

```

#include <WiFi.h>
#include <WebServer.h>
#include <Wire.h>
#include <Adafruit_MPR121.h>

// WiFi credentials - ADD YOUR CREDENTIALS HERE
const char* ssid = _____;
const char* password = _____;

WebServer server(80);
int displayCount = 0;

// I2C pins for MPR121 sensors
#define SDA_PIN 3
#define SCL_PIN 8

// MPR121 sensor instances
Adafruit_MPR121 cap1 = Adafruit_MPR121(); // sensor at 0x5A (sensor A)
Adafruit_MPR121 cap2 = Adafruit_MPR121(); // sensor at 0x5B (sensor B)
Adafruit_MPR121 cap3 = Adafruit_MPR121(); // sensor at 0x5C (sensor C)

// LED Matrix Pin assignments
const int columns[] = {43, 44, 42, 41, 40, 39}; // Negative
const int rows[] = {38, 37, 36, 35, 0, 45}; // Positive

// Current character to display and touch interaction
String currentChar = "A";
int currentPattern[6][6]; // The target pattern for the current character
int activatedPattern[6][6]; // Which LEDs have been activated by touch
unsigned long lastMultiplexTime = 0;
int currentRow = 0;

// Touch state tracking
uint16_t lastTouched1 = 0, lastTouched2 = 0, lastTouched3 = 0;
bool interactiveMode = true; // When true, only show LEDs that are both in pattern
AND touched

// Character patterns (6x6 matrix, 1 = LED ON, 0 = LED OFF)
int patterns[36][6][6] = {
  // Number 0
  {{0,1,1,1,1,0},
   {1,0,0,0,0,1},

