MINOR-I PROJECT REPORT

"INTELLIGENT SOCKET"

Easy & Cost-Effective Home Automation

Project By:

Tanishq Manuja (17802006)

Suryansh Agrawal (17102020)

Submitted To:

Dr Rachna Singh

Dept. of Electronics and Communication Engineering

Jaypee Institute of Information Technology

ABSTRACT

To develop a cost-effective human automation device using easily available components that can automate even dumb devices (device without pre-built Wi-Fi), as smart devices available in market a highly priced and also buying new appliances, will make your existing ones redundant.

Socket can be controlled manually, using mobile application or using voice commands. It will also be able to measure real time power consumption to save electricity which will benefit the environment.

CERTIFICATE

This is to certify that Tanishq Manuja and Suryansh Agrawal, students of Electronics and Communication Engineering, has successfully completed the project on the topic "Intelligent Sockets" under the guidance of **Dr. Rachna Singh** during the year 2019.

This project is absolutely genuine and does not indulge in plagiarism of any kind. The reference taken in making this project have been declared at the end of this report.

Signature

(Dr. Rachna Singh)

Signature

(Dr. Ajay Kumar)

ACKNOWLEDGEMENT

It gives us immense pleasure to express our deepest sense of gratitude and sincere thanks to my highest respected and esteemed guide **Dr. Rachna Singh** for her valuable guidance, encouragement and help for completing this work. Her useful suggestions for this whole work and co-operative behaviour are sincerely acknowledged. We would like to express our sincere thanks to her for giving us this opportunity to undertake this project. We would also like to express our indebtedness to our parents as well as our family members whose blessings and support always helped us to face the challenges ahead.

CANDIDTATE'S DECLARATION

We (Tanishq Manuja & Suryansh Agrawal) hereby declare that the work presented in this report entitled "Intelligent Socket", in fulfilment of the requirement for the award of the degree of Bachelor of Technology in Electronics and Communication, submitted in Electronics and Communication Department, affiliated to Jaypee Institute of Information Technology, Sector 62 Noida is an authentic record of our own work carried out during our degree. The work reported in this has not been by us for award of any other degree or diploma.

Signature

(Tanishq Manuja)

Signature

(Suryansh Agrawal)

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

Project Outline:

Smart Socket build around ESP8266 microcontroller packaged in NodeMCU development board which has Wi-Fi inbuilt and can control various hardware elements like relay (10Amp AC) and current sensor (5Amp ACS712) using it's onboard GPIO Pins. Combining these will allow us to control the socket from anywhere in the world via an available Wi-Fi connection.

Google firebase is used as backend and extended frontend service, example: -
Firebase real-time database as the main storage for application, Firebase hosting for web application service, Firebase functions for smart home API and data management. Also, Google's Smart Home API is used to make the socket a part of google *Home* infrastructure which allows it to be controlled by voice command in google assistant and other supported *Home* devices.

Key Features:

Wi-Fi on/off control using mobile application. Also control and query the device using voice assistant from your mobile phone. Device also has power consumption monitoring features and can handle up to 1100W of power through the device. Application also has over consumption warning feature present in the web application

COMPONENTS DESCRIPTION

Section 2.1: Node MCU

NodeMCU is an open source IoT platform. It includes firmware which runs on the **ESP8266** Wi-Fi SoC from Espressif Systems, and hardware which is based on the ESP-12 module. The term "**NodeMCU**" by default refers to the firmware rather than the development kits.

Developer	*ESP8266 Open source Community*
Type	*Single-board microcontroller*
Operating system	*XTOS*
CPU	*ESP8266(LX106)*
Memory	*128kBytes*
Storage	*4MBytes*
Power	*USB*

1. Node MCU History

NodeMCU was created shortly after the ESP8266 came out. On December 30, 2013, Espressif Systems began production of the ESP8266. The ESP8266 is a Wi-Fi SoC integrated with a Tensilica Xtensa LX106 core, widely used in IoT application. NodeMCU started on 13 Oct 2014, when Hong committed the first file of nodemcu-firmware to GitHub.

They needed to modify the Arduino IDE so that it would be relatively easy to change the IDE to support alternate toolchains to allow Arduino C/C++ to be compiled for these new processors. They did this with the introduction of the Board Manager and the SAM Core. A "core" is the collection of software components required by the Board Manager and the Arduino IDE to compile an Arduino C/C++ source file for the target MCU's machine language.

2. *Pins*

NodeMCU provides access to the GPIO (General Purpose Input/Output) and a pin mapping table is part of the API documentation.

