is a cohesive unit combining voice control, environmental sensing, and a user interface. Key features include Google Assistant for voice commands, environmental sensors for real-time data, and ESP-RainMaker for remote access, ensuring a comprehensive and accessible smart home solution.

2. System Design

2.1 Sub-system 1: ESP32 Smart Home Automation with Voice Control Integration

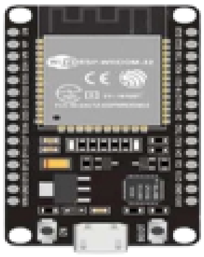
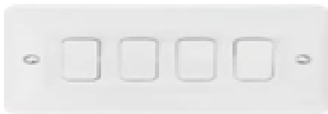
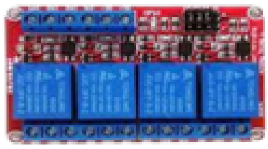
Sub-system 1 is the core of the ESP32 Smart Home Automation system, designed to integrate voice control capabilities for managing and automating home appliances. It employs the ESP32 microcontroller as the central processing unit, leveraging its Wi-Fi functionality to connect with various home devices.

The primary feature of this subsystem is its integration with Google Assistant for voice control, enabling users to issue commands verbally to control lighting, temperature, and other home appliances.

Additionally, it incorporates environmental sensors like the DHT22 for temperature and humidity monitoring, and an LDR for light sensing, allowing the system to respond intelligently to environmental changes. User interactions are facilitated through an LCD display, providing feedback and manual control options. The entire subsystem is interconnected using the ESP-RainMaker platform, ensuring seamless operation, remote accessibility, and efficient management of home automation tasks.

2.1.1 Component List

Below is the list of components

	Hardware/ Software Components	Description	Image
1	ESP-WROOM-32	This is a powerful, generic Wi-Fi+BT+BLE MCU module that targets a wide variety of applications. The ESP32 chip within this module provides robust connectivity options and supports a wide range of communication protocols.	
2	230V Light Bulbs	Standard light bulbs used in home lighting. In this project, they represent the typical home appliances being controlled.	
3	Switches (15A 250 VAC/ 20A 125 VAC)	These are high-capacity switches, capable of handling significant electrical loads, making them suitable for controlling various home appliances.	
4	5V 4-Channel Relay Module	A crucial component for interfacing high-power devices with the low-power ESP32. This module can control several devices independently and safely.	
5	10K Ohms resistor	A common resistor used in electronic circuits, often for current limiting or voltage division.	
6	LDR (Light Dependent Resistor)	A sensor that varies its resistance based on the light intensity. Used in the project to detect ambient light levels for automated control.	

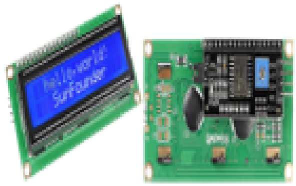
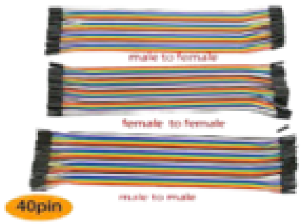
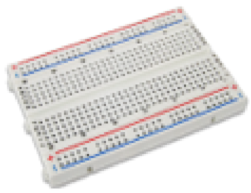
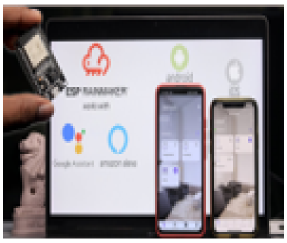
6	DHT22	A sensor used for measuring temperature and humidity. This provides environmental data to the ESP32 for monitoring or controlling other devices based on these parameters.	
7	LCD 16x02 with I2C Module	A display for providing a user interface. It can show system statuses, sensor readings, or other information. The I2C module simplifies connection to the ESP32, using just two data lines.	
8	Jumper Cables	Used for making connections between components on a breadboard or within an electronic assembly.	
9	Breadboard	A solderless device for prototyping electronic circuits. It allows for easy inserting and removing of components and jumper wires, facilitating the testing and development of the circuit design.	
10	ESP-RainMaker	This is the software platform used for the ESP32. It provides an easy way to create IoT applications by handling common functionality such as device provisioning, security, and cloud connectivity.	
11	Arudino IDE	A user-friendly integrated development environment (IDE) used for writing and uploading code to Arduino-compatible boards. Ideal for beginners and experts alike, it simplifies the coding process for various microcontroller-based projects.	

Table 1 for System Component list

2.2 Comparison between Alternate Components

In the development of an ESP32-based smart home system, selecting the appropriate components is crucial for achieving the desired functionality and performance. This section compares alternative options for three key components: the microcontroller, relay module, and temperature/humidity sensor, providing insights into their suitability for the system.

1. Microcontroller: ESP-WROOM-32 vs. Arduino Uno R3

- **ESP-WROOM-32 (ESP32):**

Its integrated Wi-Fi and Bluetooth, along with a powerful processor, make it ideal for complex, connected applications, enhancing system performance with faster processing and more efficient connectivity. The advanced capabilities may slightly steepen the learning curve, impacting the initial user setup experience.

- **Arduino Uno R3:**

User-friendly for beginners, ideal for simple tasks. Its lack of built-in Wi-Fi/Bluetooth could limit system interconnectivity and performance in advanced projects, potentially affecting the range and scope of user interaction.

Justification for ESP32: Chosen for its superior processing power and integrated connectivity, essential for a high-performance smart home system. It enhances the user experience with faster response times and seamless integration of voice control, despite the slightly more complex programming required.

2. Relay Module: 5V 4-Channel Relay Module vs. Solid State Relays

- **5V 4-Channel Relay Module:**

Its ability to control multiple devices and provide electrical isolation enhances system versatility and safety. However, the mechanical nature might affect long-term reliability and consistency in performance, impacting user experience over time.

- **Solid State Relays:**

Offer silent operation and reliability, improving the user experience in environments where noise is a concern. The higher cost and need for heat management can impact the system's affordability and complexity.

Justification for Relay Module: Selected for its balance of functionality and cost. It efficiently manages multiple devices and is suitable for a residential setting where high-frequency switching isn't critical, offering a practical solution without significantly impacting the user experience.

3. Temperature and Humidity Sensor: DHT22 vs. DHT11

- **DHT22:**

Offers higher accuracy and a broader range, leading to more precise environmental control, thereby improving the system's responsiveness and user experience. The higher cost and slower response time might be considerations for budget and speed efficiency.

- **DHT11:**

More budget-friendly and faster response times suit basic monitoring needs, but lower accuracy and a narrower range could limit the system's effectiveness in accurately responding to environmental changes, potentially affecting user comfort and system efficiency.

Justification for DHT22: Preferred for its higher accuracy and broader range, ensuring precise environmental control. This choice elevates the system's performance in effectively responding to environmental changes, significantly enhancing user comfort and interaction with the system.

