ПРИКЛАДНІ ПРОБЛЕМИ КОМП'ЮТЕРНИХ НАУК,
БЕЗПЕКИ ТА МАТЕМАТИКИ, 2023, № 2

УДК 004.333

РОЗРОБКА BACKEND FRAMEWORK ДЛЯ МІКРОПЛАТ ESP ІЗ ПІДТРИМКОЮ HTTP (TCP/IP) ПРОТОКОЛУ

Ступінь А.П. (ORCID: 0009-0002-8826-1952),
Булатецький В.В. (ORCID: 0000-0002-9883-4550)
Волинський національний університет імені Лесі Українки

DEVELOPMENT OF A BACKEND FRAMEWORK FOR MICROCONTROLLERS ESP WITH HTTP (TCP/IP) PROTOCOL SUPPORT

Stupin A.P., Bulatetskyi V.V.
Lesya Ukrainka Volyn National University

Abstract. The work is dedicated to exploring the intricacies of developing a backend framework for the ESP microplatform with support for the HTTP (TCP/IP) protocol. The growing popularity of ESP microcontrollers in the field of embedded systems and the Internet of Things underscores the relevance of studying effective methods for supporting network protocols on these platforms. The paper encompasses an analysis of the technical characteristics of ESP microplatforms, the identification of requirements for a backend framework for optimal processing of HTTP requests, and the development of corresponding algorithms and data structures. The primary focus of the work is on achieving high productivity and efficiency in the conditions of limited resources on ESP microcontrollers to provide reliable data exchange via the HTTP protocol in embedded systems.

Key features of this framework include simplifying development through ready-to-use tools for handling HTTP requests, efficient utilization of microcontroller resources, and the ability to interact with various sensors and peripheral devices, making it an effective choice for Internet of Things (IoT) projects. Implementing this framework in the development of internet applications for ESP platforms opens up broad possibilities for creating efficient and resource-saving projects. By providing flexibility in interacting with the HTTP protocol, it enables developers to quickly implement their own webservers and efficiently process HTTP requests. Optimization for resource conservation becomes a key advantage of the framework, especially in the conditions of limited capabilities of ESP microcontrollers. This makes it particularly interesting for developers working on projects where every byte of memory and every processor cycle is crucial. A framework specifically adapted for ESP microcontrollers becomes a key component in this environment as it provides the means for developing embedded programs for data processing and exchange over the network. The research results can be beneficial for developers in the field of embedded systems focused on utilizing

Keywords: backend, framework, HTTP, Internet of Things (IoT), ESP Microcontrollers, microplatphorms, embedded systems.

Вступ. З кожним днем мікроплати ESP стають все більш популярними у веброзробці [1,2]. Мікроплати ESP відкривають нові можливості для створення різноманітних систем. Однак розробка серверної частини, для мікроплат ESP, залишається актуальною проблемою, причиною цього є обмежені ресурси на мікроплаті. Це дослідження спрямоване на вивчення особливостей розробки серверних рішень, оптимізованих для потреб мікроплат ESP, з урахуванням обмежених ресурсів та особливостей їх апаратної архітектури. Основний фокус полягає у пришвидшені розробки систем, що працюють на базі протоколу HTTP за допомогою backend framework.

Постановка проблеми. Мета роботи полягає в розгляді, аналізі та розробці оптимізованого backend framework, спеціально адаптованого для мікроплат ESP.

Backend framework, як продукт, повинен відповідати наступним вимогам для мікроплат ESP8266 або ESP32:

- автоматично встановлювати та налаштовувати TCP/IP з'єднання з сервером;
- забезпечувати можливість визначення HTTP-маршрутів для обробки різних запитів;
- забезпечувати підтримку параметрів маршруту для динамічного отримання даних

лише за шляхом HTTP запиту;
- забезпечувати підтримку основних HTTP-методів, таких як GET, POST, PUT, DELETE та операцій над ними;
- забезпечувати можливість визначення моделей даних та їх взаємозв'язків для роботи з базою даних;
- забезпечувати автоматичне створення та оновлення схеми бази даних відповідно до моделей;
- забезпечувати кешування даних отриманих від бази даних для подальшого швидкого використання;
- мати механізм для обробки помилок, для полегшення відлагодження та забезпечення безпеки;
- мати можливість автоматично серіалізувати та десеріалізувати дані у форматі JSON для обміну інформацією з клієнтами;
- забезпечувати можливість формувати html відповіді;
- здатність легко розширювати функціональність за допомогою додаткових модулів або бібліотек;
- мати можливість використання оновлень програмного забезпечення через бездротові мережі з використанням технології OTA (Over The Air)[13] для зручного вдосконалення програми на ESP8266 або ESP32 без прямого підключення до комп'ютера;
- забезпечувати управління станом системи для можливості вчасно виявляти та розв'язувати проблеми;
- забезпечувати легкість та простоту коду, для можливості легкого входження та допрацювання основних модулів та елементів backend framework.