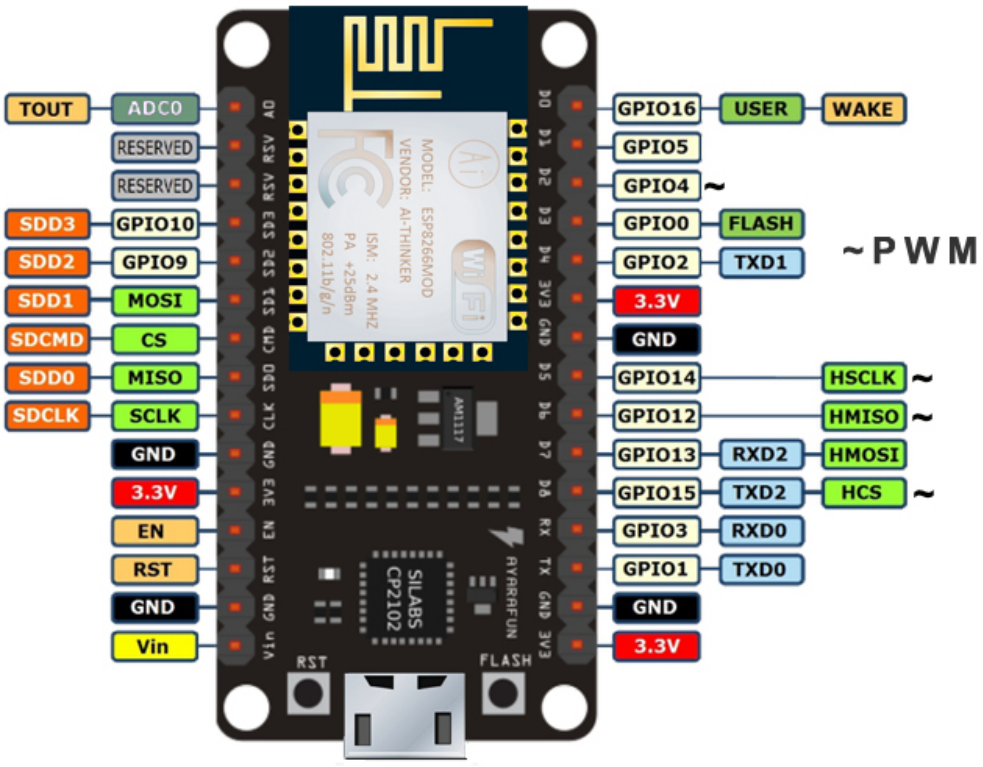

Fig 2.1.1: NodeMCU Pinout

D0 (GPIO16) can only be used for GPIO read/write. It does not support open-drain/interrupt/PWM/I²C or 1-Wire.

Section 2.2: ACS712_05B (Current Sensor)

Sensing and controlling current flow are a fundamental requirement in a wide variety of applications including, over-current protection circuits, battery chargers, switching mode power supplies, digital watt meters, programmable current sources, etc. This ACS721 current module is based on ACS712 sensor, which can accurately detect AC or DC current. The maximum AC or DC that can be detected can reach 5A, and the present current signal can be read via analog I / O port of Arduino.

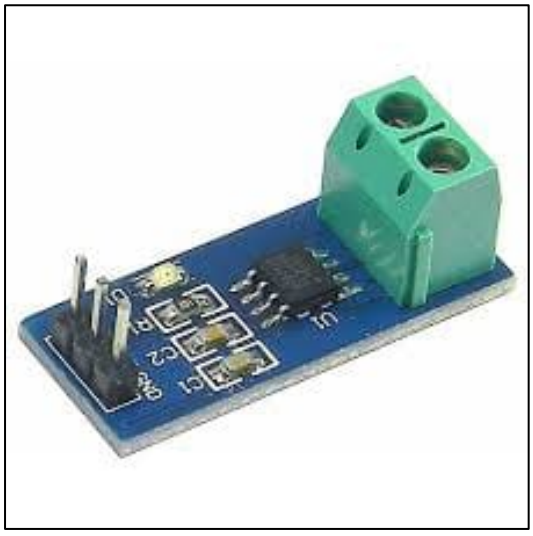

Fig 2.2.1: ACS712 5A (Current Sensor)

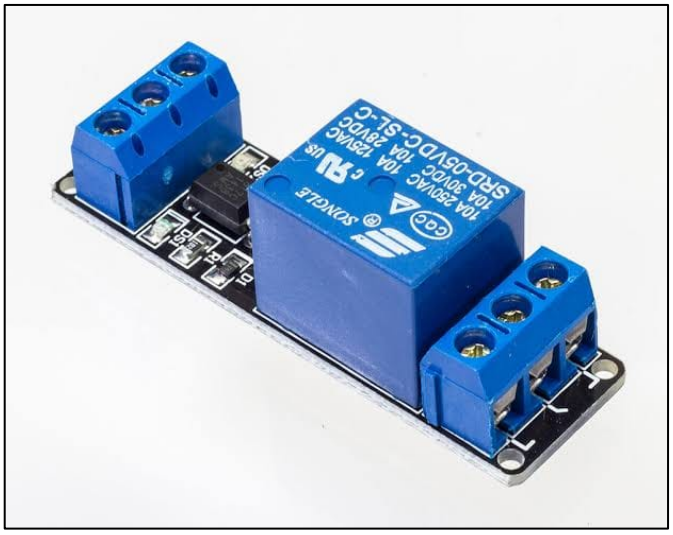

Fig 2.3.1: Electro-Mechanical Relay Module

Section 2.3: Electromagnetic Relay

A **relay** is an electrically operated switch. It consists of a set of input terminals for a single or multiple control signals, and a set of operating contact terminals. The switch may have any number of contacts in multiple contact forms, such as make contacts, break contacts, or combinations thereof.

The traditional form of a relay uses an electromagnet to close or open the contacts, but other operating principles have been invented, such as in solid-state relays which use semiconductor properties for control without relying on moving parts. Relays with calibrated operating characteristics and sometimes multiple operating coils are used to protect electrical circuits from overload or faults; in modern electric power systems these functions are performed by digital instruments still called protective relays.

