Voice-Controlled WiFi-Based Home Automation System

Project Report

Lawaz Islam

Overview

I built a WiFi-based home automation system that controls household appliances remotely through an Android app and voice commands. The system uses a NodeMCU ESP8266 microcontroller driving a four-channel relay module, with four LEDs standing in for four real appliances. Commands are sent over WiFi to the NodeMCU's IP address, which then switches the relays through its GPIO pins, turning each appliance on or off. The result is a low-cost system that lets a user control multiple devices from anywhere with an internet connection, including by voice.

System Architecture

- Controller: a NodeMCU ESP8266 (12E) WiFi module that connects to the local network and receives control signals.
- Switching interface: a four-channel relay module wired to the NodeMCU's GPIO pins, with each relay controlling one output channel.
- User interface: an Android app with ON and OFF buttons for each channel, plus voice input that maps spoken commands to those buttons.
- Communication: the app sends signals to the NodeMCU's assigned IP address over WiFi, so control works locally or remotely over the internet.

Hardware and Components

- NodeMCU ESP8266-12E development board
- Four-channel relay module
- Four LEDs as appliance stand-ins, each with a 220-ohm series resistor
- Breadboard, jumper wires, and a regulated power supply

Working Principle

On power-up, the NodeMCU connects to the local WiFi network and is reachable at its own IP address. I entered that IP address into the Android app so the two could communicate. When the user presses an ON or OFF button, or speaks a command that the app converts to the same action, the app sends a request to the NodeMCU. The NodeMCU reads the request and drives the matching GPIO pin high or low, which energizes or releases the corresponding relay and switches the connected LED, or in a real deployment, the appliance. Because the signal travels over the internet, the appliances can be controlled from anywhere, with the phone acting as the remote and the NodeMCU as the receiver.

Implementation

- Programmed the NodeMCU in the Arduino IDE to join the WiFi network, listen for incoming requests, and map each request to a GPIO pin and relay channel.
- Wired the four-channel relay module to the NodeMCU GPIO pins and connected the LEDs through 220-ohm resistors to represent four independent appliances.
- Configured the Android control app with the NodeMCU's IP address and tested each ON/OFF button and voice command against the correct channel.
- Validated all four channels switching independently and reliably over the network.

Results

The finished system controlled four independent channels reliably, both from the app's buttons and by voice, with responsive switching over WiFi. It demonstrated a complete path from a phone command to a physical switching action, the core of any practical home-automation or IoT control system.

Applications

- Home automation: switching lights, fans, and other appliances remotely.
- Accessibility: hands-free voice control for elderly or differently-abled users.
- Energy management: turning devices off remotely to avoid waste.
- IoT prototyping: a reusable base for internet-connected control projects.