OTA (Over The Air) – це механізм, який дозволяє оновлювати програмне забезпечення (прошивку) мікроконтролерів Arduino та інших мікроконтролерів із можливістю програмування через Arduino IDE (оновлення відбувається безпосередньо через бездротове з'єднання, таке як Wi-Fi чи Bluetooth, без необхідності підключення через USB-порт або інші фізичні інтерфейси).

Опис проєкту. В роботі розроблено Backend framework для мікроплат ESP із підтримкою HTTP протоколу. Цей backend framework, розроблений спеціально для мікроконтролерів ESP (наприклад, ESP8266 та ESP32), з акцентом на підтримку HTTP протоколу. Він надає вбудовані засоби для створення власних вебсерверів та обробки HTTP-запитів на рівні вбудованого програмного забезпечення. Розроблений backend framework полегшує розробку інтернет-застосунків для ESP-платформ, надаючи готові інструменти для обробки вхідних HTTP-запитів, керування маршрутами та роботу з параметрами URL. Він оптимізований для ресурсозаощадження та ефективного використання обмежених ресурсів мікроконтролерів.

Фреймворк також забезпечує можливість взаємодії з різноманітними сенсорами та периферійними пристроями, що робить його ідеальним вибором для проєктів Інтернету речей (IoT). Розробка включає декілька основних компонент, зокрема вбудований HTTP сервер для обробки HTTP-запитів та відправлення HTTP-відповідей для мікроплат ESP.

Також реалізована система маршрутизації для обробки різних HTTP-запитів, що дозволяє визначати обробники (handlers) для конкретних маршрутів та надає зручний інтерфейс для їх швидкого та легкого створення.

Фреймворк підтримує віддачу статичного вмісту (наприклад, HTML-сторінок, CSS, JavaScript) та генерацію динамічного вмісту на основі запитів. Додатково реалізовано базу даних на основі файлової системи, яка присутня на платах ESP.

Можливість інтеграції з датчиками, актуаторами та іншими пристроями для

забезпечення взаємодії через HTTP-запити. Оскільки написання вебсервера на мікроплатах з підтримкою додаткових датчиків є основним завданням.

Також надані засоби для забезпечення безпеки та аутентифікації, включаючи можливість налаштування доступу до конкретних ресурсів.

Існує можливість розширення фреймворку за допомогою модулів та додаткових функціональностей. Для кожної сутності в backend framework розроблені свої інтерфейси та засоби, які забезпечуватимуть можливість його розширення, як стороннім програмним забезпеченням, так і власно написаним.

Після детального огляду наявних backend framework та їх дослідження, використано архітектурний шаблон MVC (Model View Controller) [3]. Оскільки основні переваги цього архітектурного шаблону – це розділення логіки програми від представлення та обробки вхідних даних. Тобто, кожен компонент відповідатиме конкретно за свої завдання, а можливість їх розширення, дозволить легко та просто змінювати або замінювати основні компоненти на інші без перероблення іншого коду. Робота над кожним компонентом Модель (Model), Представлення (View), Контролер (Controller) може вестися незалежно, що дозволяє команді розробників працювати паралельно над різними частинами програми, що також полегшуватиме роботу в команді. Завдяки розділенню логіки та представленню (View), легко перевикористовувати код Моделі (Model) та Контролера (Controller) в інших частинах програми чи в інших проєктах. Це дозволить простіше писати backend для інших плат у випадку, якщо проєкт буде використовувати не лише одну плату ESP, а декілька.

Розділення на компоненти спрощує тестування. Тести можна проводити окремо для кожного компонента, що полегшує виявлення помилок та впровадження змін без впливу на інші частини системи.

