Maven Silicon

Embedded System Design Internship

Internship Project Report

Home Automation System

By
Jayanth Balan

Table of Contents

1.Student Information

Name: Jayanth Sinnakavadi Balan

University: Vellore Institute of Technology, Chennai Campus
Location: Kelambakkam - Vandalur Rd, Rajan Nagar, Chennai, Tamil Nadu 600127

Batch: ESDI - Home Automation - VIT June 2024
Submission Date: 11-07-24

2.Introduction

The idea of "smart homes" has become highly popular in recent years due to the quick development of Internet of Things (IoT) technology and the growing desire for ease and automation in daily living. The project offers a workable example of an ESP32 microcontroller-based smart home automation system. Through a web interface, the system allows for the remote control and automation of domestic appliances such as fans and lights. It also includes sensors to enable automatic modifications based on ambient conditions. This method is particularly applicable for a wide range of application scenarios, as it not only improves convenience but also contributes to energy efficiency and security.

The system guarantees that appliances are only turned on when necessary by integrating sensors that identify occupancy and ambient conditions. This helps to reduce energy waste and minimize utility bills. It presents a realistic and successful use of IoT in home automation, emphasizing the major advantages of security, ease, energy savings, and operational effectiveness. Through utilizing the ESP32 microcontroller's capabilities and incorporating many sensors, the project showcases the ability of smart home technology to enhance living standards and encourage environmentally conscious practices. Projects like this will be vital in determining the direction of intelligent and responsive living spaces as smart home automation develops further.

3.Project Specification

3.1.Functional Requirements

- The project controls 2 appliances, a light bulb and a fan. The light bulb is connected to the standard 230V supply via relay. The 5V fan is connected directly to the Vin Supply pin of the ESP32 microcontroller.

- The controller is connected to the home WiFi network and creates a webpage containing the virtual control buttons for each appliance.

- When in manual mode, the fan and bulb can be controlled by any mobile or pc connected to home WiFi network.

- When in auto mode, the status of the appliances is determined by a proximity sensor (pir), a temperature sensor and a light sensor.

- Auto or manual mode can be selected on the devices.

