



PROJECT REPORT

VOICE CONTROLLED HOME AUTOMATION

SUBMITTED BY:

BSD Manjunath babu(18bec008)

E Shiva sai (18bec012)

GVS Bhargav(18bec013)

G sai harsha (18bec016)

P sathvik (18bec038)

INDEX

- Introduction
- The Idea and procedure
- Hardware Required
- Description of Hardware
 1. Nodemcu esp8266
 2. 4-channel relay module
- Flow chart
- Pin diagram
- Code
- Setup
- Conclusion
- Reference

INTRODUCTION

- Automation plays a key role in human life. Voice Controlled Home automation allows us to control household electrical appliances like light, door, fan, AC etc using our voice commands. Home automation not only refers to reducing human efforts but also energy efficiency and time saving. The main objective of home automation is to help handicapped and old aged people who will enable them to control home appliances in critical situations.
- This project put forward the implementation of home automation using NodeMCU ESP8266 and Android smartphones. Home appliances are connected to the microprocessor and communication is established between the NodeMCU ESP8266 and Android mobile device or tablet via ESP-12E module. The device with low cost and scalable to less modification to the core is much important. It presents the design and implementation of automation system that can monitor and control home appliances via android phone or tablet.
- This project put forward the implementation of home automation using NodeMCU ESP8266 and Android smartphones. Home appliances are connected to the microprocessor and communication is established between the NodeMCU ESP8266 and Android mobile device or tablet via ESP-12E module. The device with low cost and scalable to less modification to the core is much important. It presents the design and implementation of automation system that can monitor and control home appliances via android phone or tablet.
- In simple words, Home Automation System using ESP8266 Wi-Fi module, where you can control your Home appliances using your Voice though an Android App from anywhere in the world. To make our loads (Light/Fans) turn ON or OFF by simply

using a voice command is going to come true at the end of this project.

THE IDEA AND PROCEDURE

- This project is designed to use the voice recognition technology to control home appliances i.e., light and fan.
- Alexa voice recognition facility will be utilised to input voice.
- Android phones will be used for the application software which will be connected to the Alexa voice command Input.
- Application software will be responsible for converting the voice command into the text format. Ultimately application software will transfer that command in the text format to the ESP8266 using ESP-12E module.
- This microcontroller (Node MCU ESP8266) is intelligent enough to convert the text command executable according to the programming that is involved as per requirement and after execution, the action will be performed.
- Commands which we have included are lights on/off, fan on/off, tv on/off and all on/off.
- Major home utilities for our project are light, fan and TV. Each of them follows the certain command that is programmed.
- The aim is to use this IoT technology for the people of all ages, gender and even with some disability to enjoy life in a more pleasant way.
- Now we are getting up to turn our home appliance on or off. If we are making the home controlling system more easy and reliable to the user like disable people then they will be benefited from our system.
- It will become the best system for the disabled people and for the elderly people to use their mobile phone and control the home utilities. The goal of this system is to help disable and elder people.