2.3 Pseudo Code, Diagrams and Pin Configuration

Pseudocode:

```
Start
Initialize System:
  Set GPIO modes for switches, relays, and sensors
  Initialize WiFi
  Initialize ESP RainMaker with devices
  Initialize Google Assistant Integration:
    - Use the "Google Home" app to set up and link smart home devices with Google Assistant.
    - Ensure that the devices' manufacturers provide Google Assistant compatibility.
  Initialize LCD Display:
    - Use the LiquidCrystal_I2C library to initialize the LCD display.
    - Set the I2C address of the display (e.g., 0x27).
    - Specify the number of rows and columns (e.g., 16x2).
    - Turn on the backlight if needed.
Main Loop:
  While true:
    If WiFi is not connected:
      Attempt to reconnect to WiFi with retry mechanism (Max retries, delay)
      If WiFi reconnection is not successful:
        Update WiFi LED to OFF
        Continue to the next iteration of the loop
      Else:
        Update WiFi LED to ON
    Read state of manual switches
    For each switch:
      If switch is toggled:
        Toggle the corresponding relay
        Update relay state in ESP RainMaker
    If voice command is received from Google Assistant:
      Parse the voice command
      Identify the requested action and target device
      Execute the action on the target device
      Update the state of the devices in ESP RainMaker
    If automation rule or schedule is triggered:
      Perform the scheduled action
      Update the state of the devices in ESP RainMaker
    Read sensor data from DHT22 and LDR at specified intervals
    Update sensor data in ESP RainMaker
    Display sensor data on LCD Display
    Wait for a short duration (e.g., 1 second) before the next iteration
End
```

Figure 1: Pseudocode for ESP32 Smart Home Automation with Voice Control Integration

Flow diagram

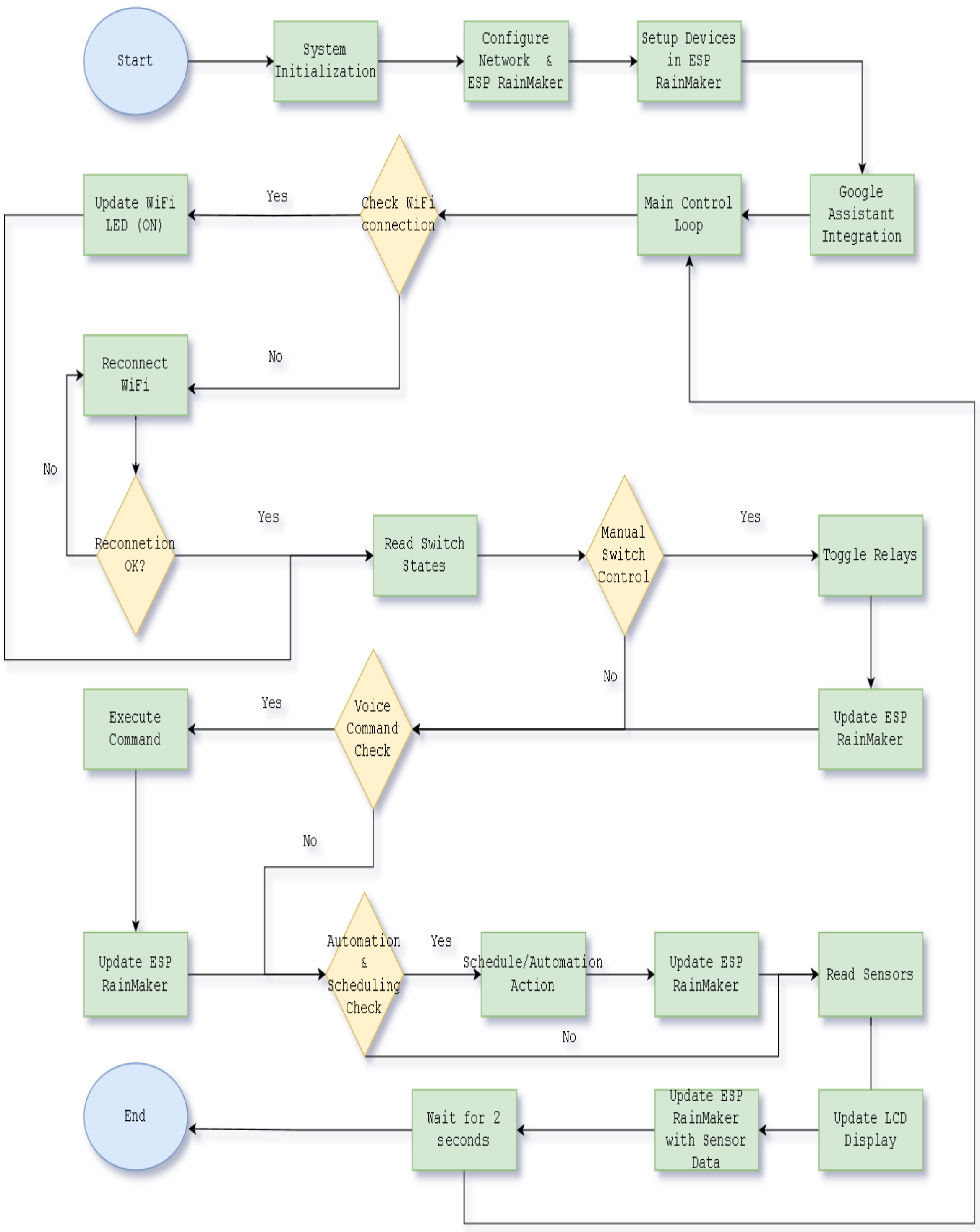


Figure 2: Flowchart diagram for ESP32 Smart Home Automation with Voice Control Integration

