## Лабораторная работа 10: Работа с HTTP и MQTT на ESP32

**Цель:**

* Изучить основы работы с HTTP-сервером на ESP32.
* Научиться создавать веб-интерфейс для управления устройством через браузер.
* Изучить протокол MQTT.
* Настроить MQTT-брокера и клиента на ESP32.
* Реализовать отправку и прием сообщений по MQTT для обмена данными с другими устройствами.

## Теоретическая часть

### HTTP сервер на ESP32

HTTP (Hypertext Transfer Protocol) - это базовый протокол передачи данных в сети, используемый для отображения веб-страниц. ESP32 имеет встроенный веб-сервер, который позволяет создавать простые веб-приложения.

### Работа с протоколом MQTT

MQTT (Message Queuing Telemetry Transport) - это протокол обмена сообщениями, предназначенный для использования в системах Интернета вещей (IoT). MQTT позволяет устройствам подключаться к брокеру MQTT и обмениваться сообщениями между собой.

#### HTTP сервер на ESP32

**Схема подключения:**

* ESP32 подключен к сети Wi-Fi.

**Код:**

#include <WiFi.h>  
#include <WiFiServer.h>  
#include <WebServer.h>  
  
const char\* ssid = "YOUR\_SSID";  
const char\* password = "YOUR\_PASSWORD";  
  
const int ledPin = 2; // GPIO pin for the LED  
  
WiFiServer server(80);  
WebServer webServer;  
  
void setup() {  
 Serial.begin(115200);  
  
 // Connect to Wi-Fi network  
 WiFi.begin(ssid, password);  
 while (WiFi.status() != WL\_CONNECTED) {  
 delay(500);  
 Serial.print("."); } Serial.println(""); Serial.println("WiFi connected.");  
  
 // Set up LED as output  
 pinMode(ledPin, OUTPUT);  
  
 // Start HTTP server  
 server.begin();  
 webServer.on("/", handleRoot);  
 webServer.on("/on", handleOn);  
 webServer.on("/off", handleOff);  
 webServer.begin();  
 Serial.println("HTTP server started.");  
}  
  
void loop() {  
 webServer.handleClient();  
}  
  
void handleRoot() {  
 String message = "<!DOCTYPE html><html><head><title>ESP32 LED Control</title></head><body>";  
 message += "<h1>ESP32 LED Control</h1>";  
 message += "<p>Current LED state: ";  
 if (digitalRead(ledPin) == HIGH) {  
 message += "ON";  
 } else {  
 message += "OFF";  
 }  
 message += "</p>";  
 message += "<a href=\"/on\">Turn ON</a>";  
 message += "<a href=\"/off\">Turn OFF</a>";  
 message += "</body></html>";  
 webServer.send(200, "text/html", message);  
}  
  
void handleOn() {  
 digitalWrite(ledPin, HIGH);  
 webServer.send(200, "text/html", "LED turned ON");  
}  
  
void handleOff() {  
 digitalWrite(ledPin, LOW);  
 webServer.send(200, "text/html", "LED turned OFF");  
}

**Описание:**

* В коде определены имя и пароль сети Wi-Fi, а также номер GPIO-пина для светодиода.
* Функция setup() подключается к сети Wi-Fi, настраивает LED как вывод и запускает HTTP-сервер.
* Функция loop() обрабатывает запросы к серверу.
* Функция handleRoot() генерирует HTML-страницу с текущим состоянием светодиода и кнопками для его включения/выключения.
* Функции handleOn() и handleOff() соответственно включают и выключают светодиод, а также отправляют HTML-ответы, подтверждающие изменения.

**Результат:**

* При подключении к ESP32 через Wi-Fi в браузере откроется веб-страница с текущим состоянием светодиода и кнопками для его управления.

### Работа с протоколом MQTT

**Схема подключения:**

* ESP32 подключен к сети Wi-Fi.
* ESP32 подключен к MQTT-брокеру.

**Код:**

#include <WiFi.h>  
#include <PubSubClient.h>  
  
const char\* ssid = "YOUR\_SSID";  
const char\* password = "YOUR\_PASSWORD";  
  
const char\* mqtt\_server = "mqtt.example.com"; // MQTT broker address  
const int mqtt\_port = 1883;  
const char\* mqtt\_user = "esp32";  
const char\* mqtt\_password = "esp32password";  
  
const char\* topic\_pub = "esp32/led"; // Topic for publishing LED state  
const char\* topic\_sub = "esp32/command"; // Topic for subscribing to commands  
  
WiFiClient espClient;  
PubSubClient client(espClient);  
  
int ledPin = 2; // GPIO pin for the LED  
  
void setup() {  
 Serial.begin(115200);  
  
 // Connect to Wi-Fi network  
 WiFi.begin(ssid, password);  
 while (WiFi.status() != WL\_CONNECTED) {  
 delay(500);  
 Serial.print(".");  
 }  
 Serial.println("");  
 Serial.println("WiFi connected.");  
  
 // Set up LED as output  
 pinMode(ledPin, OUTPUT);  
  