Існує можливість використовувати різні види подань (графічні, текстові, вебінтерфейс) без необхідності зміни логіки програми. Backend framework містить основні формати для відповідей html та json. Завдяки розділенню логіки представлення (View) можна легко та просто доробити свої формати відповідей, а зручний інтерфейс їх розширення та створення зробить цей процес максимально простим. [3]

Використання архітектурного патерну MVC дозволяє забезпечити стандартизацію в розробці проєктів, що полегшує розуміння та обслуговування коду в командному середовищі.

Маршрутизація (routing) є ключовим елементом багатьох backend framework, зокрема тих, що використовують патерн MVC. Маршрутизація дозволяє визначити, які частини програми будуть викликані при обробці конкретного HTTP-запиту. У MVC-фреймворках, таких як backend framework для мікроплат ESP, маршрутизація зазвичай відбувається на рівні контролера. [3]

View визначає, як відображати дані, включаючи їх форматування та стилізацію. Це може містити в собі визначення кольорів, шрифтів, розмірів та інших атрибутів відображення, що доволі корисно для формування відповідей у форматі HTML. У добре спроєктованій архітектурі, компонент Представлення відділений від логіки програми. Це означає, що View не містить бізнес-логіки та операцій, пов'язаних з обробкою даних. Все що йому необхідно передає Контролер та може використовувати моделі для отримання результатів з бази даних. [3]

Взаємодія з файловою системою або базою даних відбувається за допомогою ORM (Object-Relational Mapping). Вони будуть містять у собі прості інтерфейси збереження, отримання, та кешування даних. ORM дозволяє використовувати об'єктно-орієнтовані концепції, такі як класи, об'єкти, наслідування та поліморфізм, для роботи з даними, що полегшує розробку. Простота ORM полягатиме в тому, що розробникам не потрібно буде напряму видобувати щось із файлової системи, або думати, як щось

записати у базу даних для отримання або оновлення даних. Вони можуть використовувати методи та властивості об'єктів для взаємодії з базою даних.[4]

Кешування для даних отриманих за допомогою ORM відбувається на принципі lazy loading. Lazy loading – це концепція в об'єктноорієнтованому програмуванні, яка використовується зокрема в контексті роботи з базами даних та завантаженням об'єктів або даних. Основна ідея полягає в тому, щоб відкладати завантаження об'єктів або даних до того часу, коли вони дійсно потрібні для виконання операцій. Lazy loading дозволяє витратити ресурси тільки на ті дані, які реально потрібні у конкретний момент часу. Це може покращити продуктивність системи, особливо якщо таких даних дуже багато, або що важливіше їх довго отримувати, що в контексті мікроплат ESP є суттєво. [5]

Обґрунтування вибору інструментальних засобів розробки. Вибір мови програмування для роботи з платою ESP8266 та ESP32, зазвичай, залежить від підтримки та специфічних вимог платформи. Для цих цілей була обрана мова програмування C++. Мова програмування C++ широко використовується в розробці програмного забезпечення для мікроконтролерів і вбудованих систем. Існує велика кількість бібліотек та фреймворків для ESP8266, написаних на C або C++. Вони спрощують розробку та дозволяють швидше створювати програмне забезпечення для пристроїв IoT (Інтернет речей).[6]

Наступний не менш важливий інструмент це, безпосередньо, редактор самого коду. Для цілей написання та підтримки коду було обрано редактор Visual Studio Code. Редактор підтримує багато мов програмування і технологій, включаючи C, C++, JavaScript, Python, Java, HTML, CSS, і багато інших, що дозволяє писати код більшістю мов програмування. [7]

Для мікроплат ESP необхідно налаштувати середовище для вивантаження прошивок на плати ESP8266 та ESP32. Для цих цілей було обрано Arduino IDE, оскільки Arduino IDE має інтуїтивно зрозумілий та легкий у використанні інтерфейс, вбудовані приклади коду, які допомагають ознайомитися з основами програмування та роботи з конкретними пристроями, такими як ESP8266 або ESP32 [8]. Великою перевагою є те, що Arduino IDE використовує мову програмування C++, що в парі з обраною мовою програмування C++ та редактором коду Visual Studio Code утворює зручну комбінацію, яка максимально швидко прискорює процеси написання коду, його тестування та спрощує процес вивантаження нових прошивок на плати ESP8266 або ESP32.

