**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ТАРАСА ШЕВЧЕНКА**

ФАКУЛЬТЕТ РАДІОФІЗИКИ, ЕЛЕКТРОНІКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ

Кафедра комп'ютерної інженерії

До захисту допущено: «На правах рукопису»
Завідувач кафедри _____ Юрій Бойко
« _ » _____ 2023 р.

# КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему:

# «АПАРАТНО-ПРОГРАМНИЙ КОМПЛЕКС МОНІТОРИНГУ ВІДВІДУВАНЬ ЖИТЛОВИХ ТА СЛУЖБОВИХ ПРИМІЩЕНЬ»

**Виконав:**
студент 4-го курсу бакалаврату
денної форми навчання
спеціальності 123 Комп'ютерна інженерія
ОНП «_____»
Побережний Максим Русланович _____

**Науковий керівник**:
кандидат фізико-математичних наук, доцент
Загороднюк Сергій Петрович _____

**Рецензент:**
_____

Засвідчую, що у цій бакалаврській роботі
немає запозичень з праць інших авторів без
відповідних посилань
Студент _____

Робота допущена до захисту в ЕК рішенням кафедри _____
від «__»_____ 2023 р., протокол № __.

Завідувач кафедри _____,
кандидат фізико-математичних наук, доцент
Бойко Юрій Володимирович

(підпис)

Київ – 2023

## РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра містить 88 сторінок, 27 рисунків, 24 додатки, використано 9 інформаційних джерел.

Розроблено апартно-програмний комплекс моніторингу відвідувань житлових та службових приміщень. Розглянуто різні способи комунікації мікроконтролера з сервером. Розроблено веб-застосунок для користувача.

В якості мікроконтролера використовувався NodeMCU ESP8266. Для реалізації серверної частини використовувався фреймворк Django.

# ЗМІСТ

# ВСТУП

Стрімкий розвиток інформаційних технологій відкриває багато нових можливостей для покращення якості життя людей. З кожним роком обчислювальні пристрої стають меншими, потужнішими та дешевшими, що відкриває широкий спектр для їх застосування. Однією з таких сфер є інтернет речей, суть якої полягає у розробці систем пристроїв, які забезпечують спілкування фізичного світу зі світом комп'ютерних систем.

Різноманіття датчиків та мікроконтролерів дозволяє створювати системи майже для будь-яких цілей. Мета та спосіб реалізації обмежуються хіба що уявою розробника.

В даній роботі була реалізована система спостереження за відкриттям та закриттям дверей на базі мікроконтролера NodeMCU ESP8266 та датчика відкриття дверей MC-38. Також розглянуто спостереження за температурою і вологістю в примищені за допомогою датчика SHT20 з інтерфейсом підключення I2C. Для кінцевого користувача був розроблений вебінтерфейс на базі Django, стилізація виконувалась з використанням фреймворку Bootstrap 5. Спілкування мікроконтролера, сервера та кінцевої сторінки користувача реалізована за допомогою протоколу WebSocket. Сервер вебсокетів реалізований на базі Django Channels. Реалізація вебсокету на стороні сторінки користувача, а також всі динамічні елементи на сторінці реалізовані за допомогою JavaScript.

Також були реалізовані такі аспекти, як аутентифікація мікроконтролера за допомогою токена, аутентифікація користувача за допомогою унікального ідентифікатора сесії, мінімізація навантаження на мережеві канали, шляхом розробки кодів для спілкування мікроконтролера з сервером, динамічне оновлення елементів на сторінці користувача, без необхідності оновлювати сторінку, анімація деяких елементів на сторінці, здатність системи до розширення, шляхом забезпечення підтримки багатьох користувачів та

багатьох мікроконтролерів, керування мікроконтролером зі сторінки користувача, перегляд журналу, в якому записані події відкриття або закриття дверей та точний час події, перегляд журналу, в якому записані події мікроконтролера, а саме його перезавантаження, час коли відбулося перезавантаження, а також причина перезавантаження, постійне спостерження за тим чи є мікроконтролер зараз в мережі та відображення цієї інформації користувачу, збереження часу, коли мікроконтролер був останній раз у мережі, постійни моніторинг та відображення в реальному часі стану дверей – відкриті або закриті, перегляд в реальному часі даних з датчику температури та вологості повітря, оновлення в реальному часі журналу відкриття/закриття дверей та журналу подій мікроконтролера.

# 1. ЗАГАЛЬНИЙ ПРИНЦИП РОБОТИ СИСТЕМИ

## 1.1 Компоненти системи

Система складається з трьох основних компонентів: мікроконтролера NodeMCU ESP8266, сервера на базі фреймворку Django та вебсторінки для користувача.

На стороні міскроконтролера NodeMCU ESP8266 реалізовано зчитування інформації з датчика відкриття дверей MC-38 та датчика температури та вологості SHT20, керування реле, клієнтська частина вебсокету для спілкування з сервером і NTP клієнт для отримання точної дати та часу.

На стороні сервера реалізована серверна частина вебсокету для спілкування з мікроконтролером та сторінкою користувача, база даних, в якій зберігається інформація про користувача, інформація про мікроконтролер, його токен, ідентифікатор та власник, журнал для запису подій відкриття/закриття дверей, журнал для запису подій перезавантаження мікроконтролера. Також реалізований HTTP сервер для надання користувачу вебсторінки, її скриптів та стилів. Мета сервера забезпечити з’явзок користувача з мікроконтролером, аутентифікацію користувача і мікроконтролера, авторизацію користувача і надати йому доступ тільки до тих мікроконтролерів, власником яких він є, панель адміністратора для адміністраторів системи та збереження необхідних даних.

На стороні вебсторінки для користувача реалізований зручний інтерфейс для керування мікроконтролером та перегляду інформації з нього. Для спілкування з сервером реалізовані клієнтські частини вебсокетів там, де це необхідно. Дизайн сторінки спрямований на те, щоб надати користувачу мінімалістичний, приємний на око та інтуїтивно зрозумілий інструмент для роботи з мікроконтролером.

## 1.2 Спілкування між компонентами системи та зовнішніми сервісами.

### 1.2.1 Спілкування між мікроконтролером та сервером.

В процесі розробки апаратно-програмного комплексу моніторингу відвідувань житлових та службових приміщень були розглянуті два протоколи для спілкування мікроконтролера і сервера: HTTP та вебсокет.

Вебсокет – протокол сьомого рівня моделі OSI, який забезпечує двосторонній безперервний зв’язок між клієнтом і сервером для передачі даних.

HTTP/1.1 – протокол сьомого рівня моделі OSI, який переважно використовується для передачі вебсторінок користувачу.

Спілкування через HTTP/1.1 відбувається наступним чином: клієнт встановлює TCP з’єднання з сервером, потім надсилає запит на сервер, сервер дає відповідь на запит. Виходячи з цього, для забезпечення двостороннього з’єднання, потрібно реалізувати з обох сторін і HTTP/1.1 сервер і HTTP/1.1 клієнт.

У випадку з вебсокетами, спочатку створюється з’єднання між клієнтом та сервером, далі і клієнт і сервер готові приймати інформацію в будь-який момент до розриву з’єднання. Такий підхід дозволяє уникнути створення новго з’єднання кожен раз, коли відбувається обмін інформацією, серверу і клієнту потрібно лиш обмінюватися ping-pong сигналами для підтримки з’єднання. Також вебсокет займає лише один порт з кожної сторони, в той час як HTTP/1.1 клієнту і серверу потрібні два порта з кожної сторони.

### 1.2.2 Спілкування між сервером та вебсторінкою користувача

Ще однією перевагою вебсокетів, виявилась можливість динамічного оновлення інформації на вебсторінці користувача. Клієнти вебсокету реалізовуються мовою JavaScript. Вони встановлюють з’єднання з сервером і готові в будь-який момент приймати інформацію та відразу ж відображають її

на сторінці. Без вебсокетів такий функціонал було б важко реалізувати. Одним з варіантів реалізації міг би бути API клієнт, який через кожен визначений проміжок часу, наприклад 2 секунди, запитував би у сервера оновлену інформацію. Такий підхід є менш ефективним, тому що при кожному запиті потрібно було б створювати нове з'єднання, а також не завжди цей запит був би актуальний, тому що на сервері можуть лежати ті самі дані, що і дві секунди назад, тому клієнт ніякої б нової інформації не отримав і тільки марно навантажив би сервер.

### 1.2.3 Отримання точного часу по протоколу NTP

Як було зазначено в попередньому пункті, мікроконтролер, при надсиланні подій перезавантаження або відкриття/закриття дверей, додає точний час події. Для того щоб визначити точний час, був реалізований NTP клієнт на стороні мікроконтролера, який отримує його з сервісу pool.ntp.org.

### 1.2.4 HTTP сервер для того щоб користувач міг отримати вебсторінку

HTTP сервер реалізований для того щоб надіслати користувачу вебсторінку, JavaScript і стиль до неї. В цьому випадку клієнтом виступає веб-браузер користувача, який надсилає HTTP запит за певною адресою і отримує у відповідь відповідну HTML сторінку, а також JavaScript та CSS до неї. Деякі Bootstrap стилі і скрипти, які використовує вебсторінка підвантажуються з зовнішнього ресурсу.

Схема комунікації між компонентами апартно-програмного комплексу та зовнішніми сервісами зображена на рисунку 1.

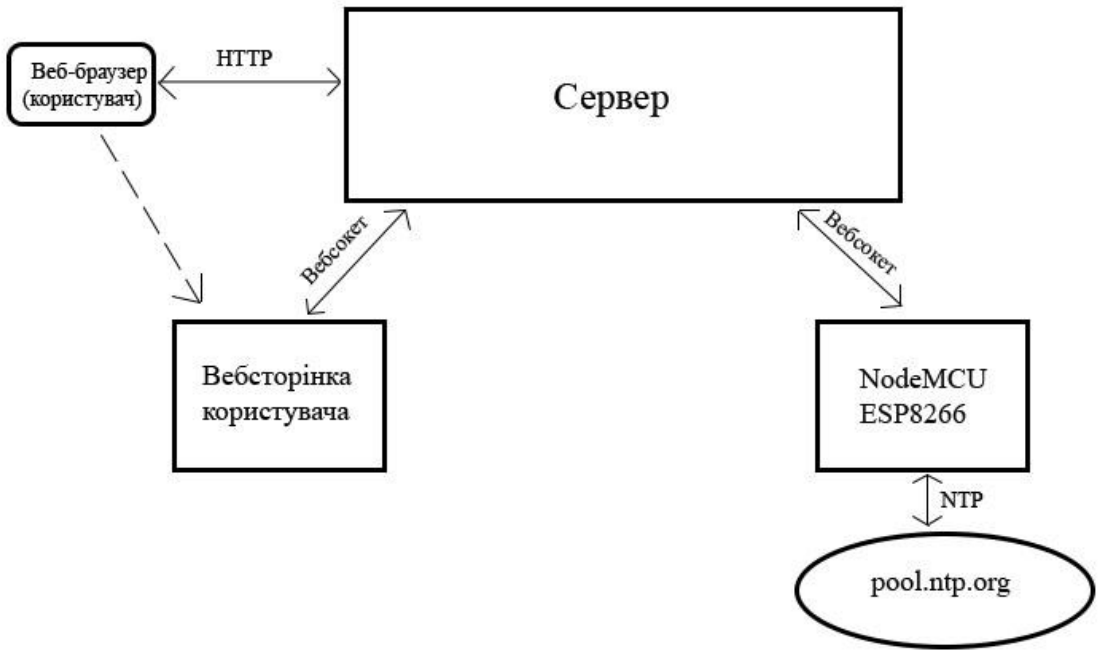


Рисунок 1 Схема комунікації між компонентами системи та зовнішніми сервісами

# 2. ДЕТАЛЬНИЙ ОГЛЯД РЕАЛІЗАЦІЇ ПІДКЛЮЧЕННЯ ДАТЧИКІВ ДО МІКРОКОНТРОЛЕРА NODEMCU ESP8266 ТА ПРОГРАМНОГО КОДУ ДЛЯ НЬОГО

## 2.1 Загальні відомості про NodeMCU ESP8266

Назва NodeMCU ESP8266 складається з двох основних частин: NodeMCU та ESP8266. ESP8266 – це чіп розроблений компанією Espressif Systems, який має вбудований Wi-Fi модуль та є низьковартісним, що робить його чудовим рішенням для сфери інтернет речей та розумного дому. Використовувати лише чіп не завжди буває зручно, тому ESP8266 розміщена на платі NodeMCU, яка надає доступ до 11 GPIO ніжок чіпа, має вбудований аналогово-цифровий перетворювач, Mini-USB інтерфейс, через який можна програмувати чіп та подавати живлення на схему. Також на платі розміщені два світлодіоди, один з яких підключений до ніжки GPIO2, а інший до GPIO16. Малюнок та підпис ніжок плати зображені на рис. 2.1.

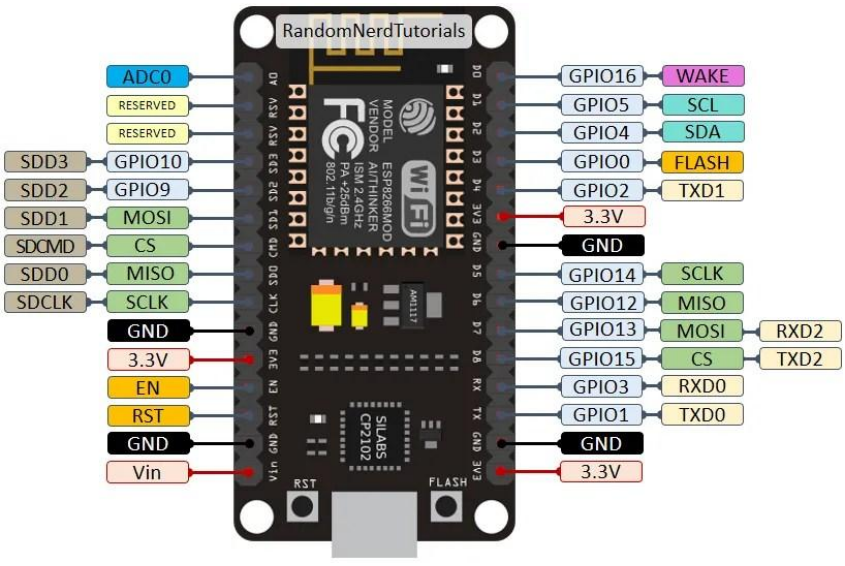


Рисунок 2.1 Малюнок та підпис ніжок плати NodeMCU ESP8266

Джерело: https://randomnerdtutorials.com/getting-started-with-esp8266-wifi-transceiver-review/#esp8266-intro

## 2.2 Середовище розробки PlatformIO

PlatfromIO – це розширення для IDE (інтегрованого середовища розробки) Visual Studio Code для програмування безлічі різних платформ, в тому числі і NodeMCU ESP8266.

Щоб встановити PlatformIO, потрібно у Visual Studio Code перейти на вкладку Extensions (Розширення) (рис. 2.2.1), у пошуку ввести PlatformIO, обрати перший варіант та настинути Install (Встановити) (рис. 2.2.2). Після встановлення, в меню зліва з’явиться іконка PlatformIO (рис. 2.2.3). Потрібно натиснути на цю іконку, Visual Studio Code почне завантаження всіх необхідних додаткових файлів для цього розширення. Після того, як завантаження завершиться, справа знизу з’явиться пропозиція перезавантажити IDE (рис. 2.2.4). Як тільки VSCode перезавантажився PlatformIO стане доступною для використання.