```

```
{1,0,0,0,0,1},
{1,0,0,0,0,1},
{1,0,0,0,0,1},
{0,1,1,1,1,0}},
// Number 1
{{0,0,1,0,0,0},
{0,1,1,0,0,0},
{0,0,1,0,0,0},
{0,0,1,0,0,0},
{0,0,1,0,0,0},
{0,1,1,1,0,0}},
// Number 2
{{1,1,1,1,1,1},
{0,0,0,0,0,1},
{1,1,1,1,1,1},
{1,0,0,0,0,0},
{1,1,1,1,1,1},
{0,0,0,0,0,0}},
// Number 3
{{1,1,1,1,1,0},
{0,0,0,0,0,1},
{0,1,1,1,1,0},
{0,0,0,0,0,1},
{0,0,0,0,0,1},
{1,1,1,1,1,0}},
// Number 4
{{1,0,0,0,1,0},
{1,0,0,0,1,0},
{1,1,1,1,1,1},
{0,0,0,0,1,0},
{0,0,0,0,1,0},
{0,0,0,0,1,0}},
// Number 5
{{1,1,1,1,1,1},
{1,0,0,0,0,0},
{1,1,1,1,1,0},
{0,0,0,0,0,1},
{0,0,0,0,0,1},
{1,1,1,1,1,0}},
// Number 6
{{0,1,1,1,1,0},
{1,0,0,0,0,0},
```

```
{1,1,1,1,1,0},
{1,0,0,0,0,1},
{1,0,0,0,0,1},
{0,1,1,1,1,0}},
// Number 7
{{1,1,1,1,1,1},
{0,0,0,0,0,1},
{0,0,0,0,1,0},
{0,0,0,1,0,0},
{0,0,1,0,0,0},
{0,1,0,0,0,0}},
// Number 8
{{0,1,1,1,1,0},
{1,0,0,0,0,1},
{0,1,1,1,1,0},
{1,0,0,0,0,1},
{1,0,0,0,0,1},
{0,1,1,1,1,0}},
// Number 9
{{0,1,1,1,1,0},
{1,0,0,0,0,1},
{1,0,0,0,0,1},
{0,1,1,1,1,1},
{0,0,0,0,0,1},
{0,1,1,1,1,0}},
// Letter A
{{0,1,1,1,1,0},
{1,0,0,0,0,1},
{1,0,0,0,0,1},
{1,1,1,1,1,1},
{1,0,0,0,0,1},
{1,0,0,0,0,1}},
// Letter B
{{1,1,1,1,1,0},
{1,0,0,0,0,1},
{1,1,1,1,1,0},
{1,0,0,0,0,1},
{1,0,0,0,0,1},
{1,1,1,1,1,0}},
// Letter C
{{0,1,1,1,1,0},
{1,0,0,0,0,1},
```

```
{1,0,0,0,0,0},
{1,0,0,0,0,0},
{1,0,0,0,0,1},
{0,1,1,1,1,0}},
// Letter D
{{1,1,1,1,0,0},
{1,0,0,0,1,0},
{1,0,0,0,0,1},
{1,0,0,0,0,1},
{1,0,0,0,1,0},
{1,1,1,1,0,0}},
// Letter E
{{1,1,1,1,1,1},
{1,0,0,0,0,0},
{1,1,1,1,0,0},
{1,0,0,0,0,0},
{1,0,0,0,0,0},
{1,1,1,1,1,1}},
// Letter F
{{1,1,1,1,1,1},
{1,0,0,0,0,0},
{1,1,1,1,0,0},
{1,0,0,0,0,0},
{1,0,0,0,0,0},
{1,0,0,0,0,0}},
// Letter G
{{0,1,1,1,1,0},
{1,0,0,0,0,1},
{1,0,0,0,0,0},
{1,0,0,1,1,1},
{1,0,0,0,0,1},
{0,1,1,1,1,0}},
// Letter H
{{1,0,0,0,0,1},
{1,0,0,0,0,1},
{1,1,1,1,1,1},
{1,0,0,0,0,1},
{1,0,0,0,0,1},
{1,0,0,0,0,1}},
// Letter I
{{1,1,1,1,1,1},
{0,0,1,1,0,0},
```

```
{0,0,1,1,0,0},
{0,0,1,1,0,0},
{0,0,1,1,0,0},
{1,1,1,1,1,1}},
// Letter J
{{0,0,0,0,0,1},
{0,0,0,0,0,1},
{0,0,0,0,0,1},
{0,0,0,0,0,1},
{1,0,0,0,0,1},
{0,1,1,1,1,0}},
// Letter K
{{0,1,0,0,0,1},
{0,1,0,0,1,0},
{0,1,0,1,0,0},
{0,1,1,0,0,0},
{0,1,0,1,0,0},
{0,1,0,0,1,0}},
// Letter L
{{1,0,0,0,0,0},
{1,0,0,0,0,0},
{1,0,0,0,0,0},
{1,0,0,0,0,0},
{1,0,0,0,0,0},
{1,1,1,1,1,1}},
// Letter M
{{1,0,0,0,0,1},
{1,1,0,0,1,1},
{1,0,1,1,0,1},
{1,0,0,0,0,1},
{1,0,0,0,0,1},
{1,0,0,0,0,1}},
// Letter N
{{1,0,0,0,0,1},
{1,1,0,0,0,1},
{1,0,1,0,0,1},
{1,0,0,1,0,1},
{1,0,0,0,1,1},
{1,0,0,0,0,1}},
// Letter O
{{0,1,1,1,1,0},
{1,0,0,0,0,1},
```

```
{1,0,0,0,0,1},
{1,0,0,0,0,1},
{1,0,0,0,0,1},
{0,1,1,1,1,0}},
// Letter P
{{1,1,1,1,1,0},
{1,0,0,0,0,1},
{1,0,0,0,0,1},
{1,1,1,1,1,0},
{1,0,0,0,0,0},
{1,0,0,0,0,0}},
// Letter Q
{{0,1,1,1,1,0},
{1,0,0,0,0,1},
{1,0,0,0,0,1},
{1,0,0,1,0,1},
{1,0,0,0,1,1},
{0,1,1,1,1,1}},
// Letter R
{{1,1,1,1,1,0},
{1,0,0,0,0,1},
{1,0,0,0,0,1},
{1,1,1,1,1,0},
{1,0,0,1,0,0},
{1,0,0,0,1,0}},
// Letter S
{{0,1,1,1,1,1},
{1,0,0,0,0,0},
{0,1,1,1,1,0},
{0,0,0,0,0,1},
{0,0,0,0,0,1},
{1,1,1,1,1,0}},
// Letter T
{{1,1,1,1,1,1},
{0,0,1,1,0,0},
{0,0,1,1,0,0},
{0,0,1,1,0,0},
{0,0,1,1,0,0},
{0,0,1,1,0,0}},
// Letter U
{{1,0,0,0,0,1},
{1,0,0,0,0,1},
```

```

    {1,0,0,0,0,1},
    {1,0,0,0,0,1},
    {1,0,0,0,0,1},
    {0,1,1,1,1,0}},
    // Letter V
    {{1,0,0,0,0,1},
    {1,0,0,0,0,1},
    {1,0,0,0,0,1},
    {0,1,0,0,1,0},
    {0,1,0,0,1,0},
    {0,0,1,1,0,0}},
    // Letter W
    {{1,0,0,0,0,1},
    {1,0,0,0,0,1},
    {1,0,0,0,0,1},
    {1,0,1,1,0,1},
    {1,1,0,0,1,1},
    {1,0,0,0,0,1}},
    // Letter X
    {{1,0,0,0,0,1},
    {0,1,0,0,1,0},
    {0,0,1,1,0,0},
    {0,0,1,1,0,0},
    {0,1,0,0,1,0},
    {1,0,0,0,0,1}},
    // Letter Y
    {{1,0,0,0,0,1},
    {0,1,0,0,1,0},
    {0,0,1,1,0,0},
    {0,0,1,1,0,0},
    {0,0,1,1,0,0},
    {0,0,1,1,0,0}},
    // Letter Z
    {{1,1,1,1,1,1},
    {0,0,0,0,1,0},
    {0,0,0,1,0,0},
    {0,0,1,0,0,0},
    {0,1,0,0,0,0},
    {1,1,1,1,1,1}}
};

// Function to map MPR121 touch to LED matrix coordinates

