Lab Report

Designing a Web Server for AP and Station-Based ESP32

Abstract:

This project aims to develop a web server using an ESP32 that operates in both Access Point (AP) and Station modes. Users can input RGB values through a web page to control an RGB LED. The web server also collects and displays temperature and humidity data from a DHT11 sensor. Additionally, an OLED display is used to show messages from the web interface. The project includes an extra task-based web page, designed using HTML, CSS, and JavaScript, for additional ESP32 functionalities. The system is tested in both AP and Station modes, ensuring smooth operation and user interaction.

---

1. Introduction

This project aims to develop an ESP32-based web server that integrates IoT functionalities with an interactive trivia game. The ESP32 connects to a Wi-Fi network and hosts a web interface where users can:

Control an RGB LED.

View real-time temperature and humidity readings.

Play a word-guessing trivia game.

Additionally, the project incorporates an OLED display to present messages and system status. The system is designed for accessibility and engagement, combining IoT sensor data visualization with an interactive user experience.

2. Components and Hardware Requirements

The following components were used in the implementation:

ESP32 Development Board - Serves as the central processing unit and web server.

DHT11 Temperature and Humidity Sensor - Captures real-time environmental data.

SSD1306 OLED Display (128x64, I2C) - Displays messages and sensor values.

WS2812B Neopixel RGB LED - Provides visual feedback and status indication.

WiFi Network - Required for web server connectivity in station mode.

Software & Tools:

Thonny IDE (for MicroPython programming)

Arduino IDE (alternative option)

GitHub (for version control)

Wokwi Simulator (optional)

---

3. System Design and Architecture

The system is designed to operate in two modes:

WiFi Station Mode (STA Mode): The ESP32 connects to an existing WiFi network, allowing users to interact with the web server via a browser.

WiFi Access Point Mode (AP Mode): The ESP32 creates its own WiFi hotspot, enabling direct user access when no external network is available.

Functional Modules:

3.1 Web Server

Hosts a responsive user interface accessible via a web browser.

Facilitates communication between users and the ESP32.

3.2 Trivia Game Module

Implements a word-guessing game.

Provides hints related to predefined categories.

Accepts user input and evaluates correctness.

3.3 IoT Dashboard

Displays real-time temperature and humidity data from the DHT11 sensor.

Updates information dynamically on the web interface.

3.4 OLED Display Module

Shows system status and sensor data.

Displays messages for user engagement.

3.5 RGB LED Control

Provides visual feedback based on game states and user interactions.

Changes color dynamically based on different conditions.

---

4. Software Implementation

The software for this project is implemented in MicroPython and consists of several modules.

4.1 WiFi Connectivity

To establish a WiFi connection, the following script is used:

```
import network
import time


def connect_to_wifi(ssid, password, timeout=10):
    sta = network.WLAN(network.STA_IF)
    sta.active(True)
    if not sta.isconnected():
        print(f"Connecting to WiFi: {ssid} ...")
        sta.connect(ssid, password)
        start = time.ticks_ms()
        while not sta.isconnected():
            if time.ticks_diff(time.ticks_ms(), start) > timeout * 1000:
```

```
            print("Connection timed out.")
            return None
        time.sleep(1)
    print("Connected to WiFi. IP:", sta.ifconfig()[0])
    return sta.ifconfig()[0]
```

Explanation: This function enables the ESP32 to connect to a WiFi network, retrying for a defined timeout before failing.