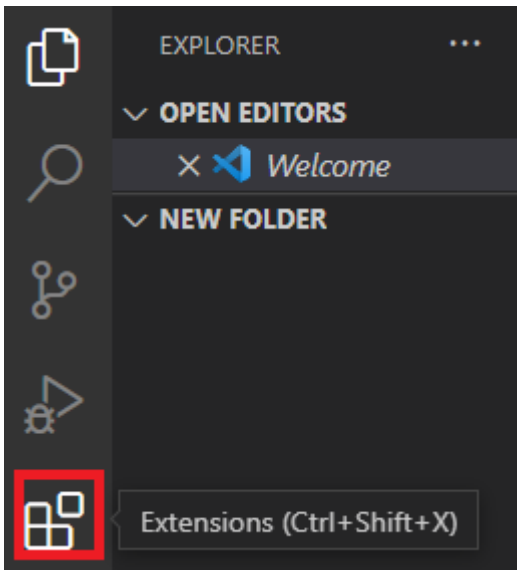


Рисунок 2.2.1 Вкладка «Розширення» у Visual Studio Code

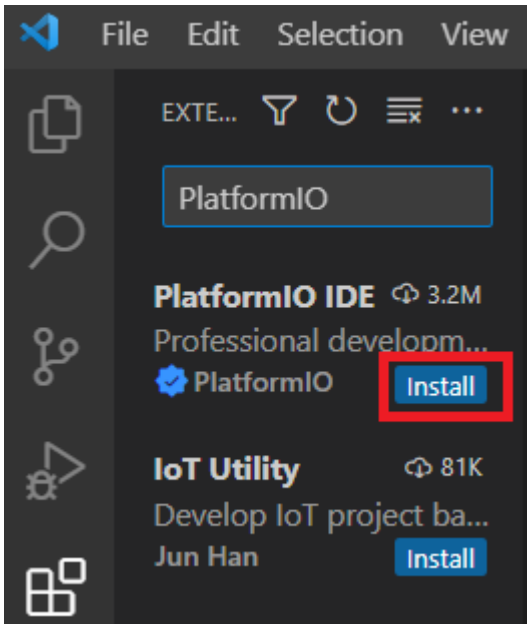


Рисунок 2.2.2 Розширення PlatformIO IDE у пошуку

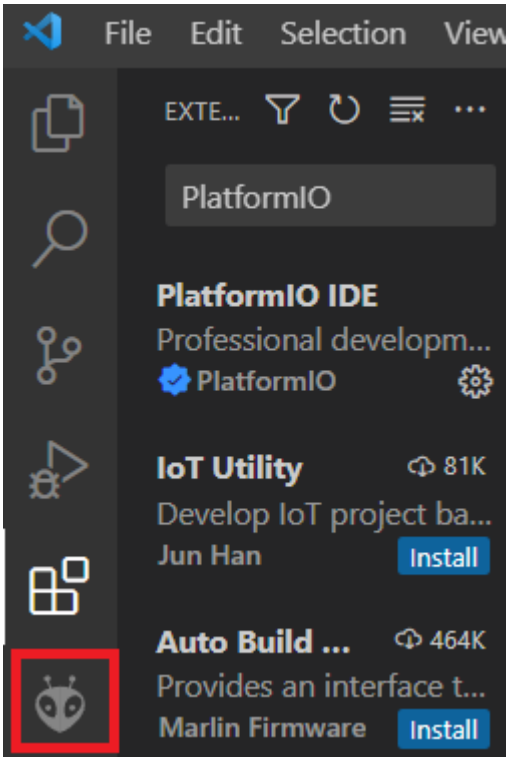


Рисунок 2.2.3 Іконка PlatformIO на боковій панелі Visual Studio Code

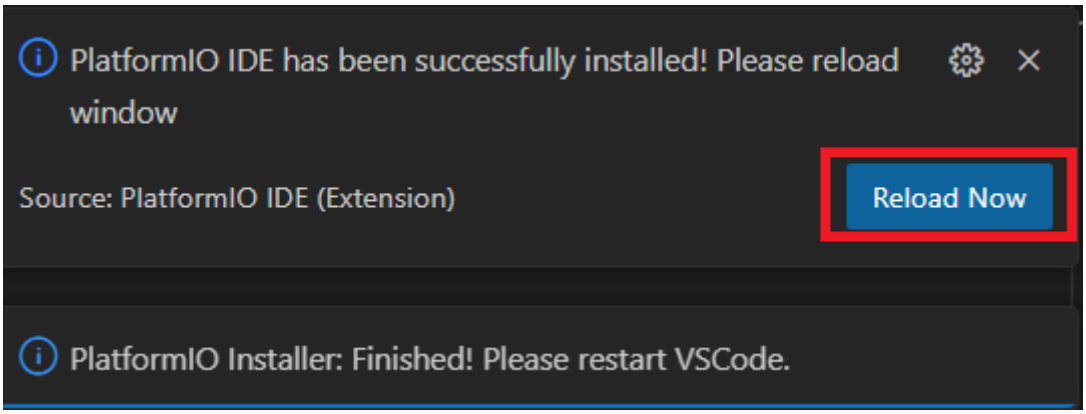


Рисунок 2.2.4 Пропозиція перезавантажити VSCode після повного встановлення PlatformIO

Для створення нового проєкту під NodeMCU ESP8266 в PlatformIO потрібно з домашньої сторінки перейти у вкладку Projects та натиснути Create New Project (рис. 2.2.5). Після чого відкриється Project Wizard (рис. 2.2.6), в якому в поле Name вписується назва проєкту, в поле Board потрібно обрати NodeMCU 1.0 (ESP-12E Module), в поле Framework – Arduino, при зніманні галочки Location, можна обрати своє власне розташування проєкту. Далі натискається кнопка Finish і PlatformIO створює новий проєкт. Основний код пишеться в main.cpp файл, який розташований в каталозі src проєкту.

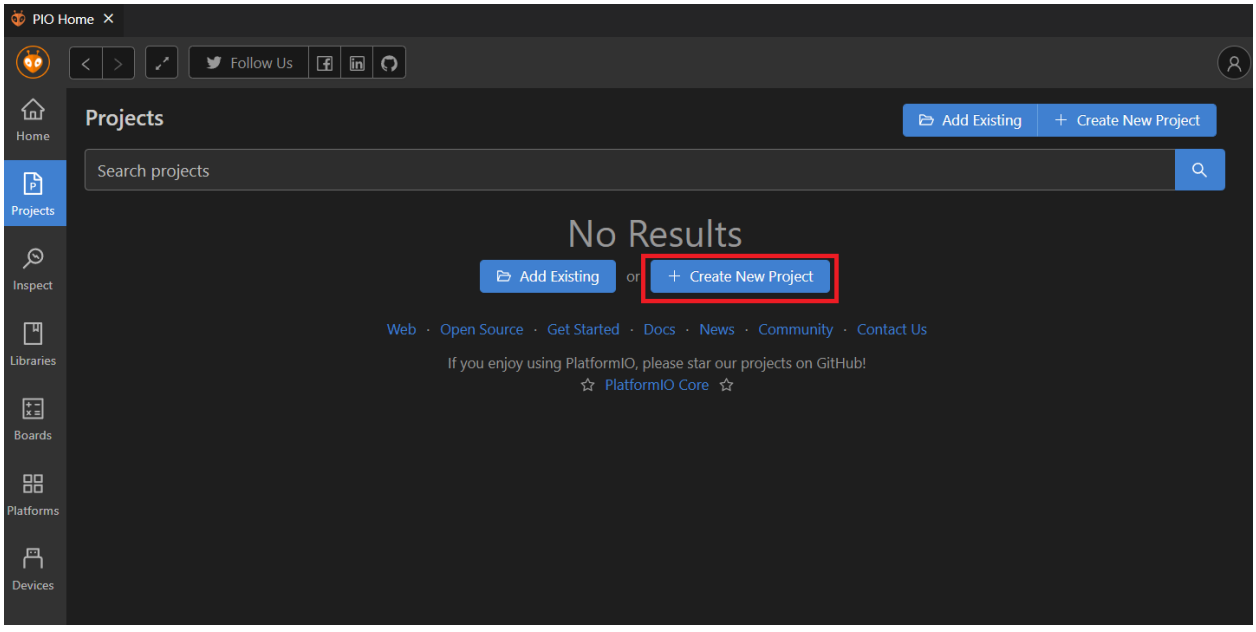


Рисунок 2.2.5 Вкладка створення нового проєкту в PlatformIO

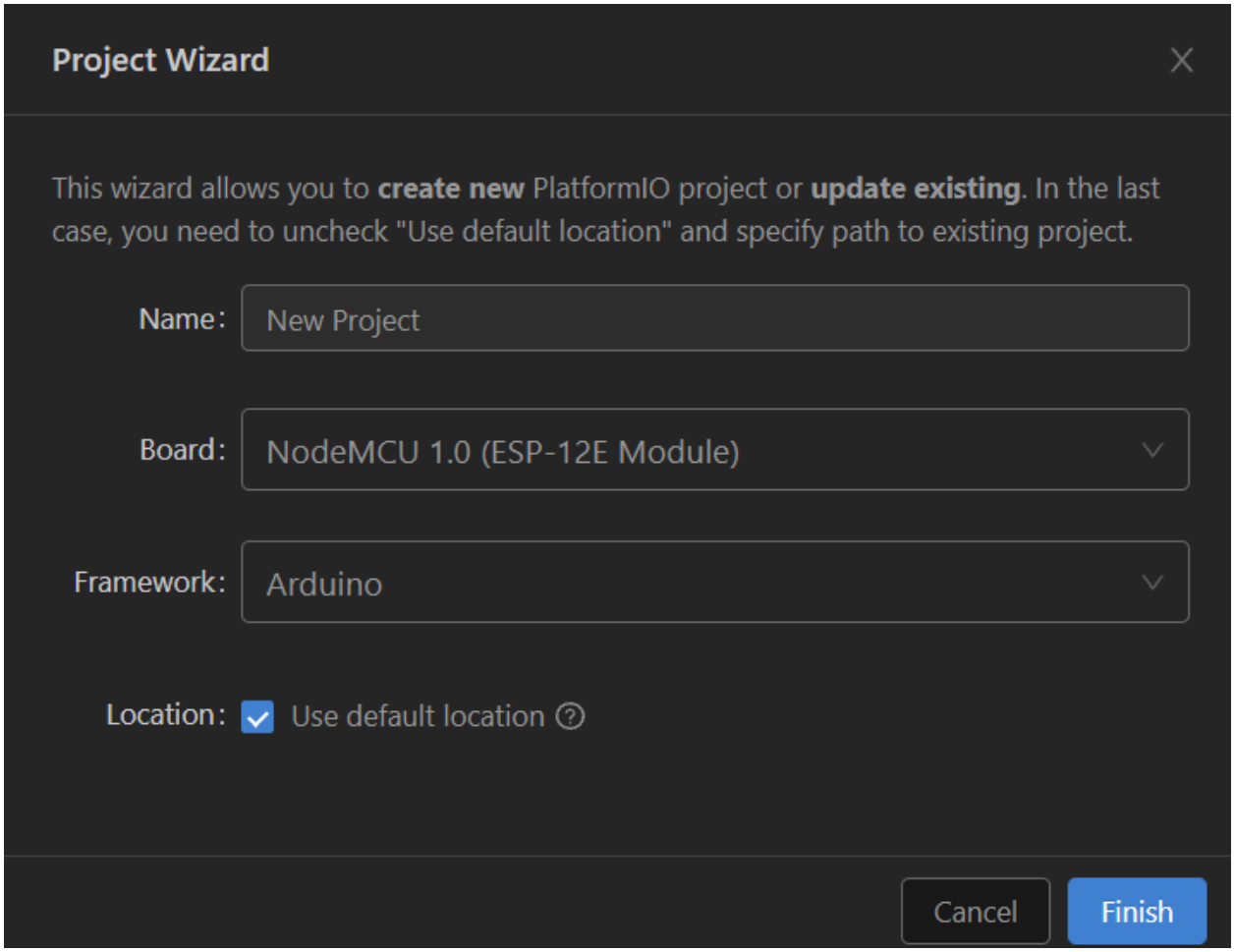


Рисунок 2.2.6 Налаштування Project Wizard для NodeMCU ESP8266

## 2.3 Огляд датчика температури та вологості повітря SHT20 та датчика відкриття дверей MC-38

### 2.3.1 Датчик температури та вологості повітря SHT20

Датчик температури та вологості повітря SHT20 (рис. 2.3.1) працює на базі чіпа CMOSens, має вбудований аналогово-цифровий перетворювач, підсилювач сигналу, OTP пам’ять і цифровий процесор. Підключення відбувається через послідовний інтерфейс $I^2C$.

Датчик підключений на ніжки GPIO14 та GPIO13 для передачі інформації, живлення забезпечується через ніжки GND та 3V3.

Підключення датчика може відбуватися не тільки на ніжки GPIO14 та GPIO13, використовувати можна фактично будь-які GPIO ніжки, потрібно лише вказати в коді які саме були використані.

Для зчитування показників з датчика, в програмному коді використовуються бібліотеки “<Wire.h>” та “DFRobot_SHT20.h”. В функції “setup()” ніжки GPIO13 та GPIO14 встановлюються на роботу по протоколу $I^2C$ командою “Wire.begin(D5, D6)”, також викликаються команди “sht20.initSHT20()” та “sht20.checkSHT20()”, де “sht20” попердньо було визначено як “DFRobot_SHT20”. Після цих команд мікроконтролер готовий в будь-який момент зчитувати показники температури та вологості повітря командами “sht20.readTemperature()” та “sht20.readHumidity()”.

Зчитування показників температури і вологості відбувається на кожній ітерації головного циклу мікроконтролера “loop()”. Перед новим зчитуванням, контролер запам’ятовує попередні показники. Якщо значення температури або вологості відрізняється  більше ніж на 0.1 від попереднього зчитування, мікроконтролер відправляє ці значення на сервер, інакше – не робить нічого. Такий механізм впроваждений для того, щоб не перевантажувати сервер та не займати канал інтернет з’єднання.

Також, перед тим, як зчитувати показники з датчика, мікроконтролер перевіряє чи є з’єднання з сервером. Це зроблено тому, що коли немає з’єднання з сервером і викликається функція “websocket.sendTXT()”, мікроконтролер відправить пакет даних на сервер і буде очікувати відповіді, що займе значний час, особливо якщо врахувати, що це відбуватиметься на кожній ітерації основного циклу програми.