```

```

void getTouchCoordinates(int sensor, int pin, int &x, int &y) {
    if (sensor == 0x5A) { // Sensor A - Top 2 rows
        if (pin == 11) { x = 0; y = 0; }
        else if (pin == 10) { x = 1; y = 0; }
        else if (pin == 9) { x = 2; y = 0; }
        else if (pin == 8) { x = 3; y = 0; }
        else if (pin == 7) { x = 4; y = 0; }
        else if (pin == 6) { x = 5; y = 0; }
        else if (pin == 5) { x = 0; y = 1; }
        else if (pin == 4) { x = 1; y = 1; }
        else if (pin == 3) { x = 2; y = 1; }
        else if (pin == 2) { x = 3; y = 1; }
        else if (pin == 1) { x = 4; y = 1; }
        else if (pin == 0) { x = 5; y = 1; }
    }
    else if (sensor == 0x5B) { // Sensor B - Middle 2 rows (was C before)
        if (pin == 6) { x = 0; y = 2; }
        else if (pin == 7) { x = 1; y = 2; }
        else if (pin == 9) { x = 2; y = 2; }
        else if (pin == 8) { x = 3; y = 2; }
        else if (pin == 10) { x = 4; y = 2; }
        else if (pin == 11) { x = 5; y = 2; }
        else if (pin == 0) { x = 0; y = 3; }
        else if (pin == 2) { x = 1; y = 3; }
        else if (pin == 1) { x = 2; y = 3; }
        else if (pin == 3) { x = 3; y = 3; }
        else if (pin == 4) { x = 4; y = 3; }
        else if (pin == 5) { x = 5; y = 3; }
    }
    else if (sensor == 0x5C) { // Sensor C - Bottom 2 rows (was B before)
        if (pin == 6) { x = 0; y = 4; }
        else if (pin == 7) { x = 1; y = 4; }
        else if (pin == 8) { x = 2; y = 4; }
        else if (pin == 9) { x = 3; y = 4; }
        else if (pin == 10) { x = 4; y = 4; }
        else if (pin == 11) { x = 5; y = 4; }
        else if (pin == 0) { x = 0; y = 5; }
        else if (pin == 1) { x = 1; y = 5; }
        else if (pin == 2) { x = 2; y = 5; }
        else if (pin == 3) { x = 3; y = 5; }
        else if (pin == 4) { x = 4; y = 5; }
        else if (pin == 5) { x = 5; y = 5; }
    }
}