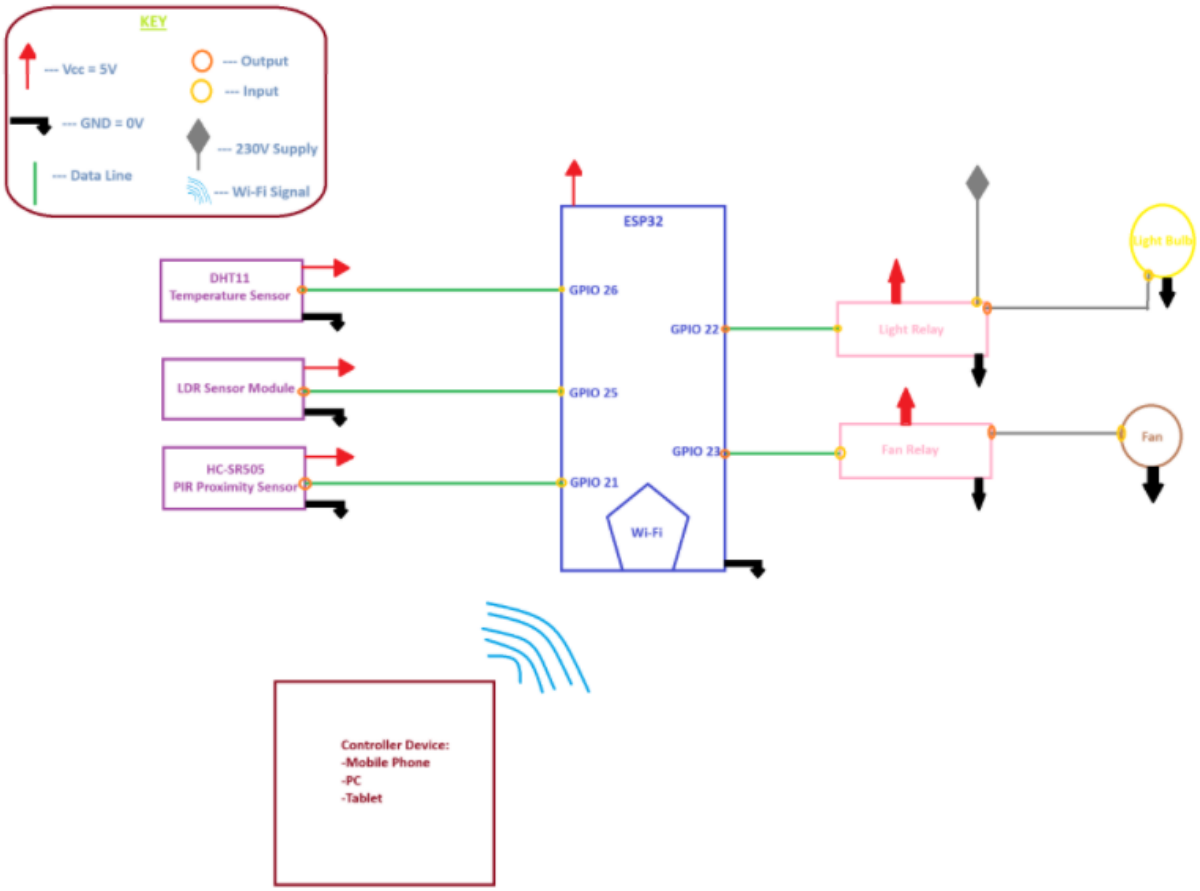
3.2.Non-Functional Requirements

- Automatic mode:
    - The bulb switches on after the entry of human in the room within 1 minute.
    - Fan switches on as soon as the threshold is detected within 1 minute.

- Manual mode:
    - The bulb switches on within 1 minute of pressing the button on the mobile phone.
    - Fan switches on within 1 minute of pressing the button on the mobile phone.
- The system works in the temperature range 0^{o} C - 60^{o} C.

4.Hardware Architecture

4.1.Block Diagram



ESP32 Development Board – It is an appropriate board for this application because of its low cost, limited power consumption, and built-in Wi-Fi capabilities and ADC of the microcontroller. It can connect to any home Wi-Fi network or work as an access point of its own.

4.2.Components

S.No.	Item	Quantity	Description
1	ESP32	1	Microcontroller with Wi-Fi capacity
2	DHT11	1	Temperature Sensor
3	LDR Module	1	Light Measurement Sensor
4	HC-SR505	1	Passive Infrared Sensor (Motion Sensor)
5	Relay	2	5V 1 Channel Optocoupler Relay
6	Light Bulb	1	230V Bulb
7	Fan	1	5V Fan
8	Bulb Holder	1	
9	Breadboard	1	

5.Software Architecture

5.1.Pseudocode

Setup

1. Initialize WiFi credentials and sensor pin assignments.
2. Initialize light and fan pin assignments and states.
3. Initialize the DHT sensor and WiFi server.
4. Initialize other variables for tracking state and timing.
5. Start the serial communication.
6. Set light and fan pins as outputs and turn them off.
7. Set sensor pins as inputs.
8. Start the WiFi access point.
9. Print the IP address to the serial monitor.
10. Begin the DHT sensor.
11. Start the WiFi server.

Loop

1. Check for a client connection to the server.
2. If a client is connected:
   - Record the current time and reset the previous time.
   - Print a message indicating a new client connection.
   - Initialize an empty string for the current line.
   - While the client is connected and timeout has not occurred:

- If the client has sent data:
  - Read the data.
  - Print the data to the serial monitor.
  - Append the data to the header string.
  - If a newline character is received:
    - If the current line is empty (indicating the end of the HTTP request):
      - Send the HTTP response header.
      - Call the `sendHTML` function to send the HTML page.
      - If the mode is "manual":
        - If the header contains "GET /light/on":
          - Turn on the light.
          - Update `lightstate` to "on".
        - Else if the header contains "GET /light/off":
          - Turn off the light.
          - Update `lightstate` to "off".
        - If the header contains "GET /fan/on":
          - Turn on the fan.
          - Update `fanstate` to "on".
        - Else if the header contains "GET /fan/off":
          - Turn off the fan.
          - Update `fanstate` to "off".
      - If the header contains "GET /mode/manual":
        - Set the mode to "manual".