HARDWARE REQUIRED

1. Nodemcu esp8266
2. 4-channel relay module
3. Bread board
4. Connecting wires
5. Home appliances

SOFTWARE REQUIRED

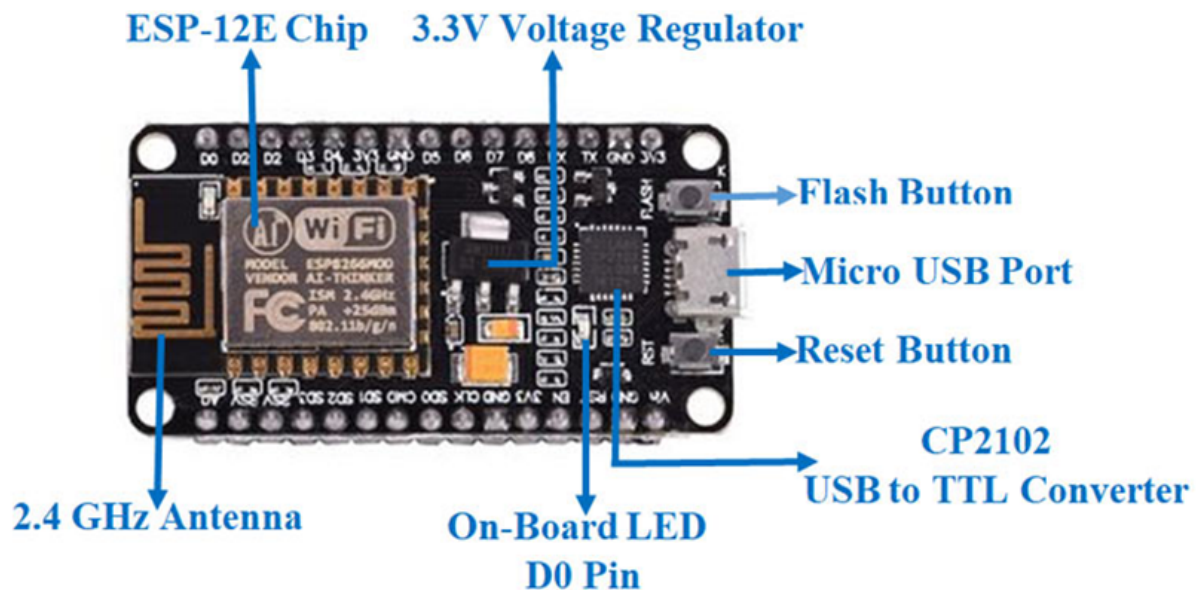
1. Arduino 1.8.13

Libraries to be included:

1. ArduinoJson library
2. Arduinowebsockets library
3. Sinricpro library

Description of Hardware

NODEMCU ESP8266



- The NodeMCU ESP8266 development board comes with the ESP-12E module containing ESP8266 chip having Tensilica Xtensa 32-bit LX106 RISC microprocessor. This microprocessor supports RTOS and operates at 80MHz to 160 MHz adjustable clock frequency. NodeMCU has 128 KB RAM and 4MB of Flash memory to store data and programs. Its high processing power with in-built Wi-Fi / Bluetooth.
- NodeMCU can be powered using Micro USB jack and VIN pin (External Supply Pin). It supports UART, SPI, and I2C interface.
- Specifications & features:

Microcontroller: Tensilica 32-bit RISC CPU Xtensa LX106

Operating Voltage: 3.3V

Input Voltage: 7-12V

Digital I/O Pins (DIO): 16

Analog Input Pins (ADC): 1

UARTs: 1

SPIs: 1

I2Cs: 1

Flash Memory: 4 MB

SRAM: 64 KB

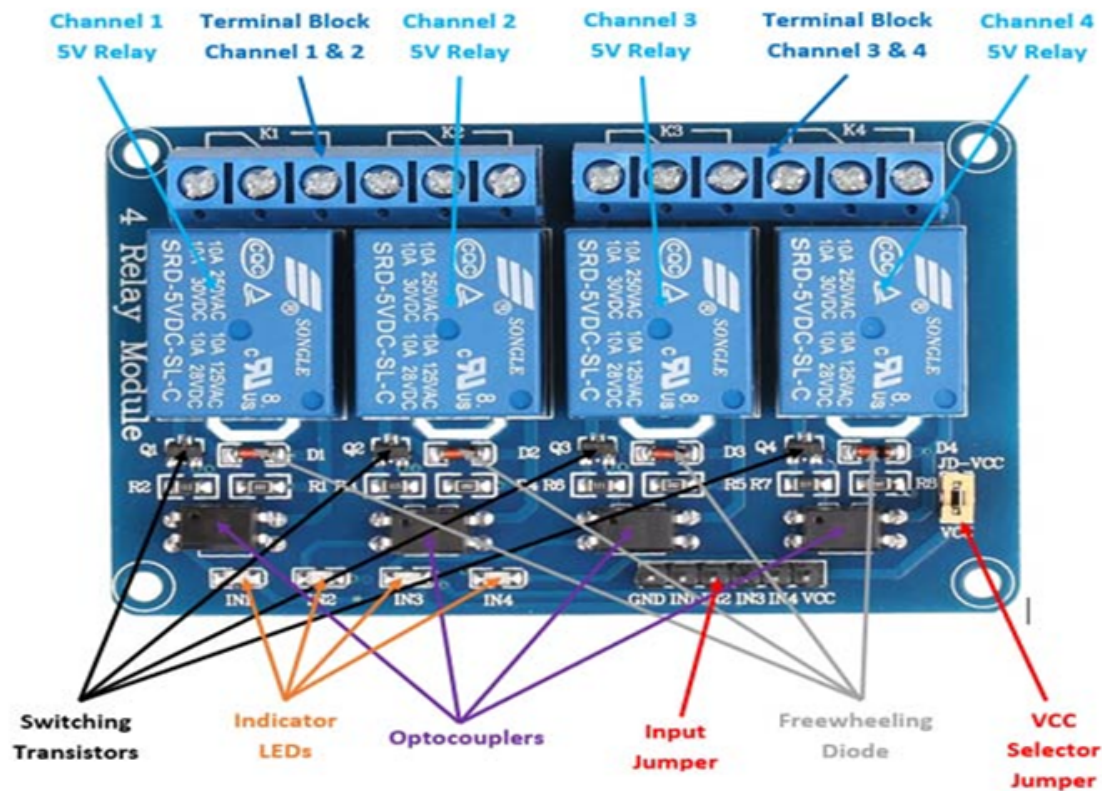
Clock Speed: 80 MHz

USB-TTL based on CP2102 is included onboard, Enabling Plug
n Play

PCB Antenna

Small Sized module to fit smartly inside your IoT projects

4-CHANNEL RELAY MODULE



- The relay is the device that open or closes the contacts to cause the operation of the other electric control. It detects the intolerable or undesirable condition with an assigned area and gives the commands to the circuit breaker to disconnect the affected area. Thus protects the system from damage.

- The four-channel relay module contains four 5V relays and the associated switching and isolating components, which makes interfacing with a microcontroller or sensor easy with minimum