```



```

}
}

// Function to handle touch input
void processTouchInput() {
    uint16_t touched1 = cap1.touched();
    uint16_t touched2 = cap2.touched();
    uint16_t touched3 = cap3.touched();
    // Process sensor A (0x5A) touches
    for (uint8_t i = 0; i < 12; i++) {
        if ((touched1 & (1 << i)) && !(lastTouched1 & (1 << i))) {
            int x, y;
            getTouchCoordinates(0x5A, i, x, y);

            // Check if this position should be lit for the current character
            if (currentPattern[y][x] == 1) {
                activatedPattern[y][x] = 1; // Activate this LED
                Serial.printf("✅ Activated LED at (%d,%d) - Sensor A, Pin %d\n", x, y, i);
                checkPatternCompletion();
            } else {
                Serial.printf("❌ Touched inactive position (%d,%d) - Sensor A, Pin %d\n", x,
y, i);
            }
        }
    }
    // Process sensor B (0x5B) touches
    for (uint8_t i = 0; i < 12; i++) {
        if ((touched2 & (1 << i)) && !(lastTouched2 & (1 << i))) {
            int x, y;
            getTouchCoordinates(0x5B, i, x, y);

            // Check if this position should be lit for the current character
            if (currentPattern[y][x] == 1) {
                activatedPattern[y][x] = 1; // Activate this LED
                Serial.printf("✅ Activated LED at (%d,%d) - Sensor B, Pin %d\n", x, y, i);
                checkPatternCompletion();
            } else {
                Serial.printf("❌ Touched inactive position (%d,%d) - Sensor B, Pin %d\n", x,
y, i);
            }
        }
    }
}

```

```

// Process sensor C (0x5C) touches
for (uint8_t i = 0; i < 12; i++) {
    if ((touched3 & (1 << i)) && !(lastTouched3 & (1 << i))) {
        int x, y;
        getTouchCoordinates(0x5C, i, x, y);

        // Check if this position should be lit for the current character
        if (currentPattern[y][x] == 1) {
            activatedPattern[y][x] = 1; // Activate this LED
            Serial.printf("✅ Activated LED at (%d,%d) - Sensor C, Pin %d\n", x, y, i);
            checkPatternCompletion();
        } else {
            Serial.printf("❌ Touched inactive position (%d,%d) - Sensor C, Pin %d\n", x,
y, i);
        }
    }
}

// Store current touch states
lastTouched1 = touched1;
lastTouched2 = touched2;
lastTouched3 = touched3;
}

// Function to check if the pattern is completed
void checkPatternCompletion() {
    int totalRequired = 0;
    int totalActivated = 0;
    for (int r = 0; r < 6; r++) {
        for (int c = 0; c < 6; c++) {
            if (currentPattern[r][c] == 1) {
                totalRequired++;
                if (activatedPattern[r][c] == 1) {
                    totalActivated++;
                }
            }
        }
    }
}

Serial.printf("Progress: %d/%d LEDs activated\n", totalActivated, totalRequired);
if (totalActivated == totalRequired && totalRequired > 0) {
    Serial.println("🎉 Pattern completed! Well done!");
    // Could add celebration effects here
}

```

```

}

const char* htmlTemplate = R"rawliteral(
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>ESP32 Touch-Sensitive LED Matrix</title>
  <style>
    body {
      margin: 0;
      background: linear-gradient(to bottom right, #dbeafe, #f0f9ff);
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
      color: #1e3a8a;
    }

    .header {
      position: fixed;
      top: 0;
      left: 0;
      width: 100%;
      background-color: #1e3a8a;
      color: white;
      padding: 12px 16px;
      font-size: 1.2em;
      font-weight: 600;
      box-shadow: 0 2px 6px rgba(0, 0, 0, 0.1);
      display: flex;
      justify-content: space-between;
      align-items: center;
      z-index: 1000;
      box-sizing: border-box;
    }

    .title {
      flex: 1;
    }

    .score {
      font-size: 1em;
      background-color: #3b82f6;