Temperature Humidity and RGB Webpage

Code

```
import network
import socket
import wifi_manager
import time
from machine import Pin, SoftI2C
from neopixel import NeoPixel
import dht
from ssd1306 import SSD1306_I2C


pin = Pin(48, Pin.OUT)
neo = NeoPixel(pin, 1)


dht_pin = Pin(4)
sensor = dht.DHT11(dht_pin)
```

```
i2c = SoftI2C(scl=Pin(9), sda=Pin(8))
oled = SSD1306_I2C(128, 64, i2c)

def display_message_on_oled(msg):
    oled.fill(0)
    max_chars_per_line = 16
    max_chars_total = 64
    msg = msg[:max_chars_total]
    words = msg.split(" ")
    lines = []
    current_line = ""
    for word in words:
        if len(current_line) + len(word) + 1 <= max_chars_per_line:
            current_line += (" " if current_line else "") + word
        else:
            lines.append(current_line)
            current_line = word
    if current_line:
        lines.append(current_line)
    y = 0
    for line in lines[:4]:
        oled.text(line, 5, y)
        y += 16
    oled.show()
```

```
display_message_on_oled("I AM SHEERAZ")
print("I AM SHEERAZ")

r = g = b = 0
try:
    sensor.measure()
    temp = str(sensor.temperature())
    humidity = str(sensor.humidity())
    print("Temperature:", temp, "°C")
    print("Humidity:", humidity, "%")
except Exception as e:
    print("Sensor Error:", e)
    temp = "N/A"
    humidity = "N/A"

SSID = "Heart Snatcher"
PASS = "iaminvincible07"

wifi_manager.connect_to_wifi(SSID,PASS)

wifi_manager.start_access_point("sherry-esp","connected")

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind(("0.0.0.0", 80))
```

```
s.listen(5)


def web_page(r=0, g=0, b=0, temp="N/A", humidity="N/A"):
    hex_color = '#{:02x}{:02x}{:02x}'.format(r, g, b)
    html = f"""<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <meta  name="viewport"  content="width=device-width,  initial-
scale=1">
    <title>ESP32 RGB & OLED Dashboard</title>
    <style>
        body {{
            font-family: 'Segoe UI', sans-serif;
            text-align: center;
            background: {hex_color};
            transition: background 0.3s ease;
            margin: 0; padding: 0;
        }}
        .container {{
            width: 90%;
            max-width: 500px;
            margin: 5vh auto;
            padding: 5vw;
            background: rgba(255,255,255,0.9);
```

```
            border-radius: 12px;
            box-shadow: 0 4px 16px rgba(0, 0, 0, 0.2);
        }}
        h1 {{ color: #333; margin-bottom: 10px; font-size: 1.8em;
}}
        p {{ font-size: 1.1em; color: #555; }}
        label {{ font-weight: bold; display: block; margin-top:
10px; }}
        .slider {{
            width: 100%;
            max-width: 300px;
        }}
        .color-preview {{
            width: 60px; height: 60px;
            margin: 10px auto;
            background-color: {hex_color};
            border-radius: 50%;
            border: 2px solid #888;
            transition: background-color 0.3s ease;
        }}
        input[type="number"] {{
            width: 60px;
            margin-left: 10px;
        }}
        input[type="text"] {{
            padding: 10px;
```

```
            width: 90%;
            max-width: 300px;
            margin-top: 15px;
            font-size: 1em;
        }}
    </style>
</head>
<body>
    <div class="container">
        <h1>🌈 ESP32 RGB Controller <br> Temperature and Humidity Monitor</h1>
        <p id="sensorData">🌡️ Temperature: {temp}°C    💧 Humidity: {humidity}%</p>
        <div class="color-preview" id="colorCircle"></div>

        <label>🔴 Red:</label>
        <input class="slider" type="range" id="red" min="0" max="255" value="{r}" oninput="updateRGB('red')">
        <input type="number" id="redValue" min="0" max="255" value="{r}" oninput="syncRGB('red')">

        <label>🟢 Green:</label>
        <input class="slider" type="range" id="green" min="0" max="255" value="{g}" oninput="updateRGB('green')">
        <input type="number" id="greenValue" min="0" max="255" value="{g}" oninput="syncRGB('green')">
```

```
        <label>🔵 Blue:</label>
        <input class="slider" type="range" id="blue" min="0" max="255" value="{b}" oninput="updateRGB('blue')">
        <input type="number" id="blueValue" min="0" max="255" value="{b}" oninput="syncRGB('blue')">

        <h2>📺 OLED Display</h2>
        <input type="text" id="msg" placeholder="Enter message (Max 64 chars)" maxlength="64" oninput="sendMessage()">
    </div>

    <script>
        function updateRGB(color) {{
            let value = document.getElementById(color).value;
            document.getElementById(color + "Value").value = value;
            syncAll();
        }}
        function syncRGB(color) {{
            let value = parseInt(document.getElementById(color + "Value").value);
            value = Math.min(255, Math.max(0, value));
            document.getElementById(color).value = value;
            syncAll();
        }}
        function syncAll() {{
            let r = document.getElementById("red").value;
```