- Else if the header contains "GET /mode/auto":
  - Set the mode to "auto".
  - Break out of the loop.
- Else:
  - Clear the current line.
- Else if the character is not a carriage return:
  - Append the character to the current line.
- Clear the header string.
- Stop the client connection.
- Print a message indicating the client disconnected.

3. If the mode is "auto":
   - Call the `automode` function.

sendHTML

1. Send the HTML structure to the client.
2. Send the CSS styles.
3. Send the body content and buttons for controlling the light, fan, and mode.
4. Update the button labels and styles based on the current states (`lightstate`, `fanstate`, `mode`).

automode

1. Delay for 2 seconds.
2. Read the proximity sensor.

3. If the proximity sensor is triggered:
   - Toggle the `userin` state.
   - If `userin` is true:
     - Delay for 2 seconds.
     - Read the temperature from the DHT sensor.
     - Read the darkness sensor.
     - If the temperature exceeds `templim1`:
       - Turn on the fan.
       - Update `fanstate` to "on".
     - Else if the temperature is below `templim2`:
       - Turn off the fan.
       - Update `fanstate` to "off".
     - If it is dark:
       - Turn on the light.
       - Update `lightstate` to "on".
     - Else:
       - Turn off the light.
       - Update `lightstate` to "off".
   - Else (if `userin` is false):
     - Turn off the light and fan.
     - Update `lightstate` to "off".
     - Update `fanstate` to "off".

5.2.Functional Blocks

‘setup()’ - The setup function initializes serial communication, configures sensor and appliance pins, and sets up the ESP32 as a WiFi access point. It also starts the DHT sensor and Wi-Fi server actions, preparing the system for client connections.

loop() - The loop function continuously checks for client connections, processes HTTP requests to update light and fan states, and sends the updated HTML page to the client. It also calls ‘automode()’ if the system is set to auto mode.

‘sendHTML(WiFiClient client)’ - The ‘sendHTML’ function constructs and sends an HTML page to the connected client, providing a web interface to control the light and fan. It updates the button states based on the current appliance states and mode.

‘automode()’ - The ‘automode’ function manages automatic control of the light and fan based on sensor readings. It adjusts the appliances according to user presence, temperature, and ambient light levels to optimize energy usage.

6.Code

```
#include <WiFi.h>
#include "DHT.h"

const char* ssid = "ESP32_ACCESS_POINT";
const char* password = "pass";

const int proxsense = 21;
const int darksense = 25;
const int tempsense = 26;

const int light = 22;
const int fan = 23;

const int templim1 = 29;
const int templim2 = 27;
int proxcheck = 0;
bool userin = false;

WiFiServer server(80);
DHT tempdht(tempsense, DHT11);

String header;
String mode = "auto";
String lightstate = "off";
String fanstate = "off";

unsigned long currentTime = millis();
unsigned long previousTime = 0;
const long timeoutTime = 10000;

void setup() {
  Serial.begin(115200);
  delay(1000);
```

```
  pinMode(light, OUTPUT);
  digitalWrite(light, LOW);
  pinMode(fan, OUTPUT);
  digitalWrite(fan, LOW);

  pinMode(proxsense, INPUT);
  pinMode(darksense, INPUT);
  pinMode(tempsense, INPUT);

  Serial.println("Setting up access point...");
  WiFi.softAP(ssid, password);

  IPAddress IP = WiFi.softAPIP();
  Serial.print("AP IP address: ");
  Serial.println(IP);

  if (IP) {
    Serial.println("Access Point created
successfully.");
  }
  else {
    Serial.println("Failed to create Access
Point.");
  }

  tempdht.begin();
  server.begin();
}


void loop() {
  WiFiClient client = server.available();

  if (client) {
    currentTime = millis();
    previousTime = currentTime;
```

```
    Serial.println("...New Client Connected...");
    String currentLine = "";

    while (client.connected() && (currentTime -
previousTime) <= timeoutTime) {

      currentTime = millis();
      if (client.available()) {
        char c = client.read();
        Serial.write(c);
        header += c;
        if (c == '\n') {
          if (currentLine.length() == 0) {
            client.println("HTTP/1.1 200 OK");
            client.println("Content-
type:text/html");
            client.println("Connection: close");
            client.println();

            sendHTML(client);

            if (mode == "manual") {
              if (header.indexOf("GET /light/on")
>= 0) {
                Serial.println("Light ON");
                lightstate = "on";
                digitalWrite(light, HIGH);
                Serial.println("Light turned ON");
              }
              else if (header.indexOf("GET
/light/off") >= 0) {
                Serial.println("Light OFF");
                lightstate = "off";
                digitalWrite(light, LOW);
                Serial.println("Light turned OFF");
              }
```