```

```
padding: 8px 16px;
border-radius: 20px;
font-weight: 500;
}

.content {
margin-top: 80px;
padding: 20px;
text-align: center;
}

.instructions {
background-color: #e0f2fe;
color: #0c4a6e;
border-radius: 12px;
padding: 20px;
margin: 0 auto 40px auto;
font-size: 1.1em;
max-width: 700px;
box-shadow: 0 4px 12px rgba(0, 0, 0, 0.06);
}

.touch-info {
background-color: #f0fdf4;
color: #166534;
border-radius: 12px;
padding: 20px;
margin: 0 auto 40px auto;
font-size: 1em;
max-width: 700px;
box-shadow: 0 4px 12px rgba(0, 0, 0, 0.06);
}

h1, h2 {
color: #1e40af;
margin: 30px 0 20px 0;
}

.section {
margin-bottom: 40px;
}
```

```
.button-container {
  display: flex;
  justify-content: center;
  flex-wrap: wrap;
  gap: 8px;
  margin-bottom: 20px;
  max-width: 800px;
  margin-left: auto;
  margin-right: auto;
}

.btn {
  background-color: #2563eb;
  color: white;
  border: none;
  padding: 12px 16px;
  font-size: 1.1em;
  border-radius: 8px;
  cursor: pointer;
  transition: all 0.2s ease;
  min-width: 50px;
  font-weight: 600;
}

.btn:hover {
  background-color: #1d4ed8;
  transform: translateY(-1px);
  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.15);
}

.btn.number {
  background-color: #059669;
}

.btn.number:hover {
  background-color: #047857;
}

.feedback {
  font-size: 1.4em;
  font-weight: bold;
  padding: 12px;
```

```

    margin-top: 20px;
}

.success {
    color: #16a34a;
}

.current-display {
    background-color: #f8fafc;
    border: 2px solid #e2e8f0;
    border-radius: 12px;
    padding: 20px;
    margin: 20px auto;
    max-width: 400px;
    font-size: 1.2em;
    color: #475569;
}

@media (max-width: 600px) {
    .btn {
        font-size: 1em;
        padding: 10px 12px;
        min-width: 40px;
    }

    .button-container {
        gap: 6px;
    }
}
</style>
</head>
<body>
<div class="header">
    <div class="title">ESP32 Touch-Sensitive LED Matrix</div>
    <div class="score">Displays: <span id="score">0</span></div>
</div>

<div class="content">
    <div class="instructions">
        <p>Click any letter or number button below to display it on the 6x6 LED
matrix!</p>

        <p>The ESP32 controls both this web interface and the LED matrix directly.</p>
    </div>

```

```

</div>

<div class="touch-info">
  <p><strong>🎯 Interactive Touch Mode Active!</strong></p>
  <p>1. Click a character button below to set the target pattern</p>
  <p>2. Touch the positions on the LED matrix that should light up</p>
  <p>3. Only the LEDs you touch (that are part of the pattern) will actually light
up!</p>
  <p>4. Try to complete the entire character by touching all the correct
positions</p>

  <div style="margin-top: 15px;">
    <button class="btn" onclick="toggleMode()" id="modeBtn">Switch to Display
Mode</button>
    <button class="btn" onclick="resetPattern()" style="background-color: #dc2626;
margin-left: 10px;">Reset Pattern</button>
  </div>
</div>

<div class="current-display" id="currentDisplay">
  Currently displaying: A
</div>

<div class="section">
  <h2>Letters (A-Z)</h2>
  <div class="button-container">
    <button class="btn" onclick="sendCharacter('A')">A</button>
    <button class="btn" onclick="sendCharacter('B')">B</button>
    <button class="btn" onclick="sendCharacter('C')">C</button>
    <button class="btn" onclick="sendCharacter('D')">D</button>
    <button class="btn" onclick="sendCharacter('E')">E</button>
    <button class="btn" onclick="sendCharacter('F')">F</button>
    <button class="btn" onclick="sendCharacter('G')">G</button>
    <button class="btn" onclick="sendCharacter('H')">H</button>
    <button class="btn" onclick="sendCharacter('I')">I</button>
    <button class="btn" onclick="sendCharacter('J')">J</button>
    <button class="btn" onclick="sendCharacter('K')">K</button>
    <button class="btn" onclick="sendCharacter('L')">L</button>
    <button class="btn" onclick="sendCharacter('M')">M</button>
    <button class="btn" onclick="sendCharacter('N')">N</button>
    <button class="btn" onclick="sendCharacter('O')">O</button>
    <button class="btn" onclick="sendCharacter('P')">P</button>
  </div>