Ще один інструмент для розробки, а саме для компілювання тестів, або ж коду backend framework є використання CMake. CMake – це кросплатформена система автоматизації генерації конфігураційних файлів для компіляції програмного забезпечення. Вона надає можливість визначення конфігураційного процесу та завдань компіляції незалежно від конкретної платформи чи компілятора [9]. CMake надає зручний спосіб визначення та керування залежностями між різними модулями та бібліотеками в проєкті. В результаті можна спокійно додавати нові бібліотеки до проєкту і забувати про їх ретельне налаштування.

Оскільки вивантаження прошивки для плати ESP8266 займає близько 1-2 хвилин, це стає проблемою, бо після мінімальних змін у коді доведеться дуже довго чекати, для отримання результатів його виконання. На допомогу цьому приходить одна з ключових функцій CMake. Однією з ключових функцій CMake є здатність генерувати конфігураційні файли для різних систем збірки та середовищ. Через це у розробці backend framework появляється нова можливість, а саме створення коду та його попередні запуски безпосередньо на тій платформі, на якій відбувається його написання, без потреби заливання проміжних прошивок на плати ESP8266 або ESP32. Це пришвидшує написання ключових алгоритмів, та пришвидшує побудову архітектури backend framework.

Особливості програмної реалізації. Основне завдання цієї роботи це реалізувати всі необхідні компоненти для backend framework. Оскільки обрано архітектурний патер MVC (Model, View, Controller), то реалізація кожного з них відбувається незалежно один від одного. Ці компоненти будуть реалізовувати основний функціонал та надавати всі необхідні інтерфейси для backend framework, тому їх винесено в окрему бібліотеку.

Спочатку, перед написанням бекенду, тобто реалізацією патерну MVC, необхідно зробити драйвера, який надасть прості та зрозумілі інтерфейси для керування платою ESP (рис. 1).

Драйвер містить обробник HTTP, що залежить напряму від мікроплати ESP8266 або ESP32, та для обробника використовується бібліотека "ESP8266WebServer" для взаємодії із платою ESP8266. Вона допомагає сконфігурувати HTTP сервера та відкрити TCP/IP з'єднання для передачі даних. Обробник файлової системи також залежить від плати ESP, тому для роботи з файловою системою використовується бібліотека "FS" та "LittleFS" для мікроплат ESP. Оскільки середовище програмування плат ESP це Arduino IDE тому використовується бібліотека "Arduino" для кращої взаємодії із середовищем Arduino IDE.

src > h++ ESP8266_NII_DRIVER.hpp > ...
You, 1 minute ago | 1 author (You)

```
1   #pragma once
2
3   #include "Backend.hpp" // Бібліотека Backend framework (MVC based)
4
5   #include <ESP8266WebServer.h>   // вебсерве HTTP (TCP/IP)
6   #include <Arduino.h>            // Середовище Arduino IDE
7
8   #include <FS.h>        // Файлова система
9   #include <LittleFS.h>  // Драйвер файлової системи
10
11
```

Рис. 1. Заголовки та підключені бібліотеки для драйвера між backend framework та ESP8266

Наступним кроком реалізації, це створення двох основних режимів роботи драйверу. Перший – це його ініціалізація (рис. 2), та другий – обробка та оновлення всіх вхідних запитів та процесів backend framework (рис. 3).

Безпосередньо сам мікроконтролер ESP не вміє підключатися до мереж інтернет, або будь-якої іншої мережі, будь то локально. Тому для його ініціалізації передається два параметри: назва та пароль до мережі wifi (рис.2, рядок 69, 70). Третій параметр не обов'язковий, можна одразу в ініціалізацію задати маршрутизацію для backend framework (рис. 2, рядок 71).

Наступним кроком після реалізації драйвера, це реалізація безпосередньо backend framework, який працює на архітектурному патерні MVC. Щоб відділити backend framework від інших бібліотек, з метою захисту від конфліктів імен, всі функції та класи описуються в просторі імен nii. Також використані підпростори, наприклад Router для отримання доступу до функції та класів маршрутизації.