Circuit Diagram

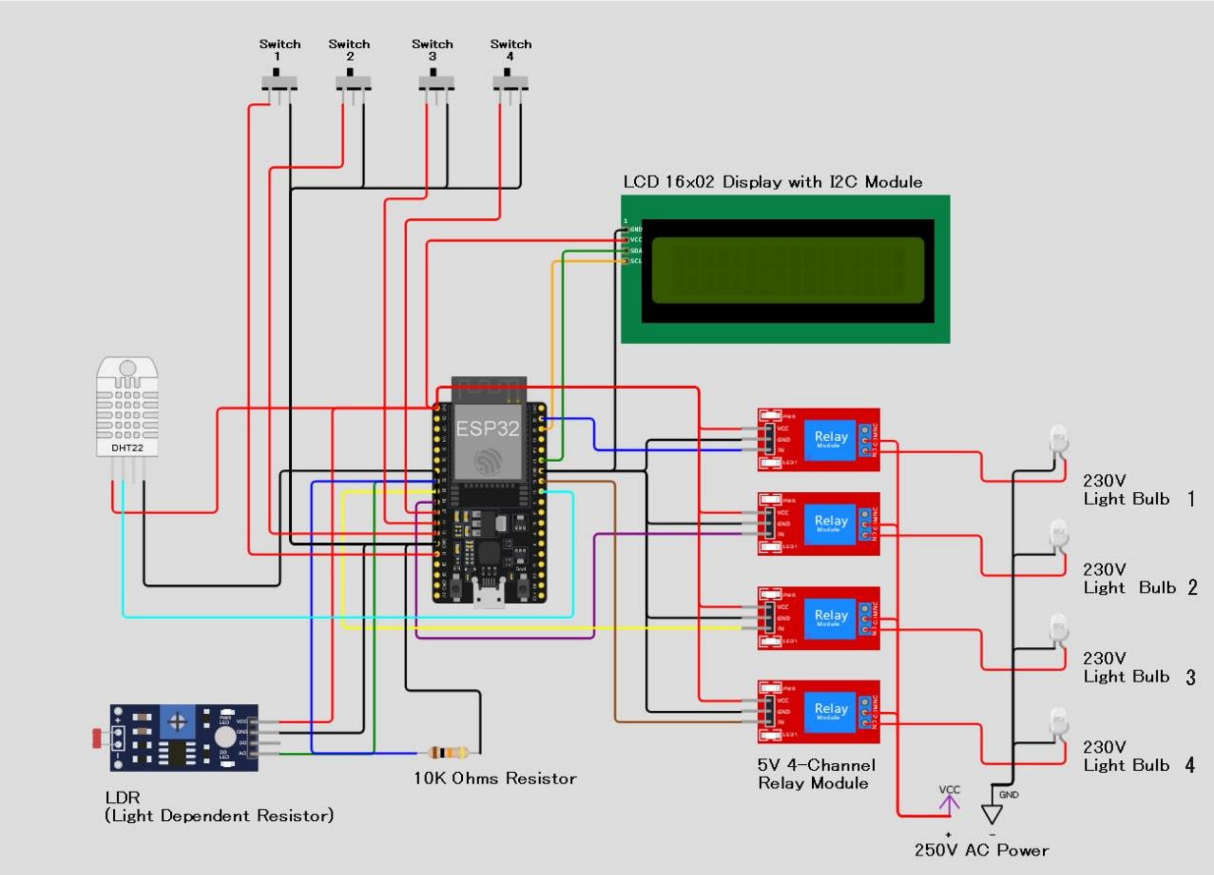


Figure 3: Circuit Diagram for ESP32 Smart Home Automation with Voice Control Integration

ESP-WROOM-32 Datasheet

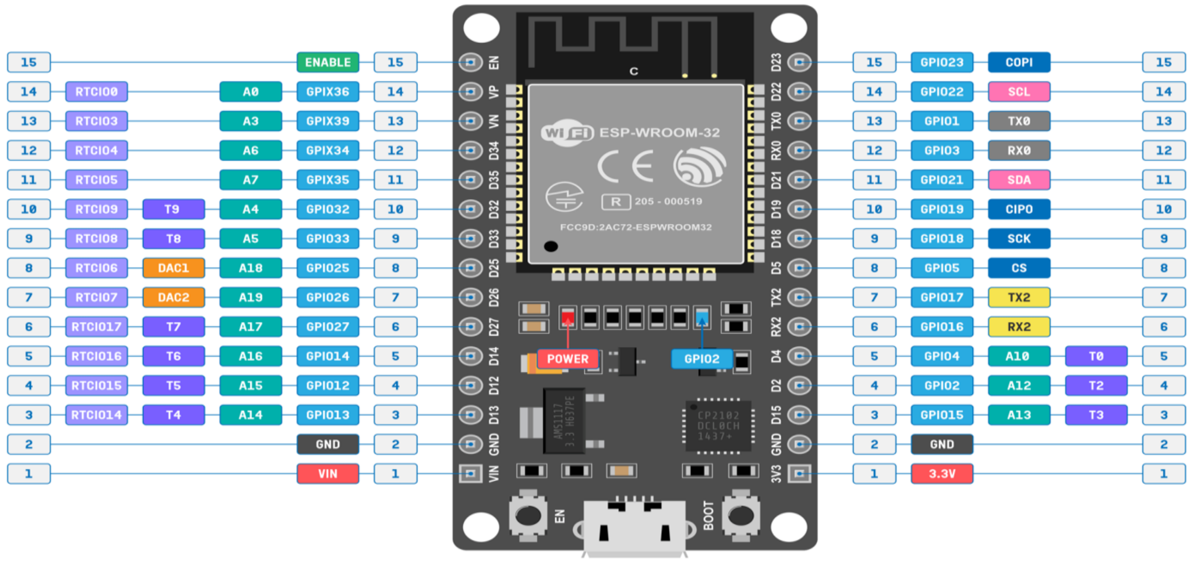


Figure 4: ESP32 Datasheet

Pin Configuration:

Component	PIN	Description	Wiring to ESP32
Switch 1-4	GND (-)	Connected to the ground of the system for completing the circuit	GND
Switch 1	VCC (+)	Live wire connected to GPIO13 of ESP32, used to control the first relay	D13
Switch 2	VCC (+)	Live wire connected to GPIO12 of ESP32, used to control the second relay	D12
Switch 3	VCC (+)	Live wire connected to GPIO14 of ESP32, used to control the third relay	D14
Switch 4	VCC (+)	Live wire connected to GPIO27 of ESP32, used to control the fourth relay	D27
DHT22	VCC (+)	Powers the DHT22 temperature and humidity sensor	3V3
	Data (Out)	Transmits the sensor data to the ESP32	D18
	GND (-)	Ground pin for the DHT22 sensor	GND
LDR	Left leg	Connected to a power source for the light-dependent resistor	3V3
	Right leg	Senses the light level and sends data to the ESP32	D33
10K Ohms Resistor	Left leg	Forms part of the voltage divider circuit for the LDR	D33
	Right leg	Completes the voltage divider circuit for accurate LDR readings	GND
5V 4-Channel Relay Module	DC (+)	Powers the relay module	3V3
	DC (-)	Ground connection for the relay module	GND
	IN1	Relay input connected to ESP32 to control the first connected device	D23
	IN2	Relay input connected to ESP32 to control the second connected device	D26
	IN3	Relay input connected to ESP32 to control the third connected device	D25
	IN4	Relay input connected to ESP32 to control the fourth connected device	D19

Table 2: Pin Configuration of ESP-32

Component	PIN	Description	Wiring to ESP32
LCD 16x02 Display with I2C Module	GND	Ground pin for the LCD module	GND
	VCC	Powers the LCD module	3V3
	SDA	Serial Data Line for I2C communication	D21
	SCL	Serial Clock Line for I2C communication	D22

Table 3: LCD display pin configuration of ESP-32

Component	PIN	Description	Wiring to 5V 4-Channel Relay Module
Light Bulb 1	VCC (+)	Connects to the first relay to control Light Bulb 1	NO1
Light Bulb 2		Connects to the second relay to control Light Bulb 2	NO2
Light Bulb 3		Connects to the third relay to control Light Bulb 3	NO3
Light Bulb 4		Connects to the fourth relay to control Light Bulb 4	NO4