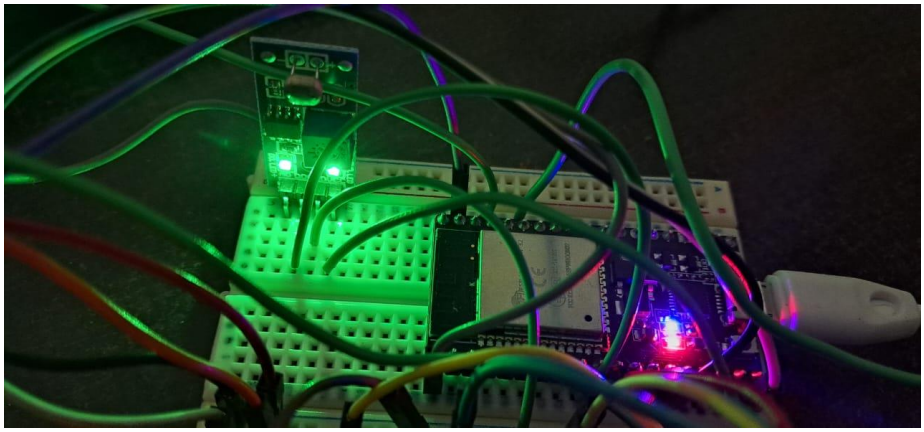
```
              if (header.indexOf("GET /fan/on") >=
0) {
                Serial.println("Fan ON");
                fanstate = "on";
                digitalWrite(fan, HIGH);
                Serial.println("Fan turned ON");
              }
              else if (header.indexOf("GET
/fan/off") >= 0) {
                Serial.println("Fan OFF");
                fanstate = "off";
                digitalWrite(fan, LOW);
                Serial.println("Fan turned OFF");
              }
            }

            if (header.indexOf("GET /mode/manual")
>= 0) {
              Serial.println("Mode MANUAL");
              mode = "manual";
              Serial.println("Mode set to MANUAL");
            }
            else if (header.indexOf("GET
/mode/auto") >= 0) {
              Serial.println("Mode AUTO");
              mode = "auto";
              Serial.println("Mode set to AUTO");
            }

            break;
          }
          else {
            currentLine = "";
          }
        }
        else if (c != '\r') {
          currentLine += c;
```

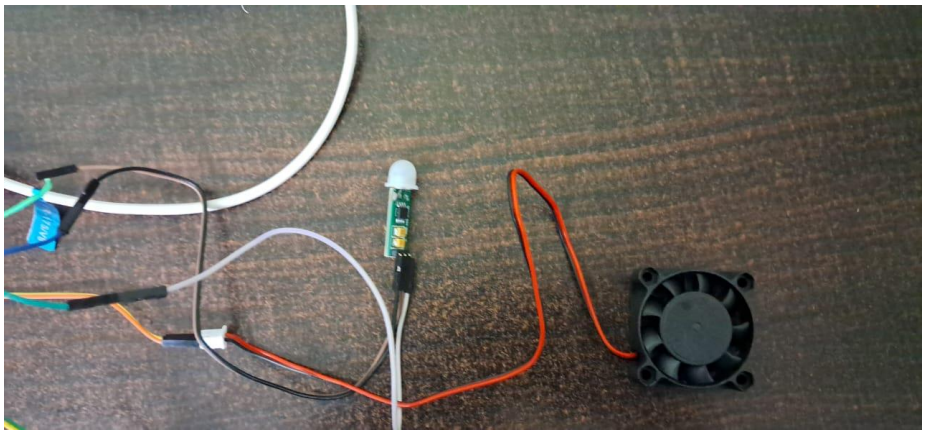
```
        }
      }
    }

    header = "";
    client.stop();
    Serial.println("Client disconnected.");
    Serial.println("");
  }

  if (mode == "auto") {
    automode();
  }
}

void sendHTML(WiFiClient client) {
  client.println("<!DOCTYPE html><html>");
  client.println("<head><meta name=\"viewport\"
content=\"width=device-width, initial-scale=1\">");
  client.println("<link rel=\"icon\"
href=\"data:,\">");

  client.println("<style>html { font-family:
monospace; display: inline-block; margin: 0px auto;
text-align: center;}");
  client.println(".button { background-color:
yellowgreen; border: none; color: white; padding:
16px 40px;");
  client.println("text-decoration: none; font-size:
32px; margin: 2px; cursor: pointer;}");
  client.println(".button2 {background-color:
gray;}</style></head>");

  client.println("<body><h1>ESP32 Web
Server</h1>");
  client.println("<p>Control Light and Fan
State</p>");
```

