Exercise 16 - 17

Jonas Pedersen, Søren Graae

January 13, 2025

Exercise 16

16a

What is a server?

A server is a computer or system that provides resources, data, services, or programs to other computers, known as clients, over a network. It can host websites, manage databases, store files, or run applications, making it central to many online and offline systems. Servers are designed to handle multiple client requests simultaneously and often operate continuously to ensure reliable access.

16b

What is a GET request? And how is it different to a POST request?

A GET request retrieves data from a server by appending parameters to the URL, typically used for fetching resources and is idempotent (doesn't alter server state). A POST request, on the other hand, sends data in the request body to the server, often used for actions like form submissions or uploading files. POST is more secure for sensitive data and can handle larger payloads, while GET is limited by URL length and is cached or logged by browsers.

16c

Follow the guidelines given in the document "Programming with Arduino - Webserver 2018"

```
/**
 * @file main.ino
 * @brief Demonstration of an ESP32 acting as a web server to toggle an LED.
 *
 * This sketch sets up a WiFi connection (using WiFiMulti to handle multiple
 * networks), starts an HTTP server, and serves a simple webpage. The user can
 * toggle an LED via a POST request.
 */

#include <WiFi.h>
#include <WiFiMulti.h> // ESP32 version of WiFiMulti
#include <WebServer.h> // ESP32 version of the WebServer library
#include <ESPmDNS.h>   // mDNS library for ESP32

/**
 * @brief WiFiMulti instance to manage multiple networks.
 */
WiFiMulti wifiMulti;

/**
 * @brief Web server instance listening on port 80.
 */
WebServer server(80);

/**
 * @brief GPIO pin used to drive the LED.
 *
 * Change to the GPIO pin connected to your LED (e.g., 2 for built-in LED on many ESP32 boards).
 */
const int led = 4;

/**
 * @brief Handles the root ("/") HTTP GET request.
 *
 * Sends an HTML page with information about IoT and a button to toggle the LED.
 */
void handleRoot();

/**
 * @brief Handles the "/LED" HTTP POST request.
 *
 * Toggles the state of the LED and redirects back to the root page.
 */
void handleLED();

/**
 * @brief Handles requests for non-existent pages.
 *
 * Sends a 404 Not Found response.
 */
void handleNotFound();

/**
 * @brief Arduino setup function.
 *
 * Initializes serial communication, configures the LED pin,
 * connects to WiFi using WiFiMulti, starts the mDNS responder,
 * sets up HTTP routes, and starts the web server.
 */
void setup() {
  Serial.begin(115200);
  delay(10);

  pinMode(led, OUTPUT);
  digitalWrite(led, HIGH); // Ensure LED is off initially

  // Connect to WiFi networks
  Serial.println();
  wifiMulti.addAP("GN-TOP-SECRET", "TOP-SECRET"); // Add Wi-Fi networks you want to connect to

  Serial.println();
  Serial.print("Connecting ...");

  // Attempt to connect to one of the WiFi networks
  while (wifiMulti.run() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected to:");
  Serial.println(WiFi.SSID());
  Serial.println("IP address:");
  Serial.println(WiFi.localIP());

  // Start mDNS responder
  if (MDNS.begin("iot")) {
    Serial.println("mDNS responder started");
  } else {
    Serial.println("Error setting up MDNS responder!");
  }

  // Set up HTTP routes
  server.on("/", HTTP_GET, handleRoot);
  server.on("/LED", HTTP_POST, handleLED);
  server.onNotFound(handleNotFound);

  // Start the server
  server.begin();
  Serial.println("Server started");
}

/**
 * @brief Arduino main loop.
 *
 * Continuously handles client requests coming in on the web server.
 */
void loop() {
  server.handleClient();
}

void handleRoot() {
  server.send(200, "text/html",
    "<html><title>Internet of Things - Demonstration</title><meta charset=\"utf-8\"/> \
     <body><h1>Velkommen til denne WebServer</h1> \
    <p>Internet of Things (IoT) er \"tingenes Internet\" - dagligdags ting kommer p  nettet og f r ny v rdi. Det kan l se mange udfordringer.</p> \
    <p>Her kommunikerer du med en webserver p  en lille microcontroller af typen ESP32, som i dette tilf lde styrer en digital udgang, som du s  igen kan bruge til at styre en lampe, en ventilator, t nde for varmen eller hvad du lyster.</p> \
    <p>Klik p  nedenst ende knap for at t nde eller slukke LED p  port GPIO2</p> \
    <form action=\"/LED\" method=\"POST\"><input type=\"submit\" value=\"Skift tilstand p  LED\" style=\"width:500px; height:100px; font-size:24px\"></form> \
    <p>Med en ESP32 kan du lave sjove projekter</p> \
    <p>Vil du vide mere: Kig p  hjemmesiden for uddannelsen : <a href=\"www.dtu.dk/net\">Netv rksteknologi og it</a></p> \
    </body></html>"
  );
}

void handleLED() {
  // Toggle the LED
  digitalWrite(led, !digitalRead(led));

  // Redirect to the home page
  server.sendHeader("Location", "/");
  server.send(303); // HTTP 303 See Other
}

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

The code has been modified to work on an ESP32 Wroom.

Exercise 17