Після того як драйвер ініціалізувався, потрібно надати інформацію про маршрути. Це можна зробити за допомогою RouteBuilder, створити інстанс якого, можна за допомогою виклику nii::Router::builder(). Далі потібно вказати шлях метод path() та виклик контролера (обробника) метод call(). Обробник може бути трьох типів: обробник методу класу; обробник функції; обробник лямбда функції (рис. 4).

```
66      ESP8266WebServer server(80);
67
68      inline void setup(
69          const char *ssid = nullptr,
70          const char *passPhrase = nullptr,
71          void (*defineRoutes)() = nullptr
72      )
73      {
74          delay(1000);
75          Serial.begin(115200);
76          delay(1000);
77          TRACE("Start up..\n");
78
79          WiFi.mode(WIFI_STA);
80          if (strlen(ssid) == 0) {
81              WiFi.begin();
82          } else {
83              WiFi.begin(ssid, passPhrase);
84          }
85
86          TRACE("Mounting the filesystem...\n");
87          if (!LittleFS.begin()) {
88              TRACE("could not mount the filesystem...\n");
89              delay(2000);
90              ESP.restart();
91          }
92
93          TRACE("Connecting to WiFi...\n");
94          while (WiFi.status() != WL_CONNECTED) {
95              delay(500);
96              TRACE(".");
97          }
98          TRACE("connected.\n");
99
100         server.addHandler(new Driver(&server));
101
102         server.enableCORS(true);
103
104         if (defineRoutes) {
105             defineRoutes();
106         }
107
108         server.begin();
109         TRACE("hostname=%s\n", WiFi.getHostname());
110
111         TRACE("server_url=%s\n", WiFi.localIP().toString().c_str());
112     }
```

Рис. 2. Ініціалізація драйвера backend framework для ESP8266

```
106
107     inline void update()
108     {
109         server.handleClient();
110     }
111
```

Рис. 3. Обробка та оновлення всіх процесів backend framework

```
nii::Router::builder()->path("/")->call(&MainController::index);

nii::Router::builder()->path("/foo")->call(foo);

nii::Router::builder()->path("/foo")->call([] () {
    return ...
});
```

Рис. 4. Створення маршрутизації для backend framework із вказанням подальших контролерів (обробників) та їх шляху.

Передача параметрів у маршрути відбувається за допомогою символу $ (рис. 5).

```
Router::builder()->path("/test/$")->call([] (int parameter) {
    return  parameter + 10;
});
```

Рис.5. Передача параметрів маршруту та їх обробка

Обробник параметрів маршруту, автоматично дивиться на параметри які приймає на вхід обробник, та намагається їх обробити, а у випадку коли параметри не коректні, запуститься обробник 404 помилки.

На рис 6 подана структура маршрутизації. Це безпосередньо сам маршрут (файл RouteHolder.hpp), інформація про те який контролер (обробник) потрібно викликати (файл Callable.hpp), будівельник маршрутів (файл RouteBuilder.hpp), обробник параметрів (файл Binder.hpp).

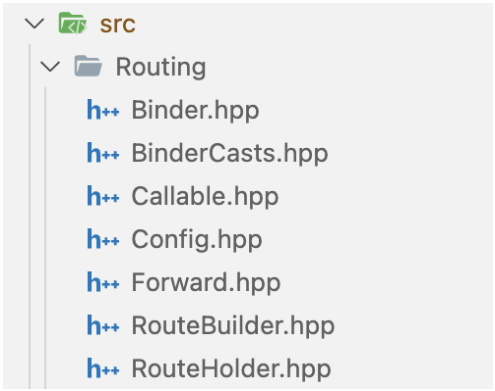

Рис. 6. Структура файлів маршрутизації

Обробник параметрів за замовчуванням обробляє параметри звичайних типів (числовий, рядковий, логічний). Підтримка інших типів відбувається за допомогою інтерфейсу розширення (рис.7).

Отже, простота у контролерах та їх обробниках здебільшого полягає у маршрутизації та параметрів які передаються. В той час як формування відповідей, тобто компонент представлення у MVC відбувається безпосередньо в контролерах (рис. 8).

```
template<>
struct BinderCaster<int>
{
    inline static int cast(Parameter &&parameter)
    {
        return std::stoi(parameter.raw());
    }
};
```

Рис. 7. Приклад розширення обробника параметрів для типу ціле число (int)

```
43      inline nii::Response *zones()
44      {
45          auto res = new nii::JsonResponse();
46
47          auto items = DB<Zone>::get();
48
49          for (int i = 0; i < items.size(); i++) {
50            res->json()["data"][i]["key"] = items[i].key;
51            res->json()["data"][i]["mode"] = items[i].mode;
52            res->json()["data"][i]["from"] = items[i].from;
53            res->json()["data"][i]["to"] = items[i].to;
54          }
55
56          return res;
57      }
```

Рис. 8. Формування відповіді формату json у контролері

Рядок 45 (рис. 8) створює представлення (View) у форматі json, якщо бути конкретнішим з цього моменту починається формування відповіді на запит, що попадає до обробника zones() та повертається уже сформована відповідь для подальшої обробки запиту, рядок 56 (рис. 8).

