Міністерство освіти і науки України Харківський національний університет радіоелектроніки

Кафедра програмної інженерії

Звіт

З лабораторної роботи №4

З дисципліни «Аналіз та рефакторинг коду» на тему: «РОЗРОБКА IoT КЛІЄНТА»

|  |  |
| --- | --- |
| Виконав  ст. гр. ПЗПІ-22-4  Попов Б.С. | Перевірив ст.  викладач каф. ПІ Сокорчук І. П. |

Харків 2024

# 1 МЕТА РОБОТИ

Розробити клієнт IoT, який здійснює збір, аналіз і передачу даних на сервер відповідно до принципів Інтернету речей. Реалізувати бізнес-логіку, функціонал налаштування пристрою та забезпечити інтеграцію з іншими компонентами системи.

# 2 ХІД РОБОТИ

У ході виконання лабораторної роботи була реалізована IoT-система для моніторингу стану транспортних засобів на базі мікроконтролера ESP32. Система включає кілька ключових інженерних рішень, що забезпечують ефективність та стабільну роботу пристрою:

* Розроблено програмний код для збору та обробки даних з різних сенсорів, зокрема для вимірювання температури за допомогою датчика DS18B20, та інших параметрів, таких як GPS-координати та тиск у шинах.
* Використано Wi-Fi модуль ESP32 для підключення до мережі та передавання даних на MQTT брокер для віддаленого моніторингу стану транспортного засобу.
* Реалізовано механізм збору даних з датчика температури, GPS-модуля (для визначення широти та довготи) і датчика тиску в шинах, що дозволяє здійснювати точне вимірювання та передачу даних.
* Розроблено систему публікації зібраних даних у форматі JSON через MQTT на сервер для моніторингу стану транспортного засобу в реальному часі.
* Забезпечено тестування пристрою через серійний монітор для перевірки коректності з'єднання з Wi-Fi мережею та MQTT сервером, а також для виведення діагностичної інформації.

2.1 Текстовий опис інженерних рішень

Програмний код IoT-пристрою розроблений за допомогою Arduino IDE з використанням таких бібліотек:

* WiFi.h – для з'єднання з Wi-Fi мережею.
* DallasTemperature.h та OneWire.h – для роботи з датчиком температури DS18B20.
* PubSubClient.h – для передачі даних через MQTT.
* TinyGPS++ (або інша бібліотека для роботи з GPS) – для отримання координат з GPS-модуля.
* MQTT.h – для роботи з MQTT сервером, через який дані передаються до віддаленого клієнта для моніторингу.

Основні етапи виконання роботи:

* Підключення до Wi-Fi: Автоматичне з'єднання з заданою мережею для передачі даних.
* Зчитування та обробка даних сенсорів: Збір даних про температуру, GPS-координати та тиск у шинах.
* Передача даних на MQTT сервер: Збір та відправка зібраних даних у форматі JSON для віддаленого моніторингу.
* Логування: Виведення діагностичної інформації для перевірки працездатності системи через серійний монітор.

Таким чином, система дозволяє ефективно моніторити та передавати стан транспортного засобу для віддаленого контролю.

Изображение выглядит как электроника, текст, Электронная техника, Электронный компонент

Автоматически созданное описание

Рисунок 2.1 – Схема IoT пристрою

* 1. Фрагменти програмного коду
     1. Бізнес-логіка IoT-клієнта

Фрагмент коду зчитування даних сенсорів: Код наведено у [додатку А](#_bookmark0).

* + 1. Налаштування IoT-клієнта

Фрагмент коду підключення до Wi-Fi:

1 void setup() {

2   **Serial**.begin(115200);

3   WiFi.begin(ssid, password);

4

5   while (WiFi.status() != WL\_CONNECTED) {

6     **Serial**.println("Connecting to WiFi...");

7   }

8   **Serial**.println("Connected to WiFi");

9   client.setServer(mqtt\_server, mqtt\_port);

10   sensors.begin();

11 }

* 1. UML Діаграми
     1. Діаграма прецедентів IoT клієнта

Изображение выглядит как зарисовка, рисунок, диаграмма, линия

Автоматически созданное описание

Рисунок 2.3.1 – UML діаграма прецедентів IoT клієнта

Діаграма прецедентів описує взаємодію IoT-клієнта з користувачем, Wi-Fi, MQTT-брокером, датчиком температури та акселерометром. Основні прецеденти включають підключення до Wi-Fi, збір даних із сенсорів та відправлення даних на MQTT-брокер.