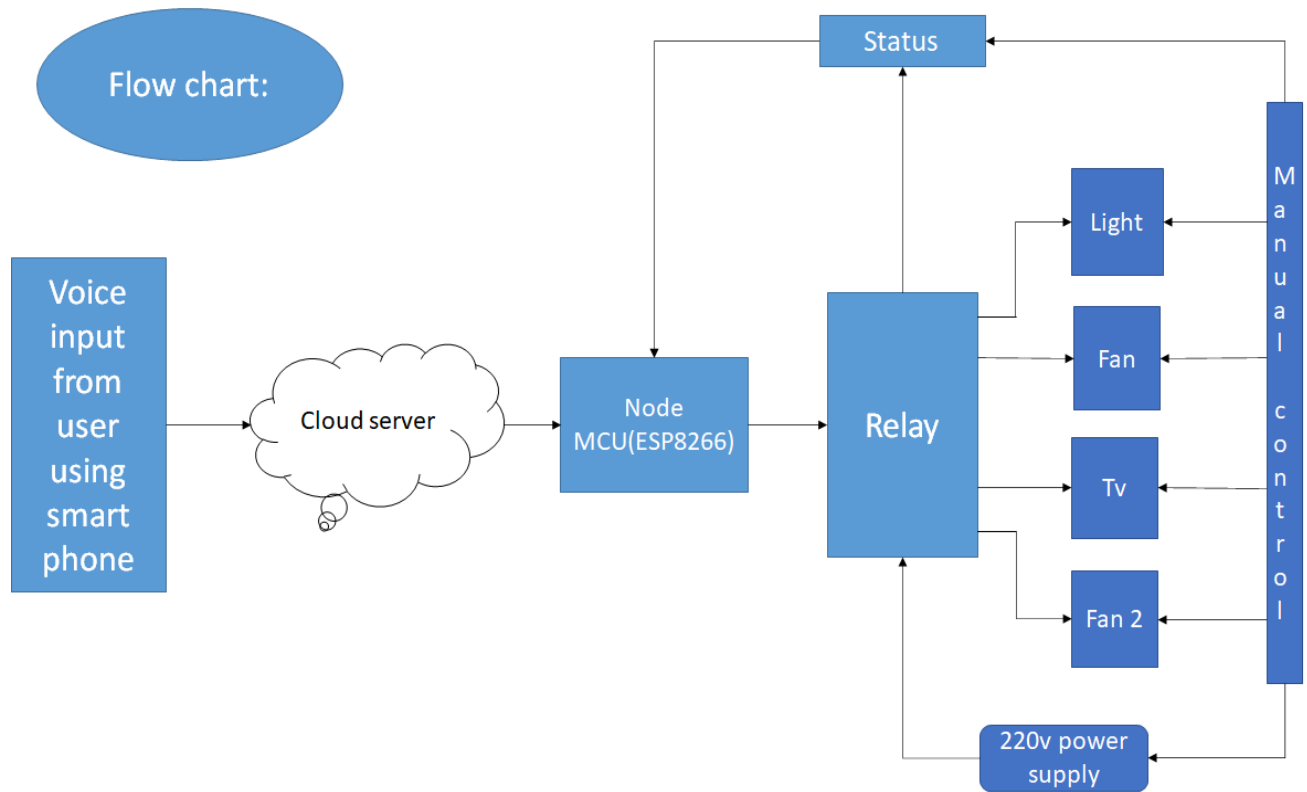
components and connections. The contacts on each relay are specified for 250VAC and 30VDC and 10A in each case, as marked on the body of the relays.

Specifications:

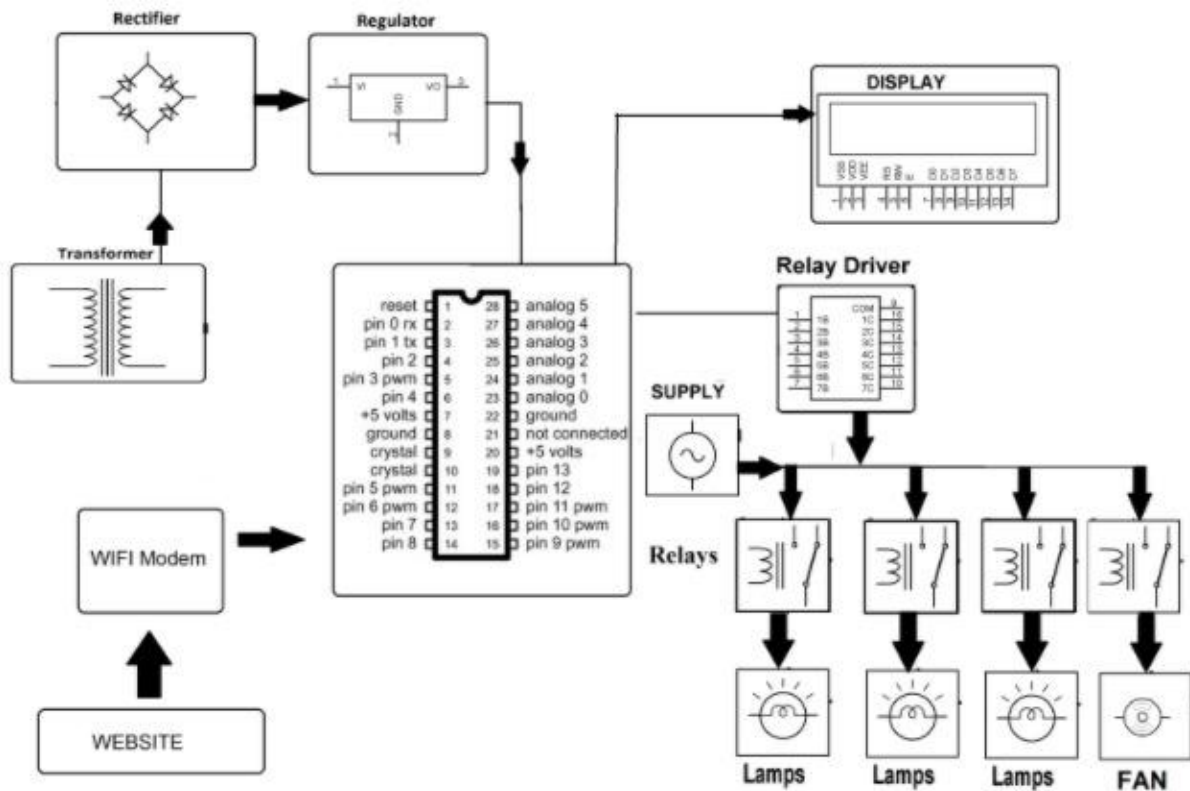
- Supply voltage – 3.75V to 6V
- Trigger current – 5mA
- Current when the relay is active - ~70mA (single), ~300mA (all four)
- Relay maximum contact voltage – 250VAC, 30VDC
- Relay maximum current – 10A