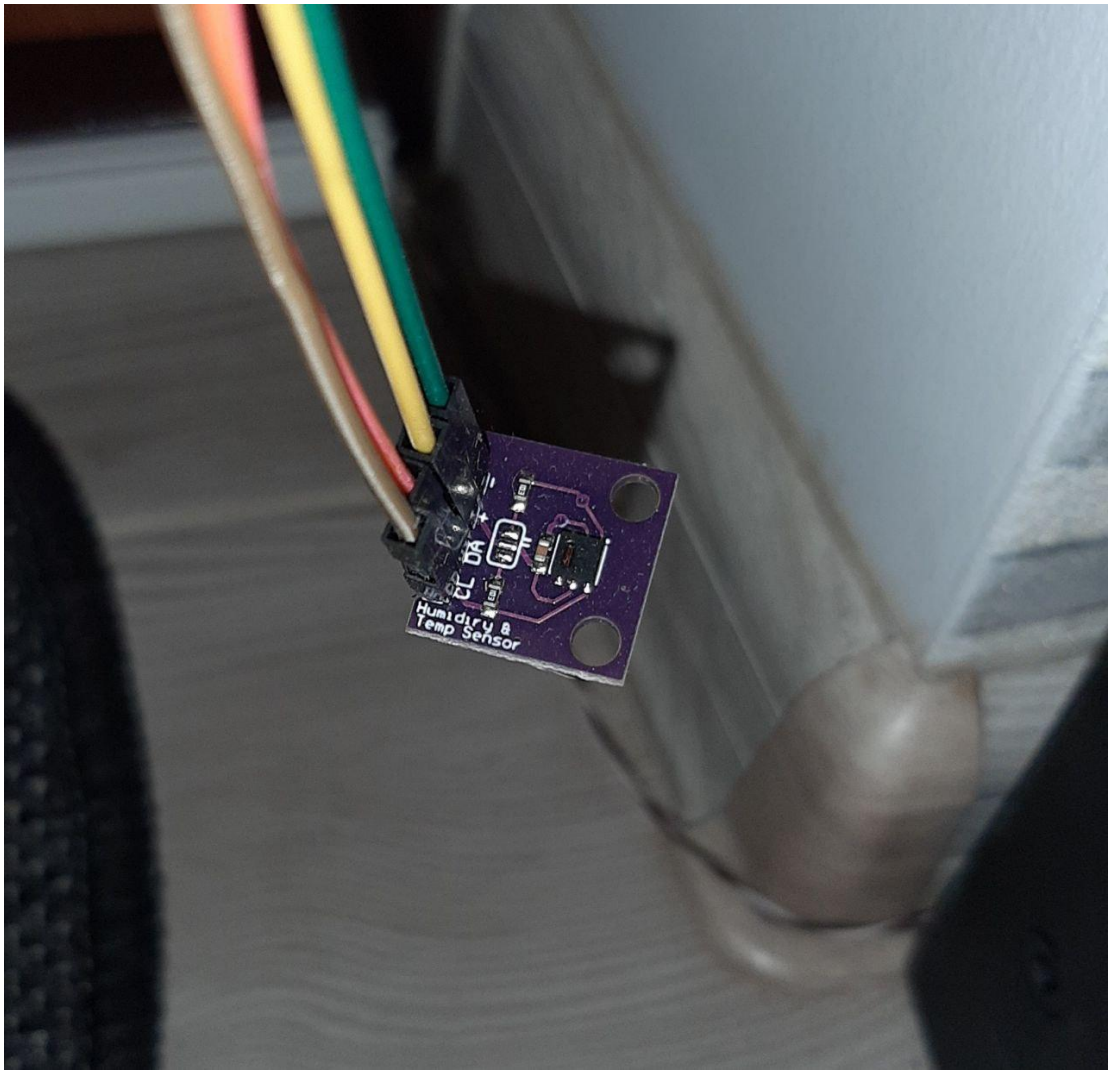


Рисунок 2.3.1 Датчик температури та вологості повітря SHT20

### 2.3.2 Датчик відкриття дверей MC-38

Датчик відкриття дверей MC-38 складається з геркону та магніту, на виході має два проводи (рис. 2.3.2). Підключення датчику відбується шляхом під’єднання одного провода до GND, іншого – до GPIO4. В коді, ніжка GPIO4 встановлюється в режим "INPUT PULLUP". “INPUT PULLUP” – це режим підтягуючого резистора, який забезпечує високий рівень сигналу, коли датчик пропускає через себе електричний струм і низький рівень сигналу, коли датчик не пропускає через себе струм. Також цей режим допомагає уникнути невизначеного стану на ніжці GPIO4, коли сигнал не являє собою ні логчний нуль, ні логічну одиницю. Встановлення режиму ніжки відбувається в функції

“setup()” командою “pinMode(GER, INPUT_PULLUP)”, де “GER” попередньо було визначено як “D2” командою “#define GER D2”.

Зчитування стану ніжки, до якої підключений датчик відбувається командою “digitalRead(GER)”. Як і у випадку з датчиком температури, мікроконтролер запам’ятовує попереднє зчитування з ніжки і надсилає на сервер інформацію, тільки якщо стан ніжки змінився відносно попереднього.

Перед надсиланням інформації про стан ніжки GPIO4, мікроконтролер перевіряє чи з’єднаний він з сервером. Якщо з’єднання немає, тоді мікроконтролер ставить відправку повідомлення у чергу до того часу, поки не з’явиться з’єднання.

Рисунок 2.3.2 Датчик відкриття дверей MC-38

## 2.4 Віддалене керування мікроконтролером. Обробка контролюючих команд

### 2.4.1 Обробка команд, які надходядь з сервера

Кожен раз коли мікрокотролер отримує повідомлення через вебсокет, запускається функція “messageHandler”, в які розписані всі команди та яку дію потрібно виконати при їх отриманні.

### 2.4.2 Керування вбудованим світлодіодом

Вбудований в плату NodeMCU ESP8266 світлодіод підключений до ніжки GPIO16, тому встановлюючи високий або низький сигнал на цій ніжці, можна його вмикати або вимикати. Для того щоб встановити ніжку в режим керування, використовується команда “pinMode(LED, OUTPUT)”, де “LED” попередньо був визначений як “D0”.

Керування світлодіодом може бути корисним для того щоб зрозуміти чи працює мікрокотролер і чи приймає команди.

Також ввімкнення/вимкнення світлодіода відбувається на подію відкриття/закриття дверей відповідно, що дає зрозуміти чи спрацював датчик.

### 2.4.3 Керування реле

Підключення реле відбувається на ніжку GPIO5, яка встановлюється на режим керування командою “pinMode(LED, OUTPUT)”. Живлення реле відбувається через ніжки Vin та GND. Реле використовується у нормально закритому режимі, при надходжені команди на керування реле, ніжка GPIO5 встановлюється у стан, зазначений в команді.

Через реле можна підключити будь-який пристрій, який захоче користувач, наприклад, світлодіодну ленту.

### 2.4.4 Команда перезавантаження мікрокотролера

При надходжені команди на перезавантаження мікроконтролера, виконується команда “ESP.reset()”, після якої мікроконтролер повністю перезавнтажується.

## 2.5 Підключення NodeMCU ESP8266 до Wi-Fi

Для підключення мікроконтролера до Wi-Fi мережі, використовується бібліотека “<ESP8266WiFi.h>”. На початку коду, в змінних “ssid” та “password” вказуються відповідно SSID та пароль мережі, далі в “setup()” викликається “WiFi.begin()”, в яку параметрами передаються SSID та пароль, після чого запускається нескінченний цикл до того часу, поки “WiFi.status()” не стане “WL_CONNECTED”. В середині циклу відбувається блимання світлодіодом, щоб дати зрозуміти користувачу, що мікроконтролер знаходиться у стані підключення до мережі.

## 2.6 Реалізація клієнта вебсокета на NodeMCU ESP8266

Для реалізації клієнта вебсокета використовується бібліотека “<WebSocketsClient.h>”. Для зручності, на початку коду “WebSocketsClient” визначається як “webSocket” командою “WebSocketsClient webSocket;”. В коді має бути визначена функція, яка відповідає за обробку подій, пов’язаних з вебсокетом. В ній визначається що повинно відбуватися при встановлені з’єднання з сервером, при розриві з’єднання з сервером, при отримані текстового повідомлення, при отримані бінарного повідомлення, при отриманні команд “PING” та “PONG”. У функції “setup()” відбувається запуск клієнта, шляхом виконання команди “webSocket.begin();”, в яку параметрами передаються адреса сервера, порт та URL. Ідентифікатор пристрою та токен

аутентифікації передаються в команду “webSocket.setExtraHeaders();” перед викликом “webSocket.begin();”. Ця команда додає заголовок до всіх повідомлень, які будуть відправлені з цього мікрокотролера. Також викликається команда “webSocket.onEvent();”, в яку передається функція, яка обробляє події вебсокета.

В рамках цієї роботи, єдина подія, яка потребує додаткової обробки є отримання текстового повідомлення (“WStype_TEXT”). З сервера може прийти або команда керування, яка обробляється в функції “messageHandler”, або запит “getState”, який запитує у мікроконтролера про стан реле (відкрите/закрите) та стан світлодіода (ввімкнений/вимкнений). Відповідно, в середині обробника події “WStype_TEXT” відбувається або виклик функції “sendState”, яка відповідає на запит “getState”, або виклик “messageHandler”, в залежності від того, яке повідомлення прийшло.

Для того, щоб мікроконтролеру зрозуміти чи є з’єднання з сервером, він обмінюється з сервером “PING” та “PONG” пакетами. За замовченням мікроконтролер не відправляє “PING” пакети на сервер. Для того щоб налаштувати “пінгування” серверу, використовується команда “webSocket.enableHeartbeat(1000, 1000, 1);”. Ця команда налаштовує надсилання “PING” пакетів кожну секунду та очікування відповіді протягом однієї секунди, без перенадсилання пакету, якщо відповідь не була отримана. Інтервал, через який мікроконтролер спробує перепідключитися до сервера, встановлюється в 10 секунд командою “webSocket.setReconnectInterval(10000);”. Такий інтервал був обраний тому, що мікроконтролер намагається перепідключитися до сервера приблизно протягом чотирьох секунд. Якщо сервер буде вимкнений на довгий час, то спроби перепідключення будуть відбуватися на кожній ітерації головного циклу програми, що майже не залишить часу на фіксування відкриття/закриття дверей.

## 2.7 Отримання точної дати та часу за допомогою протоколу NTP

NTP (Network Time Protocol) – протокол для синхронізації дати та часу пристроїв в мережі. В даній роботі, цей протокол використовується для збереження точного часу подій відкриття/закриття дверей та подій перезавантаження мікроконтролера, навіть коли сервер вимкнений.

В якості NTP сервера використовується ресурс pool.ntp.org. Для реалізації NTP клієнта на стороні мікрокотнролера використовується бібліотека “<NTPClient.h>”.

Кожен раз, перед тим, як відправити на сервер подію відкриття/закриття дверей або перезавантаження, викликається функція “dateTime()”, яка отримує точні дату та час з NTP сервера та перетворює їх на строку, яка додається до інформації про подію. Якщо з будь-яких причин мікроконтролеру не вдалося отримати дату та час з NTP сервера, то він відправить за замовчуванням дату 1970-01-01 та час 00:00:00, при отримані яких, сервер замінить їх на дату та час, в який він отримав цей пакет.

## 2.8 Мінімізація розмірів пакетів даних, які надсилаються з мікроконтролера на сервер, шляхом кодування слів в цифри

Всі повідомлення з мікроконтролера на сервер надсилаються у вигляді JSON. JSON – формат даних, який складається з пари ключ – значення та масивів, наприклад [{“key1”: “value1”, “key2”: “value2”}, {“key3”: “value3”, “key4”: “value4”}]. Так як сервер та мікроконтролер спілкуються між собою чітко визначеними повідомленнями, з метою економії інтернет трафіка, назви ключів та стандартні значення можна закодувати в цифри.

Всього визначено чотири типи повідомлень, які мікроконтролер може надіслати на сервер: “device”, “state”, “gerkon”, “temperature_and_humidity”, – вони закодовані в цифри 0, 1, 2 і 3 відповідно. Для того щоб сервер міг зрозуміти який тип повідомлення він опрацьовує, першим ключем в JSON

передається саме тип повідомлення. Ключ має назву “type” та кодується як 0. Тобто якщо мікроконтролер передає тип повідомлення “device”, то першою парою ключа і значення в JSON буде “0:0”.

Наступними ключами йдуть “date” та “time”, які передають дату та час повідомлення. Вони кодуються як 2 та 3 відповідно, та використовуються тільки в повідомленнях типу “device” та “gerkon”. Безпосередні значення дати та часу закодувати неможливо, тому вони передаються без кодування. Наприклад, якщо мікроконтролер передає тип повідомлення “device”, дату “2023-05-05” та час “18:00:00”, JSON буде виглядати наступним чином «{“0”:”0”, “2”: “2023-05-05”, “3”: “18:00:00”}».

Ключі “action” та “actionType” використовуються в типі повідомлення “device” та кодуються цифрами 4 і 5 відповідно. Ключ “action” передає повний текст події, яка відбулася на мікроконтролері, наприклад “Device rebooted, reason: Power On”. Ключ “actionType” може набувати двох значень: “INFO” та “ERROR”. Ці значення кодуються цифрами 0 та 1 відповідно та сигналізують про тип події – інформаційна або помилка в роботі мікроконтролера. Отже, для передачі повідомлення про перезавантаження мікроконтролера, через те, що зникло живлення, а потім з’явилося знову, повний JSON буде виглядати наступним чином «{“0”:”0”, “2”: “2023-05-05”, “3”: “18:00:00”, “4”: “Device rebooted, reason: Power On”, “5”: “INFO”}».

Ключі “led” та “relay” кодуються цифрами 6 та 7 відповідно і використовуються щоб надати відповідь на запит з серверу “getState” повідомленням типу “state”. Значення цих ключів можуть набувати або 0 або 1, в залежності від стану ніжки реле та світлодіода. Повідомлення типу “state” не потребує надсилання точної дати та часу. Повне JSON повідомлення цього типу буде виглядати як «{“0”: “1”, “6”: “0”, “7”: “1”}».

Ключ “gerkonState” кодується цифрою 8 та позначає стан геркона – 0 або 1. Повне JSON повідомлення типу “gerkon” буде виглядати як «{“0”:”0”, “2”: “2023-05-05”, “3”: “18:00:00”, “8”: “1”}».

Ключі “temperature” та “humidity” кодуються цифрами 9 та 10 відповідно та позначають температуру та вологість повітря, отримані з датчика SHT20. Повне JSON повідомлення типу “temperature_and_humidity” буде виглядати наступним чином «{“0”: “3”, “9”: “21.4”, “10”: “65.41”}».

## 2.9 Відкладена відправка повідомлень на сервер, коли відсутнє з’єднання

Постійна перевірка з’єднання з сервером, описана в пункті 2.6, дає змогу відкласти відправку повідомлення на сервер, якщо з’єднання відсутнє. Перед кожною відправкою повідомлення, мікроконтролер перевіряє чи є з’єднання з сервером, якщо немає, тоді це повідомлення додається в масив відкладених повідомлень. Масив відкладених повідомлень може містити в собі максимум 200 повідомлень. Якщо цей масив повністю заповнений, то всі наступні повідомлення будуть втрачені.

В основному циклі мікроконтролера викликається функція “handleDelayedMessages()”, яка перевіряє чи є з’єднання з сервером, якщо так, тоді вона відправляє повідомлення по черзі з масиву.

## 2.10 Сторожовий таймер

Зі збільшенням розміру і функціональності системи, зростає шанс повного зависання мікроконтролера. Так як система планується бути автономною та віддаленою, повинен бути забезпечений механізм, який перезавантажить мікроконтролер у такому випадку. Для цієї цілі ESP8266 має вбудований сторожовий таймер.

Сторожовий таймер – апаратна схема контролю за зависанням системи. Сторожовий таймер, як можна зрозуміти з назви, містить в собі таймер, який запускається заново, коли на нього приходить сигнал з мікроконтролеру. Якщо таймер не був перезапущений вчасно, сторожовий таймер перезавантажує мікроконтролер.

ESP8266 має апаратний і логічний сторожові таймери. Логічний сторожовий таймер допомогає, наприклад, коли програма зайшла в нескінченний цикл, в той час як апартний допомагає, при повному зависанні системи.

Сторожовий таймер потребує щоб на нього надсилали періодично сигнали. Це можна зробити використавши команду “ESP.wdtFeed()”, вона оновлює таймер і на апаратному і на логічному сторожовому таймері. Проте в більшості випадків, ця команда не знадобиться, хіба що в коді реалізовані великі складні частини, які займають значний час. Сторожові таймери автоматично оновлюються на початку основного циклу програми “loop()”, а також при використанні “delay()” та інших команд.

## 2.11 Основний цикл програми

Повний лістинг коду на NodeMCU ESP8266 наведений в Додатку А. Основний цикл програми містить в собі наступні елементи:

- “webSocket.loop()” відповідає за обробку всього, що пов’язано з вебсокетами: пінг-понг з сервером, прослуховування команд з серверу та інше.
- “checkGetrkon()” здійснює перевірку стану датчика відкриття дверей МС-38 та відправку інформації на сервер, якщо стан змінився.
- “handleSHT20()” відповідає за зчитування показників температури та вологості повітря та відправку інформації на сервер, якщо один з показників змінився більше ніж на 0.1 одиницю.
- “handleDelayedMessages()” здійснює відправку всіх відкладених, через відсутність з’єднання з сервером повідомлень.
- “ArduinoOTA.handle()” дозволяє завантажувати новий код через WiFi.
- “delay(100)” – затримка 100 мс.

## 2.12 Підключення та розміщення елементів системи

Підключення та розміщення елементів системи зображені на рис. 2.12.1. Датчик МС-38 розміщений на нижній частині рамки вхідної двері, з лівого боку, так, щоб його не могли випадково зачіпити ногою. Два проводи на виході з датчику були припаєні до двох інших проводів для подовження, на іншому кінці подовжуючі проводи припаєні до проводів з роз'ємом «мама» (рис. 2.12.2) для підключення їх безпосередньо до ніжок мікроконтролера. Ніжки датчика SHT20 підключені проводами «мама»-«мама» (рис. 2.12.3) до ніжок мікроконтролера. Реле підключено проводами «тато»-«мама» (рис. 2.12.4). Плата NodeMCU живиться через USB роз'єм і блок живлення 5В з розетки 220В.