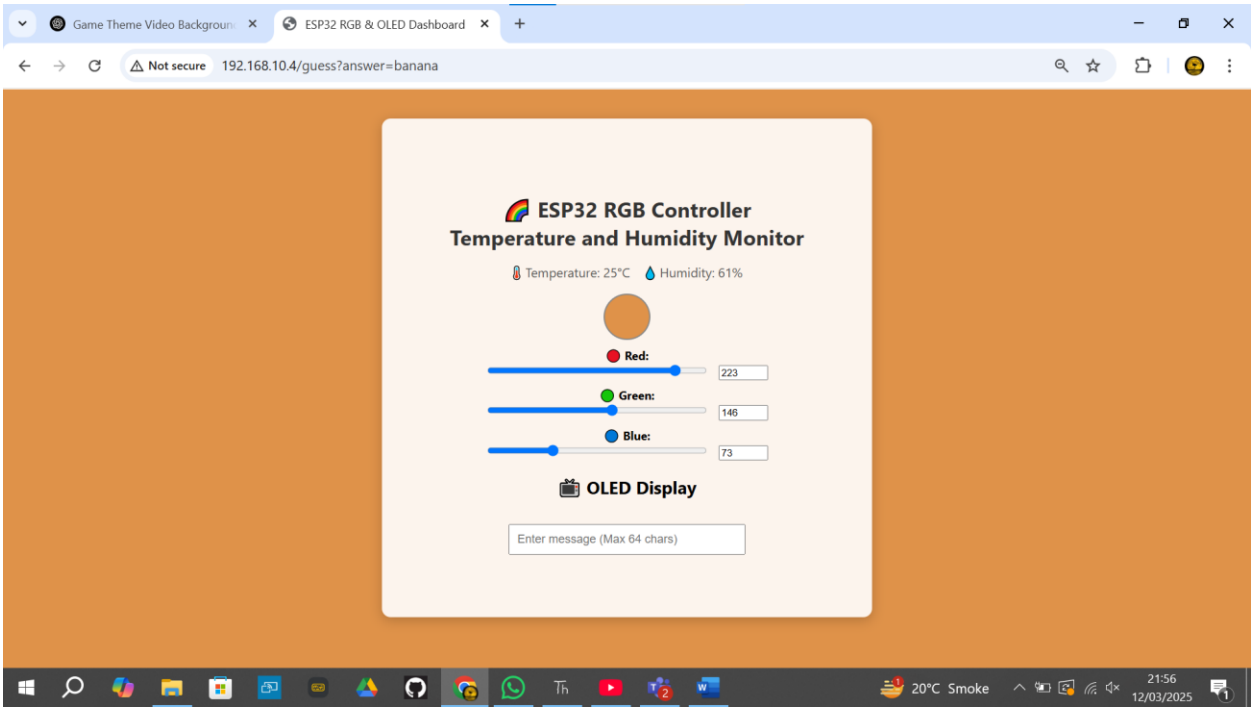
```
            let g = document.getElementById("green").value;
            let b = document.getElementById("blue").value;
            fetch("/?r=" + r + "&g=" + g + "&b=" + b);
            let colorHex = "#" + [r,g,b].map(x => ("0" + parseInt(x).toString(16)).slice(-2)).join('');

document.getElementById("colorCircle").style.backgroundColor = colorHex;
            document.body.style.backgroundColor = colorHex;
        }}
        function sendMessage() {{
            let msg = document.getElementById("msg").value;
            fetch("/?msg=" + encodeURIComponent(msg));
        }}

        setInterval(() => {{
            fetch("/sensor").then(resp => resp.json()).then(data => {{
                document.getElementById("sensorData").innerHTML =
                    `🌡️ Temperature: ${{data.temp}}°C    💧 Humidity: ${{data.humidity}}%`;
            }}).catch(err => console.log("Sensor fetch error:", err));
        }}, 2000);
    </script>
</body>
</html>"""
```

```
    return html


while True:
    conn, addr = s.accept()
    print("Connection from:", addr)
    request = conn.recv(1024).decode()
    print("Request:", request)


    if "/?r=" in request and "&g=" in request and "&b=" in request:
        try:
            parts = request.split("/?")[1].split(" ")[0]
            params = {kv.split("=")[0]: kv.split("=")[1] for kv in
parts.split("&")}
            r = min(255, max(0, int(params["r"])))
            g = min(255, max(0, int(params["g"])))
            b = min(255, max(0, int(params["b"])))
            neo[0] = (r, g, b)
            neo.write()
        except Exception as e:
            print("RGB Error:", e)


    elif "/?msg=" in request:
        try:
            msg = request.split("/?msg=")[1].split(" ")[0]
            msg = msg.replace("%20", " ")
```