RESOURCE FLOW

The Google firebase is connected to Node MCU and the Android Web App. In the socket device, the ESP8266 Node MCU is connected to Relay, which is further connected to Current Sensor.

Actions on Google are being performed via Firebase. The Firebase is basically acting as the link between Android Web app and the Node MCU.

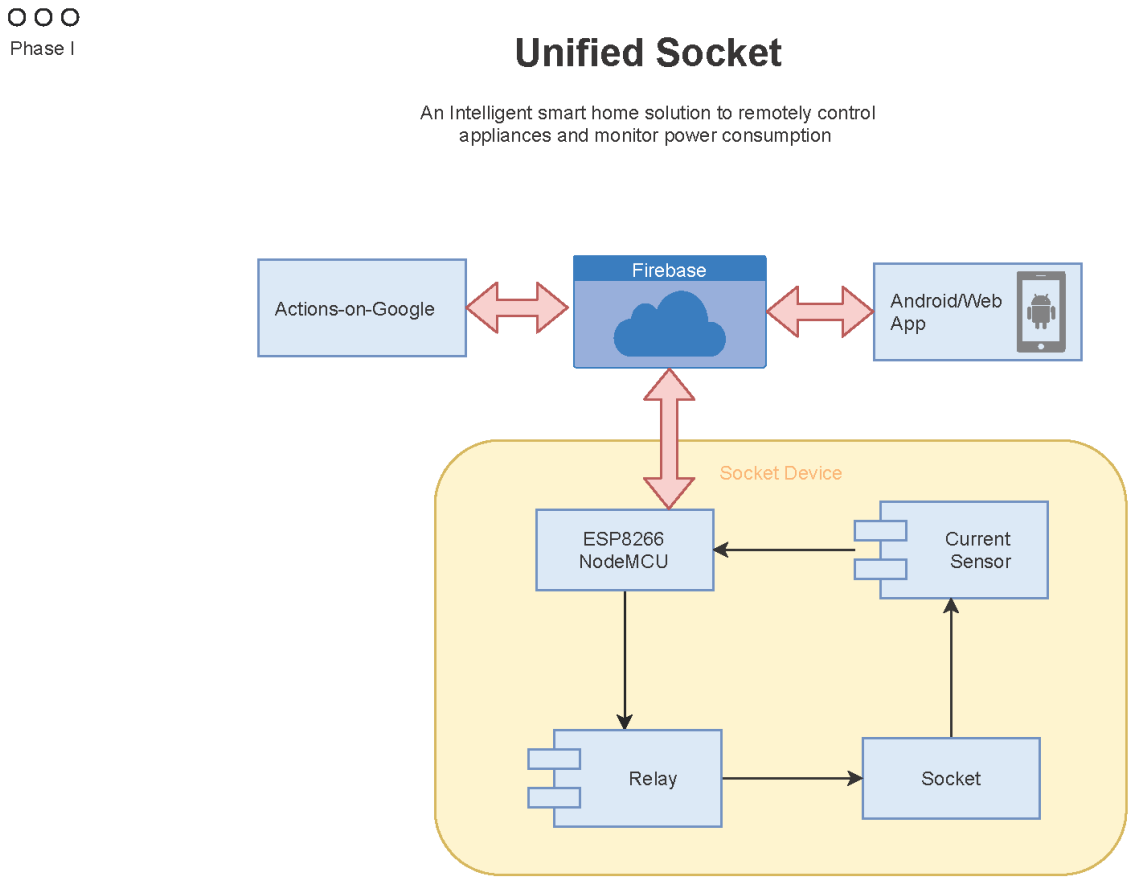

Fig 3.1: Basic Visualization of Socket

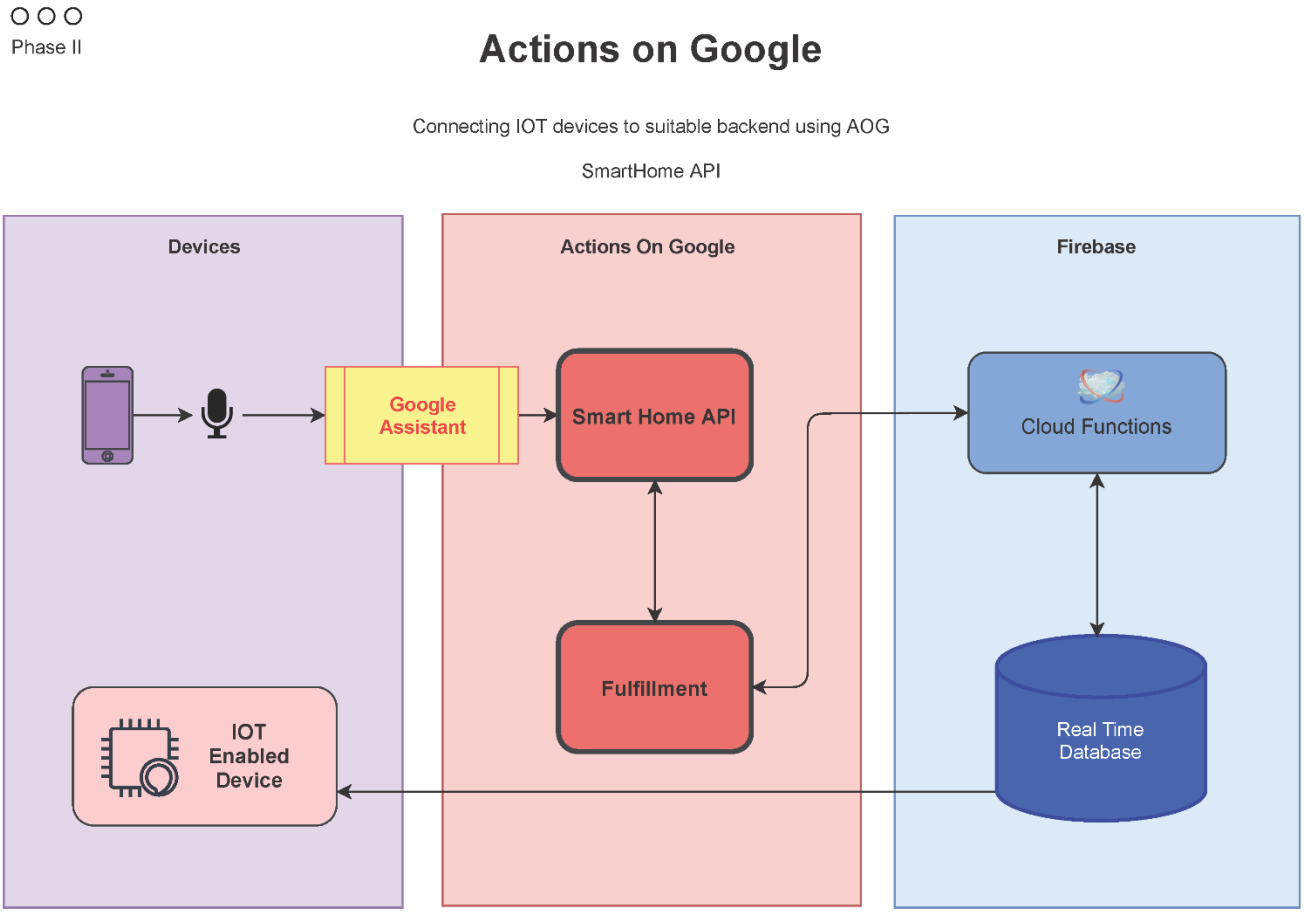

Fig 3.2: Working of Back-End

The device is connected to Smart Home API through Google Assistant. Then, fulfilment will check the information and pass it on to the Cloud Functions and further to Real Time Database. When the above information is processed, Node MCU will switch on the socket.

WORKING

This section explains the working of final product made, mostly using pictorial representation for easy understanding.