Рисунок 2.11.1 Підключення та розміщення елементів системи

Рисунок 2.11.2 Роз’єм «мама»

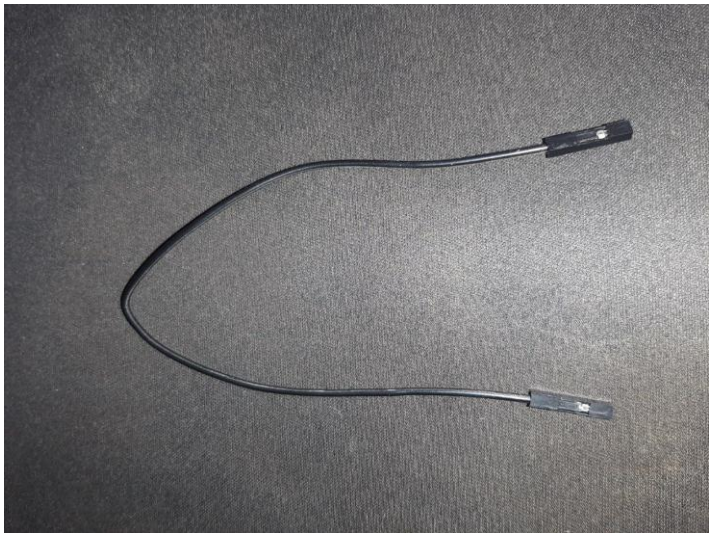

Рисунок 2.11.3 Провід «мама»-«мама»

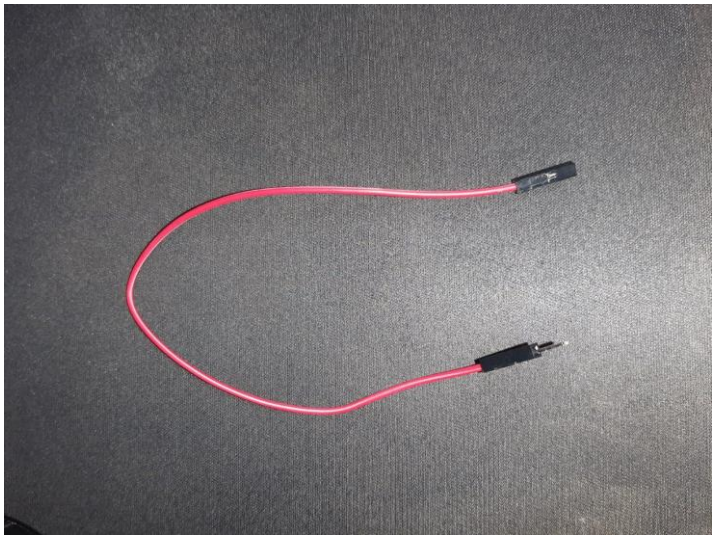

Рисунок 2.11.4 Провід «тато»-«мама»

# 3. ДЕТАЛЬНИЙ ОГЛЯД РЕАЛІЗАЦІЇ СЕРВЕРНОЇ ЧАСТИНИ

## 3.1 Мова програмування Python. Фреймворк Django. Django Channels.

Для реалізації серверної частини була обрана мова програмування Python. Станом на 2022 рік, Python займає друге місце в списку найпопулярніших мов програмування на платформі GitHub, поступаючись тільки JavaScript. Динамічна типизація та зручний синтаксис полегшують та прискорюють процес розробки, а широка спільнота забезпечує велику кількість інформації в мережі інтернет та готові рішення до багатьох задач.

Вебсервер написаний за допомогою фреймворку Django. Django побудований на MVT (model-view-template) архітектурі, яка складається з трьох компонентів:

- Model – абстрактний рівень структуризації та маніпуляції данними. Цей рівень також відповідає за взаємодію з базою даних. В Django використовується Django ORM для взаємодії з базою даних.
- View відповідає за обробку запитів та надсилання відповідей клієнту. Django надає можливість вибору підходу до побудови views. Можна використовувати функціональний підхід, де кожна view – це функція, в якій прописується логіка обробки запитів та відповіді на них, або використовувати класовий підхід, де кожна view – це клас з певними атрибутами та методами.
- Template – рівень, який відповідає за генерування потрібних даних на сторінці. В Django використовується Django Template Language, який дозволяє передавати змінні з views, використовувати цикли, умови та інше для генерації html сторінки для користувача.

Django Channels – розширення до фреймворку Django, яке дозволяє

окрім HTTP протоколу також використовувати інші протокли, наприклад вебсокети.

### 3.2 Реалізація HTTP сервера

Як було зазначено в попередньому пункті, Django використовує MVT архітектуру, отже весь програмний код можна розбити на три основні частини: Models, Views та Templates.

Лістинг файлу models.py наведений в Додатку Б. Всього створено 3 моделі: ESPDevice, DoorLog та DeviceLog. ESPDevice містить в собі інформацію про власника мікроконтролера, його унікальний ідентифікатор, токен, дату та час внесення в систему, останній час, коли мікрокотнролер був в мережі та чи є в мережі зараз. DoorLog потрібен для ведення журналу відкриття дверей та містить в собі інформацію про дату та час події, стан дверей – відкриті/закриті та унікальний ідентифікатор мікроконтролера, з якого прийшла інформація. DeviceLog має схожу структуру з DoorLog, він містить в собі дату та час події, унікальний ідентифікатор пристрою, з якого прийшла інформація, тип повідомлення – INFO/ERROR та текст повідомлення.

Лістинг файлу views.py наведений в Додатку В. Всього створено 6 views: LoginPageView, IndexPageView, DoorLogs, LogsPageView, ControlDevicePageView та SelectDeviceView. Кожен view відповідає за надання користувачу відповідну сторінку у веб застосунку. Кожен view, окрім LoginPageView вимагає від користувача бути залогіненим в систему, інакше він буде перенаправлений на сторінку логіна. Кожен view, окім SelectDeviceView та LoginPageView вимагає, щоб в сесії користувача був вказаний унікальний ідентифікатор мікроконтролера з яким він хоче працювати, інакше він буде перенаправлений на SelectDeviceView. Маршрутизація запиту користувача на потрібний view відбувається у файлі urls.py, який наведений в Додатку К. LoginPageView відповідає за надання користувачеві сторінки логіна та сам логін користувача в систему. Після успішного логіна, користувач

перенаправляється на наступний view – SelectDeviceView, який відповідає за вибір користувачем мікроконтролера, з яким він хоче працювати та зберігає цю інформацію в сессію користувача, але тільки при умові, що обраний користувачем мікроконтролер існує в базі даних. Після чого користувач перенаправляється на IndexPageView, який відповідає за надання користувачу Dashboard сторінки. DoorLogs та LogsPageView надають користувачу сторінки журналів відкриття дверей та мікроконтролера відповідно. ControlDevicePageView надає користувачу сторінку для керування мікроконтролером.

Лістинг файлів templates наведений в додатках Г, Ґ, Д, Е, Є, Ж, З. Всього є 7 шаблонів. Шаблонів на 1 більше ніж views тому, що є основний base.html шаблон, від якого наслідуються всі інші. Цей шаблон містить в собі верхню навігаційну панель та посилання на Bootstrap 5 стилі та скрипти. Інші 6 шаблонів відповідають сторінкам, докладніше про які – в розділі 4.

## 3.3 Реалізація вебсокет сервера

Серверна частина вебсокетів реалізована за допомогою Django Channels. Основа логіка написана в файлі consumers.py (Додаток І), маршрутизацією запитів займається файл routing.py (Додаток Ї).

### 3.3.1 Робочі простори

Всього в системі реалізовано 6 різних клієнтів вебсокета, відповідно для спілкування з кожним з них повинна бути реалізована логіка на стороні сервера. Для досягнення цієї мети, були введені робочі простори (workspaces). Всього є 6 робочих просторів: “esp”, “doorlog”, “devicelog”, “device_control”, “dashboard” та “status_select”, – вони кодуються в URL адресу запиту. URL адреса, на яку сервер приймає запити складається з двох частин: загальної адреси вебсокета “ws/main/” та назви робочого простору. Наприклад клієнт

вебсокета в мікроконтролері підключається на адресу “*адреса сервера*/ws/main/esp”.

### 3.3.2 Групи

Кожному підключеному клієнту надається унікальний ідентифікатор та присвоюється група. Група потрібна для того, щоб можна було відправляти повідомлення всім учасникам групи. Наприклад існує група для сторінки з журналом події, до якої підключено 4 клієнти, коли з’являється нова подія, то вона відправляється всім учасникам групи, тому всі 4 клієнти зможуть отримати актуальну інформацію. Група складається з двох частин: назви робочого простору та унікального ідентифікатору мікроконтролера. Унікальний ідентифікатор мікроконтролера додається до назви групи для того, щоб користувач який працює з одним мікроконтролером не отримав повідомлення, які стосуються іншого мікроконтролера. Присвоєння групи клієнту відбувається наступним чином: клієнт надсилає запит, наприклад, на “ws/main/doorlog”, із інформації про сесію дістається унікальний ідентифікатор мікроконтролера, з яким працює користувач, наприклад, “111111”, клієнту присвоюється група з ім’ям “doorlog_111111”. У випадку з клієнтом вебсокета мікроконтролера, унікальний ідентифікатор дістається з заголовку запиту. Єдиним робочим простором, якому в назву групи не присвоюється унікальний ідентифікатор мікроконтролера, є status_select.

### 3.3.3 Робочий простір “esp”

Робочий простір “esp” відповідає за спілкування з мікроконтролером. Коли мікроконтролер надсилає запит на з’єднання, на сервері відбувається перевірка унікального ідентифікатору та токену мікроконтролера. Якщо запис з таким токеном і унікальним ідентифікаторм існує в базі даних, сервер приймає з’єднання та встановлює в базі даних статус мікроконтролера “в мережі”. Коли

відбувається розривання з’єднання, сервер вставновлює статус мікроконтролера “не в мережі” та запам’ятовує час коли відбувся розрив з’єднання. При кожному запуску сервера (якщо він був вимкнений), всім мікроконтролерам автоматично встановлюються статуси “не в мережі”. Це відбувається у файлі apps.py (Додаток Й). При кожному новому з’єднанні або розриві з’єднання сервер надсилає повідомлення всім клієнтам в групі “dashboard_*ідентифікатор мікроконтролера*” про те що мікроконтролер з’явився в мережі або зник з мережі.

Після встановлення успішного з’єднання та призначення мікроконтролеру групи, сервер очікує від нього повідомлення. Коли приходить повідомлення про подію відкриття/закриття дверей, сервер записує цю події в базу даних і відразу надсилає її всім клієнтам в групі “doorlog_*ідентифікатор мікроконтролера*”. Те ж саме відбувається і з подіями перезавантаження мікроконтролера, після отримання повідомлення, сервер зберігає його в базу даних та надсилає всім учасникам групи “devicelog_*ідентифікатор мікроконтролера*”.

Ще одним типом повідомлення, яке сервер чекає від мікроконтролера, є показники датчика температури та вологості повітря. При отримані такого повідомлення, сервер перенаправляє його всім клієнтам у групі “dashboard_*ідентифікатор мікроконтролера*”.

Останній тип комунікації між сервером і мікроконтролером – це запит від сервера “getState”. При відправці такого запиту, сервер очікує від мікроконтролера повідомлення типу “state” та відразу перенаправляє його всім клієнтам в групі “device_control _*ідентифікатор мікроконтролера*”.

Також варто зазначити, що перед тим як обробляти будь-яке повідомлення від мікроконтролера, сервер спочатку перевіряє ідентифікатор та токен в заголовках повідомлення. Якщо вони не співпадуть з жодним записом у базі даних, сервер закриє з’єднання з цим клієнтом.

### 3.3.4 Робочий простір “dashboard”

Робочий простір “dashboard” відповідає за надання користувачу такої інформації, як температура та вологість повітря, статус мікроконтролера (в мережі або ні) та стан дверей (відкриті або закриті).

Як було зазначено в попередньому пункті, сервер перенаправляє інформацію з датчика температури та вологсті повітря в групу “dashboard_*ідентифікатор мікроконтролера*”. Передача статусу мікроконтролера, також була описана в попередньому пункті – при кожному встановленні чи розриванні з’єднання з мікроконтролером сервер надсилає інформацію у відповідну групу. Також сервер надсилає поточний стан мікроконтролера при встановлені першого з’єднання з клієнтом робочого простору “dashboard”. Стан дверей передається при першому встановлені з’єднання та при кожній зміні стану дверей. Інформація про стан дверей береться із останнього запису в базі даних, де зберігаються події відкриття/закриття дверей.

### 3.3.5 Робочі простіри “doorlog” та “devicelog”

Робочі простори “doorlog” та “devicelog” майже ідентичні. В обох випадках інформація з бази даних надсилається при першому встановленні з’єднання та, як було описано у пункті 3.3.3, при надходжені нової події з мікроконтролера. Також в обох робочих просторах є можливість встановлення проміжку часу, за який користувач хоче отримати інформацію з журналу. Для цього користувач може за допомогою інтерфейсу встановити на сторінці проміжок часу та надіслати його на сервер, натиснувши відповідну кнопку (докладніше про інтерфейс користувача в розділі 4), після чого ця інформація буде збережена в сесії користувача. Події, які користувач буде отримувати через вебсокет, будуть фільтруватися по цьому проміжку часу.

### 3.3.6 Робочий простір “device_control”

Робочий простір “device_control” відповідає за перенаправлення команд з інтерфейсу користувача на мікроконтролер, а також отримання з мікроконтролера інформації про поточний стан реле та світлодіода. Як було зазначено в пункті 3.3.3, сервер надсилає запит “getState” відповідному мікроконтролеру і перенаправляє відповідь відповідній групі робочого простору “device_control”. Запит “getState” надсилається один раз, при першому підключені клієнта “device_control” до сервера.

Команди отримані від клієнта “device_control” просто перенаправляються на відповідний мікроконтролер.

### 3.3.7 Робочий простір “status_select”

Робочий простір “status_select” має одну функцію – відобразити статуси мікроконтролерів, які доступні користувачу на сторінці вибору мікроконтролера.

### 3.3.8 Аутентифікація клієнта

З точки зору аутентифікації, в цій системі всього можливі два типи клієнта: мікроконтролер та користувач. Аутентифікація мікроконтролера описана в пункті 3.3.3 та в пункті 2.6 – через токен та унікальний ідентифікатор. Аутентифікацією користувача займається Django Authentication Middleware. З точки зору цієї роботи нас цікавить аутентифікація користувача через унікальний ідентифікатор сесії. Керуванням сесіями займає Django Session Middleware. Коли приходить запит на з’єднання через вебсокет, перевіряється чи є на сервері збережений “_auth_user_id”, якщо такий існує, то з’єднання приймаєтсья, якщо ні – то відхиляється. Той самий процес перевірки відбувається під час отримання повідомлень з клієнтів.

## 3.4 Налаштування Django. ASGI. Розгортання серверу за допомогою Nginx, uvicorn та gunicorn.

### 3.4.1 Налаштування Django

Налаштування Django відбувається у файлі settings.py, який наведений у додатку Л. Там вказуються додатки, які використовуються, часовий пояс, розташування статичних файлів та багато іншого. Загалом в цьому файлі визначаються глобальні змінні, які використовують ті чи інші компоненти Django. В рамках даної роботи в налаштування “INSTALLED_APPS” був доданий додаток “main”. Також додані:

- “LOGIN_URL” – те, куди перенаправляється незалогінений користувач
- “STATIC_URL” та “STATIC_ROOT” – місце розташування статичних файлів (js, css, favicon)
- “ASGI_APPLICATION” – потрібно для вебсокетів
- “CHANNEL_LAYERS” – Reddis для зберігання груп та клієнтів вебсокетів
- “LOGGING” – для збереження логів сервера у файл

Варто відзначити, що в налаштуваннях встановлені “DEBUG = True”, “ALLOWED_HOSTS = [‘*’]”, “SECRET_KEY” – залишився стандартний. Так як система розгорнута в локальній мережі, ці налаштування можна лишити такими, як є, але при розгортанні цього сервера в публічній мережі ці налаштування повинні бути змінені з міркувань безпеки.