Table 4: Pin Configuration of 4-Channel Relay module

Component	PIN	Description	Wiring to 250V AC Power
Relay Module 1	COM1	Common terminal of the first relay	Live wire (+)
Relay Module 2	COM2	Common terminal of the second relay	
Relay Module 3	COM3	Common terminal of the third relay	
Relay Module 4	COM4	Common terminal of the fourth relay	
Light Bulb 1-4	GND (-)	Ground terminal for Light Bulbs	Ground (GND -)

Table 5: Pin Configuration of 250V AC Power

2.4 Source Code Explanation

Code Breakdown and Explanation:

1. Library Inclusions and Constants:

- The code begins by including essential libraries required for the functionality of the ESP32, Wi-Fi management, sensor data handling, and LCD display operations.
- Constants for the ESP RainMaker service (service_name, pop) and device names are defined. These constants are crucial for the identification and management of the devices in the RainMaker platform.

```
// Include necessary libraries
#include "RMaker.h"
#include "WiFi.h"
#include "WiFiProv.h"
#include <DHT.h>
#include <SimpleTimer.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Define constants for WiFi Provisioning
const char *service_name = "PROV_SmartHome";
const char *pop = "1234";

// Define chip ID and node name for ESP-RainMaker
uint32_t espChipId = 5;
char nodeName[] = "ESP32_Smarthome";

// Define names for the switch devices
char deviceName_1[] = "Switch1";
char deviceName_2[] = "Switch2";
char deviceName_3[] = "Switch3";
char deviceName_4[] = "Switch4";
```

Figure 5: Source Code for Library Inclusion and Constants

2. GPIO Pin Definitions and Variable Initialization:

- GPIO pins are assigned to various components like relays, switches, and sensors. This setup is fundamental for interfacing the ESP32 with these physical components.
- Variables for storing the state of relays, switches, and sensor data (temperature, humidity, LDR value) are initialized. These variables are critical for tracking the state of each component in the system.

```
// Define GPIO pins connected to relays and switches
static uint8_t RelayPin1 = 23;
static uint8_t RelayPin2 = 26;
static uint8_t RelayPin3 = 25;
static uint8_t RelayPin4 = 19;
static uint8_t SwitchPin1 = 13;
static uint8_t SwitchPin2 = 12;
static uint8_t SwitchPin3 = 14;
static uint8_t SwitchPin4 = 27;
static uint8_t wifiLed      = 2;   // WiFi LED Indicator
static uint8_t gpio_reset   = 0;   // Reset button

// Define pins and variables for sensors
static uint8_t DHTPIN       = 18; // D18  pin connected with DHT
static uint8_t LDR_PIN      = 33; // D33  pin connected with LDR

// Relay State
bool toggleState_1 = LOW; //Define integer to remember the toggle state for relay 1
bool toggleState_2 = LOW; //Define integer to remember the toggle state for relay 2
bool toggleState_3 = LOW; //Define integer to remember the toggle state for relay 3
bool toggleState_4 = LOW; //Define integer to remember the toggle state for relay 4

// Switch State
bool SwitchState_1 = LOW;
bool SwitchState_2 = LOW;
bool SwitchState_3 = LOW;
bool SwitchState_4 = LOW;

// Sensors Readings
float temperature1 = 0;
float humidity1   = 0;
float ldrVal  = 0;
```

Figure 6: Source Code for GPIO Pin Definitions & Variable Initialization

3. Sensor and Display Initialization:

- The DHT22 sensor for temperature and humidity measurements and the LCD display for output are initialized with their specific settings. This ensures that the sensor readings can be accurately taken and displayed.

```
// Initialize DHT22 sensor
DHT dht(DHTPIN, DHT22);

// Initialize LCD display
LiquidCrystal_I2C lcd(0x27, 16, 2); // Use the correct address as found by your I2C scanner

```

Figure 7: Source Code for DHT22 & LCD Display Initialization

4. RainMaker Configuration:

- Devices (switches and sensors) are configured within the ESP RainMaker platform. This allows for remote monitoring and control of these devices through the RainMaker application.

```
// Initialize a timer
SimpleTimer Timer;

// Define switch devices and sensors in ESP-RainMaker
//The framework provides some standard device types like switch, lightbulb, fan, temperature sensor.
static Switch my_switch1(deviceName_1, &RelayPin1);
static Switch my_switch2(deviceName_2, &RelayPin2);
static Switch my_switch3(deviceName_3, &RelayPin3);
static Switch my_switch4(deviceName_4, &RelayPin4);
static TemperatureSensor temperature("Temperature");
static TemperatureSensor humidity("Humidity");
static TemperatureSensor ldr("LDR");

```

Figure 8: Source Code for Devices Initialization in ESP-RainMaker

5. System Provisioning Event Handler:

- A function (sysProvEvent) is defined to handle provisioning events. This includes actions to be taken when the system starts the provisioning process and when it connects to Wi-Fi.

```
// Function to handle system provisioning events
void sysProvEvent(arduino_event_t *sys_event)
// Handle different events like provisioning start and WiFi connection
{
  // Callback function for handling changes in device parameters
  switch (sys_event->event_id) {
    case ARDUINO_EVENT_PROV_START:
#if CONFIG_IDF_TARGET_ESP32
      Serial.printf("\nProvisioning Started with name \"%s\" and PoP \"%s\" on BLE\n", service_name, pop);
      printQR(service_name, pop, "ble");
#else
      Serial.printf("\nProvisioning Started with name \"%s\" and PoP \"%s\" on SoftAP\n", service_name, pop);
      printQR(service_name, pop, "softap");
#endif
      break;
    case ARDUINO_EVENT_WIFI_STA_CONNECTED:
      Serial.printf("\nConnected to Wi-Fi!\n");
      digitalWrite(wifiLed, true);
      break;
  }
}
```