•Major components are 5V relay terminal blocks, male headers, transistors, optocouplers, diodes, and LEDs

FLOW CHART



PIN DIAGRAM



CODE

```
#include <Arduino.h>

#include <ESP8266WiFi.h>

#include "SinricPro.h"
#include "SinricProSwitch.h"

#include <map>

#define WIFI_SSID    "Iot"
```

```
#define WIFI_PASS      "123456789"
#define APP_KEY        "2ef16e54-e6b2-4efe-ab39-91f768f0d92f"
#define APP_SECRET     "8cb4a75c-2098-4c45-a4ee-52a657b39d1a-5ea72f68-be81-4709-a65a-a9b48e84ba2d"

//Enter the device IDs here
#define device_ID_1  ""
#define device_ID_2  "60855c82d3827f1ea82d1969"
#define device_ID_3  "60855c63c54b6f1eb0aa446d"
#define device_ID_4  "608549c0c54b6f1eb0aa4412"


// define the GPIO connected with Relays and switches
#define RelayPin1 5 //D1
#define RelayPin2 4 //D2
#define RelayPin3 14 //D5
#define RelayPin4 12 //D6


#define SwitchPin1 10 //SD3
#define SwitchPin2 0 //D3
#define SwitchPin3 13 //D7
#define SwitchPin4 3 //RX


#define wifiLed 16 //D0


#define TACTILE_BUTTON 1


#define BAUD_RATE 9600


#define DEBOUNCE_TIME 250


typedef struct { // struct for the std::map below
    int relayPIN;
    int flipSwitchPIN;
} deviceConfig_t;
```

```

std::map<String, deviceConfig_t> devices = {
    //{deviceId, {relayPIN, flipSwitchPIN}}
    {device_ID_1, { RelayPin1, SwitchPin1 }},
    {device_ID_2, { RelayPin2, SwitchPin2 }},
    {device_ID_3, { RelayPin3, SwitchPin3 }},
    {device_ID_4, { RelayPin4, SwitchPin4 }}
};

```

```

typedef struct
{
    // struct for the std::map below
    String deviceId;
    bool lastFlipSwitchState;
    unsigned long lastFlipSwitchChange;
}
flipSwitchConfig_t;

```

```

std::map<int, flipSwitchConfig_t> flipSwitches;
// this map is used to map flipSwitch PINs to deviceId and handling debounce and last flipSwitch state checks
// it will be setup in "setupFlipSwitches" function, using informations from devices map