### 3.4.2 Налаштування ASGI

За замовчуванням Django сервер запускається через WSGI – інтерфейс для перенаправлення запитів на веб додаток або фрейморк, написаний мовою

python. Він є синхронним, що обмежує швидкість обробки запитів та їх кількість. Щоб обійти цю проблему зазвичай запускається декілька WSGI сутностей, щоб обробляти запити паралельно.

Той факт, що WSGI є синхронним не дозволяє використовувати його для вебсокетів, так як вебсокет потребує безперервного з'єднання, WSGI був би повністю занятий на обробку одного вебсокет підключення і не зміг би обробляти інші запити. З цієї причини використовується ASGI, який виконує ту саму функцію, що і WSGI, але є асинхронним.

Налаштування ASGI є досить простим та виконується у файлі asgi.py, який наведений у додатку М. По суті в цьому файлі лиш визначається, що всі вебсокет запити повинні бути перенаправлені на вебсокет сервер, а всі звичайні HTTP запити – на HTTP сервер.

### 3.4.3 Розгортання серверу за допомогою Nginx, uvicorn та gunicorn

Nginx – вебсервер, який також може бути використаний як проксі сервер для перенаправлення запитів на інший вебсервер. В даній роботі Nginx слугує як вебсервер для надання статичних файлів користувачу, а також як проксі сервер для перенаправлення всіх інших запитів на uvicorn сервер.

Nginx може бути встановлений на операційну систему Ubuntu шляхом виконання команди "sudo apt install nginx". Для додавання нового сайту в Nginx виконується команда "sudo nano /etc/nginx/sites-available/*назва сайту*", після чого відкривається редактор тексту nano, куди потрібно вписати налаштування сайту (Додаток Н), в якому вказується порт, розташування статичних файлів, проксіювання інших запитів на сервер uvicorn та інше. Налаштування потрібно зберегти, після чого створити symbolic link між цим сайтом в "sites-available" та цим же самим сайтом в "sites-enabled" командою "sudo ln -s /etc/nginx/sites-available/*назва сайту* /etc/nginx/sites-enabled". Після чого робимо перезавантаженя nginx командою "sudo systemctl restart nginx" і сервер повинен бути доступним на вказаному в налаштуваннях порту.

Uvicorn – ASGI сервер, gunicorn – WSGI сервер, вони запускаються разом командою “gunicorn *назва Django проекту*.asgi:application -w 4 -k uvicorn.workers.UvicornWorker”, але перед цим мають бути встановлені через pip в віртуальне середовище (venv) або глобальне (залежить від того як відбувається розгортання серверу). Теоретично на цьому можна закінчити розгортання сервера і при кожному запуску сервера вводити команду запуску uvicorn та gunicorn, єдине, що потрібно вказати порт на якому їх запускати, наприклад, 8000 та в додатку Н замість “proxy_pass http://unix:/run/gunicorn.sock” вказати “proxy_pass http://localhost:8000”.

В даній роботі замість звичайного TCP/IP сокета було прийнято рішення використовувати ICP сокет, або іншими словами Unix сокет. ICP сокет схожий за своєю суттю на TCP/IP сокет, різниця полягає в тому, що ICP сокет створений для комунікації всередині операційної системи. Він не займає жодного TCP/IP порта та є швидшим, тому що не потребує виконання TCP/IP протоколів, які потрібні для гарантування комунікації між різними пристроями в мережі, але не є такими важливими в межах однієї операційної системи. Загалом і використання TCP/IP сокета і використання ICP сокета є однаково прийнятними рішеннями для розгортання вебсерверів, на які Nginx буде перенаправляти запити.

Для створення Unix сокета, потрібно створити файл командою “sudo nano /etc/systemd/system/gunicorn.socket” та вписати туди код з додатку О. Після чого запустити командами “sudo systemctl start gunicorn.socket” та “sudo systemctl enable gunicorn.socket”.

І останнє, але не менш важливе – це створення сервісу, який буде займатися запуском ASGI та WSGI серверів. Для цього потрібно створити ще один файл командою “sudo nano /etc/systemd/system/gunicorn.service” та вставити туди код з додатку П. Далі перезапускаємо daemon, gunicorn та nginx командами “sudo systemctl daemon-reload”, “sudo systemctl restart gunicorn”, “sudo services nginx restart” і сервер має бути доступний за адресою і портом, вказаними в nginx.

# 4. ІНТЕРФЕЙС КОРИСТУВАЧА

Інтерфейс користувача включає в себе 6 сторінок: login, select device, dashboard, door’s logs, device’s logs, control device, – та верхньої панелі навігації, яка доступна на всіх сторінках. Верхня панель навігації прописана в файлі base.html (Додаток З). Зліва на ній відображено всі сторінки, на які можна перейти, справа – поточного користувача (якщо користувач вже зайшов під своїм аккаунтом), поточно обраний мікроконтролер (якщо такий вже був обраний користувачем) та кнопку виходу з аккаунту.

## 4.1 Сторінка Login

Сторінка складається з маленького вікна посередині сторінки, яке має два поля – для логіна і пароля, а також кнопку логін (рис. 4.1.1). Поля доступні для автозаповнення з менеджера паролів будь-якого браузера, поле для паролю ховає символа за крапочками. При вдалому введені логіна і пароля користувач перенаправляється на сторінку Select Device, при невдалому перенаправляється знову на сторінку логіна. Шаблон сторінки знаходиться у файлі login.html (Додаток Г). Для відправки введених даних використовується форма з CSRF токеном, який є вбудованим в Django і допомагає запобігти CSRF атакам.

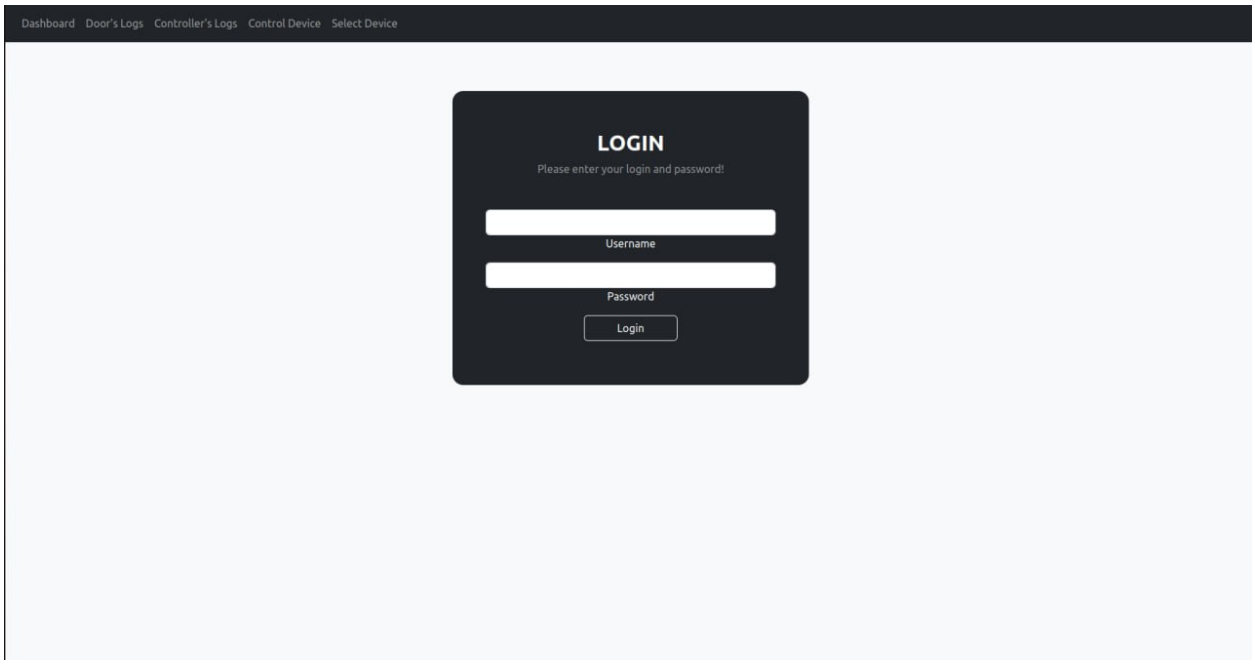


Рисунок 4.1.1 Пуста сторінка Login

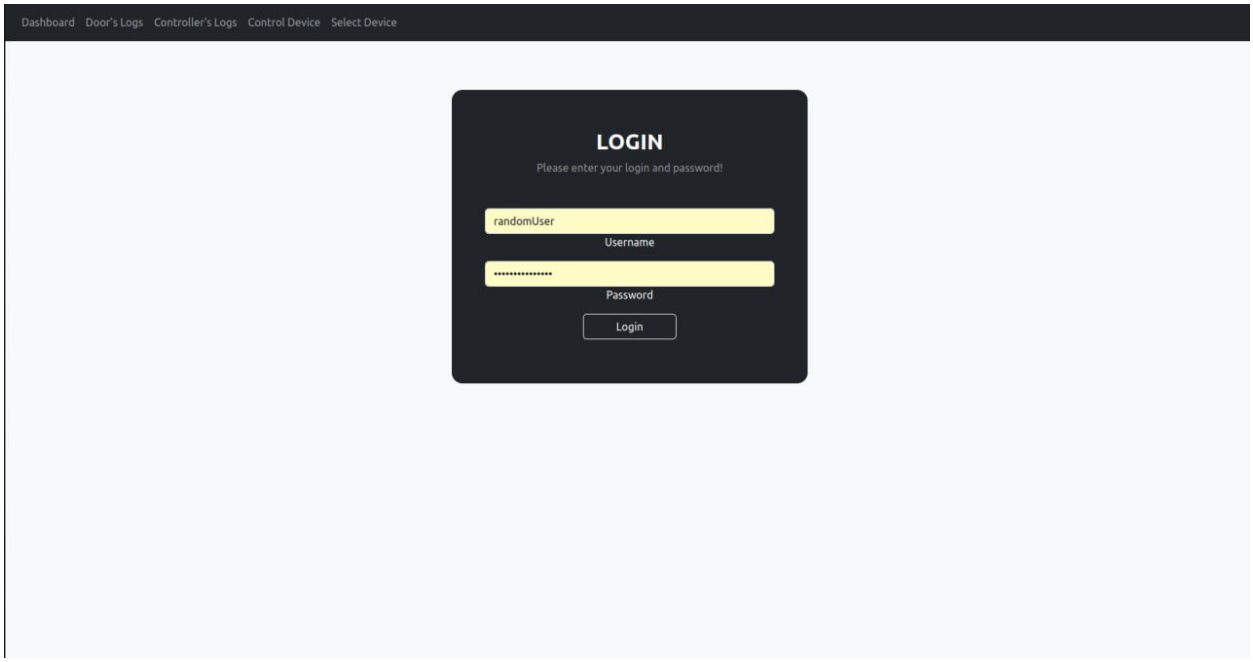


Рисунок 4.1.2 Сторінка Login з автозаповненим логіном і паролем

## 4.2 Сторінка Select Device

Сторінка Select Device слугує для того, щоб користувач міг обрати мікроконтролер, з яким хоче працювати. Вона відображає користувачу список мікроконтролерів, які йому доступні. Кожен елемент списку містить в собі унікальний ідентифікатор мікроконтролеру та його статсу – в мережі або не в мережі (рис. 4.2.1). Так як передача інформації про стан мікроконтролеру відбувається через вебсокет і можлива затримка при підключені до серверу, користувач бачить текст “Loading…” (рис. 4.2.2) до того часу, поки з сервера не прийде необхідна інформація. Відправка вибору користувача відбувається також через форму. Щоб обрати мікроконтролер, користувачу потрібно клікнути на нього в списку. Після того, як користувач обрав мікроконтролер, він буде перенаправлений на сторінку Dashboard. Шаблон сторінки знаходиться у файлі select_device.html (Додаток Ґ).

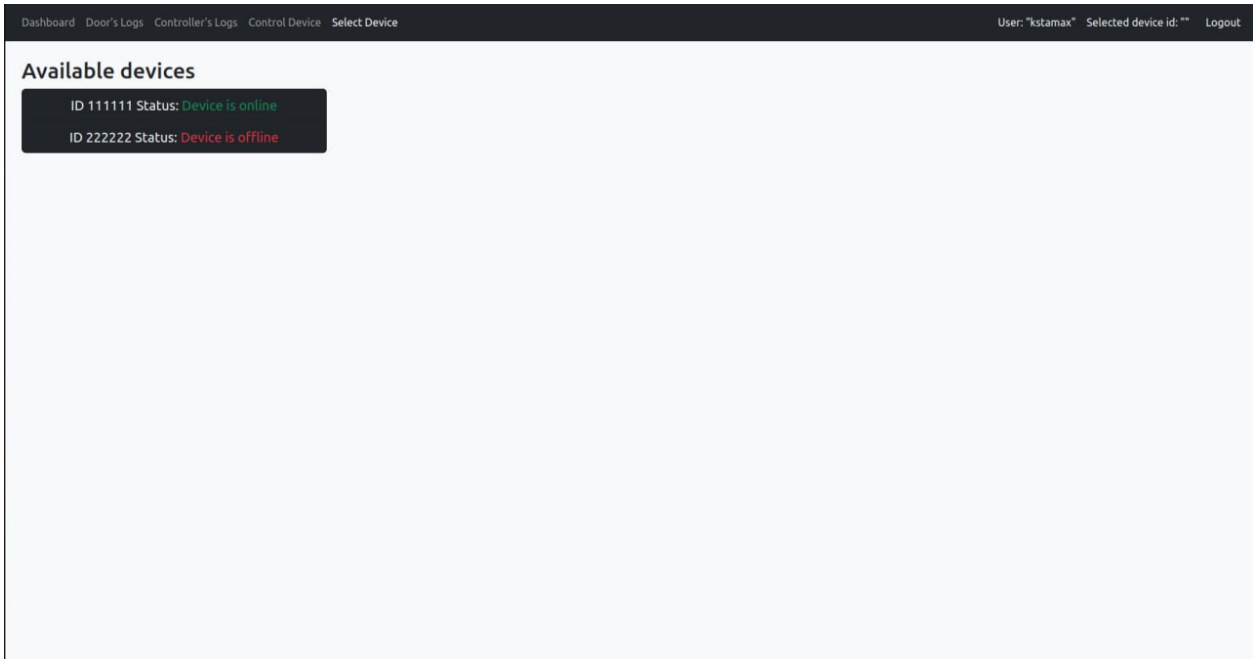


Рисунок 4.2.1 Сторінка Select Device

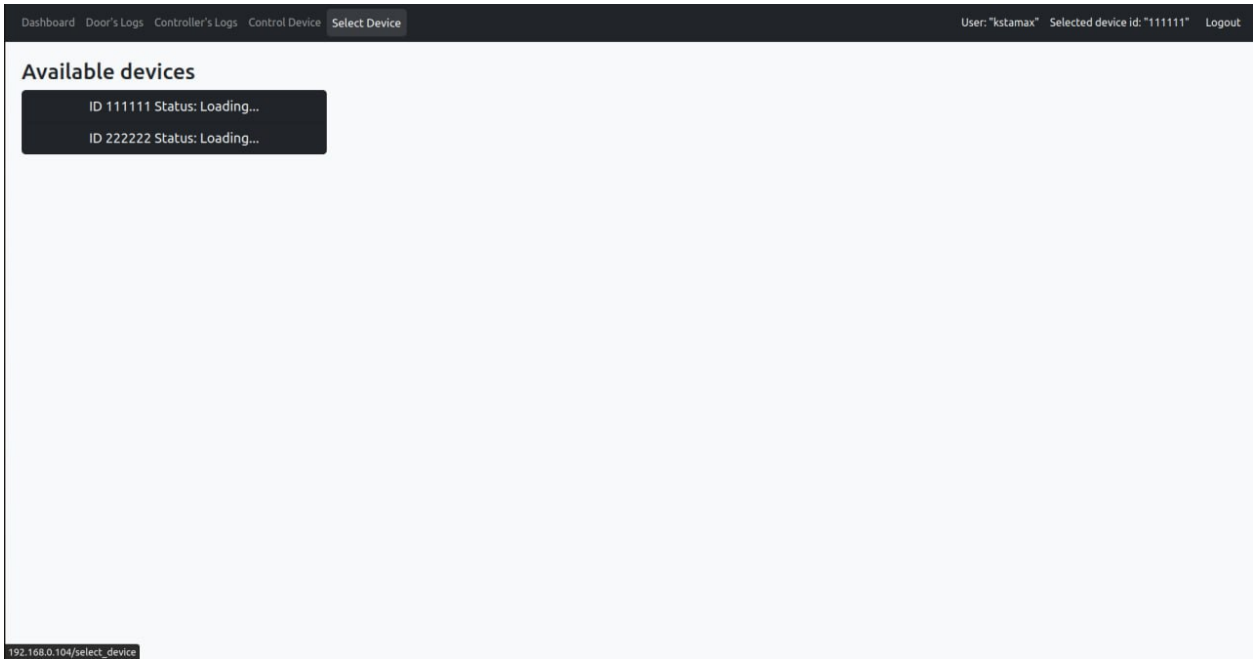


Рисунок 4.2.2 Сторінка Select Device до того, як відбулося підключення до вебсокета

## 4.3 Сторінка Dashboard

Сторінка Dashboard слугує як початкова сторінка, де користувач може подивитися чи є мікроконтролер в мережі, якщо ні, то коли останній раз був, відкриті чи закриті двері, температуру та вологість повітря в кімнаті (рис. 4.3.1). Вся ця інформація отримується через вебсокет, тому поки сторінка не отримала жодної інформації від сервера, користувач бачить текст “Loading..” (рис. 4.3.2). Шаблон сторінки знаходиться у файлі index.html (Додаток Д).