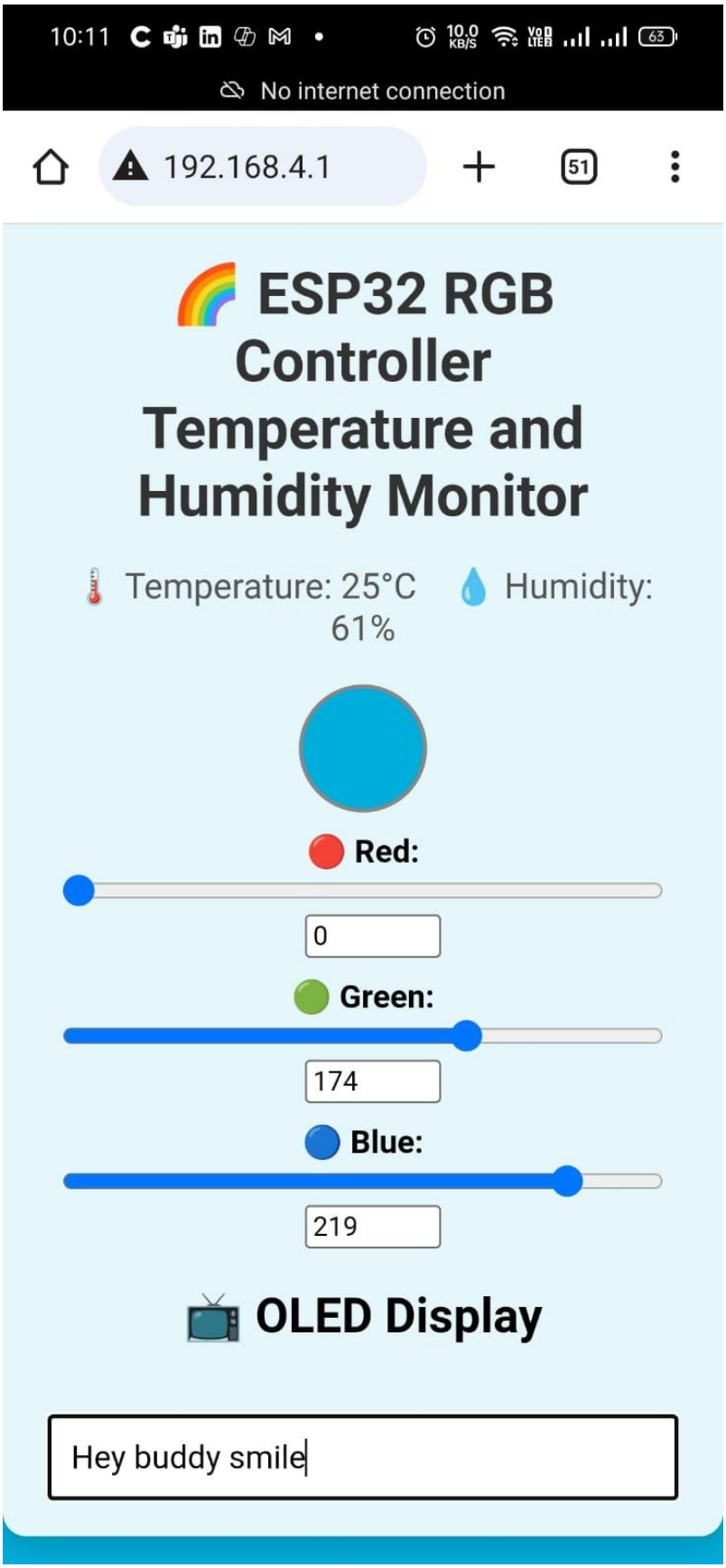
```
            display_message_on_oled(msg)
        except Exception as e:
            print("OLED Msg Error:", e)

    elif "GET /sensor" in request:
        try:
            sensor.measure()
            temp = str(sensor.temperature())
            humidity = str(sensor.humidity())
        except:
            temp = "N/A"
            humidity = "N/A"
        json_data     =     '{{"temp":     "{0}",     "humidity":
"{1}"}}'.format(temp, humidity)
        conn.send("HTTP/1.1          200          OK\nContent-Type:
application/json\n\n" + json_data)
        conn.close()
        continue

    try:
        sensor.measure()
        temp = str(sensor.temperature())
        humidity = str(sensor.humidity())
        print("Updated Temp:", temp, "Humidity:", humidity)
    except Exception as e:
        print("Sensor Read Error:", e)
```

```
        temp = "N/A"
        humidity = "N/A"

    response = web_page(r, g, b, temp, humidity)
    conn.send("HTTP/1.1 200 OK\nContent-Type: text/html\n\n" + response)
    conn.close()
```

Station Mode Desktop View



Access Point Mode Mobile View


10:11
No internet connection
192.168.4.1
51
🌈 ESP32 RGB Controller Temperature and Humidity Monitor
🌡️ Temperature: 25°C
💧 Humidity: 61%
🔴 Red:
0
🟢 Green:
174
🔵 Blue:
219
📺 OLED Display
Hey buddy smile

4.2 Trivia Game Logic

This module manages the game logic by randomly selecting a word and providing hints:

```
import urandom


def initialize_game():
    trivia_data = [
        {"category": "Science", "word": "Einstein", "hints": ["Relativity", "Physicist", "E=mc^2", "Nobel Prize"]},
        {"category": "Math", "word": "Pythagoras", "hints": ["Triangle", "Theorem", "Greek", "Right angle"]},
        {"category": "Programming", "word": "Python", "hints": ["Snake", "Language", "Indentation", "Popular"]}
    ]
    question = urandom.choice(trivia_data)
    return {
        "category": question["category"],
        "word": question["word"],
        "hints": question["hints"],
        "hints_given": 0,
        "guesses": 0,
        "status": "playing"
    }
```

Explanation: The function initializes the game by selecting a random word and assigning hints.