```

```

void setupRelays()
{
    for (auto &device : devices)
    {
        // for each device (relay, flipSwitch combination)
        int relayPIN = device.second.relayPIN;    // get the relay pin
        pinMode(relayPIN, OUTPUT);                // set relay pin to OUTPUT
        digitalWrite(relayPIN, HIGH);
    }
}

```

```

void setupFlipSwitches() {
    for (auto &device : devices) {                // for each device (relay / flipSwitch combination)
        flipSwitchConfig_t flipSwitchConfig;      // create a new flipSwitch configuration
    }
}

```

```

flipSwitchConfig.deviceId = device.first;    // set the deviceId
flipSwitchConfig.lastFlipSwitchChange = 0;    // set debounce time
flipSwitchConfig.lastFlipSwitchState = true;  // set lastFlipSwitchState to false (LOW)--

int flipSwitchPIN = device.second.flipSwitchPIN; // get the flipSwitchPIN

flipSwitches[flipSwitchPIN] = flipSwitchConfig; // save the flipSwitch config to flipSwitches map
pinMode(flipSwitchPIN, INPUT_PULLUP);          // set the flipSwitch pin to INPUT
}
}

bool onPowerState(String deviceId, bool &state)
{
    Serial.printf("%s: %s\r\n", deviceId.c_str(), state ? "on" : "off");
    int relayPIN = devices[deviceId].relayPIN;    // get the relay pin for corresponding device
    digitalWrite(relayPIN, !state);              // set the new relay state
    return true;
}

void handleFlipSwitches() {
    unsigned long actualMillis = millis();        // get actual millis
    for (auto &flipSwitch : flipSwitches) {      // for each flipSwitch in flipSwitches map
        unsigned long lastFlipSwitchChange = flipSwitch.second.lastFlipSwitchChange;
        // get the timestamp when flipSwitch was pressed last time (used to debounce / limit events)

        if (actualMillis - lastFlipSwitchChange > DEBOUNCE_TIME) {    // if time is > debounce time...

            int flipSwitchPIN = flipSwitch.first;    // get the flipSwitch pin from configuration
            bool lastFlipSwitchState = flipSwitch.second.lastFlipSwitchState;    // get the lastFlipSwitchState
            bool flipSwitchState = digitalRead(flipSwitchPIN);    // read the current flipSwitch state
            if (flipSwitchState != lastFlipSwitchState) {    // if the flipSwitchState has changed...

#ifdef TACTILE_BUTTON
                if (flipSwitchState) {    // if the tactile button is pressed

```

```

#endif

    flipSwitch.second.lastFlipSwitchChange = actualMillis;           // update lastFlipSwitchChange time
    String deviceId = flipSwitch.second.deviceId;                     // get the deviceId from config
    int relayPIN = devices[deviceId].relayPIN;                       // get the relayPIN from config
    bool newRelayState = !digitalRead(relayPIN);                     // set the new relay State
    digitalWrite(relayPIN, newRelayState);                           // set the trelay to the new state

    SinricProSwitch &mySwitch = SinricPro[deviceId];                 // get Switch device from SinricPro
    mySwitch.sendPowerStateEvent(!newRelayState);                   // send the event
#ifdef TACTILE_BUTTON
}
#endif

    flipSwitch.second.lastFlipSwitchState = flipSwitchState;        // update lastFlipSwitchState
}
}
}
}

void setupWiFi()
{
    Serial.printf("\r\n[Wifi]: Connecting");
    WiFi.begin(WIFI_SSID, WIFI_PASS);

    while (WiFi.status() != WL_CONNECTED)
    {
        Serial.printf(".");
        delay(250);
    }
    digitalWrite(wifiLed, LOW);
    Serial.printf("connected!\r\n[Wifi]: IP-Address is %s\r\n", WiFi.localIP().toString().c_str());
}

void setupSinricPro()
{