Future Improvements

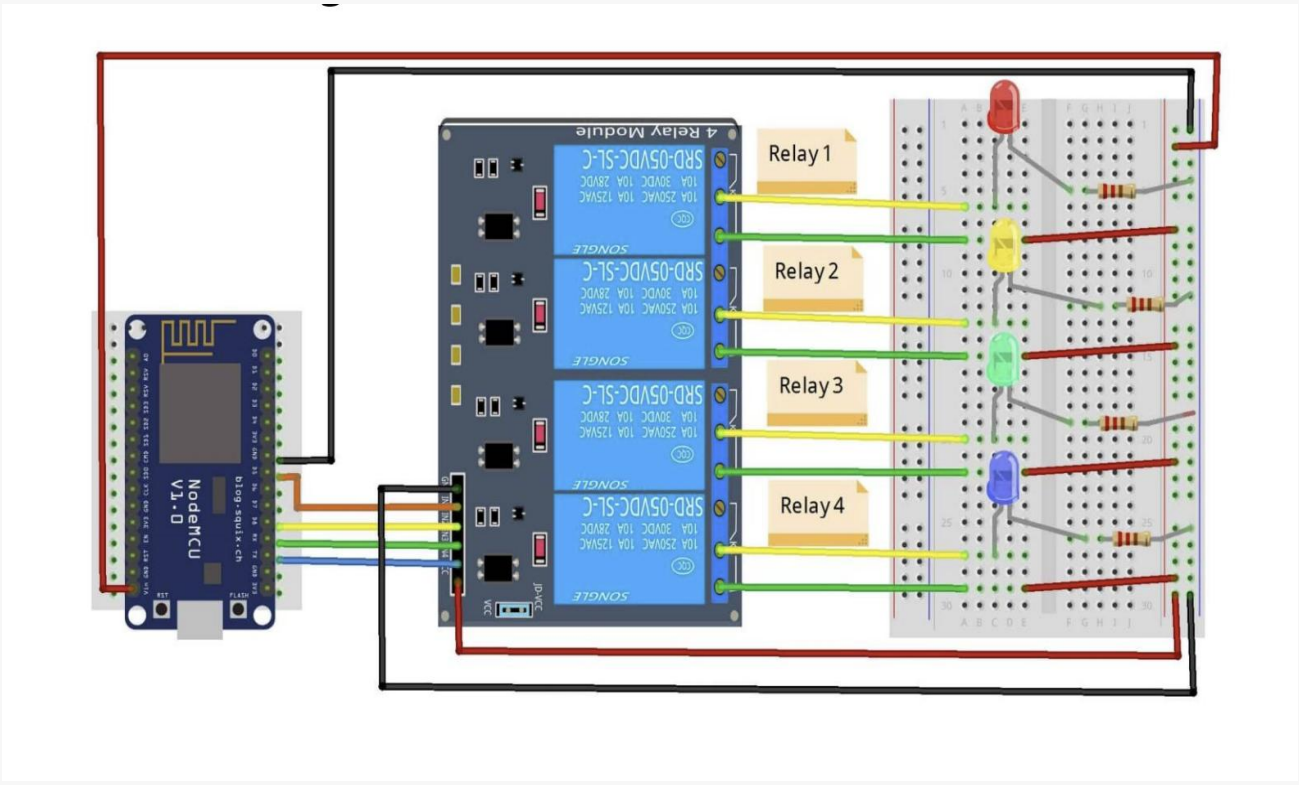
- Add status feedback so the app shows the real on/off state of each channel.
- Move control to a cloud dashboard (for example Blynk or Firebase) for reliable remote access.
- Add authentication so only authorized users can switch devices.
- Integrate with Google Assistant or Alexa for natural voice control.

Tech Stack

- **Microcontroller:** NodeMCU ESP8266-12E
- **Firmware:** Arduino IDE (C/C++)
- **Interface:** Four-channel relay module, Android control app
- **Communication:** WiFi, IP-based control over the internet

Circuit Diagram

Replace the box below with the circuit diagram screenshot from the project.



Firmware (NodeMCU Source Code)