Trivia Game Webpage

Code

```
import network
import wifi_manager
import socket
import time
from machine import Pin, SoftI2C
from neopixel import NeoPixel
import dht
from ssd1306 import SSD1306_I2C
import urandom

dht_pin = Pin(4)
sensor = dht.DHT11(dht_pin)

i2c = SoftI2C(scl=Pin(9), sda=Pin(8))
oled = SSD1306_I2C(128, 64, i2c)

rgb_pin = Pin(48, Pin.OUT)
rgb_led = NeoPixel(rgb_pin, 1)

SSID = "Heart Snatcher"
PASS = "iaminvincible07"

trivia_data = [
    {"category": "Science", "word": "Einstein", "hints": ["Relativity", "Physicist", "E=mc^2", "Nobel Prize", "Theory of Relativity"]},
```

```
    {"category": "Geography", "word": "Nile", "hints": ["Longest river", "Africa", "Egypt", "Flows north", "Cairo"]},

    {"category": "History", "word": "Napoleon", "hints": ["French", "Emperor", "Waterloo", "Short", "Corsica"]},

    {"category": "Technology", "word": "Internet", "hints": ["WWW", "Online", "Browsing", "Data", "Google"]},

    {"category": "Math", "word": "Pythagoras", "hints": ["Triangle", "Theorem", "Greek", "Right angle", "a^2 + b^2 = c^2"]},

    {"category": "Sports", "word": "Messi", "hints": ["Football", "Argentina", "Barcelona", "PSG", "GOAT"]},

    {"category": "Literature", "word": "Shakespeare", "hints": ["Playwright", "Hamlet", "Romeo", "Macbeth", "Elizabethan"]},

    {"category": "Biology", "word": "Cell", "hints": ["Basic unit", "Life", "Organelles", "Microscope", "Membrane"]},

    {"category": "Physics", "word": "Gravity", "hints": ["Apple", "Force", "Earth", "Newton", "Pull"]},

    {"category": "Movies", "word": "Titanic", "hints": ["Ship", "Iceberg", "Jack", "Rose", "Sank"]},

    {"category": "Programming", "word": "Python", "hints": ["Snake", "Language", "Indentation", "Easy", "Popular"]},

    {"category": "Invention", "word": "Telephone", "hints": ["Communication", "Alexander Bell", "Call", "Wires", "Device"]},

    {"category": "Computer", "word": "Keyboard", "hints": ["Typing", "QWERTY", "Keys", "Input", "Device"]},

    {"category": "Space", "word": "Moon", "hints": ["Crater", "Orbit", "Night", "NASA", "Apollo"]},

    {"category": "Animals", "word": "Elephant", "hints": ["Large", "Trunk", "Tusks", "Grey", "Herbivore"]},

    {"category": "Music", "word": "Beethoven", "hints": ["Composer", "Deaf", "Symphony", "Classical", "Piano"]},
```

```
    {"category": "Countries", "word": "Japan", "hints": ["Sushi", "Samurai", "Tokyo", "Anime", "Island"]},
    {"category": "Games", "word": "Chess", "hints": ["Board", "Checkmate", "King", "Queen", "Knight"]},
    {"category": "Art", "word": "Mona Lisa", "hints": ["Leonardo", "Painting", "Smile", "Louvre", "Famous"]},
    {"category": "Plants", "word": "Rose", "hints": ["Red", "Thorn", "Flower", "Love", "Petals"]},
    {"category": "Ocean", "word": "Pacific", "hints": ["Largest", "Ocean", "Asia", "Calm", "Water"]},
    {"category": "Insects", "word": "Bee", "hints": ["Honey", "Buzz", "Yellow", "Hive", "Pollinate"]},
    {"category": "Transport", "word": "Bicycle", "hints": ["Pedal", "Wheels", "Helmet", "Chain", "Ride"]},
    {"category": "Fruits", "word": "Banana", "hints": ["Yellow", "Monkey", "Peel", "Fruit", "Sweet"]},
    {"category": "Chemistry", "word": "Oxygen", "hints": ["Gas", "Breath", "Air", "O2", "Life"]},
]


# Connect to WiFi (STA mode)
wifi_manager.connect_to_wifi(SSID, PASS)


# Start ESP32's own WiFi (AP mode)
wifi_manager.start_access_point("sherry-esp", "connected")


def display_message_on_oled(msg):
    oled.fill(0)
```

```
    y = 0
    for line in msg.split("\n")[:4]:
        oled.text(line, 0, y)
        y += 16
    oled.show()

def flash_rgb(color):
    rgb_led[0] = color
    rgb_led.write()
    time.sleep(0.5)
    rgb_led[0] = (0, 0, 0)
    rgb_led.write()

def initialize_game():
    question = urandom.choice(trivia_data)
    return {
        "category": question["category"],
        "word": question["word"],
        "hints": question["hints"],
        "hints_given": 0,
        "guesses": 0,
        "status": "playing"
    }

def determine_intelligence_score(attempts):
```

```
    if attempts == 1:
        return "You are more intelligent than 90% of people! 🎉"
    elif attempts <= 3:
        return "Great job! You performed better than 70% of people!"
    else:
        return "Nice try! Keep improving and challenge yourself!"

def web_page(game_state):
    hint = game_state["hints"][game_state["hints_given"]] if game_state["hints_given"] < len(game_state["hints"]) else "No more hints!"
    status_message = ""
    bg_color = "#00FFFF"

    if game_state["status"] == "won":
        status_message = f"<div class='result win'>🎉 Correct!<br><b>{game_state['word']}</b><br>{determine_intelligence_score(game_state['guesses'])}</div>"
        bg_color = "#4CAF50"
        display_message_on_oled("You Won!\nAnswer: " + game_state['word'])
    elif game_state["status"] == "lost":
        status_message = f"<div class='result lose'>❌ Game Over!<br>The correct answer was <b>{game_state['word']}</b></div>"
        bg_color = "#FF0000"
        display_message_on_oled("You Lost!\nAnswer: " + game_state['word'])
    else:
```

```
        red_intensity = min(255, game_state["guesses"] * 50)
        bg_color = f"rgb({red_intensity}, 255, 255)"
        display_message_on_oled(f"Hint: {hint}")

    html = f"""\
HTTP/1.1 200 OK

<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>ESP32 Trivia Game</title>
  <style>
    body {{
      font-family: 'Segoe UI', sans-serif;
      background-color: {bg_color};
      margin: 0; padding: 0;
      display: flex; justify-content: center; align-items: center;
      min-height: 100vh;
      transition: background-color 0.8s ease;
    }}
    .card {{
      background: #fff;
```

```
  max-width: 400px;
  width: 90%;
  margin: 20px;
  padding: 25px;
  box-shadow: 0 4px 12px rgba(0,0,0,0.15);
  border-radius: 16px;
  text-align: center;
}}
h1 {{
  font-size: 1.8em;
  color: #333;
}}
p {{
  font-size: 1.1em;
  margin: 10px 0;
}}
.hint {{
  font-size: 1.2em;
  color: #006;
  font-weight: bold;
}}
.attempts {{
  color: #555;
}}
input[type="text"] {{
```

```
  width: 90%;
  max-width: 300px;
  padding: 12px;
  margin-top: 10px;
  border: 2px solid #ccc;
  border-radius: 8px;
  font-size: 1em;
  outline: none;
}}
button {{
  margin-top: 15px;
  padding: 12px 24px;
  font-size: 1em;
  background: linear-gradient(45deg, #2196F3, #00BCD4);
  color: white;
  border: none;
  border-radius: 8px;
  cursor: pointer;
  transition: transform 0.2s ease;
}}
button:hover {{
  transform: scale(1.05);
  background: linear-gradient(45deg, #1976D2, #00ACC1);
}}
.result {{
```