 // Connect to MQTT broker  
 client.setServer(mqtt\_server, mqtt\_port);  
 client.setCallback(callback);  
  
 while (!client.connected()) {  
 if (client.connect("ESP32Client", mqtt\_user, mqtt\_password)) {  
 Serial.println("Connected to MQTT broker.");  
 client.subscribe(topic\_sub);  
 } else {  
 Serial.print("Failed to connect to MQTT broker. Retrying in 5 seconds...");  
 delay(5000);  
 }  
 }  
}  
  
void loop() {  
 // Check for incoming MQTT messages  
 client.loop();  
}  
  
void callback(char\* topic, byte\* payload, unsigned int length) {  
 Serial.print("Message arrived on topic: ");  
 Serial.print(topic);  
 Serial.print(" Message: ");  
 for (int i = 0; i < length; i++) {  
 Serial.print((char)payload[i]);  
 }  
 Serial.println();  
  
 // Handle incoming commands  
 String command = String((char\*)payload, length);  
 if (command.equals("ON")) {  
 digitalWrite(ledPin, HIGH);  
 } else if (command.equals("OFF")) {  
 digitalWrite(ledPin, LOW);  
 }  
}  
  
void publishLedState() {  
 // Publish LED state to MQTT topic  
 if (client.connected()) {  
 String message = digitalRead(ledPin) == HIGH ? "ON" : "OFF";  
 client.publish(topic\_pub, message.c\_str());  
 Serial.println("Published LED state: " + message);  
 }  
}

**Описание:**

* В коде добавлены константы для адреса MQTT-брокера, порта, имени пользователя и пароля.
* Определены темы для публикации состояния светодиода (topic\_pub) и подписки на команды (topic\_sub).
* Функция setup() подключается к сети Wi-Fi, MQTT-брокеру и подписывается на тему topic\_sub.
* Функция loop() проверяет наличие входящих MQTT-сообщений.
* Функция callback() обрабатывает входящие сообщения, проверяет команды и управляет светодиодом.
* Функция publishLedState() публикует текущее состояние светодиода в теме topic\_pub.

**Результат:**

* При подключении к ESP32 через Wi-Fi и MQTT-брокеру:
  + Состояние светодиода будет публиковаться в теме topic\_pub.
  + Вы можете отправить сообщения “ON” или “OFF” в тему topic\_sub на MQTT-брокере, чтобы управлять светодиодом.

## Задачи:

**Задачи:**

**1. Управление умным домом через веб-интерфейс и MQTT:**

* Создайте веб-интерфейс на ESP32, который будет отображать текущее состояние нескольких устройств (например, светодиоды, датчики температуры, влажности) и позволять управлять ими.
* Реализуйте обмен данными между веб-интерфейсом и устройствами по MQTT.
* Добавьте возможность отправлять команды управления устройствами с MQTT-брокера.

**2. Система мониторинга параметров окружающей среды:**

* Подключите к ESP32 датчики температуры, влажности, освещенности.
* Считывайте данные с датчиков и публикуйте их в теме MQTT.
* Создайте приложение (например, на Android), которое будет подключаться к MQTT-брокеру и отображать данные с датчиков в виде графика или таблицы.
* Добавьте возможность отправлять команды управления устройствами (например, включение/выключение вентилятора) с приложения на MQTT-брокер.

## Примеры кода для лабораторной работы:

## Решения для задач:

**1. Управление умным домом через веб-интерфейс и MQTT:**

**Решение:**

**Система на ESP32:**

* Используйте библиотеку AsyncWebServer для создания веб-интерфейса.
* Создайте HTML-страницы с элементами управления для каждого устройства (кнопки, ползунки, выпадающие списки).
* Обрабатывайте запросы от веб-интерфейса, обновляйте состояние устройств и публикуйте изменения в теме MQTT.
* Подпишитесь на тему MQTT для получения команд управления от веб-интерфейса или MQTT-брокера.
* Обрабатывайте полученные команды и управляйте соответствующими устройствами.

**Приложение на MQTT-брокере:**

* Подпишитесь на тему MQTT, чтобы получать данные о состоянии устройств.
* Отображайте данные на веб-интерфейсе брокера (например, в виде таблицы или графика).
* Предоставьте элементы управления для отправки команд управления устройствами на ESP32.

**Пример кода на ESP32:**

#include <AsyncWebServer.h>  
#include <PubSubAsync.h>  
  
// ... (остальной код из предыдущего примера)  
  
void setup() {  
 // ... (остальной код setup() из предыдущего примера)  
  
 // Create AsyncWebServer object on port 80  
 AsyncWebServer server(80);  
  
 // Route to handle root page  
 server.on("/", HTTP\_GET, [](AsyncHttpServerRequest \*request) {  
 // ... (HTML-код для отображения состояния устройств)  
 });  
  
 // Route to handle device control requests  
 server.on("/control/:device/:action", HTTP\_POST, [](AsyncHttpServerRequest \*request) {  
 // ... (обработка запросов на управление устройствами)  
 });  
  
 // Start AsyncWebServer  
 server.begin();  
 Serial.println("HTTP server started.");  
}  
  
void loop() {  
 // ... (остальной код loop() из предыдущего примера)  
  
 // Check for incoming MQTT messages  
 client.loop();  
}  
  
void callback(char\* topic, byte\* payload, unsigned int length) {  
 // ... (обработка входящих команд)  
}  
  
void publishLedState() {  
 // ... (публикация состояния светодиода)  
}

**2. Система мониторинга параметров окружающей среды:**

**Решение:**

**Система на ESP32:**

* Подключите датчики температуры, влажности и освещенности к ESP32.
* Считывайте данные с датчиков с помощью соответствующих библиотек (например, Adafruit\_BME280).
* Формируйте JSON-строку с данными датчиков.
* Публикуйте JSON-строку в теме MQTT.