The NodeMCU runs the following Arduino sketch. It connects to WiFi, starts a small web server, and switches each of the four relay channels on or off based on requests from the Android app or a browser. Set your WiFi name and password at the top before uploading.

```
/* ========================================================================
   Voice-Controlled WiFi Home Automation - NodeMCU ESP8266 Firmware
   ------------------------------------------------------------------------
   Controls a 4-channel relay module over WiFi. Each relay switches one
   appliance (here, one LED on the breadboard). The NodeMCU runs a small web
   server; the Android app (and its voice input) sends HTTP requests to turn
   each channel ON or OFF. You can also control it from any browser on the
   same network by opening the NodeMCU's IP address.

   How control works:
     - Browser/app calls:  http://<NodeMCU-IP>/set?relay=1&state=1   (relay 1 ON)
                           http://<NodeMCU-IP>/set?relay=1&state=0   (relay 1 OFF)
     - Opening http://<NodeMCU-IP>/ shows on/off buttons for all 4 channels.

   Board: NodeMCU 1.0 (ESP-12E). Install the ESP8266 core in Arduino IDE,
   set your WiFi name/password below, upload, then open the Serial Monitor
   at 115200 baud to see the assigned IP address.
   ======================================================================== */

#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>

// ---- 1. Set your WiFi credentials ------------------------------------------
const char* WIFI_SSID = "YOUR_WIFI_NAME";
const char* WIFI_PASS = "YOUR_WIFI_PASSWORD";

// ---- 2. Wiring (match your four jumper wires to these pins) ----------------
//   Relay module IN1  ->  NodeMCU D1
//   Relay module IN2  ->  NodeMCU D2
//   Relay module IN3  ->  NodeMCU D5
//   Relay module IN4  ->  NodeMCU D6
//   Relay module VCC  ->  NodeMCU Vin (5V)
//   Relay module GND  ->  NodeMCU GND   (grounds MUST be common)
// D1, D2, D5, D6 are safe general-purpose pins. Avoid D3, D4, and D8 for
// relay inputs: they are boot-sensitive and can stop the NodeMCU from starting.
const int relayPins[4] = { D1, D2, D5, D6 };

// Most 4-channel relay boards are ACTIVE-LOW: a LOW signal turns the relay ON.
// If your relays behave backwards, change this to false.
const bool ACTIVE_LOW = true;

// Current state of each channel (false = OFF, true = ON).
bool relayState[4] = { false, false, false, false };

ESP8266WebServer server(80);

// ---- Drive one relay to match its stored state -----------------------------
void applyRelay(int i) {
  int level;
  if (ACTIVE_LOW) level = relayState[i] ? LOW : HIGH;   // active-low board
  else            level = relayState[i] ? HIGH : LOW;   // active-high board
  digitalWrite(relayPins[i], level);
}

// ---- Build the simple control web page -------------------------------------
String buildPage() {
  String html = F("<!DOCTYPE html><html><head><meta name='viewport' "
                  "content='width=device-width,initial-scale=1'>"
                  "<title>Home Automation</title></head><body>"
                  "<h2>WiFi Home Automation</h2>");
  for (int i = 0; i < 4; i++) {
    int n = i + 1;
    html += "<p>Relay " + String(n) + ": <b>" +
            (relayState[i] ? "ON" : "OFF") + "</b>   ";
    html += "<a href='/set?relay=" + String(n) + "&state=1'>[ON]</a> ";
    html += "<a href='/set?relay=" + String(n) + "&state=0'>[OFF]</a></p>";
```

```
  }
  html += F("</body></html>");
  return html;
}

void handleRoot() {
  server.send(200, "text/html", buildPage());
}

// ---- Handle on/off requests from the app or browser ------------------------
void handleSet() {
  if (server.hasArg("relay") && server.hasArg("state")) {
    int r = server.arg("relay").toInt();   // 1..4
    int s = server.arg("state").toInt();   // 1 = ON, 0 = OFF
    if (r >= 1 && r <= 4) {
      relayState[r - 1] = (s == 1);
      applyRelay(r - 1);
      Serial.printf("Relay %d -> %s\n", r, relayState[r - 1] ? "ON" : "OFF");
    }
  }
  // Send the user back to the main page.
  server.sendHeader("Location", "/");
  server.send(303);
}

void handleNotFound() {
  server.send(404, "text/plain", "Not found");
}

void setup() {
  Serial.begin(115200);

  // Start every relay in a known OFF state.
  for (int i = 0; i < 4; i++) {
    pinMode(relayPins[i], OUTPUT);
    relayState[i] = false;
    applyRelay(i);
  }

  // Connect to WiFi.
  Serial.print("\nConnecting to ");
  Serial.println(WIFI_SSID);
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nConnected.");
  Serial.print("Open this IP in your app or browser: ");
  Serial.println(WiFi.localIP());   // <-- put this IP into the Android app

  // Register routes and start the server.
  server.on("/", handleRoot);
  server.on("/set", handleSet);
  server.onNotFound(handleNotFound);
  server.begin();
  Serial.println("HTTP server started.");
}

void loop() {
  server.handleClient();   // process incoming requests
}
```