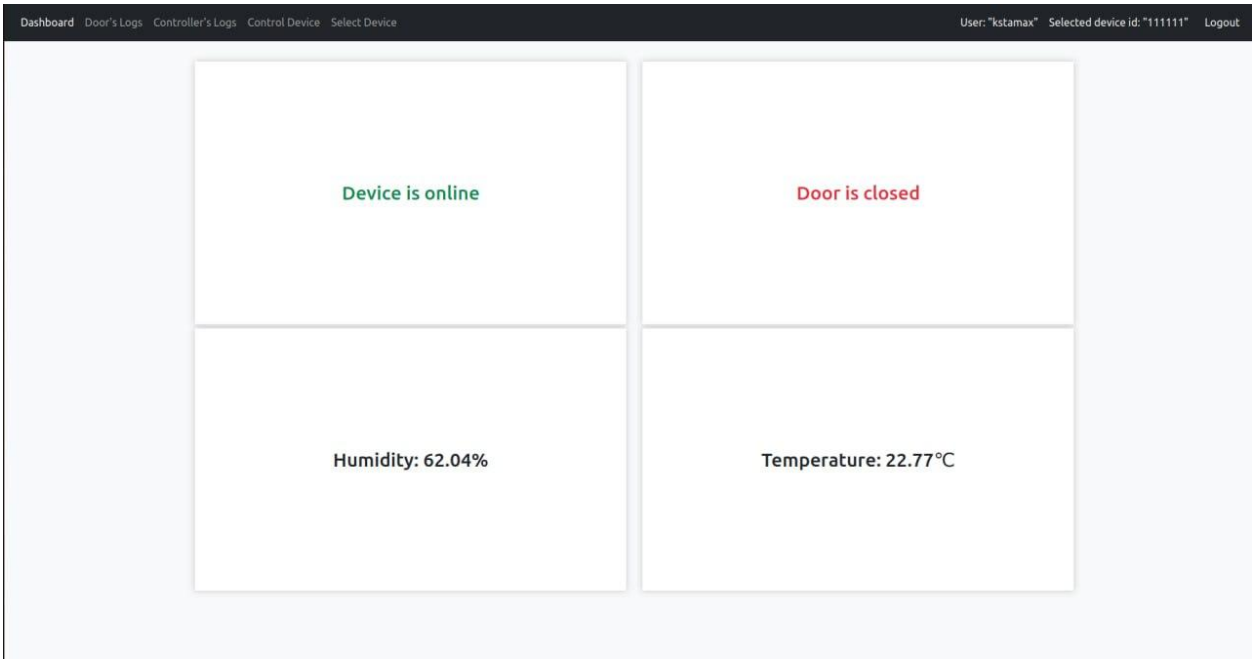


Рисунок 4.3.1 Сторінка Dashboard

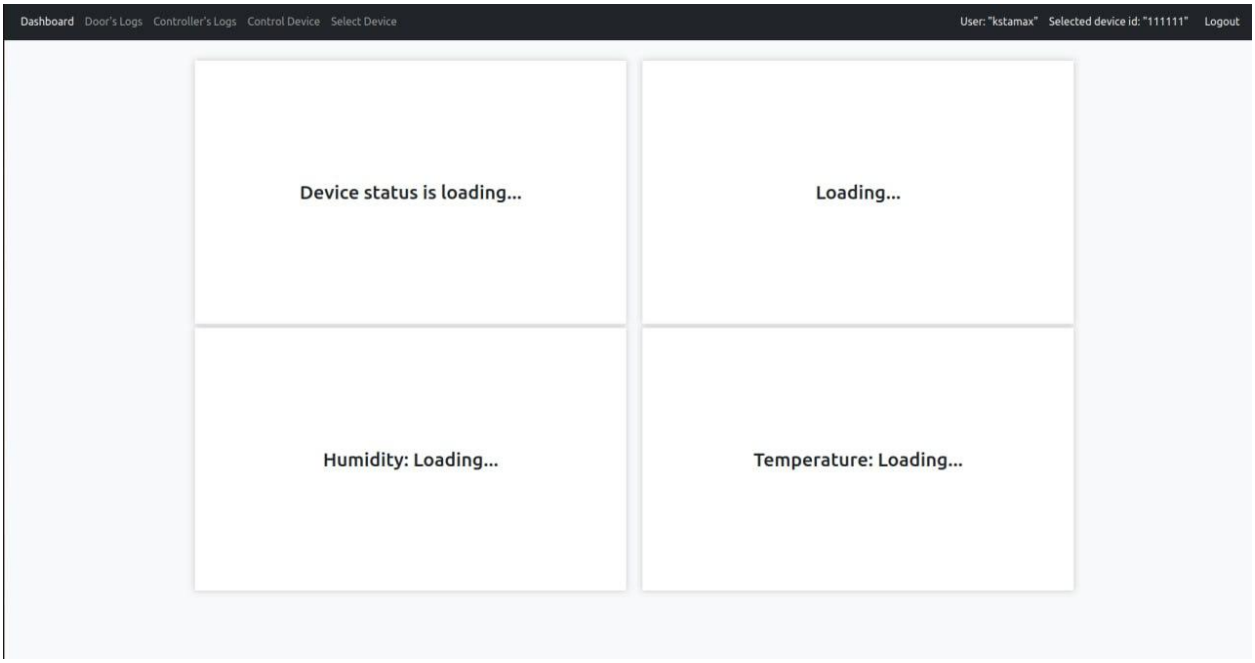


Рисунок 4.3.2 Сторінка Dashboard до встановлення з’єднання з вебсокетом

## 4.4 Сторінки Device’s Logs та Door’s Logs

Сторінки device’s logs (рис. 4.4.1) та door’s logs (рис. 4.4.2) майже ідентичні та використовуються для відображення користувачу подій перезавантаження мікроконтролера та подій відкривання/закривання дверей

відповідно. Інформація на сторінку передається через вебсокети. Нові записи на сторінку додаються зверху. Поки вебсокети не підключились, на сторінці відображується анімовани кружечок завантажженя (рис. 4.4.3 та рис. 4.4.4). Також на сторінках присутній фільтр, в який можна ввести проміжок часу, за який відображати події в журналах. Шаблони сторінок знаходяться в файлах doorlog.html та c_log.html (Додатки Е, Є).

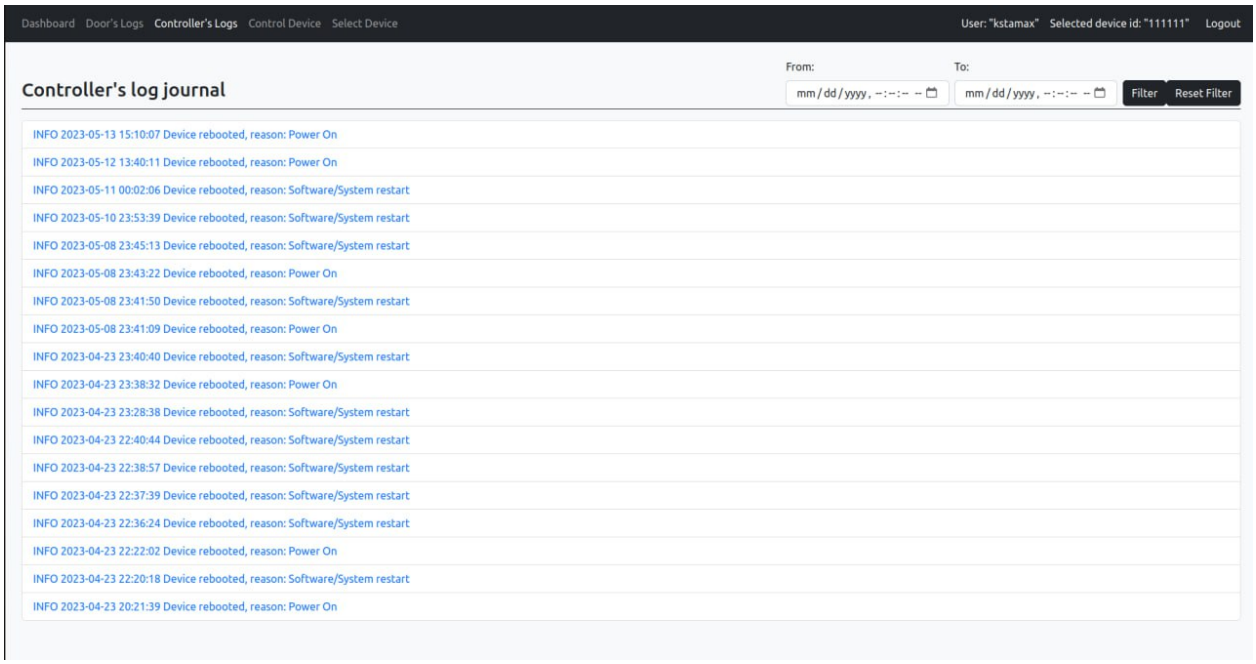


Рисунок 4.4.1 Сторінка Device’s Logs

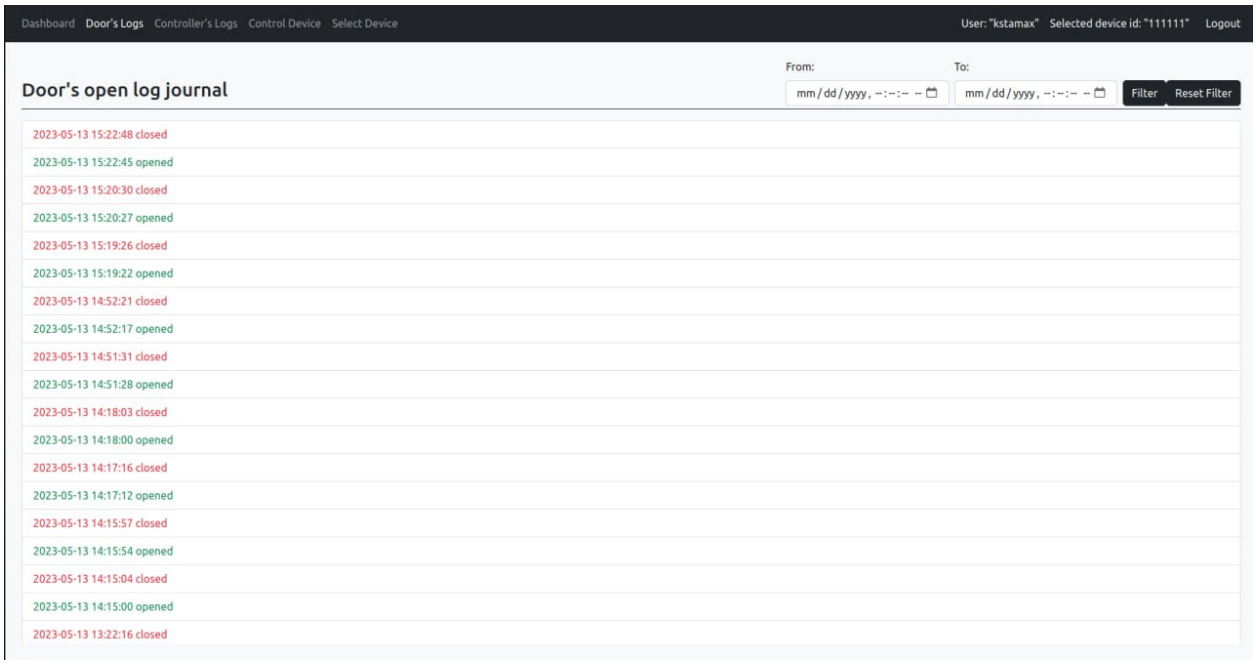

Рисунок 4.4.2 Сторінка Door’s Logs

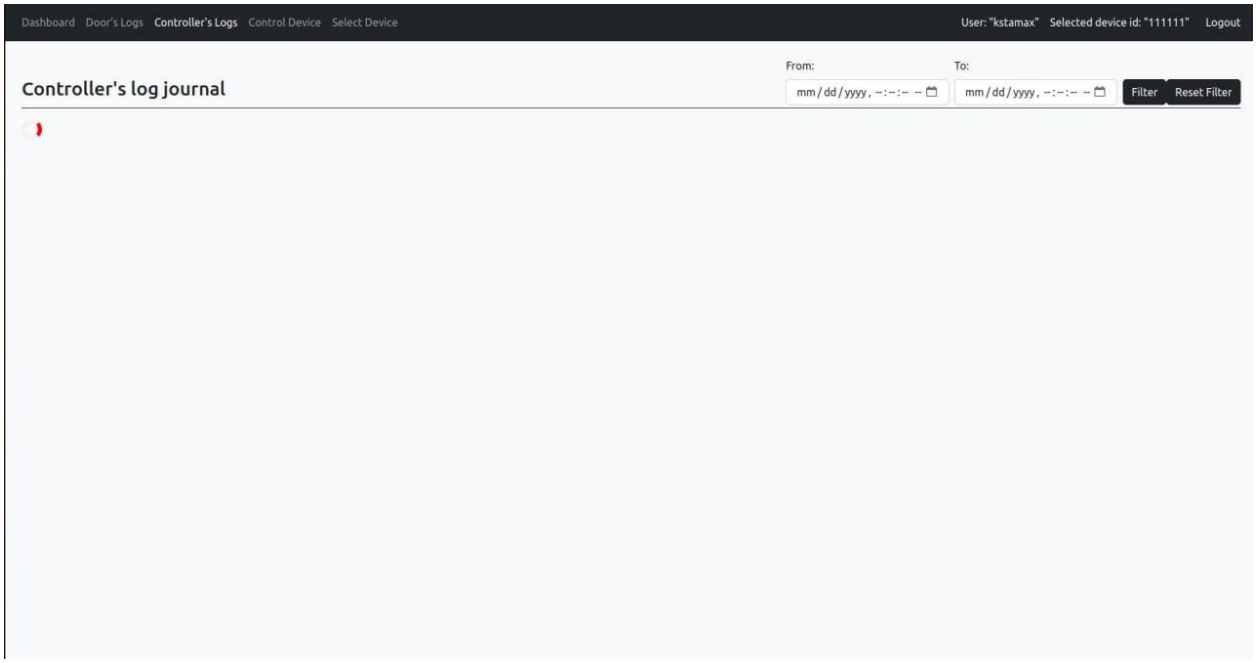


Рисунок 4.4.3 Кружечок завантаження до підключення вебсокетів на сторінці Device’s Logs

