**Fundação valeparaibana de ensino**

**Colégio técnico industrial - CTI**

**COLÉGIO TÉNICO ANTÔNIO TEIXEIRA FERNANDES**colégios univap – unidade centro

Curso técnico em eLETRôNICA

nome do autor

LISTA DE EXERCÍCIOS 4º bimestre

MICROCONTROLADORES

Lista apresentada ao Curso Técnico de eletrônica como composição de nota.

Prof. Me. Hélio Esperidião

SÃO JOSÉ DOS CAMPOS

2024

**Microcontroladores  
LISTA DE EXERCÍCIOS II BIMESTRE**

Tenha como base o código abaixo para responder as questões:

|  |
| --- |
| #include <WiFi.h>  #include <WebServer.h>  // Substitua pelas suas credenciais de rede  const char \*ssid = "lula\_livre";  const char \*senha = "batataBrasil1945a";  // Define o estado do LED  int estado\_led = 0;  // Define o pino do LED  const int pinoLed = 2;  void ligar\_led();  void exibir\_menu\_html();  void desligar\_led();  void obter\_estado\_led(); // Rota para enviar o estado atual do LED  // Cria o objeto do servidor web  WebServer servidor(80);  void setup()  {  Serial.begin(115200);  // Configura o pino do LED como saÃ­da  pinMode(pinoLed, OUTPUT);  digitalWrite(pinoLed, LOW); // LED desligado inicialmente  // Conecta ao Wi-Fi  WiFi.begin(ssid, senha);  Serial.print("Conectando ao WiFi..");  // Aguarda a conexÃ£o  while (WiFi.status() != WL\_CONNECTED)  {  delay(1000);  Serial.print(".");  }  Serial.println();  Serial.print("Conectado! EndereÃ§o IP: ");  Serial.println(WiFi.localIP());  // Define as rotas  servidor.on("/", exibir\_menu\_html); // PÃ¡gina principal  servidor.on("/led/1", ligar\_led); // Ligar o LED  servidor.on("/led/0", desligar\_led); // Desligar o LED  servidor.on("/estado\_led", obter\_estado\_led); // Rota para obter o estado do LED  // Inicia o servidor  servidor.begin();  Serial.println("Servidor iniciado");  }  void loop()  {  // Trata as requisiÃ§Ãµes dos clientes  servidor.handleClient();  digitalWrite(pinoLed, estado\_led); // Liga/desliga o LED  }  // FunÃ§Ã£o para exibir a pÃ¡gina principal com AJAX para atualizaÃ§Ã£o do estado do LED  void exibir\_menu\_html()  {  String html = "<html><body><h1>Servidor Web ESP32</h1>";  html += "<p>Clique nos botoes para controlar o LED.</p>";  html += "<p>Estado atual do LED: <span id=\"estado\">Carregando...</span></p>";  html += "<p><a href=\"/led/1\"><button>Ligar LED</button></a></p>";  html += "<p><a href=\"/led/0\"><button>Desligar LED</button></a></p>";  html += "<script>";  html += "function atualizarEstado() {";  html += " var xhttp = new XMLHttpRequest();";  html += " xhttp.onreadystatechange = function() {";  html += " if (this.readyState == 4 && this.status == 200) {";  html += " document.getElementById('estado').innerHTML = this.responseText;";  html += " }";  html += " };";  html += " xhttp.open('GET', '/estado\_led', true);";  html += " xhttp.send();";  html += "}";  html += "setInterval(atualizarEstado, 5000);"; // Atualiza a cada 5 segundos  html += "</script>";  html += "</body></html>";  servidor.send(200, "text/html", html);  }  // FunÃ§Ã£o para ligar o LED  void ligar\_led()  {  estado\_led = 1;  Serial.println("LED ligado");  // Redireciona automaticamente de volta para a pÃ¡gina inicial  servidor.sendHeader("Location", "/", true);  servidor.send(303);  }  // FunÃ§Ã£o para desligar o LED  void desligar\_led()  {  estado\_led = 0;  Serial.println("LED desligado");  // Redireciona automaticamente de volta para a pÃ¡gina inicial  servidor.sendHeader("Location", "/", true);  servidor.send(303);  }  // FunÃ§Ã£o para enviar o estado atual do LED  void obter\_estado\_led()  {  servidor.send(200, "text/plain", (estado\_led == 1) ? "Ligado" : "Desligado");  } |

1. **Explicação das Bibliotecas**  
   Explique a função de cada biblioteca importada (WiFi.h e WebServer.h). Por que elas são necessárias para o funcionamento do código?
2. **Conexão Wi-Fi**  
   O código utiliza WiFi.begin(ssid, senha); para conectar-se à rede Wi-Fi. Modifique o código para que ele informe ao usuário quando a conexão falhar após um certo tempo (ex.: 10 segundos) e reinicie automaticamente o ESP32.
3. **Personalização do HTML**  
   Altere o HTML exibido na página principal para incluir um título mais descritivo, como "Controle do LED via ESP32". Adicione também um botão para atualizar manualmente o estado do LED sem precisar aguardar o intervalo de 5 segundos do setInterval.
4. **Mudança de Porta do Servidor**  
   O código cria um servidor na porta 80 (WebServer servidor(80);). Modifique o código para que o servidor opere em uma porta diferente (ex.: 8080) e explique como isso impactaria o acesso ao servidor.
5. **Ajuste da Frequência de Atualização do Estado do LED**  
   O setInterval(atualizarEstado, 5000); define um intervalo de atualização de 5 segundos para o estado do LED. Modifique o código para que esse intervalo seja reduzido para 1 segundo e explique como isso pode afetar o desempenho do servidor.
6. **Inclusão de Controle de Segurança**  
   O código atual permite que qualquer dispositivo na mesma rede controle o LED. Adicione uma autenticação simples, onde é necessário passar um token (ex.: token=12345) na URL para acessar as rotas de controle do LED. Por exemplo, /led/1?token=12345 deve ligar o LED apenas se o token estiver correto.
7. **Função para Alternar Estado do LED**  
   Crie uma nova rota /led/toggle que alterne o estado atual do LED (de ligado para desligado ou vice-versa). Implemente a função alternar\_led() que faz essa alternância e registre-a no servidor.
8. **Otimização para Armazenar Estado do LED**  
   Atualmente, o estado do LED é mantido em uma variável global estado\_led. Modifique o código para salvar o estado em uma preferência persistente (usando a biblioteca Preferences.h), para que o LED retorne ao seu último estado após uma reinicialização.
9. **Controle de Vários LEDs**  
   Expanda o código para controlar dois LEDs, usando os pinos 2 e 4, respectivamente. Adicione novos endpoints (/led1/1, /led1/0, /led2/1, /led2/0) para controlar cada LED separadamente e ajuste o HTML para incluir botões de controle para cada LED.