Figure 9: Source Code for System Provisioning Event Handler

6. Device Write Callback Function:

- The write_callback function is responsible for handling changes to the device's state (like a switch being toggled) and updating these changes in the RainMaker platform. This is essential for ensuring that the system's physical state is synchronized with its digital representation on RainMaker.

```
void write_callback(Device *device, Param *param, const param_val_t val, void *priv_data, write_ctx_t *ctx)
// Code to handle changes in switch states and update relay states accordingly
{
  const char *device_name = device->getDeviceName();
  const char *param_name = param->getParamName();
  if (strcmp(device_name, deviceName_1) == 0) {
    Serial.printf("Lightbulb = %s\n", val.val.b ? "true" : "false");
    if (strcmp(param_name, "Power") == 0) {
      Serial.printf("Received value = %s for %s - %s\n", val.val.b ? "true" : "false", device_name, param_name);
      toggleState_1 = val.val.b;
      (toggleState_1 == false) ? digitalWrite(RelayPin1, HIGH) : digitalWrite(RelayPin1, LOW);
      param->updateAndReport(val);
    }
  } else if (strcmp(device_name, deviceName_2) == 0) {
    Serial.printf("Switch value = %s\n", val.val.b ? "true" : "false");
    if (strcmp(param_name, "Power") == 0) {
      Serial.printf("Received value = %s for %s - %s\n", val.val.b ? "true" : "false", device_name, param_name);
      toggleState_2 = val.val.b;
      (toggleState_2 == false) ? digitalWrite(RelayPin2, HIGH) : digitalWrite(RelayPin2, LOW);
      param->updateAndReport(val);
    }
  } else if (strcmp(device_name, deviceName_3) == 0) {
    Serial.printf("Switch value = %s\n", val.val.b ? "true" : "false");
    if (strcmp(param_name, "Power") == 0) {

Serial.printf("Received value = %s for %s - %s\n", val.val.b ? "true" : "false", device_name, param_name);
      toggleState_3 = val.val.b;
      (toggleState_3 == false) ? digitalWrite(RelayPin3, HIGH) : digitalWrite(RelayPin3, LOW);
      param->updateAndReport(val);
    }
  } else if (strcmp(device_name, deviceName_4) == 0) {
    Serial.printf("Switch value = %s\n", val.val.b ? "true" : "false");
    if (strcmp(param_name, "Power") == 0) {
      Serial.printf("Received value = %s for %s - %s\n", val.val.b ? "true" : "false", device_name, param_name);
      toggleState_4 = val.val.b;
      (toggleState_4 == false) ? digitalWrite(RelayPin4, HIGH) : digitalWrite(RelayPin4, LOW);
      param->updateAndReport(val);
    }
  }
}
```

Figure 10: Source Code for Write Call back function

7. Sensor Reading and Data Sending:

- Functions for reading sensor data (readSensor) and sending this data to RainMaker (sendSensor) are defined. These functions ensure that the environmental data is regularly updated and available for remote monitoring.

```
// Function to read sensor data
void readSensor() {
  // Read and process data from DHT22 and LDR sensors
  ldrVal = map(analogRead(LDR_PIN), 400, 4200, 0, 100);
  Serial.print("LDR - "); Serial.println(ldrVal);

  float h = dht.readHumidity();
  float t = dht.readTemperature(); // or dht.readTemperature(true) for Fahrenheit
  if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
  else {
    humidity1 = h;
    temperature1 = t;
    Serial.print("Temperature - "); Serial.println(t);
    Serial.print("Humidity - "); Serial.println(h);
  }
}
// Function to send sensor data to ESP-RainMaker
void sendSensor()
{
  // Read sensor data and update ESP-RainMaker parameters
  readSensor();
  temperature.updateAndReportParam("Temperature", temperature1);
  humidity.updateAndReportParam("Temperature", humidity1);
  ldr.updateAndReportParam("Temperature", ldrVal);
}
```

Figure 11: Source Code for Sensor Reading & Data Sending

8. Manual Control Handling:

- The manual_control function checks the state of manual switches and updates the corresponding relay states. This allows for local control of devices, alongside remote control via RainMaker.

```
// Function to handle manual control of switches
void manual_control()
{
  // Check the state of physical switches and update the system state
  if (digitalRead(SwitchPin1) == LOW && SwitchState_1 == LOW) {
    digitalWrite(RelayPin1, LOW);
    toggleState_1 = 1;
    SwitchState_1 = HIGH;
    my_switch1.updateAndReportParam(ESP_RMAKER_DEF_POWER_NAME, toggleState_1);
    Serial.println("Switch-1 on");
  }
  if (digitalRead(SwitchPin1) == HIGH && SwitchState_1 == HIGH) {
    digitalWrite(RelayPin1, HIGH);
    toggleState_1 = 0;
    SwitchState_1 = LOW;
    my_switch1.updateAndReportParam(ESP_RMAKER_DEF_POWER_NAME, toggleState_1);
    Serial.println("Switch-1 off");
  }
  if (digitalRead(SwitchPin2) == LOW && SwitchState_2 == LOW) {
    digitalWrite(RelayPin2, LOW);
    toggleState_2 = 1;
    SwitchState_2 = HIGH;
    my_switch2.updateAndReportParam(ESP_RMAKER_DEF_POWER_NAME, toggleState_2);
    Serial.println("Switch-2 on");
  }
  if (digitalRead(SwitchPin2) == HIGH && SwitchState_2 == HIGH) {
    digitalWrite(RelayPin2, HIGH);
    toggleState_2 = 0;
    SwitchState_2 = LOW;
    my_switch2.updateAndReportParam(ESP_RMAKER_DEF_POWER_NAME, toggleState_2);
    Serial.println("Switch-2 off");
  }
  if (digitalRead(SwitchPin3) == LOW && SwitchState_3 == LOW) {
    digitalWrite(RelayPin3, LOW);
    toggleState_3 = 1;
    SwitchState_3 = HIGH;
    my_switch3.updateAndReportParam(ESP_RMAKER_DEF_POWER_NAME, toggleState_3);
    Serial.println("Switch-3 on");
  }
  if (digitalRead(SwitchPin3) == HIGH && SwitchState_3 == HIGH) {
    digitalWrite(RelayPin3, HIGH);
    toggleState_3 = 0;
    SwitchState_3 = LOW;
    my_switch3.updateAndReportParam(ESP_RMAKER_DEF_POWER_NAME, toggleState_3);
    Serial.println("Switch-3 off");
  }
  if (digitalRead(SwitchPin4) == LOW && SwitchState_4 == LOW) {
    digitalWrite(RelayPin4, LOW);
    toggleState_4 = 1;
    SwitchState_4 = HIGH;
    my_switch4.updateAndReportParam(ESP_RMAKER_DEF_POWER_NAME, toggleState_4);
    Serial.println("Switch-4 on");
  }
  if (digitalRead(SwitchPin4) == HIGH && SwitchState_4 == HIGH) {
    digitalWrite(RelayPin4, HIGH);
    toggleState_4 = 0;
    SwitchState_4 = LOW;
    my_switch4.updateAndReportParam(ESP_RMAKER_DEF_POWER_NAME, toggleState_4);
    Serial.println("Switch-4 off");
  }
}
```