```

```
for (auto &device : devices)
{
    const char *deviceId = device.first.c_str();
    SinricProSwitch &mySwitch = SinricPro[deviceId];
    mySwitch.onPowerState(onPowerState);
}

SinricPro.begin(APP_KEY, APP_SECRET);
SinricPro.restoreDeviceStates(true);
}

void setup()
{
    Serial.begin(BAUD_RATE);

    pinMode(wifiLed, OUTPUT);
    digitalWrite(wifiLed, HIGH);

    setupRelays();
    setupFlipSwitches();
    setupWiFi();
    setupSinricPro();
}

void loop()
{
    SinricPro.handle();
    handleFlipSwitches();
}
```





SETUP

- Sinric interface

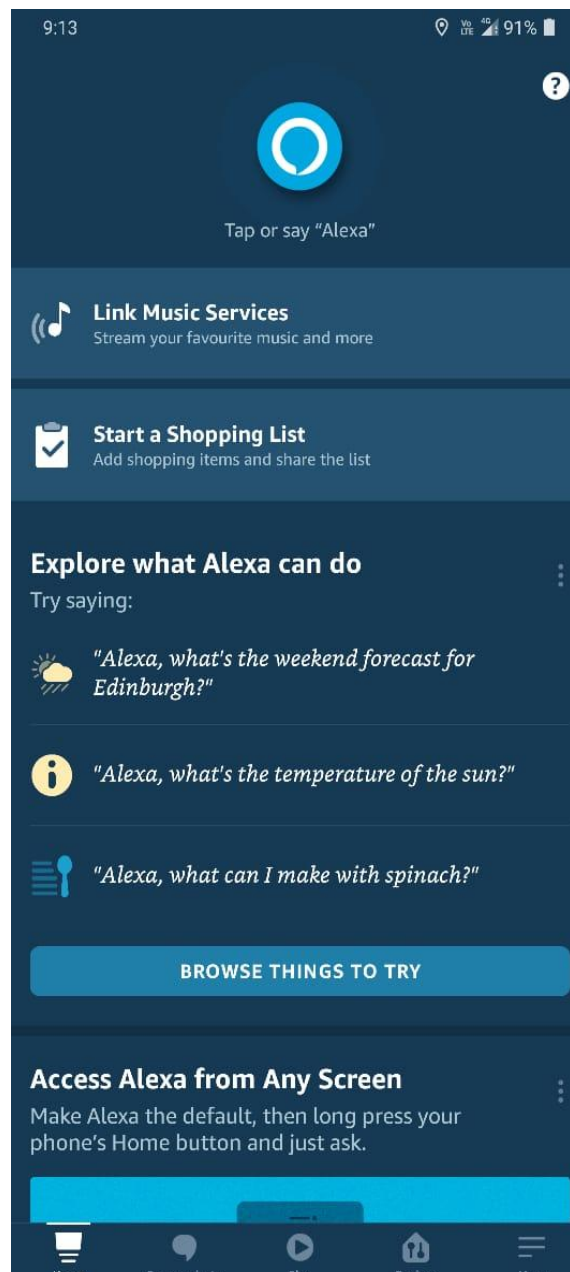
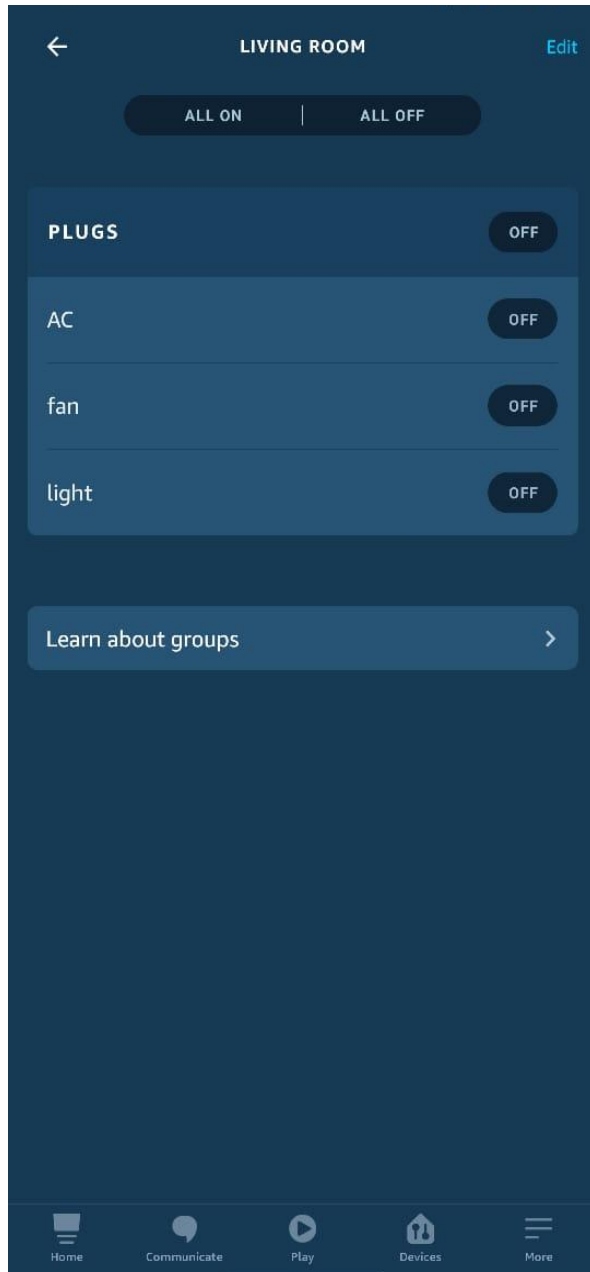
Create a device in sinric