1) Circuit Diagram

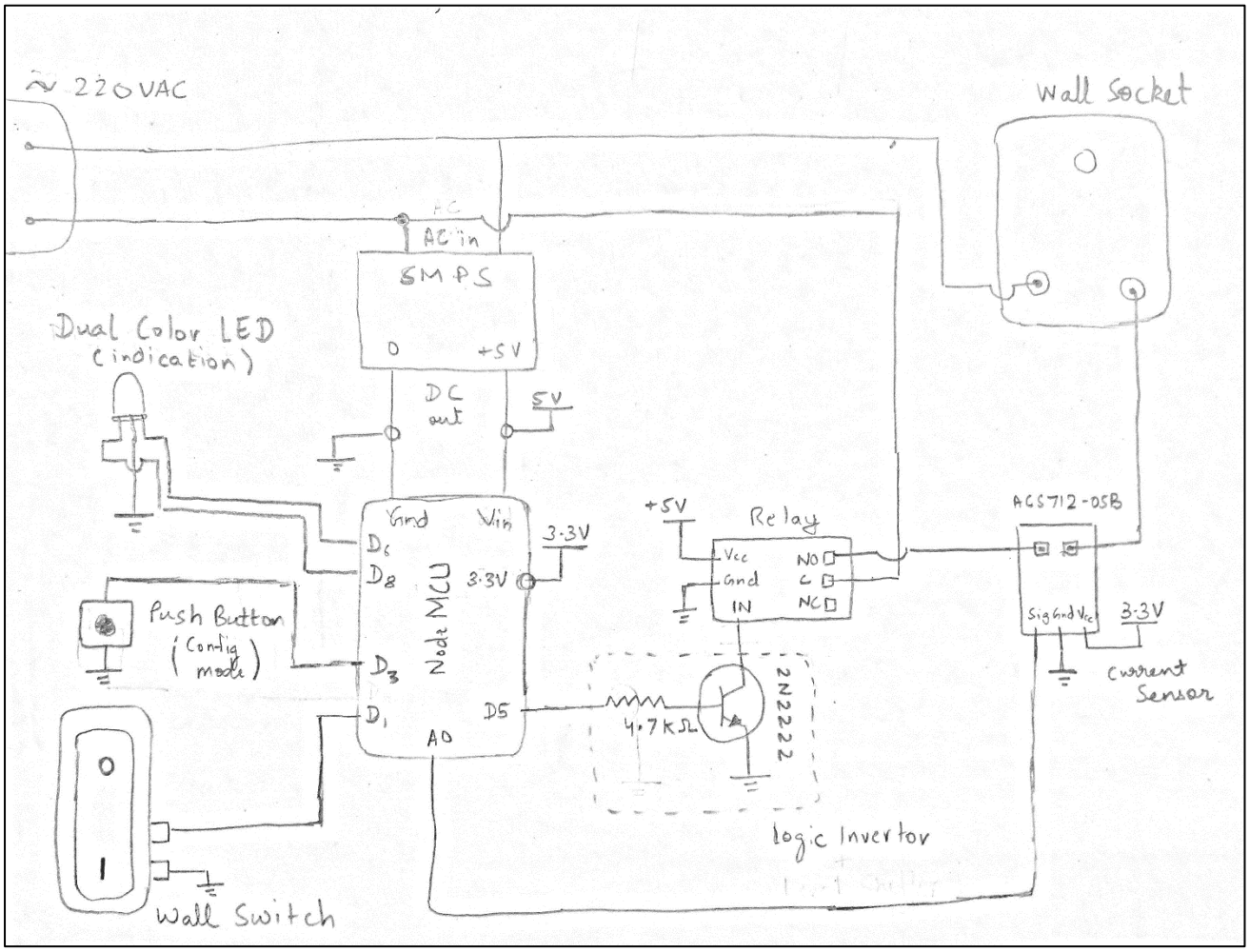

Fig 4.1: Circuit Diagram

2) Actual Circuit Construction

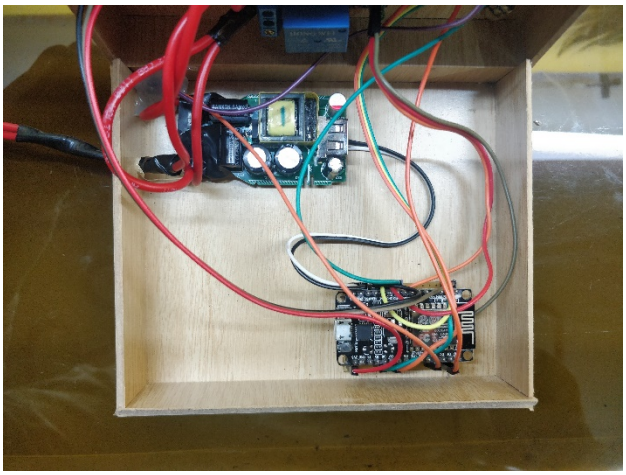

Fig 4.2i: Actual Circuit

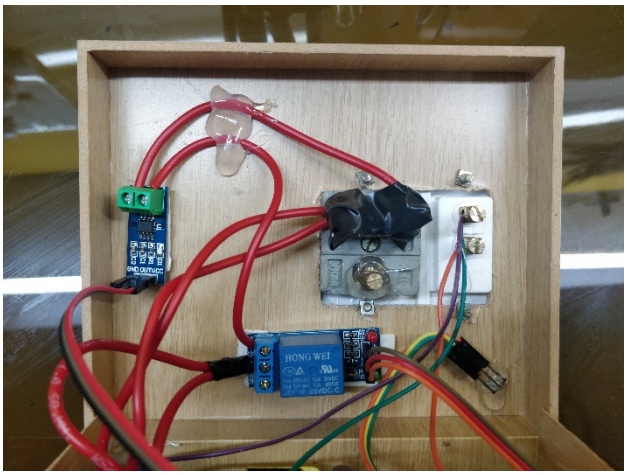