За бажанням можна розширити або створити власну відповідь як результат роботи представлення (View) у патерні MVC. Для цього потрібно наслідувати клас Response (рис. 9) або ж один із його дочірніх класів, наприклад JsonResponse (відповідь у форматі json). Базовий клас відповіді повертають усі контролери (обробники), тобто вимога до кожного з обробників, така, що він мусить повертати саме один із дочірніх класів Response (рис. 10).

Наступна компонента в архітектурному патерні MVC яку залишилося розробити – це модель (Model). На рис. 8, рядок 47 показано приклад запиту до бази даних, для отримання моделі Zone. ORM має простий інтерфейс (рис. 11) з основними функціями, для отримання, збереження, синхронізації, кешування, видалення та оновлення даних.

Сама ORM працює лише з моделями й клас DB є шаблоном, тобто він описує лише поведінку, а не точну реалізацію, зокрема так само як і ORM. Ми можемо створити модель та отримати до неї доступ за допомогою DB<НазваМоделі> після чого зробити виклик однієї з функцій доступних для ORM. Будь-яка модель для того, щоб могла працювати з ORM повинна задовольняти певні вимоги, зокрема реалізацію інтерфейсу взаємодії з базою (рис. 12).

```
10    class Response
11    {
12     public:
13        Response();
14        virtual ~Response();
15
16        // virtual int code() = 0;
17        virtual int code()
18        {
19            return 200;
20        }
21
22        virtual const char* contentType()
23        {
24            return "text/plain";
25        }
26
27        virtual const char* data()
28        {
29            return "OK";
30        }
31
32        virtual size_t contentLength()
33        {
34            return 3;
35        }
36
37        virtual std::vector<std::pair<std::string, std::string>> headers()
38        {
39            return {};
40        }
41
42     private:
43    };
```

Рис.9. Базовий клас відповіді

```
inline nii::Response *redirect()
{
    return new nii::RedirectResponse("/zones");
}
```

Рис. 10. Приклад обробника що робить перенаправлення клієнта на іншу сторінку

Контроль файлів та бази даних здійснюється за допомогою поєднання двох компонентів, конкретної моделі із реалізованим інтерфейсом (рис. 12) та класу DB для роботи викликів ORM (рис. 11). Вказання таблиці для моделі є важливим, оскільки саме назва таблиці буде ідентифікатором файлу у файловій системі ESP8266 (рис. 13), а реалізація функції db_write надасть розуміння того, що з моделі потрібно зберегти у базу даних.

Кешування для ORM здійснюється за допомогою паралельного введення масиву запитаних моделей (рис. 11 рядок 12) за допомогою механізму Lazy loading, після першого запиту до бази даних конкретної таблиці, ORM закешує отриманий результат

в оперативній пам'яті ESP8266 та дозволить швидко отримувати доступ до полів моделі, зокрема користуватися нею як звичайним класом в C++.

ORM вміє не тільки віддавати результати з бази даних, а ще їх записувати, для цього існує метод save() (рис. 13) він починає збереження всіх наявних на даних момент моделей того класу (таблиці), до якого був викликаний.

```
 9  template<class T>
10  struct DB
11  {
12      inline static std::vector<T> items = {};
13
14      inline static bool cached = false;
15
16  >   inline static void sync(T &elem, bool forceSave = true)…
35
36  >   inline static void remove(T &elem, bool forceSave = true)…
56
57  >   inline static void refresh()…
61
62  >   inline static void save()…
76
77  >   inline static  std::vector<T> & get()…
102 };      You, 6 days ago • db complete
```

Рис. 11. Основні методи ORM

```
bool db_same(const Zone &other)…

std::string db_write()…

static Zone db_from(const std::string &stream)…

static std::string db_table()…
```