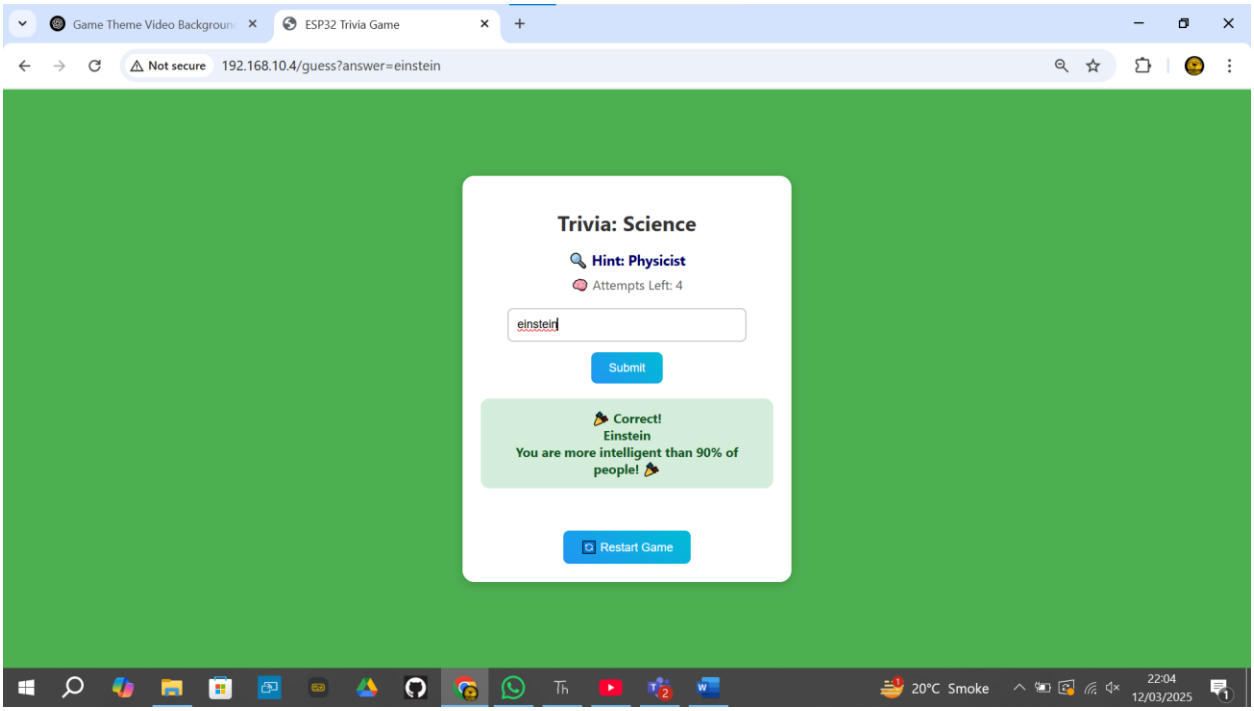
```
      margin-top: 20px;
      padding: 15px;
      font-weight: bold;
      border-radius: 12px;
      font-size: 1.1em;
    }}
    .win {{
      background-color: #d4edda;
      color: #155724;
    }}
    .lose {{
      background-color: #f8d7da;
      color: #721c24;
    }}
  </style>
</head>
<body>
  <div class="card">
    <h1>Trivia: {game_state['category']}</h1>
    <p class="hint">🔍 Hint: {hint}</p>
    <p     class="attempts">🧠     Attempts     Left:     {5     -
game_state["guesses"]}</p>

    <form action="/guess">
```

```
      <input  type="text"  name="answer"  placeholder="Enter  your
guess" required><br>
      <button type="submit">Submit</button>
    </form>

    {status_message}

    <br><br>
    <a href="/restart"><button>🔄 Restart Game</button></a>
  </div>
</body>
</html>
"""
    return html



def handle_client(conn, addr, game_state):
    request = conn.recv(1024).decode()
    if "GET / " in request:
        response = web_page(game_state)
    elif "GET /guess?answer=" in request:
        answer        =        request.split("answer=")[1].split("
")[0].replace("%20", " ").lower()
        correct = answer == game_state["word"].lower()
        if correct:
            game_state["status"] = "won"
```