The screenshot shows the 'New Device' setup interface in Sinric Pro v2.13.6. The left sidebar contains navigation links: Dashboard, Devices, Rooms, Scenes, Schedules, Credentials, Device Templates, Activity Log, Account, Subscriptions, and What's New. The main content area is titled 'New Device' and features a progress bar with four steps: 1. Device Information, 2. Notifications (current step), 3. Timers, and 4. Other. Below the progress bar, there is a section titled 'Send a push notification to my mobile'. Under 'When this device', there are two toggle switches: 'Connect' and 'Disconnect'. Under 'When this device turned', there are two toggle switches: 'On' and 'Off'. A 'Next' button is located at the bottom left, and a link 'How to connect my device?' is at the bottom right.

The screenshot shows the 'Devices' management interface in Sinric Pro v2.13.6. The left sidebar is the same as in the previous screenshot. The main content area is titled 'Devices' and includes a '+ Add Device' button and a search bar. Below is a table listing the devices:

DEVICE	DESCRIPTION	POWER STATE	ROOM	DEVICE ACCESS KEY	LAST CONNECTED	NO OF TIMES CONNECTED	
 AC ID: 60855c82d3827f1ea82d1969 Copy	AC	Off	Living Room	default	12 hours ago	8	⋮
 fan ID: 60855c83c54b6f1eb0aa446d Copy	fan	Off	Living Room	default	12 hours ago	8	⋮
 light ID: 608549c0c54b6f1eb0aa4412 Copy	light	Off	Living Room	default	12 hours ago	8	⋮

- ALEXA interface



CONCLUSION

Through this project we were able to control our house appliances and know the status of home appliances from anywhere in the world.

Reference

1. Voice controlled home automation by Rejwan Bin Sulaiman
2. Voice-activated home automation using NodeMCU by K.Loga priya and s.saranya
3. GOOGLE ASSISTANT CONTROLLED HOME AUTOMATION By Mr. Kalyan Chenumalla, Mr. Srikanth Gottam, Mr. Prashanth Kusuma, Ms. P. Bhavya Shri - IEEE VEC SB