Fig 4.2ii: Actual Circuit

3) Overview of the Socket

a. Front View

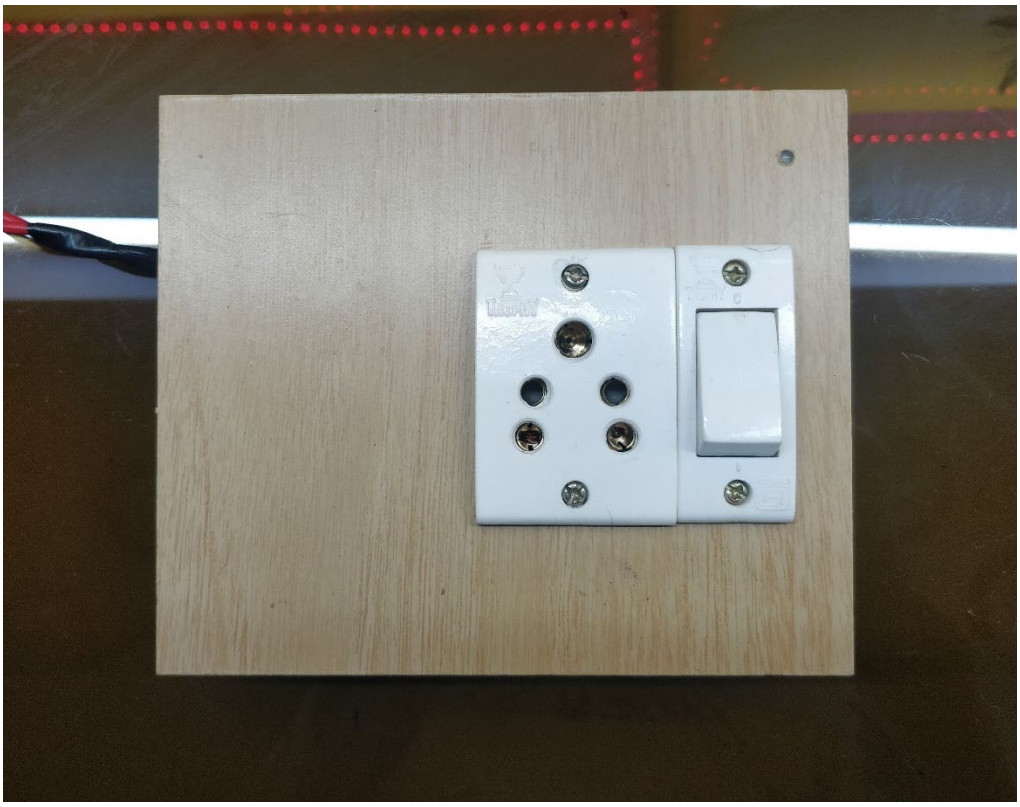

Fig 4.3a: Front View of Socket

b. Wi-Fi Configuration Button

Fig 4.3b: Side View of Socket (showing config button)