Figure 12: Source Code for Manual Control Function

9. Setup Routine:

- The setup function includes initializing the serial communication, setting GPIO modes, starting sensors, configuring the RainMaker node and devices, and beginning Wi-Fi provisioning. This routine is crucial for starting up the system with the correct configurations.

```
// Setup function - initializes the system
// Initialize LCD, Serial communication, GPIOs, ESP-RainMaker node, devices, and sensors
// Configure WiFi provisioning
void setup()
{
  // Initialize the LCD and turn on the backlight
  lcd.init();
  lcd.backlight();
  // Start the serial communication
  Serial.begin(115200);
  // Set the Relays GPIOs as output mode
  pinMode(RelayPin1, OUTPUT);
  pinMode(RelayPin2, OUTPUT);
  pinMode(RelayPin3, OUTPUT);
  pinMode(RelayPin4, OUTPUT);
  pinMode(wifiLed, OUTPUT);
  // Configure the input GPIOs
  pinMode(SwitchPin1, INPUT_PULLUP);
  pinMode(SwitchPin2, INPUT_PULLUP);
  pinMode(SwitchPin3, INPUT_PULLUP);
  pinMode(SwitchPin4, INPUT_PULLUP);
  pinMode(gpio_reset, INPUT);
// Write to the GPIOs the default state on booting
  digitalWrite(RelayPin1, !toggleState_1);
  digitalWrite(RelayPin2, !toggleState_2);
  digitalWrite(RelayPin3, !toggleState_3);
  digitalWrite(RelayPin4, !toggleState_4);
  digitalWrite(wifiLed, LOW);
  dht.begin();    // Enabling DHT sensor

  Node my_node;
  my_node = RMaker.initNode(nodeName);
  //Standard switch device
  my_switch1.addCb(write_callback);
  my_switch2.addCb(write_callback);
  my_switch3.addCb(write_callback);
  my_switch4.addCb(write_callback);
  //Add switch device to the node
  my_node.addDevice(my_switch1);
  my_node.addDevice(my_switch2);
  my_node.addDevice(my_switch3);
  my_node.addDevice(my_switch4);
  my_node.addDevice(temperature);
  my_node.addDevice(humidity);
  my_node.addDevice(ldr);
  Timer.setInterval(20000);
  //This is optional
  RMaker.enableOTA(OTA_USING_PARAMS);
  //If you want to enable scheduling, set time zone for your region using setTimeZone().
  //The list of available values are provided here https://rainmaker.espressif.com/docs/time-service.html
  // RMaker.setTimeZone("Asia/Shanghai");
  // Alternatively, enable the Timezone service and let the phone apps set the appropriate timezone
  RMaker.enableTZService();
  RMaker.enableSchedule();
  //Service Name
  for (int i = 0; i < 17; i = i + 8) {
    espChipId |= ((ESP.getEfuseMac() >> (40 - i)) & 0xff) << i;
  }
  Serial.printf("\nChip ID:  %d Service Name: %s\n", espChipId, service_name);
  Serial.printf("\nStarting ESP-RainMaker\n");
  RMaker.start();
  WiFi.onEvent(sysProvEvent);
#if CONFIG_IDF_TARGET_ESP32
  WiFiProv.beginProvision(WIFI_PROV_SCHEME_BLE, WIFI_PROV_SCHEME_HANDLER_FREE_BTDM, WIFI_PROV_SECURITY_1, pop, service_name);
#else
  WiFiProv.beginProvision(WIFI_PROV_SCHEME_SOFTAP, WIFI_PROV_SCHEME_HANDLER_NONE, WIFI_PROV_SECURITY_1, pop, service_name);
#endif
  my_switch1.updateAndReportParam(ESP_RMAKER_DEF_POWER_NAME, false);
  my_switch2.updateAndReportParam(ESP_RMAKER_DEF_POWER_NAME, false);
  my_switch3.updateAndReportParam(ESP_RMAKER_DEF_POWER_NAME, false);
  my_switch4.updateAndReportParam(ESP_RMAKER_DEF_POWER_NAME, false);
}
```

Figure 13: Source Code for Setup function

10. Main Loop:

The main loop of the program includes:

- Checking and handling the reset button functionality.
- Monitoring Wi-Fi connection status and updating an LED indicator based on this status.
- Continuously performing manual control checks, sensor readings, and updating the display.
- This loop is where the system continuously monitors and responds to changes in its environment and control inputs.

```
void loop() {
  // Handle reset button, WiFi status, manual control, sensor readings, and display updates
  // Read GPIO0 (external button to reset device)
  if (digitalRead(gpio_reset) == LOW) { // Push button pressed
    Serial.printf("Reset Button Pressed!\n");
    // Key debounce handling
    delay(100);
    int startTime = millis();
    while (digitalRead(gpio_reset) == LOW) delay(50);
    int endTime = millis();
    if ((endTime - startTime) > 10000) {
      // If key pressed for more than 10secs, reset all
      Serial.printf("Reset to factory.\n");
      RMakerFactoryReset(2);
    } else if ((endTime - startTime) > 3000) {
      Serial.printf("Reset Wi-Fi.\n");
      // If key pressed for more than 3secs, but less than 10, reset Wi-Fi
      RMakerWiFiReset(2);
    }
  }
  // Check Wi-Fi connection status and update the LED accordingly
  if (WiFi.status() != WL_CONNECTED) {
    digitalWrite(wifiLed, false);
  } else {
    digitalWrite(wifiLed, true);
    if (Timer.isReady()) {
      sendSensor();
      Timer.reset(); // Reset the timer
    }
  }
  // Perform manual control of the switches
  manual_control();

  // Read humidity and temperature values
  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature();

  // Check if any reads failed and exit early (to try again)
  if (isnan(humidity) || isnan(temperature)) {
    Serial.println("Failed to read from DHT sensor!");
  } else {

      // Update the display with the new temperature and humidity values
      lcd.setCursor(0, 0); // First line of the LCD
      lcd.print("Temp: ");
      lcd.print(temperature, 1); // One decimal place for temperature
      lcd.write(223); // Degree symbol
      lcd.print("C");

      lcd.setCursor(0, 1); // Second line of the LCD
      lcd.print("Humidity: ");
      lcd.print(humidity, 1); // One decimal place for humidity
      lcd.print("%");
    }

    delay(2000); // Wait for 2 seconds
  }
```

Figure 14: Source Code for Loop Function

3. Proof of Concept (Demo)

3.1 Manual Switch

The system includes manual switches that provide a traditional way of interacting with home appliances. This feature ensures that users can control their devices even in the absence of voice commands or remote access.

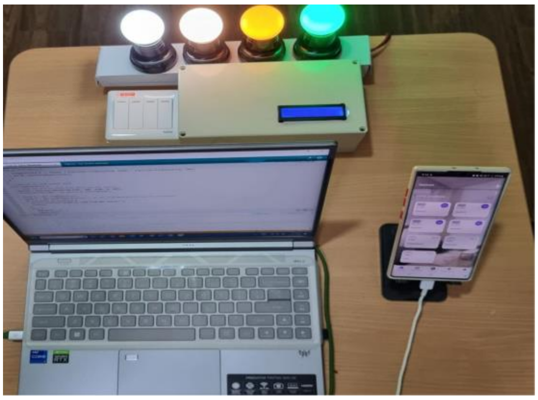


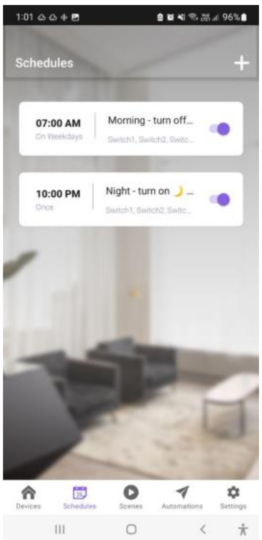
Figure 15: Prototype for ESP32 Smart Home Automation with Voice Control Integration