Рис.12. Інтерфейс взаємодії з базою даних для моделі Zone

```
inline static void save()
{
    DynamicJsonDocument json(2048);

    for (auto &item : items) {
        json["items"].add(item.db_write());
    }

    std::ofstream fs(T::db_table()+".db.json");

    serializeJson(json, fs);

    fs.close();
}
```

Рис.13. Функція збереження для ORM

Основний режим роботи як backend framework полягає у тому, щоб надати користувачам, тобто іншим програмістам, зручний та простий інтерфейс створення backend для їх серверів на мікроплатах ESP8266. Архітектурний патерн MVC дозволяє розділяти логіку додатків, а прості та зручні інтерфейси дозволяють швидко приступити до безпосереднього написання логічної складової програми (бізнес-логіки) замість обдумування того як воно працює в середині.

Умови застосування backend framework для мікроплат ESP із підтримкою HTTP протоколу.

Мінімальні вимоги сервера: плата ESP8266, ESP32 або Node MCU ESP8266.

Мінімальні вимоги для запуску Arduino IDE:

- операційна система Windows, Linux, MacOS;
- 256 Мб оперативної пам'яті;
- процесор Pentium 4 або вище.

Додаткові бібліотеки, які необхідно встановити для Arduino IDE: ESP8266WebServer.

Додаткові менеджери плат які необхідно встановити для Arduino IDE: http://arduino.esp8266.com/stable/package_esp8266com_index.json.

Для отримання доступу до всіх функцій та модулів системи, необхідно під'єднати бібліотеку #include <ESP8266_NII_DRIVER>.

nii::backend::setup(1, 2, 3) – ініціалізація backend framework. Де 1 – логін до wifi, 2 – пароль до wifi, 3 – необов'язковий, функція виклику створення маршрутів.

nii::backend::update() – оновлення backend framework, необхідне для обробки вхідних запитів.

nii::Router::builder() – створення інстансу для будівельника маршрутів.

Будівельник маршрутів має наступні методи:

- path(1) – задати шлях маршруту (1 – це рядок, в якому через знак $ можна задавати параметри маршруту);
- call(1) – вказати обробника для маршруту (1 – це функція, метод класу або лямбда вираз, який в свою чергу на вхід приймає параметри маршруту та повинен повертати вказівник на інстанс класу для відповіді).

nii::Response – клас для відповіді, його різновиди:

- nii::HtmlResponse – відповідь у форматі html;
- nii::JsonResponse – відповідь у форматі json;
- nii::RedirectResponse – відповідь для перенаправлення клієнта на сторінку.

DB<1> – ORM будівельник для отримання результатів, або проведення дій над базою даних (1 - клас моделі. Доступні статичні методи:

- sync(1,2) – синхронізувати інстанс класу моделі передану у параметрі 1, де параметр 2 – булеве, чи потрібно одразу записати дані у базу даних, якщо істина дані одразу синхронізуються з базою даних, якщо хиба, дані синхронізуються з базою даних при наступному викликові збереження;
- remove(1,2) – видалити інстанс класу моделі передану у параметрі 1, де параметр 2 – булеве, чи потрібно одразу видалити дані з бази даних, якщо істина дані одразу видаляться з бази даних, якщо хиба, дані видаляться при наступному викликові збереження;
- save() – зберегти поточний стан бази даних для класу моделі;
- get() – отримати всі результати з бази даних для класу моделі;
- refresh() – оновити всі записи з бази даних, та відновити їх структуру на початкову, ту яка була у базі даних після останнього збереження.

Введення моделей, відбувається за допомогою реалізації наступних методів:

- bool db_same(const MODEL &other) – повертає булеве, на вхід приймає, де MODEL клас моделі. Результатом перевірка чи модель яка подається на вхід аналогічна як і поточний інстанс в плані запису у базі даних;
- std::string db_write() – метод результат якого є, те яким чином модель має серіалізуватися у базу даних, повертає рядок;
- static MODEL db_from(const std::string &stream) – статичний метод повертає інстанс MODEL, який є класом моделі, з рядка, який є параметром;
- static std::string db_table() – повертає рядок, назва таблиці до якої належть модель.