```
/**
```

```
 * @file Exercise_17.ino
 * @brief Demonstration of connecting to WiFi and sending data to ThingSpeak.
 *
 * This sketch connects the ESP32 to the specified WiFi, then periodically sends
 * two fields of data (RSSI and a digital pin reading) to a ThingSpeak channel.
 * It also demonstrates reading back from the channel.
 */

#include <WiFi.h>
#include <ThingSpeak.h>

/**
 * @brief WiFi credentials.
 */
const char* ssid = "GN-TOP-SECRET";  ///< SSID of the WiFi network.
const char* pass = "TOP-SECRET"; ///< Password of the WiFi network.

/**
 * @brief WiFi client object for ThingSpeak.
 */
WiFiClient client;

/**
 * @brief ThingSpeak API information.
 */
const char* APIKey = "TOP-SECRET";    ///< Your ThingSpeak API key.
const char* server = "api.thingspeak.com";  ///< ThingSpeak server.
unsigned long channelID = 0123456;          ///< Your ThingSpeak channel ID.

/**
 * @brief Variables to store measured data.
 */
float data1; ///< Measured data field 1.
float data2; ///< Measured data field 2.

```

```
/**
 * @brief Post delay in milliseconds (send data every 5 seconds).
 */
#define postDelay 5 * 1000

/**
 * @brief Arduino setup function. Connects to WiFi and starts serial communication.
 */
void setup() {
  pinMode(4, INPUT_PULLUP);

  Serial.begin(115200);
  Serial.print("Connecting to WiFi");
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(’.’);
    delay(1000);
  }

  Serial.println("");
  Serial.println(WiFi.localIP());
}

/**
 * @brief Arduino main loop. Reads data, checks WiFi connection, sends data to ThingSpeak, and reads back.
 */
void loop() {
  // Gather your data (example: RSSI and digital pin read)
  data1 = WiFi.RSSI();
  data2 = digitalRead(4);

  // Ensure WiFi is connected
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(’.’);
    delay(1000);
  }

  // Initialize ThingSpeak
  ThingSpeak.begin(client);

  // Connect to ThingSpeak server
  client.connect(server, 80);

  // Set fields and write them
  ThingSpeak.setField(1, data1);
  ThingSpeak.setField(2, data2);
  Serial.println("Writing to ThingSpeak");
  ThingSpeak.writeFields(channelID, APIKey);

  // Read data back from ThingSpeak
  Serial.println("Reading from ThingSpeak");
  data1 = ThingSpeak.readIntField(channelID, 1, APIKey);
  Serial.print("Read RSSI: ");
  Serial.println(data1);

  client.stop();

  // Delay before the next post
  delay(postDelay);
}
```

The code has been modified to fit an ESP32 Wroom.