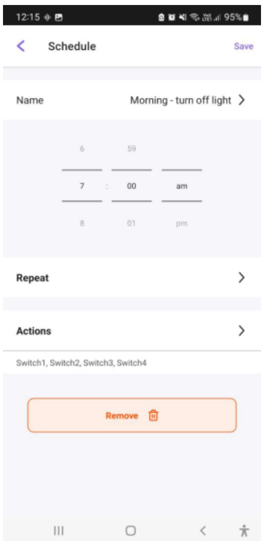
3.2 ESP Rainmaker App

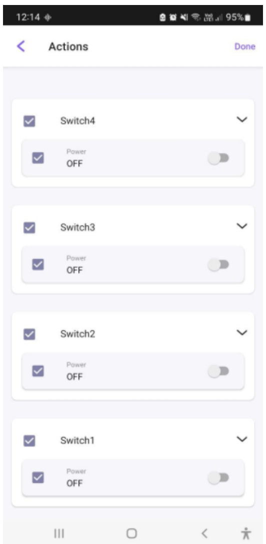
The ESP Rainmaker app plays a crucial role in the system for scheduling and automation.

- **Scheduling:**

Users can set specific times for certain actions, like turning lights on or off at predetermined times.

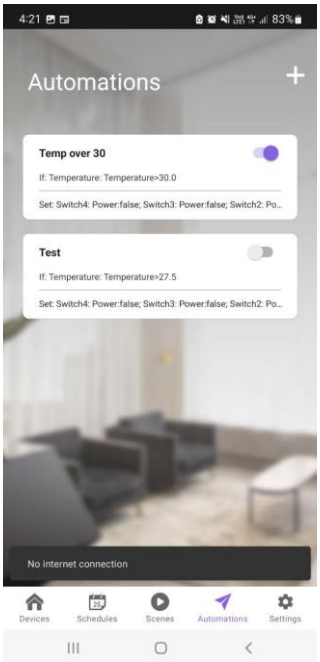


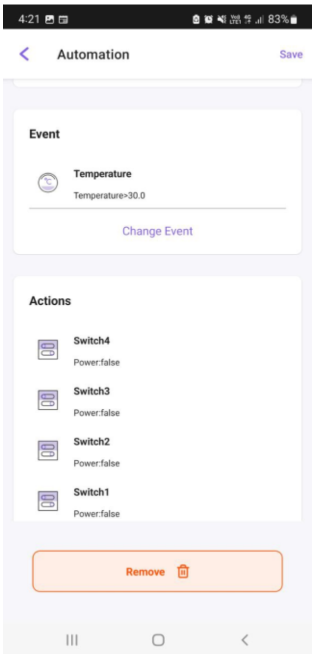




- **Automation:**

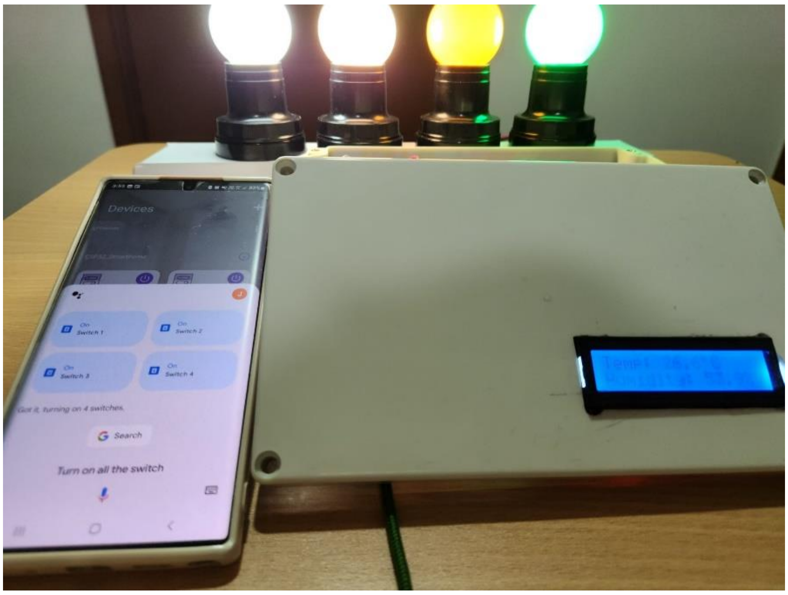
The app also allows for automation rules, where the system can perform actions based on certain triggers or conditions. For example, turning off lights automatically when the temperature exceeds 30 degrees calcium and there will be notification to notify the user.





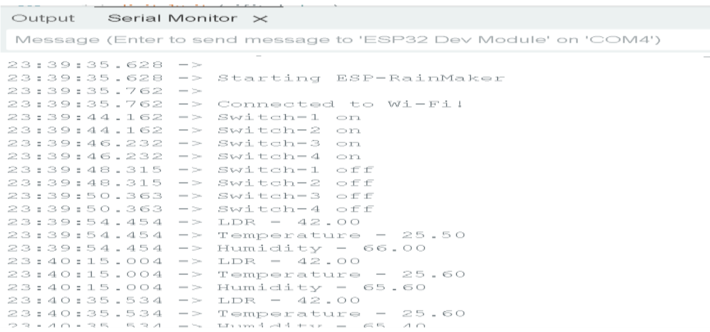
3.3 Google Assistant Voice Control Integration

A key feature of the system is the integration with Google Assistant, enabling voice control. Users can issue voice commands to control various home devices.



3.4 Arduino IDE Serial Monitor

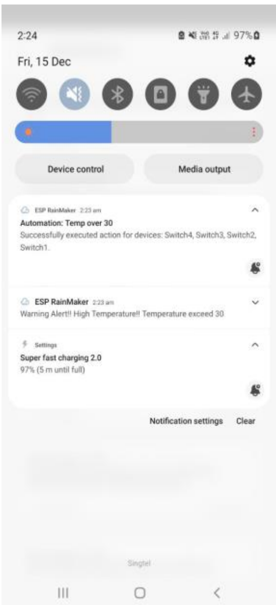
The Arduino IDE’s serial monitor is used to display system messages for real-time feedback and diagnostics. This feature is crucial for troubleshooting and understanding the system's responses to various commands and actions.

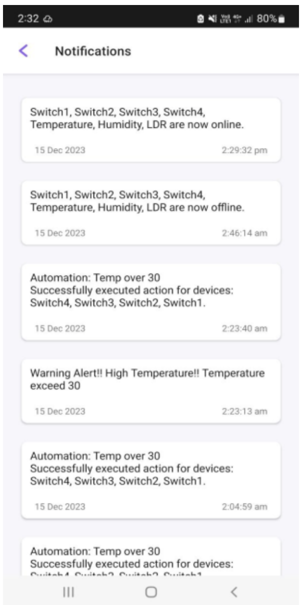


3.5 Notifications

The notification system in the ESP Rainmaker app plays a crucial role in informing users about the status and activities of their smart home system:

- **Automation Triggers:** The app notifies users when automation rules are activated, like "Light turned off when temperature exceed 30”
- **System Status Updates:** Notifications also provide updates on the Rainmaker system's connectivity, displaying messages like "Rainmaker System: Online" or "Rainmaker System: Offline". This keeps users informed about the operational status of their smart home system.