```

```

    <button class="btn" onclick="sendCharacter('Q')">Q</button>
    <button class="btn" onclick="sendCharacter('R')">R</button>
    <button class="btn" onclick="sendCharacter('S')">S</button>
    <button class="btn" onclick="sendCharacter('T')">T</button>
    <button class="btn" onclick="sendCharacter('U')">U</button>
    <button class="btn" onclick="sendCharacter('V')">V</button>
    <button class="btn" onclick="sendCharacter('W')">W</button>
    <button class="btn" onclick="sendCharacter('X')">X</button>
    <button class="btn" onclick="sendCharacter('Y')">Y</button>
    <button class="btn" onclick="sendCharacter('Z')">Z</button>
  </div>
</div>

<div class="section">
  <h2>Numbers (0-9)</h2>
  <div class="button-container">
    <button class="btn number" onclick="sendCharacter('0')">0</button>
    <button class="btn number" onclick="sendCharacter('1')">1</button>
    <button class="btn number" onclick="sendCharacter('2')">2</button>
    <button class="btn number" onclick="sendCharacter('3')">3</button>
    <button class="btn number" onclick="sendCharacter('4')">4</button>
    <button class="btn number" onclick="sendCharacter('5')">5</button>
    <button class="btn number" onclick="sendCharacter('6')">6</button>
    <button class="btn number" onclick="sendCharacter('7')">7</button>
    <button class="btn number" onclick="sendCharacter('8')">8</button>
    <button class="btn number" onclick="sendCharacter('9')">9</button>
  </div>
</div>

<div class="feedback" id="feedback"></div>
</div>

<script>
  let isInteractiveMode = true;

  function sendCharacter(char) {
    fetch(`/display?char=${char}`)
      .then(response => response.json())
      .then(data => {
        const feedback = document.getElementById("feedback");
        const scoreSpan = document.getElementById("score");
        const currentDisplay = document.getElementById("currentDisplay");

```



```

        if (data.success) {
            if (isInteractiveMode) {
                feedback.textContent = `Pattern set to "${char}" - Touch the LED
positions to activate them!`;
            } else {
                feedback.textContent = `Displaying "${char}" on LED matrix`;
            }
            feedback.className = "feedback success";
            scoreSpan.textContent = data.count;
            currentDisplay.textContent = `Currently displaying: ${char}`;
        } else {
            feedback.textContent = "Error displaying character";
            feedback.className = "feedback";
        }
    })
    .catch(error => {
        console.error('Error:', error);
        document.getElementById("feedback").textContent = "Connection error";
    });
}

function toggleMode() {
    const newMode = isInteractiveMode ? 'display' : 'interactive';

    fetch(`/mode?mode=${newMode}`)
        .then(response => response.json())
        .then(data => {
            if (data.success) {
                isInteractiveMode = !isInteractiveMode;
                const modeBtn = document.getElementById("modeBtn");
                const touchInfo = document.querySelector(".touch-info");

                if (isInteractiveMode) {
                    modeBtn.textContent = "Switch to Display Mode";
                    touchInfo.innerHTML = `
                        <p><strong>🎯 Interactive Touch Mode Active!</strong></p>
                        <p>1. Click a character button below to set the target pattern</p>
                        <p>2. Touch the positions on the LED matrix that should light up</p>
                        <p>3. Only the LEDs you touch (that are part of the pattern) will
actually light up!</p>