c. Wi-Fi Status LED

3 colours showing different Wi-Fi status

- Green: Wi-Fi Connected [Fig 8.3c Left]
- Orange: Wi-Fi Configuration Mode [Fig 8.3c Middle]
- Red: Wi-Fi Disconnected [Fig 8.3c Right]

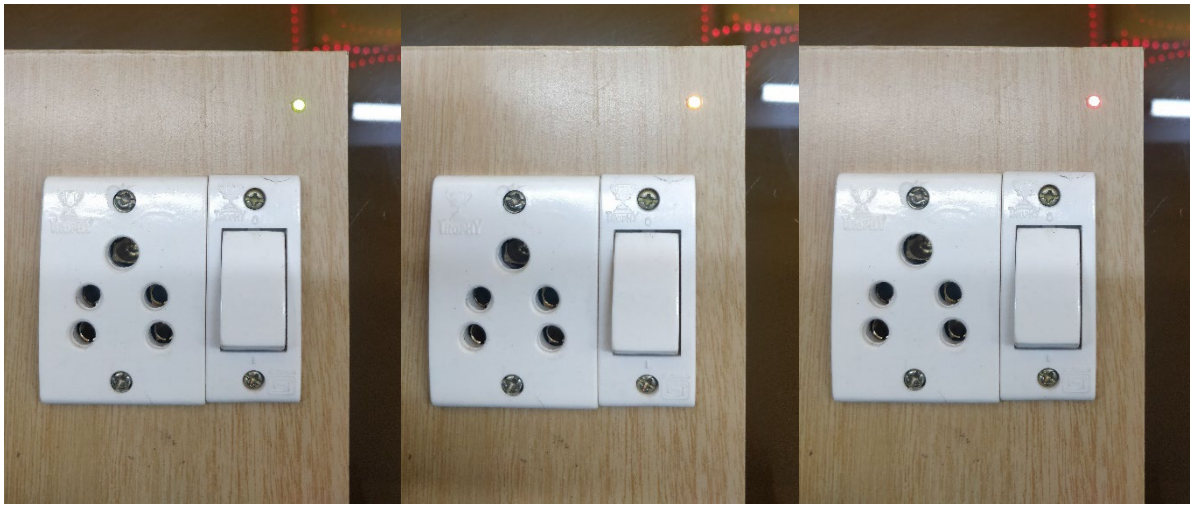

Fig 4.3c: Different LED Colours

4) Overview of App

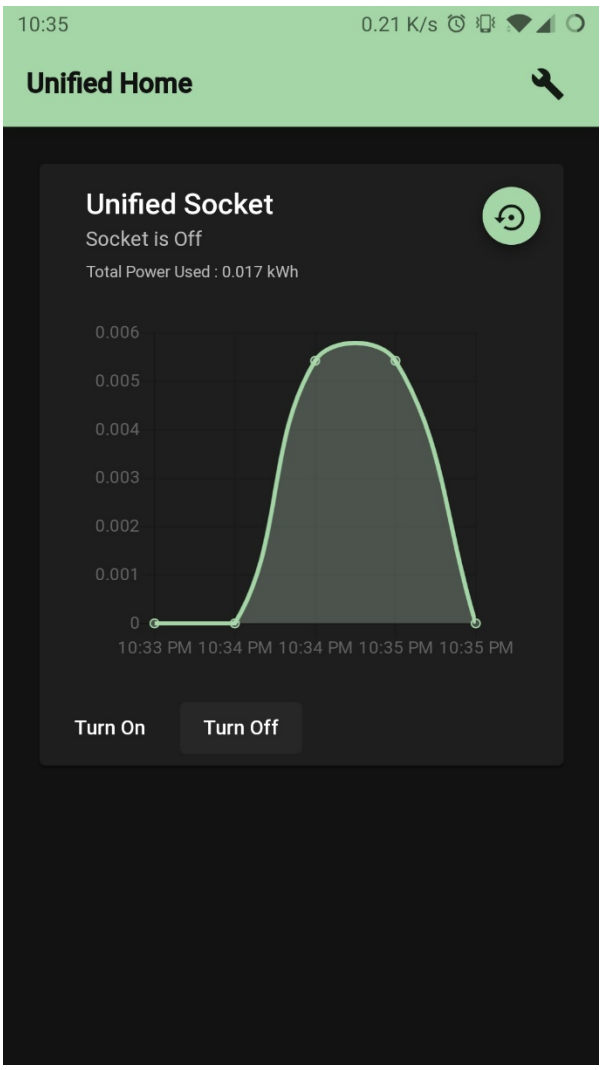

Fig 4.4.1: Power Consumption in App

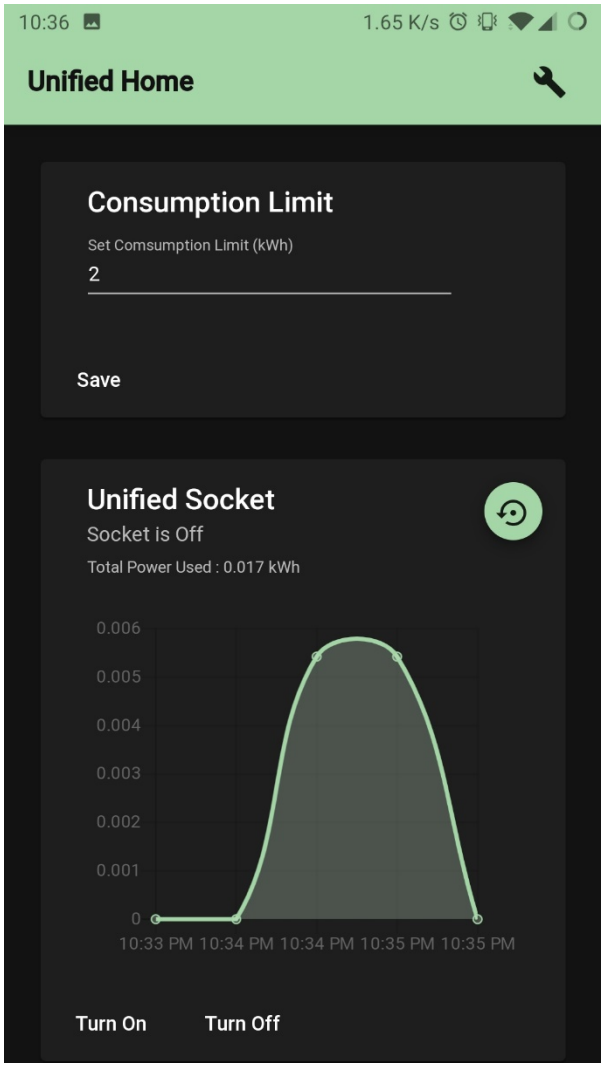

Fig 4.4.2: Set Power Consumption Limit

CONCLUSION

The objective of this project has been achieved which was developing the hardware and software for the *Intelligent Socket*. From observation that has been made, it clearly shows that it's working is accurate, & is easy to control and user friendly to use. The device has been developed successfully. The device also has fail-safe features to handle Wi-Fi failure.

FUTURE SCOPE

- Connecting Multiple Sockets in a Mesh Network
- Increasing Device/MCU Ratio
- Fail-Safe feature if Microcontroller fails
- Shrinking Size and Power Consumption of socket

REFERENCES

Websites:

- https://developers.google.com/actions/smarthome/develop
- https://firebase.google.com/docs
- https://angular.io/docs
- https://material.io/

ANNEXURE

1) NodeMCU Code:

```
#include <ACS712.h>
#include <ESP8266WiFi.h>
#include <WiFiManager.h>
#include <DNSServer.h>
#include <ESP8266WebServer.h>
#include <FirebaseArduino.h>
#include <SimpleTimer.h>
#include <Bounce2.h>

#define AP_SSID "Unfied-Socket"

#define FIREBASE_HOST "unified-home.firebaseio.com"
#define FIREBASE_AUTH "CiUsJ5C3mKYJ1TaSMm2vRw3THcO1uzIf5p7Lypdc"

#define relayPin D5
#define sensorPin A0
#define errorLEDPin D6
#define successLEDPin D8
#define switchPin D1
#define configPin D3

bool deviceState = 0;
bool switchState = 0;
bool wentOffline = 0;
float iPower = 0;

SimpleTimer timer;
Bounce debouncer = Bounce();

ACS712 sensor(ACS712_05B,sensorPin);

void setup() {
  sensor.calibrate();

  //Init Pins