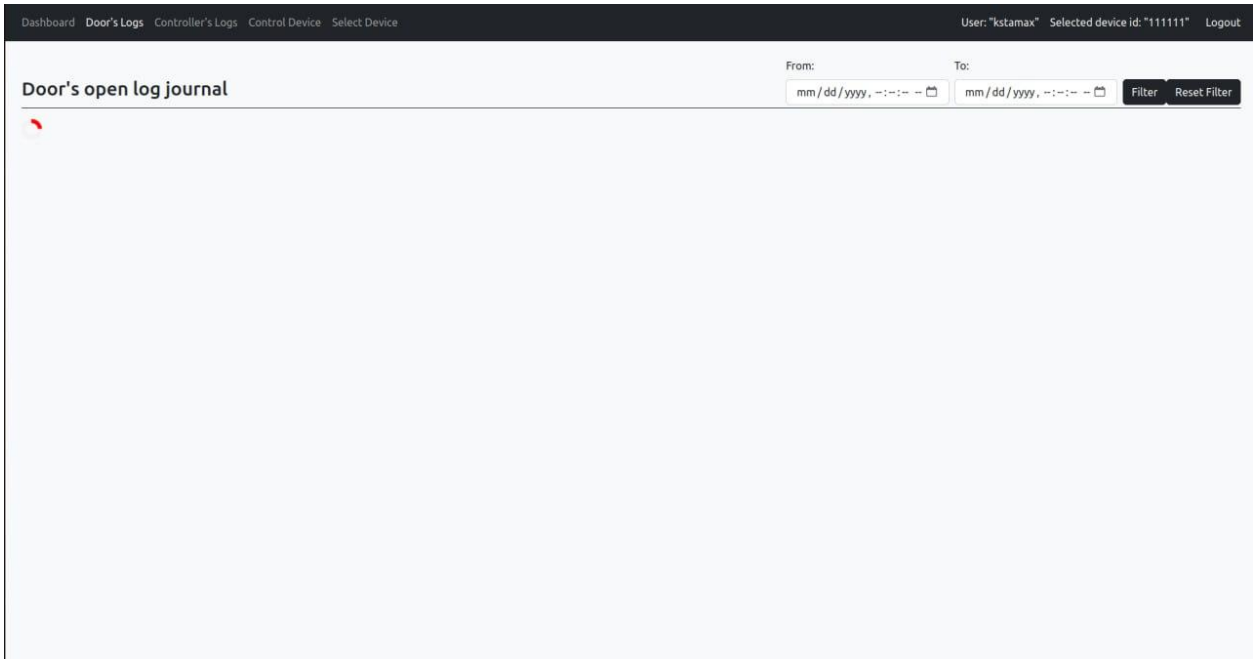


Рисунок 4.4.4 Кружечок завантаження до підключення вебсокетів на сторінці Door’s Logs

## 4.5 Сторінка Control Device

Сторінка Control Device (рис. 4.5.1) відповідає за керуванням мікроконтролером. Вона містить в собі 3 кнопки – зміна стану світлодіода, зміна стану реле та перезавантаження пристрою. Кнопки зміну станів світлодіода та реле оновлюється динамічно. При заході на сторінку відправляється запит на мікроконтролер “getState”, який надсилає поточні стани світлодіоду та реле, на основі чого генерується кнопка, яка пропонує вимкнути діод, якщо він увімкнений і навпаки, так само з реле (рис. 4.5.1 та рис. 4.5.2). Запит “getState” відбувається через вебсокети, тому поки не відбулося підключення, кнопки неактивні, пофарбовані в сірий та мають текст “Loading…” (рис. 4.5.3). Шаблон сторінки знаходиться в файлі control_device.html (Додаток Ж).

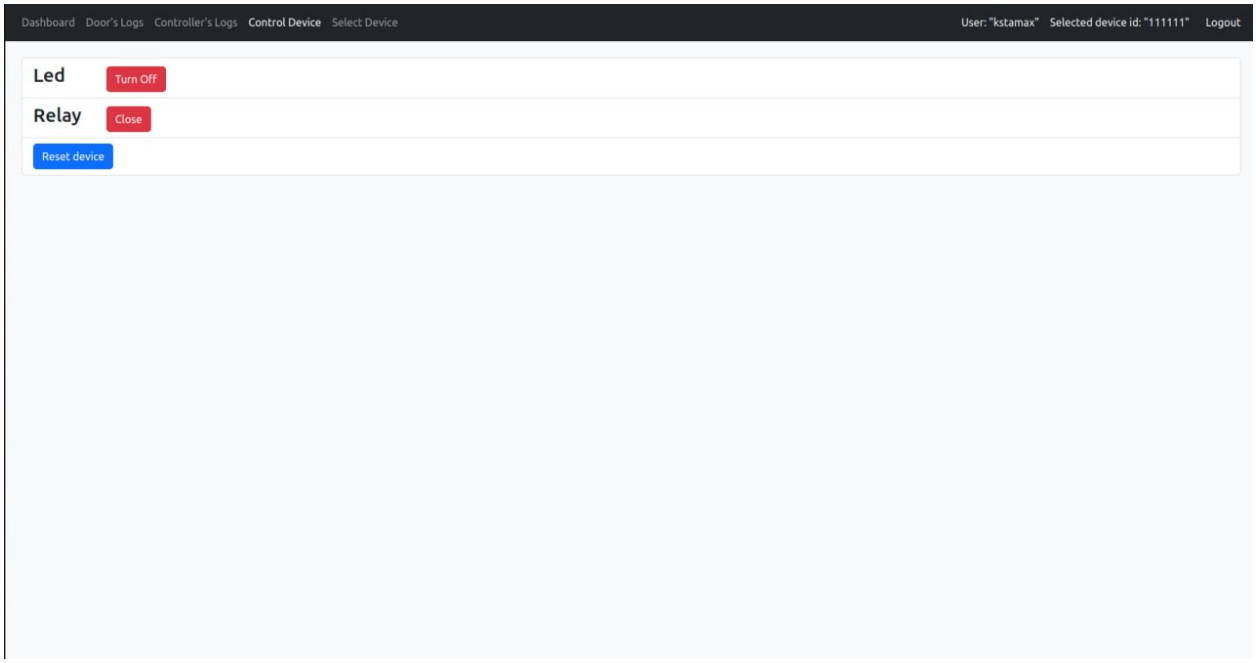


Рисунок 4.5.1 Сторінка Control Device, реле перебуває у відкритому стані, діод увімкнений

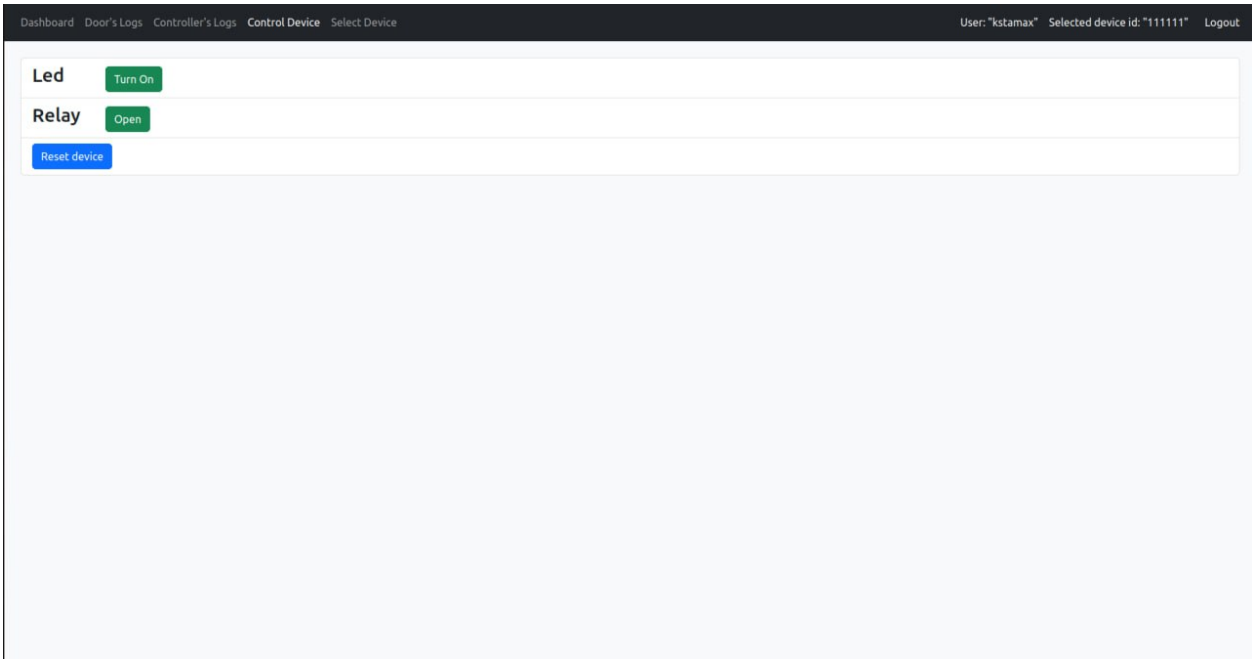


Рисунок 4.5.2 Сторінка Control Device, реле перебуває у закритому стані, діод вимкнений.

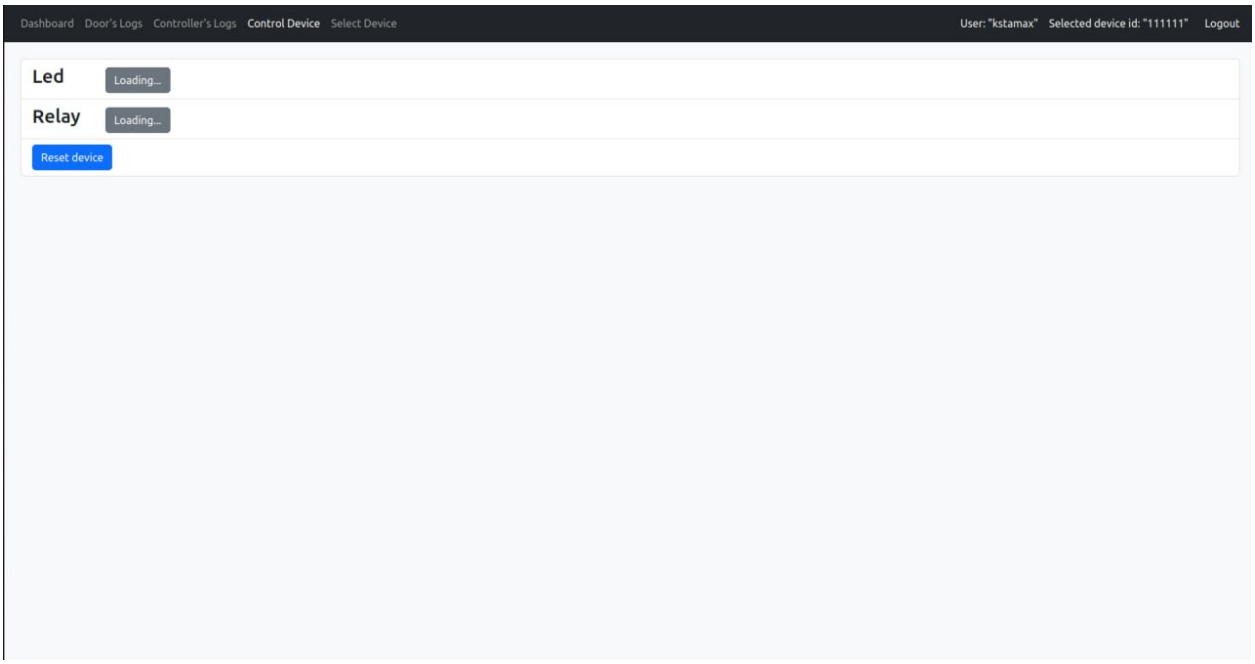


Рисунок 4.5.3 Сторінка Control Device, неактивні кнопки, поки не відбулося підключення вебсокета.

## 4.6 Скрипти та стилі

Для реалізації клієнтів вебсокета на стороні сторінок користувача використовується JavaScript. Для всіх сторінок код майже однаковий, відрізняється тільки динамічний рендеринг елементів на сторінці. Файли скріптів наведені в додатках Р, С, Т, У, Ф.

Файл стилю всього один – header_style.css (Додаток Х). В ньому пропасаний анімований кружечок завантаження та додана анімація – при наведені на активний елемент в панелі навігації, він трошки збільшується.

# ВИСНОВКИ

В ході виконання дипломної роботи, було розроблено систему нагляду за відкриттям/закриттям дверей на базі мікроконтролера NodeMCU ESP8266. На роль датчика відкриття дверей був обраний МС-38, який складається з двух частин – геркону та магніту. Датчик має пластмасовий корпус, на задню частину якого нанесений клейкий матеріал, що дозволило зручно приклеїти одну частину датчика на двері, а іншу – на рамку дверей. Датчик на виході має два проводи, до проводів були припаєні роз'єми «мама», що дало змогу зручно підключити один провід на ніжку GND, інший – на ніжку D1 контролера. Оголені частини проводів були ізольовані термоусадочними трубками.

Для того щоб надати доступ користувачу до інформації, яку отримав мікроконтролер NodeMCU ESP8266 з датчика, був розроблений вебсервер мовою Python на базі фреймворку Django. Вебсервер складається з двох великих частин:

- Перша частина відповідає за забезпечення користувача вебсторінкою, на якій буде відображений журнал зчитувань показників датчика відкриття/закриття дверей, журнал перезавантажень мікроконтролера та причин, через які він перезавантажився.
- Друга частина відповідає за спілкування з мікроконтролером: отримання з нього інформації та надсилання йому команд.

Спілкування сервера і мікроконтролера відбувається по протоколу WebSocket. На сервері реалізована серверна частина вебсокету, яка приймає підключення клієнта вебсокета мікроконтролера, обробляє повідомлення з нього та надсилає йому команди. На ESP8266 реалізована клієнтська частина вебсокету, яка надсилає запит на підключення до серверу і потім відправляє на сервер всі необхідні дані та обробляє отримані команди.

Основною перевагою такої моделі є безперервне з'єднання. Хоча іноді це може бути і мінусом, так як один з портів сервера буде постійно зайнятий, але у

цьому випадку воно забезпечує швидке спілкування контролера з сервером, а також дозволяє майже в режимі реального часу визначати доступність або недоступність сервера чи контролера.

Також була розглянута модель, де і на стороні вебсервера і на стороні мікроконтролера запускається HTTP сервер. Відповідно коли вебсерверу потрібно надіслати команду на контролер, він через HTTP клієнт робить запит на HTTP сервер контролера, і навпаки коли контролеру потрібно відправити інформацію.

Така модель може забезпечувати двохстороннє спілкування між контролером і сервером, але в порівнянні з вебсокетом вона має декілька недоліків, а саме:

1) Кожен раз коли контролеру або серверу потрібно відправити інформацію, повинно створитися нове HTTP з’єднання, яке займає додатковий час на проходження процесу рукостискання. В свою чергу, для вебсокета встановлюється тільки одне з’єднання, яке лишається відкритим до моменту розриву.
2) Через те, що немає постійного з’єднання, сервер зможе зрозуміти чи є зараз контролер в мережі тільки коли відправить якусь команду. У випадку з вебсокетами, сервер постійно відслідковує стан з’єднання з контролером, в режимі реального часу. Визначення доступності контролера в мережі є корисним для інформування користувача про стан контролера, а також запобігає марним спробам надіслати команду на контролер, що економить ресурси сервера.
3) Реалізація лише клієнтської частини на контролері займає менше пам’яті та вимагає менше ресурсів, ніж реалізація HTTP клієнту та серверу.
4) Реалізація вебсокет серверу відкрила можливість для реалізації також вебсокет клієнтів на вебсторінках, які будуть миттєво оновлювати дані для користувача, по мірі їх надходження з контролера.