```

```

    <p>4. Try to complete the entire character by touching all the correct
positions</p>

    <div style="margin-top: 15px;">
        <button class="btn" onclick="toggleMode()" id="modeBtn">Switch to
Display Mode</button>
        <button class="btn" onclick="resetPattern()" style="background-color:
#dc2626; margin-left: 10px;">Reset Pattern</button>
    </div>
    `;
} else {
    modeBtn.textContent = "Switch to Interactive Mode";
    touchInfo.innerHTML = `
        <p><strong>📺 Display Mode Active!</strong></p>
        <p>Characters will be displayed immediately when you click the buttons
below.</p>
        <p>The full pattern will light up automatically without requiring touch
interaction.</p>

        <div style="margin-top: 15px;">
            <button class="btn" onclick="toggleMode()" id="modeBtn">Switch to
Interactive Mode</button>
        </div>
        `;
    }

    document.getElementById("feedback").textContent = `Switched to ${newMode}
mode`;
    document.getElementById("feedback").className = "feedback success";
    }
    })
    .catch(error => {
        console.error('Error:', error);
    });
}

function resetPattern() {
    fetch('/reset')
        .then(response => response.json())
        .then(data => {
            if (data.success) {

```

```

        document.getElementById("feedback").textContent = "Pattern reset - all LEDs
deactivated";
        document.getElementById("feedback").className = "feedback success";
    }
})
.catch(error => {
    console.error('Error:', error);
});
}
</script>
</body>
</html>
)rawliteral";

```

```

int getCharacterIndex(char c) {
    if (c >= '0' && c <= '9') {
        return c - '0'; // Numbers 0-9 are indices 0-9
    } else if (c >= 'A' && c <= 'Z') {
        return (c - 'A') + 10; // Letters A-Z are indices 10-35
    }
    return 10; // Default to 'A' if character not found
}

```

```

void setCharacterPattern(String character) {
    if (character.length() > 0) {
        char c = character.charAt(0);
        int index = getCharacterIndex(c);

        // Copy the pattern to currentPattern
        for (int r = 0; r < 6; r++) {
            for (int col = 0; col < 6; col++) {
                currentPattern[r][col] = patterns[index][r][col];
                activatedPattern[r][col] = 0; // Reset activated pattern
            }
        }
        currentChar = character;
        Serial.println("LED Matrix target pattern set to: " + character);
        Serial.println("Touch the positions that should light up to activate them!");

        // Count total LEDs needed for this character
        int totalLEDs = 0;
        for (int r = 0; r < 6; r++) {

```

```

        for (int c = 0; c < 6; c++) {
            if (currentPattern[r][c] == 1) totalLEDs++;
        }
    }
    Serial.printf("Character '%s' requires touching %d positions\n", character.c_str(),
totalLEDs);
}
}

void updateLEDMatrix() {
    unsigned long currentTime = micros();
    // Much faster refresh rate for video recording - update every 500 microseconds
    (0.5ms)
    // This gives us 2000Hz refresh rate instead of 500Hz
    if (currentTime - lastMultiplexTime >= 500) {
        // Turn off all rows first
        for (int i = 0; i < 6; i++) digitalWrite(rows[i], LOW);

        // Set column states for current row
        for (int c = 0; c < 6; c++) {
            bool ledOn = false;

            if (interactiveMode) {
                // Only light LED if it's both in the pattern AND has been activated by touch
                ledOn = (currentPattern[currentRow][c] == 1) &&
(activatedPattern[currentRow][c] == 1);
            } else {
                // Traditional mode - show full pattern
                ledOn = (currentPattern[currentRow][c] == 1);
            }

            if (ledOn) {
                digitalWrite(columns[c], LOW); // LED ON
            } else {
                digitalWrite(columns[c], HIGH); // LED OFF
            }
        }

        // Turn on current row
        digitalWrite(rows[currentRow], HIGH);

        // Move to next row
    }
}