  pinMode(relayPin, OUTPUT);
  debouncer.attach(switchPin, INPUT_PULLUP);
  debouncer.interval(25);
  pinMode(configPin, INPUT_PULLUP);
  pinMode(successLEDPin, OUTPUT);
  digitalWrite(successLEDPin, LOW);
```

```
  pinMode(errorLEDPin, OUTPUT);
  digitalWrite(errorLEDPin, HIGH);

  //Init Switch
  switchState = digitalRead(switchPin);

  //Init WiFi
  WiFiManager wifiManager;

  //Init Firebase
  Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
  Firebase.setBool("Socket/OnOff", deviceState);

  //Init Timer
  timer.setInterval(30000, sendStats);
}

void loop() {
  checkConfig();
  timer.run();
  if (debouncer.update()) syncSwitch();
  if (WiFi.status() == WL_CONNECTED) syncFirebase();
}

void checkConfig() {
  if (digitalRead(configPin) == LOW) {
    WiFiManager wifiManager;
    wifiManager.setAPCallback(configModeCallback);
    if (!wifiManager.startConfigPortal(AP_SSID)) {
      delay(3000);
      ESP.reset();
      delay(5000);
    }
  }
}

void configModeCallback (WiFiManager *myWiFiManager) {
  digitalWrite(errorLEDPin, HIGH);
  digitalWrite(successLEDPin, HIGH);
}

void syncSwitch() {
  digitalWrite(relayPin, switchState);
  Firebase.setBool("Socket/OnOff", switchState);
  if (Firebase.failed()) wentOffline = 1;
  switchState = !switchState;
```

```
}

void syncFirebase() {
  FirebaseObject Socket = Firebase.get("Socket");
  if (Firebase.failed()) {
    digitalWrite(successLEDPin, LOW);
    digitalWrite(errorLEDPin, HIGH);
  } else {
    deviceState = Socket.getBool("OnOff");
    if (wentOffline) {
      deviceState = (!switchState);
      Firebase.setBool("Socket/OnOff", deviceState);
      wentOffline = 0;
    }
    digitalWrite(relayPin, deviceState);
    digitalWrite(successLEDPin, HIGH);
    digitalWrite(errorLEDPin, LOW);
  }
}

void sendStats() {
  iPower = sensor.getCurrentAC(50)*220/1000;
  Firebase.setFloat("Socket-Stats/iPower_temp", deviceState ? iPower : 0);
}
```