Робота з json опрацьовує бібліотека ArduinoJson.h. Основні її функції, це серіалізація даних (рис. 14) та їх десеріалізація (рис. 15).

```
DynamicJsonDocument doc(1024);

doc["sensor"] = "gps";
doc["time"]   = 1351824120;
doc["data"][0] = 48.756080;
doc["data"][1] = 2.302038;

serializeJson(doc, Serial);
// This prints:
// {"sensor":"gps","time":1351824120,"data":[48.756080,2.
```

Рис. 14. Приклад серіалізації даних у бібліотеці ArduinoJson.h

```
char json[] = "{\"sensor\":\"gps\",\"time\":1351824120,\'

DynamicJsonDocument doc(1024);
deserializeJson(doc, json);

const char* sensor = doc["sensor"];
long time          = doc["time"];
double latitude    = doc["data"][0];
double longitude   = doc["data"][1];
```

Рис. 15. Приклад десеріалізації даних у бібліотеці ArduinoJson.h

Додаткові методи для інтерфейсу відповіді nii::Response:
- setCode(1) – встановлює код відповіді на той що передано у параметрі 1;
- contentType() - повертає тип контенту;
- data() – повертає сирий вказівник на контент відповіді;
- code() – повертає поточний код відповіді.

Додаткові метод для html відповіді, клас nii::HtmlResponse:
- setData(1) – встановлює контент відповіді, який передано у параметрі 1;
- addData(1) – додає контент до відповіді, який передано у параметрі 1;
- clear() – очищує поточний контент відповіді.

Додаткові метод для json відповіді, клас nii::JsonResponse:
- json() – повертає поточний json документ, для його подальших змін.

Висновки. У зв'язку зі зростанням кількості пристроїв, які входять до складу Інтернету речей (IoT), розробники шукають ефективні та надійні інструменти для створення програмного забезпечення для таких систем. Дослідження особливостей розробки backend framework для мікроплат ESP із підтримкою HTTP (TCP/IP) протоколу свідчить про важливість розширення можливостей цих пристроїв у сфері Інтернету речей (IoT). Розробка спеціалізованого backend-рішення дозволяє оптимізувати роботу

плат ESP8266 та ESP32, забезпечуючи ефективніше використання ресурсів та поліпшення загальної продуктивності.

Впровадження цього фреймворку в розробку інтернет-застосунків для ESP-платформ відкриє широкі можливості для створення ефективних та ресурсозаощаджувальних проєктів. Забезпечуючи гнучкість у взаємодії з HTTP протоколом, він дозволить розробникам швидко реалізувати власні вебсервери та ефективно обробляти HTTP-запити.

Оптимізація для ресурсоощадження стає ключовою перевагою фреймворку, особливо в умовах обмежених можливостей мікроконтролерів ESP. Це робить його особливо цікавим для розробників, які працюють над проєктами, де кожен байт пам'яті та кожен цикл процесора має значення. Загалом, враховуючи його властивості та можливості, цей backend framework стає не лише ефективним інструментом для розробки на мікроконтролерах ESP, але й стратегічним вибором для інноваційних проєктів в галузі Інтернету речей (IoT).

References

1. Litayem N., Al-Sa'di A. Exploring the Programming Model, Security Vulnerabilities, and Usability of ESP8266 and ESP32 Platforms for IoT Development. *2023 IEEE 3rd International Conference on Computer Systems (ICCS)*, Qingdao, China, 22–24 September 2023. 2023. URL: https://doi.org/10.1109/iccs59700.2023.10335558.
2. Comparison between ESP32 and ESP8266: Which is the best option? - Polaridad.es. *Polaridad.es*. URL: https://polaridad.es/en/comparacion-entre-esp32-y-esp8266-cual-es-la-mejor-opcion/.
3. Model-View-ViewModel - .NET. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm.
4. Editor. Understanding Object-Relational Mapping. *AltexSoft*. URL: https://www.altexsoft.com/blog/object-relational-mapping/.
5. Що таке lazy loading або «ліниве завантаження» контенту і як воно впливає на SEO?. *IdeaDigital Agency*. URL: https://ideadigital.agency/blog/lazy-loading-dlya-seo/.
6. Microsoft C/C++ Documentation. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/cpp/?view=msvc-170.
7. Microsoft. Visual Studio Code - Code Editing. Redefined. *Visual Studio Code - Code Editing. Redefined*. URL: https://code.visualstudio.com/.
8. Arduino Docs | Arduino Documentation. *Arduino Docs | Arduino Documentation*. URL: https://docs.arduino.cc/.
9. CMake: The Standard Build System. *CMake - Upgrade Your Software Build System*. URL: https://cmake.org/features/.