```

```

    currentRow = (currentRow + 1) % 6;

    lastMultiplexTime = currentTime;
}
}

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

void handleDisplay() {
    if (!server.hasArg("char")) {
        server.send(400, "application/json", "{\"error\":\"Missing character\"}");
        return;
    }

    String character = server.arg("char");
    character.toUpperCase();
    // Update the LED matrix pattern immediately
    setCharacterPattern(character);
    displayCount++;
    server.send(200, "application/json", "{\"success\":true, \"count\":\"" +
String(displayCount) + ", \"character\":\"" + character + "\"}");
}

// Add a new endpoint to toggle between interactive and display modes
void handleMode() {
    if (server.hasArg("mode")) {
        String mode = server.arg("mode");
        if (mode == "interactive") {
            interactiveMode = true;
            Serial.println("Switched to interactive mode - touch to activate LEDs");
        } else if (mode == "display") {
            interactiveMode = false;
            Serial.println("Switched to display mode - showing full pattern");
        }
        server.send(200, "application/json", "{\"success\":true, \"mode\":\"" + mode +
"\"}");
    } else {
        server.send(400, "application/json", "{\"error\":\"Missing mode parameter\"}");
    }
}
}

```

```

// Add endpoint to reset current pattern
void handleReset() {
    // Clear activated pattern
    for (int r = 0; r < 6; r++) {
        for (int c = 0; c < 6; c++) {
            activatedPattern[r][c] = 0;
        }
    }
    Serial.println("Pattern reset - all LEDs deactivated");
    server.send(200, "application/json", "{\"success\":true, \"message\":\"Pattern reset\"}");
}

void setup() {
    Serial.begin(115200);
    while (!Serial) delay(10);
    // Initialize I2C for MPR121 sensors
    Wire.begin(SDA_PIN, SCL_PIN);
    // Initialize MPR121 sensors
    Serial.println("Initializing MPR121 touch sensors...");
    if (!cap1.begin(0x5A)) {
        Serial.println("✗ MPR121 not found at 0x5A, check wiring!");
    } else {
        Serial.println("✓ MPR121 found at 0x5A (Sensor A)");
    }

    if (!cap2.begin(0x5B)) {
        Serial.println("✗ MPR121 not found at 0x5B, check wiring!");
    } else {
        Serial.println("✓ MPR121 found at 0x5B (Sensor B)");
    }

    if (!cap3.begin(0x5C)) {
        Serial.println("✗ MPR121 not found at 0x5C, check wiring!");
    } else {
        Serial.println("✓ MPR121 found at 0x5C (Sensor C)");
    }

    // Setup LED matrix pins
    for (int i = 0; i < 6; i++) {
        pinMode(columns[i], OUTPUT);
        pinMode(rows[i], OUTPUT);
    }
}

```

```

    digitalWrite(columns[i], HIGH); // Columns off
    digitalWrite(rows[i], LOW);      // Rows off
}

// Initialize activated pattern
for (int r = 0; r < 6; r++) {
    for (int c = 0; c < 6; c++) {
        activatedPattern[r][c] = 0;
    }
}

// Initialize with letter 'A'
setCharacterPattern("A");

// Connect to WiFi
WiFi.begin(ssid, password);
Serial.print("Connecting to WiFi");
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
    updateLEDMatrix(); // Keep LED matrix running during WiFi connection
    processTouchInput(); // Process touch input during WiFi connection
}
Serial.println();
Serial.println("WiFi connected!");
Serial.println("IP Address: " + WiFi.localIP().toString());
Serial.println("LED Matrix ready and displaying 'A'");
Serial.println("Touch sensors active - touch any position on the matrix!");

// Setup web server routes
server.on("/", handleRoot);
server.on("/display", handleDisplay);
server.on("/mode", handleMode);
server.on("/reset", handleReset);
server.begin();

    Serial.println("Web server started");
    Serial.println("Open http://" + WiFi.localIP().toString() + " in your browser");
    // Wait for sensor calibration
    delay(2000);
    Serial.println("Touch sensors calibrated and ready!");
}

void loop() {
    server.handleClient();
    updateLEDMatrix(); // Continuously update LED matrix display
}

```

```
processTouchInput(); // Process touch sensor input  
}
```