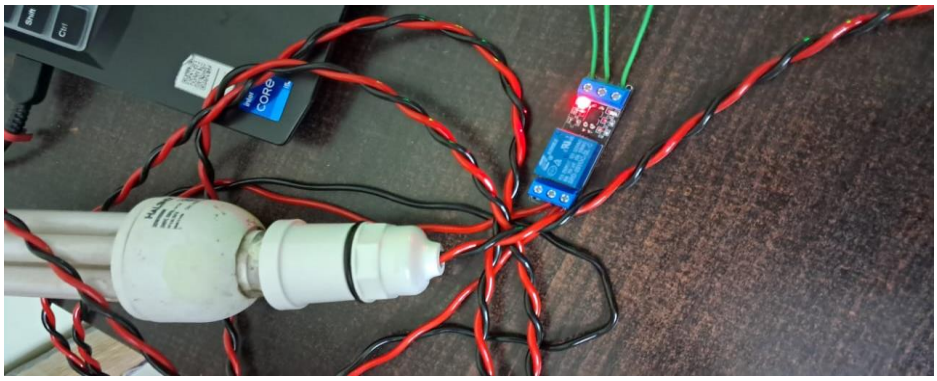
```
  if (lightstate == "off") {
    client.println("<p><a
href=\"/light/on\"><button
class=\"button\">ON</button></a></p>");
  }
  else {
    client.println("<p><a
href=\"/light/off\"><button class=\"button
button2\">OFF</button></a></p>");
  }

  if (fanstate == "off") {
    client.println("<p><a href=\"/fan/on\"><button
class=\"button\">ON</button></a></p>");
  }
  else {
    client.println("<p><a href=\"/fan/off\"><button
class=\"button button2\">OFF</button></a></p>");
  }

  if (mode == "auto") {
    client.println("<p><a
href=\"/mode/manual\"><button class=\"button
button2\">AUTO</button></a></p>");
  }
  else {
    client.println("<p><a
href=\"/mode/auto\"><button
class=\"button\">MANUAL</button></a></p>");
  }

  client.println("</body></html>");

  client.println();
}
```

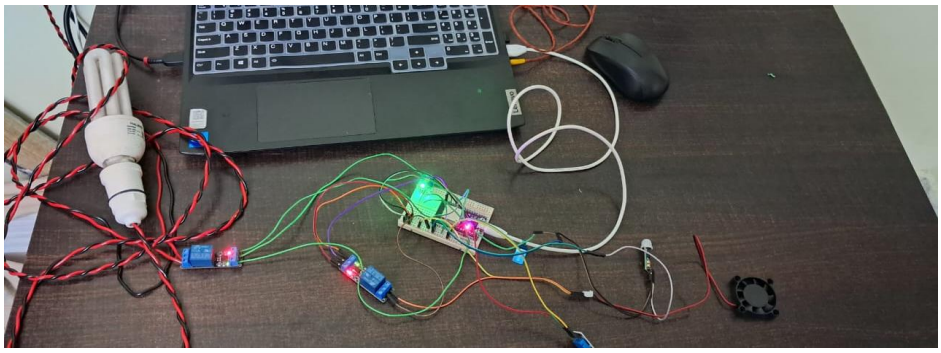
```
void automode() {
  delay(2000);
  proxcheck = digitalRead(proxsense);
  if (proxcheck == 0) {
    proxcheck = 1;
    userin = !userin;

    if (userin == true) {
      delay(2000);
      float temp = tempdht.readTemperature();
      int dark = digitalRead(darksense);

      if (temp >= templim1) {
        digitalWrite(fan, HIGH);
        fanstate = "on";
      }
      else if (temp <= templim2) {
        digitalWrite(fan, LOW);
        fanstate = "off";
      }

      if (dark == 0) {
        digitalWrite(light, HIGH);
        lightstate = "on";
      }
      else {
        digitalWrite(light, LOW);
        lightstate = "off";
      }
    }
    else {
      digitalWrite(light, LOW);
      digitalWrite(fan, LOW);
    }
  }
  delay(2000);
}
```