* + 1. Діаграма діяльності IoT клієнта

Діаграма діяльності ілюструє повний процес роботи IoT-клієнта для моніторингу стану транспортного засобу. Спочатку система ініціалізується та встановлює з'єднання з Wi-Fi мережею. При успішному підключенні клієнт активує сенсори для збору даних, таких як температура двигуна, тиск у шинах і геолокація.

Після отримання даних від сенсорів система їх обробляє і готує до передачі. Далі IoT-клієнт намагається відправити ці дані на MQTT-брокер. У разі успішної передачі даних, система відображає повідомлення про успішне відправлення. Якщо передача не вдалася, система повторює спроби надсилання до досягнення успіху.

Изображение выглядит как текст, диаграмма, снимок экрана, чек

Автоматически созданное описание

Рисунок 2.3.2– UML діаграма діяльності IoT клієнта

# 3 ВИСНОВКИ

У процесі виконання лабораторної роботи було успішно розроблено IoT-клієнт для моніторингу стану транспортних засобів, що відповідає сучасним вимогам до смарт-пристроїв та IoT-систем. Клієнт забезпечує ефективний збір, обробку та передачу даних від сенсорів для вимірювання температури двигуна, тиску в шинах і геолокації. Завдяки використанню протоколу MQTT для передачі даних, пристрій може бути інтегрований у більші системи для віддаленого моніторингу стану автомобілів. Це підкреслює практичну значущість розробленого IoT-клієнта для застосування в транспортних та логістичних системах.

Посилання на відео:

# Додаток А

1. #include <WiFi.h>

2. #include <PubSubClient.h>

3. #include <OneWire.h>

4. #include <DallasTemperature.h>

5. const char\* ssid = "Wokwi-GUEST";

6. const char\* password = "";

7. const char\* mqtt\_server = "broker.hivemq.com";

8. const int mqtt\_port = 1883;

9. const char\* mqtt\_topic = "iot/data";

10. const char\* mqtt\_user = "";

11. const char\* mqtt\_password = "";

12. WiFiClient espClient;

13. PubSubClient client(espClient);

14. #define ONE\_WIRE\_BUS 23

15. OneWire oneWire(ONE\_WIRE\_BUS);

16. DallasTemperature sensors(&oneWire);

17. void setup() {

18. Serial.begin(115200);

19. WiFi.begin(ssid, password);

20. while (WiFi.status() != WL\_CONNECTED) {

21. Serial.println("Connecting to WiFi...");

22. }

23. Serial.println("Connected to WiFi");

24. client.setServer(mqtt\_server, mqtt\_port);

25. sensors.begin();

26. }

27. void loop() {

28. if (!client.connected()) {

29. reconnect();

30. }

31. client.loop();

32. // Запрашиваем температуру

33. sensors.requestTemperatures();

34. float temperature = sensors.getTempCByIndex(0);

35. float latitude = 48.8566 + random(-100, 100) / 10000.0;

36. float longitude = 2.3522 + random(-100, 100) / 10000.0;

37. float tirePressure = random(200, 350) / 10.0;

38. publishData(temperature, latitude, longitude, tirePressure);

39. delay(5000);

40. }

41. void publishData(float temperature, float latitude, float longitude, float tirePressure) {

42. String payload = String("{\"temperature\":") + temperature +

43. ",\"latitude\":" + latitude +

44. ",\"longitude\":" + longitude +

45. ",\"tirePressure\":" + tirePressure + "}";

46. if (client.publish(mqtt\_topic, payload.c\_str())) {

47. Serial.println("Data published successfully");

48. } else {

49. Serial.println("Failed to publish data");

50. }

51. }

52. void reconnect() {

53. while (!client.connected()) {

54. Serial.print("Attempting MQTT connection...");

55. if (client.connect("ESP32Client")) {

56. Serial.println("Connected to MQTT broker");

57. } else {

58. Serial.print("Failed, rc=");

59. Serial.print(client.state());

60. }

61. }

}