4. Implementation Cost

S/N	Hardware Components	Location/URL	Cost (SGD)
1	ESP-WROOM-32	**Shopee**, https://shp.ee/vyfiwg2	5.89
2	4 Light Bulb 230V	**CONTINENTIAL ELECTRONICS**, 10 Jalan Besar, Sim Lim Tower, #B1-25, 208787	16
3	4 Manual Switches		12
4	4 E27 holder (Light Bulb)		6
5	Project Box Waterproof		12
6	DHT22	**Kuriosity**, 10 Jalan Besar, Sim Lim Tower, #03-49, 208787	8.20
7	LDR		0.40
8	5V 4-Channel Relay Module		9.20
9	LCD 16x02 Display with I2C Module		10.60
10	Solderless Breadboard 830 Tie		4
11	10K Ohms Resistor	**Shopee**, https://shp.ee/gknr4wq	3.78
12	Jumper Wires (40pcs M-M, 40pcs, F-M, 40pcs, F-F)	**Shopee**, https://shp.ee/ht6hnaw	4.19
TOTAL			**92.26**

Table 6 for Implementation costs

5. System’s Advantages and Disadvantages

Advantages:

1. **Enhanced Accessibility:**

The integration of voice control significantly increases the system's accessibility, making it extremely user-friendly for individuals with diverse abilities, including those with mobility or visual impairments.

2. **Convenience and Efficiency:**

Users can effortlessly control and automate home appliances through voice commands, which enhances the convenience and efficiency of managing daily household tasks.

3. **Cost-Effective Solution:**

Utilizing the ESP32 microcontroller, noted for its affordability, in conjunction with other economical components, positions this system as a budget-friendly alternative to many commercial home automation systems.

4. **Customization and Scalability:**

The flexibility provided by the open-source Arduino IDE and the adaptable nature of the ESP32 allows for extensive customization and scalability to meet specific user needs and preferences.

5. **Remote Control and Monitoring:**

The ESP-RainMaker integration enables users to monitor and control their home systems remotely, adding a layer of convenience and enhanced security, particularly beneficial when users are not physically present at home.

Disadvantages:

1. **Internet Dependence:**

The system's performance is heavily reliant on a stable internet connection. This dependency is particularly critical for remote operations and processing voice commands, which could be impaired during connectivity issues.

2. **Complex Setup for Novices:**

For individuals without prior experience in IoT or microcontroller-based systems, the initial setup and customization of the system might be challenging and require a steep learning curve.

3. **Voice Recognition Limitations:**

The accuracy of voice recognition can be affected by various external factors, such as ambient noise, different accents, or speech variances, occasionally leading to misunderstandings or incorrect execution of commands.

4. **Security and Privacy Concerns:**

As with many IoT systems, there exists a risk of security breaches. Implementing robust security measures is critical to safeguard user data and maintain privacy.

5. **Limited MQTT Support in ESP-RainMaker:**

While ESP-RainMaker provides several advantages for managing IoT devices, its limited support for MQTT, a prevalent IoT communication protocol, could restrict the system's integration with certain devices or systems reliant on MQTT.

6. Alternative Solutions and Suggestions:

- **Enhanced Connectivity Solutions:**

For mitigating internet dependency, integrate alternative connectivity options like LAN or mesh networks to provide backup connectivity solutions.

- **User-Friendly Setup Guides:**

Develop comprehensive tutorials and guides to assist novices in setting up and customizing the system, potentially paired with a user-friendly app for easier configuration.

- **Advanced Voice Recognition:**

Incorporate more sophisticated voice recognition software capable of better handling diverse accents and reducing errors due to ambient noise.

- **Strengthened Security Protocols:**

Implement advanced encryption and regular security updates to protect against potential cyber threats and enhance user trust in the system.

- **MQTT Protocol Integration:**

To address the limited MQTT support in ESP-RainMaker, consider the possibility of integrating additional software layers or platforms that offer broader protocol support, ensuring greater compatibility with a wide range of IoT devices and services.

7. Summary

This report details the development of an ESP32-Based Home Automation system, an innovative project that seamlessly integrates voice control using Google Assistant into home management. The system is designed to address the challenges in current home automation solutions, such as complexity, high costs, and lack of universal accessibility, especially for individuals with disabilities.

The project leverages the ESP-WROOM-32 module for its Wi-Fi capabilities and cost-effectiveness, making it ideal for IoT applications. The integration with Google Assistant allows for intuitive voice commands, enhancing user interaction. Key components include switches, relay modules, DHT22 and LDR sensors, and an LCD display, all chosen for their functionality and affordability.

The system architecture combines these components into a unified subsystem, providing a cohesive and user-friendly experience. It features environmental sensing for automated responses, manual controls, and remote access via ESP-RainMaker.

Key challenges addressed include the system's reliance on a stable internet connection, the complexity of setup for novices, limitations in voice recognition accuracy, and security concerns. To mitigate these, the report suggests solutions such as alternative connectivity options, user-friendly setup guides, advanced voice recognition technologies, and robust security protocols.

The total implementation cost of SGD 92.26 highlights the system's cost-effectiveness. Overall, the ESP32-Based Home Automation system demonstrates a significant advancement in making smart home technology more accessible, user-friendly, and affordable, paving the way for broader adoption in residential spaces.

8. References

- *Arduino IDE 2 | Arduino Documentation*. https://docs.arduino.cc/software/ide-v2
- Arduino Library List. Arduino Libraries. https://www.arduinolibraries.info/
- *DHT22 – Temperature and humidity Sensor*. Components101. https://components101.com/sensors/dht22-pinout-specs-datasheet
- *ESP RainMaker with ESP32 – Voice Assistant Integration*. https://circuitdigest.com/microcontroller-projects/esp-rainmaker-tutorial-esp32-alexa-google-voice-assistant
- *Introduction | ESP RAINMAKER*. https://rainmaker.espressif.com/docs/intro
- Lab, M. (2022, December 5). *ESP RainMaker Getting Started Tutorial with ESP32 and Arduino IDE*. Microcontrollers Lab. https://microcontrollerslab.com/esp-rainmaker-tutorial-esp32-arduino-ide/
- *LDR (Light Dependent Resistor) or photoresistor*. Components101. https://components101.com/resistors/ldr-datasheet
- Mohanan, V. (2023, November 10). *DOIT ESP32 DevKit V1 Wi-Fi Development Board - Pinout Diagram & Arduino Reference - CIRCUITSTATE*. CIRCUITSTATE Electronics. https://www.circuitstate.com/pinouts/doit-esp32-devkit-v1-wifi-development-board-pinout-diagram-and-reference/