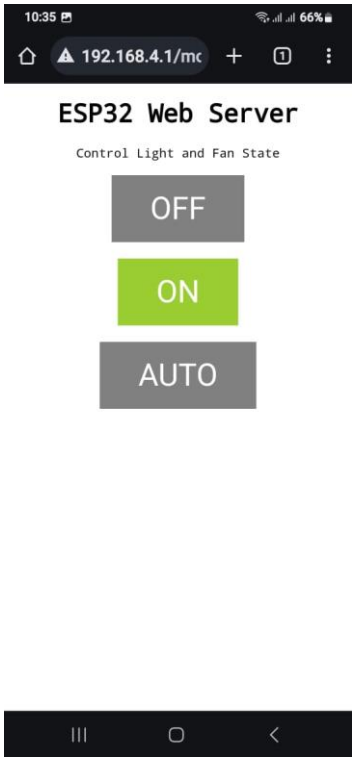
7.Testing and Results

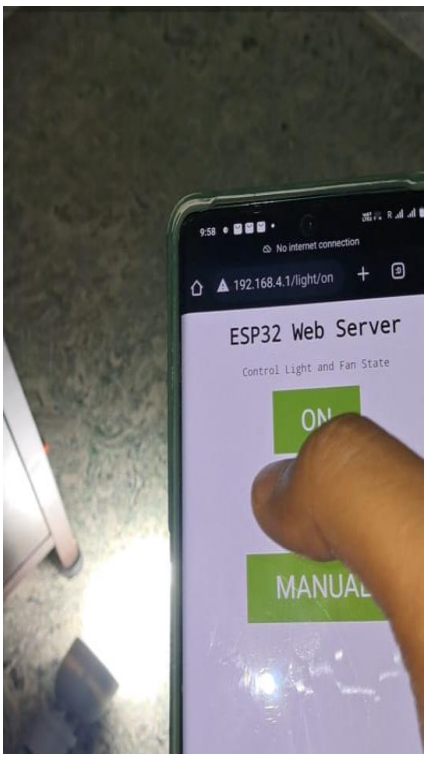












During the testing phase, each individual component was tested separately using the sample codes available in the IDE. After those tests returned successful, the manual mode functionality was tested by setting up the light bulb and fan. Through a mobile device, the appliances were controlled without any issues. Following that the auto mode functionalities were tested, with the pir, ldr and temperature sensors. Any errors occurred were quickly resolved in this debug period.

There were several bugs when testing both the modes such as timing errors, recurring reloads due to fluctuating power supply and a connection mistake. These issues were resolved, and the system is now working without any glitches or issues. It can fulfill all the required functionalities in their entirety. Therefore, the project was successfully concluded.

8.Future Work

Further Enhancements could focus on improving its usability and functionality. Accuracy and responsiveness could be heightened by integrating more intricate sensors, such as more sophisticated temperature sensors. Incorporating a dedicated mobile application will also improve accessibility and convenience for consumers by providing simpler control and real-time monitoring. Instead of collecting digital data from the ldr sensor, analog data could be processed to decide the intensity of light required. Similarly, the analog data from the temperature sensor could be used to determine the speed of the fan. The intensity of the light bulb and the speed of the fan could be controlled via a motor driver (ideally an L298n) using PWM techniques or built-in DAC instead of simple on/off operations. This would allow the most ideal ambience and environment for any situation and a better user experience.

It is also possible to integrate machine learning algorithms to anticipate user behavior and enhance the appliances' automation feature. A complete smart home solution would also include integrating more home automation devices to the system and enabling remote access via cloud services. Finally, the system would become more dependable and user-friendly by enhancing security measures to guard against illegal access and guaranteeing strong performance under varied network conditions.

THANK YOU

<<<←{([])}-/|\-{([])}→ >>>