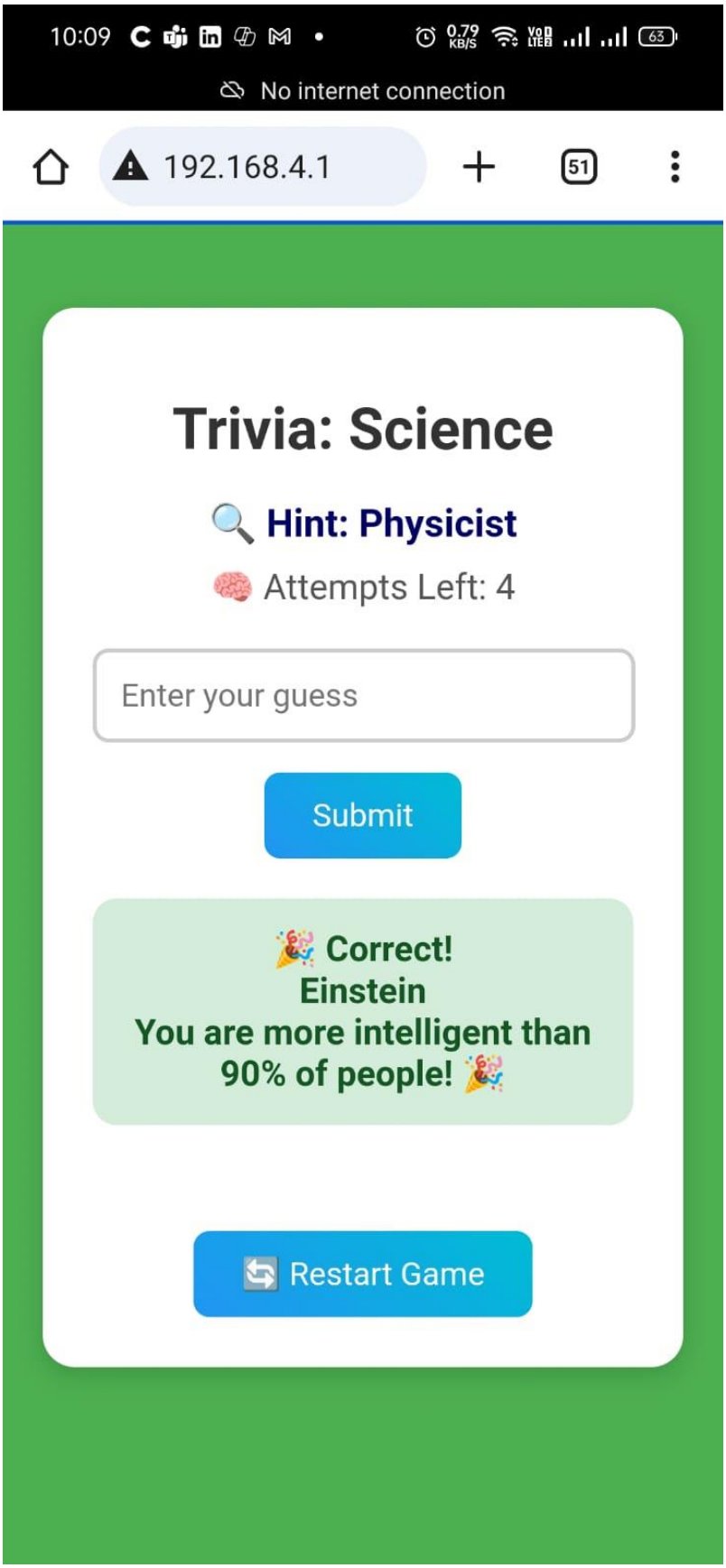
```
            flash_rgb((0, 255, 0))  # Green for correct
        else:
            game_state["guesses"] += 1
            flash_rgb((255, 0, 0))  # Red for wrong
            if game_state["guesses"] >= 5:
                game_state["status"] = "lost"
            else:
                game_state["hints_given"]                          =
min(game_state["hints_given"] + 1, len(game_state["hints"]) - 1)
        response = web_page(game_state)
    elif "GET /restart" in request:
        game_state.clear()
        game_state.update(initialize_game())
        response = web_page(game_state)
    else:
        response = "HTTP/1.1 404 Not Found\n\nPage Not Found"
    conn.send(response)
    conn.close()


game_state = initialize_game()


addr = socket.getaddrinfo('0.0.0.0', 80)[0][-1]
s = socket.socket()
s.setsockopt(socket.SOL_SOCKET,  socket.SO_REUSEADDR,  1)   #  Allow
address reuse
s.bind(addr)
```

```
s.listen(1)

print("Trivia Game server running...")

while True:
    conn, addr = s.accept()
    handle_client(conn, addr, game_state)
```