**Приложение:**

* Подключитесь к MQTT-брокеру.
* Подпишитесь на тему MQTT, по которой публикуются данные с датчиков.
* Разберите JSON-строку и извлеките данные о температуре, влажности и освещенности.
* Отобразите данные на экране в виде графика или таблицы.
* Предоставьте элементы управления для отправки команд управления устройствами на ESP32 (например, включение/выключение вентилятора).

**Пример кода на ESP32:**

#include <Adafruit\_BME280.h>  
#include <PubSubAsync.h>  
  
// ... (остальной код из предыдущего примера)  
  
void setup() {  
 // ... (остальной код setup() из предыдущего примера)  
  
 // Initialize BME280 sensor  
 if (bme280.begin(0x76)) {  
 Serial.println("Sensor connected.");  
 } else {  
 Serial.println("Sensor not found.");  
 while (1);  
 }  
  
 // Set callback functions for sensor data updates  
 bme280.setTemperatureCallback([&](float temperature) {  
 updateSensorData();  
 });  
  
 bme280.setHumidityCallback([&](float humidity) {  
 updateSensorData();  
 });  
  
 bme280.setPressureCallback([&](float pressure) {  
 updateSensorData();  
 });  
}  
  
void loop() {  
 // ... (остальной код loop() из предыдущего примера)  
  
 // Check for incoming MQTT messages  
 client.loop();  
}  
  
void updateSensorData() {  
 // Read sensor data  
 float temperature = bme280.readTemperatureC();  
 float humidity = bme280.readHumidity();  
 float pressure = bme280.readPressurePa();  
  
 // Format JSON string with sensor data  
 String jsonData = "{\"temperature\": " + String(temperature) + ", \"humidity\": " + String(humidity) + ", \"pressure\": " + String(pressure) + "}";  
  
 // Publish JSON string to MQTT topic  
 if (client.connected()) {  
 client.publish(topic\_pub, jsonData.c\_str());  
 Serial.println("Published sensor data: " + jsonData);  
 }  
}

**Пример кода приложения:**

import paho.mqtt.client as mqtt  
import matplotlib.pyplot as plt  
  
# Define MQTT broker address and topic  
broker\_address = "mqtt.example.com"  
topic = "esp32/sensor\_data"  
  
# Define callback function for incoming MQTT messages  
def on\_message(client, userdata, msg):  
 # Parse JSON data from MQTT message  
 data = json.loads(msg.payload.decode())  
  
 # Extract sensor data  
 temperature = data["temperature"]  
 humidity = data["humidity"]  
 pressure = data["pressure"]  
  
 # Print sensor data to console  
 print("Temperature:", temperature, "°C")  
 print("Humidity:", humidity, "%")  
 print("Pressure:", pressure, "Pa")  
  
 # Update plot with new data  
 x.append(time.time())  
 y\_temp.append(temperature)  
 y\_hum.append(humidity)  
 y\_pres.append(pressure)  
  
 # Update plot axes  
 plt.xlim([x[0], x[-1]])  
 plt.ylim([min(y\_temp + y\_hum + y\_pres), max(y\_temp + y\_hum + y\_pres)])  
  
 # Redraw the plot  
 plt.pause(0.01)  
  
# Create MQTT client object  
client = mqtt.Client()  
  
# Connect to MQTT broker  
client.connect(broker\_address)  
  
# Subscribe to MQTT topic  
client.subscribe(topic)  
  
# Set callback function for incoming messages  
client.on\_message = on\_message  
  
# Initialize plot  
x = []  
y\_temp = []  
y\_hum = []  
y\_pres = []  
  
plt.ion()  
plt.figure(figsize=(12, 6))  
  
plt.subplot(3, 1, 1)  
plt.plot(x, y\_temp, label='Temperature')  
plt.ylabel('Temperature (°C)')  
plt.grid(True)  
  
plt.subplot(3, 1, 2)  
plt.plot(x, y\_hum, label='Humidity (%)')  
plt.ylabel('Humidity (%)')  
plt.grid(True)  
  
plt.subplot(3, 1, 3)  
plt.plot(x, y\_pres, label='Pressure (Pa)')  
plt.xlabel('Time (s)')  
plt.ylabel('Pressure (Pa)')  
plt.grid(True)  
  
# Start infinite loop for receiving and processing MQTT messages  
client.loop\_forever()