2) Google Cloud Functions Code:

```
const functions = require("firebase-functions");
const admin = require("firebase-admin");
admin.initializeApp();
const { smarthome } = require("actions-on-google");
const firebaseRef = admin.database().ref("/");

//*---------------------------------SmartHome Functions
const app = smarthome({
  debug: true,
  key: "<api-key>"
});

//*-----On Sync
app.onSync(body => {
  return {
    requestId: body.requestId,
    payload: {
      agentUserId: "123",
      devices: [
        {
          id: "Socket",
          type: "action.devices.types.SWITCH",
          traits: ["action.devices.traits.OnOff"],
          name: {
            defaultNames: ["Socket"],
            name: "Socket",
            nicknames: ["Socket", "Smart Socket"]
          },
          willReportState: false,
          deviceInfo: {
            manufacturer: "Stark Industries",
            model: "UH-Socket",
            hwVersion: "1.0",
            swVersion: "1.0"
          }
        }
      ]
    }
  };
});
```

```
//*-----On Query
app.onQuery(async body => {
  const { requestId } = body;
  const payload = {
    devices: {}
  };
  const queryPromises = [];
  for (const input of body.inputs) {
    for (const device of input.payload.devices) {
      const deviceId = device.id;
      queryPromises.push(
        // eslint-disable-next-line promise/always-return
        queryDevice(deviceId).then(data => {
          payload.devices[deviceId] = data;
        })
      );
    }
  }
  // Wait for all promises to resolve
  await Promise.all(queryPromises);
  return {
    requestId: requestId,
    payload: payload
  };
});

//*-----On Execute
app.onExecute(body => {
  const { requestId } = body;
  const payload = {
    commands: [
      {
        ids: [],
        status: "SUCCESS",
        states: {
          online: true
        }
      }
    ]
  };
  for (const input of body.inputs) {
    for (const command of input.payload.commands) {
      for (const device of command.devices) {
        const deviceId = device.id;
        payload.commands[0].ids.push(deviceId);
        for (const execution of command.execution) {
```

```
        const execCommand = execution.command;
        const { params } = execution;
        switch (execCommand) {
          case "action.devices.commands.OnOff":
            firebaseRef.child(`${deviceId}`).update({
              OnOff: params.on
            });
            payload.commands[0].states.on = params.on;
            break;
        }
      }
    }
  }
  return {
    requestId: requestId,
    payload: payload
  };
});

exports.smarthome = functions.https.onRequest(app);

//Used Functions

const queryFirebase = async deviceId => {
  const snapshot = await firebaseRef.child(deviceId).once("value");
  const snapshotVal = snapshot.val();
  return {
    on: snapshotVal.OnOff
  };
};

const queryDevice = async deviceId => {
  const data = await queryFirebase(deviceId);
  return {
    online: true,
    on: data.on
  };
};

//*---------------------------------Fake Auth & Token
const util = require("util");

exports.fakeauth = functions.https.onRequest((request, response) => {
  const responseurl = util.format(
    "%s?code=%s&state=%s",
```

```
    decodeURIComponent(request.query.redirect_uri),
    "xxxxxx",
    request.query.state
  );
  console.log(responseurl);
  return response.redirect(responseurl);
});

exports.faketoken = functions.https.onRequest((request, response) => {
  const grantType = request.query.grant_type
    ? request.query.grant_type
    : request.body.grant_type;
  const secondsInDay = 86400; // 60 * 60 * 24
  const HTTP_STATUS_OK = 200;
  console.log(`Grant type ${grantType}`);

  let obj;
  if (grantType === "authorization_code") {
    obj = {
      token_type: "bearer",
      access_token: "123access",
      refresh_token: "123refresh",
      expires_in: secondsInDay
    };
  } else if (grantType === "refresh_token") {
    obj = {
      token_type: "bearer",
      access_token: "123access",
      expires_in: secondsInDay
    };
  }
  response.status(HTTP_STATUS_OK).json(obj);
});

//*---------------------------------Database Management

exports.manageReadings = functions.database
  .ref("Socket-Stats/iPower_temp")
  .onCreate(snap => {
    const val = snap.val();
    snap.ref.remove();

    let tPowerRef = admin.database().ref("Socket-Stats/tPower");
    tPowerRef.once("value",snapshot=>{
      tPowerRef.set(snapshot.val() + val*(30/3600));
    });
```

```
  return admin
    .database()
    .ref("Socket-Stats/iPower")
    .push({
      value: val,
      timestamp: admin.database.ServerValue.TIMESTAMP
    });
});
```