Station Mode Laptop View



Access Point Mode Mobile View


10:09
No internet connection
192.168.4.1
51
Trivia: Science
🔍 Hint: Physicist
🧠 Attempts Left: 4
Enter your guess
Submit
🎉 Correct!
Einstein
You are more intelligent than
90% of people! 🎉
🔄 Restart Game

4.3 Web Server Implementation (Socket Programming)

The ESP32 runs an HTTP server using socket programming to handle user requests:

```
import socket


def start_server():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('0.0.0.0', 80))
    server_socket.listen(5)
    print("Web server running...")
    while True:
        conn, addr = server_socket.accept()
        request = conn.recv(1024).decode()
        response = web_page()
        conn.send('HTTP/1.1 200 OK\nContent-Type: text/html\n\n' + response)
        conn.close()
```

Explanation: The function initializes an HTTP server that listens for client requests, processes them, and serves the corresponding HTML page.

4.4 Web Page Generation

```
def web_page():
    html = """
    <html>
    <head>
        <title>ESP32 Trivia Game</title>
    </head>
```

```
    <body>
        <h1>ESP32 Web Server</h1>
        <p>Interactive trivia game and IoT dashboard.</p>
    </body>
    </html>
    """
    return html
```

Explanation: This function works for the webpage functionalities.