Для зручного відображення користувачу даних з контролера був створений вебсайт. Вебсайт має наступні сторінки: Dashboard, Door’s Logs, Controller’s Logs, Control Device та Select Device.

Доступ до вебсайту дозволений тільки аутентифікованим користувачам за допомогою логіну і паролю. Так як система передбачалася закритою, додати нового користувача може тільки адміністратор, використовуючи панель адміністратора Django, яка доступна за адресою http://«адреса сервера»/admin. Ця панель доступна тільки користувачам, які мають права адміністратора.

Кожен девайс має свій унікальний ідентифікатор, який складається з 6 символів, наприклад «111111». Також девайс має власника та список користувачів, які мають доступ до нього. Для забезпечення захищеного з’єднання з сервером, кожен девайс має свій унікальний ключ, який складається з 40 символів. Для того щоб сервер дозволив нове з’єднання по вебсокету для девайса, має співпасти і ідентифікатор девайса і ключ.

Після того, як користувач коректно ввів свої логін і пароль, він потрапляє на сторінку зі списком девайсів, до яких він має доступ, де він може обрати девайс, з яким хоче працювати. Інформація про обраний девайс зберігається в сессії користувача.

На сторінці Dashboard користувач може побачити наступну інформацію:

1) Статус девайса – в мережі, чи ні. Якщо девайс не в мережі, там також будуть вказані дата та час, коли девайс був останній раз в мережі.
2) Актуальний стан датчика відкриття/закриття дверей – відкритий або закритий

Вся інформація на цій сторінці оновлюється динамічно і отримується через вебсокет. Поки йде підключення до вебсокету, користувач побачить текст “Loading…”, що дасть йому зрозуміти, що тут має з’явитися якась інформація і варто почекати.

На сторінці Door’s Logs користувач може побачити журнал відкриття/закриття дверей, де будуть всі зафіксовані відкриття або закриття дверей а також точні дата та час цих подій.

На сторінці Controller’s Logs користувач може побачити журнал, де записані всі перезавантаження контролера, а також їх причини. Це дає змогу користувачу виявити можливу несправність контролера, серйозні зависання (якщо такі трапляться), проблеми з живленням та інше.

Обидві сторінки з журналами також мають опцію фільтру по даті та часу, користувач може вказати за який період часу він хоче подивитися записи в журналі та легко скинути цей фільтр одним натисканням кнопки, якщо він хоче повернути назад журнал за весь період часу. Також обидві стоірнки журналу оновлюються динамічно, кожен раз коли контролер відправляє новий запис в журнал, у користувача він миттєво з’явиться на сторінці, без необхідності перезавантажувати сторінку. Інформація на ці сторінки приходить через вебсокети, поки йде підключення до вебсокету, користувач побачить на сторінці анімований кружечок завантаження.

На сторінці Control Device користувач може ввімкнути або вимкнути вбудований світлодіод на платі, перезавантажити девайс, або перемкнути стан реле, яке також може бути використане для потреб користувача. Якщо девайс не в мережі, то користувач замість кнопок керування побачить текст “Loading…”. Статуси кнопок оновлюються динамічно через вебсокети. При заході на сторінку через вебсокет відправляється сигнал на сервер, про те, що потрібно запитати у девайса про стан реле та діода, що сервер і робить, девайс в свою чергу відправляє цю інформацію на сервер, яка перенаправляється на сторінку і користувач бачить актуальну інформацію про реле(закрите/відкрите) та діод(ввімкнений/вимкнений).

На сторінці Select Device користувач в будь-який момент може переключитися на інший контролер, який може бути встановлений в іншому місці і підключений до сервера. Ідентифікатор обраного контролера буде доступний користувачу у верхній панелі, щоб він завжди міг подивитися, з яким контролером працює на даний момент. Також на цій сторінці користувач може побачити стан всіх контролерів – в мережі вони чи ні.

Всі сторінки стилізовані за допомогою Bootstrap5 та власного CSS коду. Зверху користувачу завжди доступна навігація між сторінками. Всі сторінки відчуваються живими, так як мають анімовані елементи, завантажуються динамічно, при наведені мишкою на посилання – збільшується розмір тексту та підсвічується зона з текстом, при наведенні мишкою на кнопки - вони підсвічуються. Також сторінки мають мінімалістичний стиль, користувач не бачить нічого зайвого, немає яскравих кольорів, які б кидалися в очі. Все це покращує досвід роботи користувача з системою та залишає приємні враження.

Збереженнях всіх даних відбувається у базу даних SQLite. При бажанні вона може бути легко замінена на MySQL або PostgreSQL або на будь-яку іншу базу даних, яку підтримує Django. Це можливо через те, що Django має вбудовану ORM для роботи з базою і її заміна на будь-яку іншу зводиться лише до внесення невеликих змін в налаштування, розгортання самої бази та написання двох команд в консоль. Створення всіх необхідних таблиць Django виконує самостійно.

Загалом ця система контролю є хорошим рішенням для виконання задачі стеження за відкриттям/закриттям дверей тому, що вона має:

1) Зручний, стилізований, динамічний веб інтерфейс для взаємодії користувача з системою через персональний комп’ютер або смартфон
2) Можливість масштабування – можна додавати багато користувачів та девайсів, які потенційно можуть стояти на різних дверях, в різних кімнатах, приміщеннях або навіть містах чи країнах, якщо сервер має публічну IP адресу та SSL сертифікат для безпечного публічного з’єднання.
3) Можливість розширення функціоналу – в цій системі реалізований універсальний принцип спілкування девайса та сервера через вебсокети. Використовуючи цей принцип можна потенційно додавати безліч різних датчиків чи інших електронних приладів (як наприклад були додані реле та датчик температури та вологості повітря) і дописувати реалізацію

роботи з ними як в код девайса, так і в код серверу без необхідності внесення сильних змін в існуючий код.

4) Невибаглива – загальна вартість системи є дуже низькою, так як ESP8266 є популярним, промисловим, дешевим контролером, датчик відкриття дверей також коштує небагато. Сервер не повинен постійно працювати, так як контролер здатен запам’ятовувати події в свою оперативну пам’ять і відправляти їх на сервер коли він буде доступний. Так само і контролер не повинен постійно працювати, користувач матиме доступ до журнала дверей і контролера, навіть якщо він вимкнений.
5) Автономна – після встановлення, контролер не вимагає жодної уваги, він працює надійно та без перебоїв. В разі несподіваних випадків, може бути перезавантажений дистанційно через веб інтерфейс користувача
6) Надійна – контролер пройшов тестування на вхідних дверях в квартиру, пропрацювавши два тижні поспіль і за цей час жодного разу не перезавантажився та не пропустив жодного відкриття/закриття дверей. При цьому вебсервер вмикався тільки раз на день – ввечері. Всі записи в журналі подій мають точні час і дату, які контролер отримував з NTP серверу при записі події в журнал. Жоден запис в журналі не є пошкодженим чи викривленим.

Недоліком системи є те, що вона не підходить для використання в цілях охорони якогось серйозного об’єкту, так як:

А) Контролер не має захисту від заглушення – частоти WiFi мережі можуть бути заглушені спеціальним пристроєм, через що контролер втратить з’єднання з сервером та не зможе сповістити про вторгнення

Б) Контролер не має автономного живлення

В) Немає оперативного сповіщення про вторгнення (СМС або будь-яке інше сповіщення на телефон)

Г) Немає механізму, щоб сповістити, що контролер або датчик були зірвані та і взагалі немає ніякого хоча б пластикового корпусу щоб забезпечити цілісність системи

# ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Getting Started with ESP8266 NodeMCU Development Board [Електронний ресурс]
URL: https://randomnerdtutorials.com/getting-started-with-esp8266-wifi-transceiver-review/#esp8266-intro

2. Django Channels [Електронний ресурс]
URL: https://channels.readthedocs.io/en/stable/

3. SHT20 Datasheet [Електронний ресурс]
URL: https://pdf1.alldatasheet.com/datasheet-pdf/view/756285/ETC2/SHT20.html

4. Basics of the I2C Communication Protocol [Електронний ресурс]
URL: https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/

5. MC-38 Magnetic Contact Switch Sensor: Here is How to Use It with ESP8266 and Arduino. [Електронний ресурс]
URL: https://juanstechblog.blogspot.com/2022/06/magnetic-contact-switch-sensor-mc-38-esp8266-arduino.html

6. ESP8266 NodeMCU NTP Client-Server: Get Date and Time (Arduino IDE) [Електронний ресурс]
URL: https://randomnerdtutorials.com/esp8266-nodemcu-date-time-ntp-client-server-arduino/

7. ESP8266 Watchdogs in Arduino [Електронний ресурс]
URL: https://sigmdel.ca/michel/program/esp8266/arduino/watchdogs_en.html

8. The top programming languages [Електронний ресурс]
URL: https://octoverse.github.com/2022/top-programming-languages

9. Django Model View Template (MVT) Overview [Електронний ресурс]
URL: https://www.onlinetutorialspoint.com/django/django-model-view-template-mvt-overview.html

# ДОДАТКИ

## Додаток А

Лістинг програмного коду мікроконтролера NodeMCU ESP8266

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <WiFiClient.h>
#include <NTPClient.h>
#include <WiFiUdp.h>
#include <WebSocketsClient.h>
#include <ArduinoOTA.h>
#include <Wire.h>
#include "DFRobot_SHT20.h"

#define LED D0
#define GER D2
#define RELAY D1


const char* ssid = "TP-Link_CC35";
const char* password = "66433187";

const long utcOffsetInSeconds = 0;
const char* auth_token = "authorization:
M9p7o0hlcDmsyLzEHL3u12AUFkqkfsg1j2uxVjiP\r\ndevice: 111111";

WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "pool.ntp.org", utcOffsetInSeconds);

DFRobot_SHT20    sht20;
float prev_temp = 0;
float prev_humid = 0;

WebSocketsClient webSocket;

bool prev_gerkon_read = LOW;

bool rebootFlag;
bool ledState;
bool relayState;
String delayedMessages[200];
int delayedMessagesCount = 0;
const int MAX_DELAYES = 200;
```

```
String dateTime(){
  timeClient.begin();
  timeClient.update();
  //Get a date structure
  time_t epochTime = timeClient.getEpochTime();
  struct tm *ptm = gmtime ((time_t *)&epochTime);
  int monthDay = ptm->tm_mday;
  int currentMonth = ptm->tm_mon+1;
  int currentYear = ptm->tm_year+1900;
  String currentDate = String(currentYear) + "-" + String(currentMonth) + "-" +
String(monthDay);
  // 1-api_key, 2-date, 3-time
  String resultStr =
"\"2\":\""+currentDate+"\",\"3\":\""+timeClient.getFormattedTime()+"\"";
  timeClient.end();
  return resultStr;
}

String deviceLog(String action, String actionType){
  //0-type:0-device, 4-action, 5-actionType
  return "{\"0\":\"0\","+dateTime()+",\"4\":\""+action+"\",\"5\":\""+actionType+"\"}";
}

String temp_and_humidity(float temperature, float humidity){
  char buffer1[10];
  dtostrf(temperature, 4, 2, buffer1);
  String s_temperature = String(buffer1);
  char buffer2[10];
  dtostrf(humidity, 4, 2, buffer2);
  String s_humidity = String(buffer2);

  //0-type:0-device, 4-action, 5-actionType
  return "{\"0\":\"3\",\"9\":\""+s_temperature+"\",\"10\":\""+s_humidity+"\"}";
}

void rebootLog(){
  String action = "Device rebooted, reason: " + ESP.getResetReason();
  String actionType;
  if((ESP.getResetReason()== "External System") || (ESP.getResetReason()==
"Software/System restart") || (ESP.getResetReason()== "Power On")){
    //0-INFO
    actionType = "0";
  }
  else{
```

```
    //1-ERROR
    actionType = "1";
  }
  String msg = deviceLog(action, actionType);
  if (webSocket.isConnected()){
    webSocket.sendTXT(msg);
  }
  else if (delayedMessagesCount < MAX_DELAYES) {
    delayedMessages[delayedMessagesCount] = msg;
    delayedMessagesCount++;
  }
}

void sendState(){
  String led_s = ledState ? "1" : "0";
  String relay_s = relayState ? "1" : "0";
  //0-type:1-state, 6-led, 7-relay
  String msg = "{\"0\":\"1\",\"6\":\""+led_s+"\",\"7\":\""+relay_s+"\"}";
  if (webSocket.isConnected()){
    webSocket.sendTXT(msg);
  }
  else if (delayedMessagesCount < MAX_DELAYES) {
    delayedMessages[delayedMessagesCount] = msg;
    delayedMessagesCount++;
  }
}

void handleSHT20(){
  if (webSocket.isConnected()){
    float temp = sht20.readTemperature();
    float humid = sht20.readHumidity();
    if((abs(temp-prev_temp) >= 0.1) || (abs(humid-prev_humid) >= 0.1)){
      prev_temp = temp;
      prev_humid = humid;
      String msg = temp_and_humidity(temp, humid);
      webSocket.sendTXT(msg);
    }
  }
}

void messageHandler(String msg){
  if (msg[0] == '0') {
    if (msg[1] == '0') {
      digitalWrite(LED, LOW);
      ledState = false;
```

```
    }
    else if (msg[1] == '1') {
      digitalWrite(LED, HIGH);
      ledState = true;
    }
  }
  else if (msg[0] == '1') {
    if (msg[1] == '0') {
      digitalWrite(RELAY, LOW);
      relayState = false;
    }
    else if (msg[1] == '1') {
      digitalWrite(RELAY, HIGH);
      relayState = true;
    }
  }
  else if (msg[0] == '2') {
    ESP.reset();
  }
}

void webSocketEvent(WStype_t type, uint8_t * payload, size_t length) {
    switch(type) {
      case WStype_DISCONNECTED:
        {
          Serial.printf("[WSc] Disconnected!\n");
          break;
        }
      case WStype_CONNECTED:
        {
          Serial.printf("[WSc] Connected to url: %s\n", payload);
          if (rebootFlag){
            rebootLog();
            rebootFlag = false;
          }
          sendState();
          break;
        }
      case WStype_TEXT:
        {
          Serial.printf("[WSc] RESPONSE: %s\n", payload);
          char* payload_converter = (char*) payload;
          String payload_str = payload_converter;
          if (payload_str == "getState") {
            sendState();
```

```
        }
        else{
          messageHandler(payload_str);
        }
        break;
      }
    case WStype_BIN:
      {
        Serial.printf("[WSc] get binary length: %u\n", length);
        hexdump(payload, length);
        break;
      }
    case WStype_PING:
      {
        Serial.printf("[WSc] get ping\n");
        break;
      }
    case WStype_PONG:
      {
        Serial.printf("[WSc] get pong\n");
        break;
      }
  }
}

void checkGerkon(){
  bool gerkon_read = digitalRead(GER);
  //0-type:2-gerkon, 8-state
  if (gerkon_read != prev_gerkon_read) {
    if (gerkon_read == HIGH) {
      String msg = "{\"0\":\"2\","+dateTime() + ",\"8\":\"1\"}";
      if (webSocket.isConnected()){
        webSocket.sendTXT(msg);
      }
      else if (delayedMessagesCount < MAX_DELAYES) {
        delayedMessages[delayedMessagesCount] = msg;
        delayedMessagesCount++;
      }
      digitalWrite(LED, LOW);
    } else {
      String msg = "{\"0\":\"2\","+dateTime() + ",\"8\":\"0\"}";
      if (webSocket.isConnected()){
        webSocket.sendTXT(msg);
      }
      else if (delayedMessagesCount < MAX_DELAYES) {
```

```
      delayedMessages[delayedMessagesCount] = msg;
      delayedMessagesCount++;
     }
     digitalWrite(LED, HIGH);
    }
    prev_gerkon_read = gerkon_read;
  }
}

void handleDelayedMessages(){
  int temp = delayedMessagesCount;
  for (int i = 0; i < temp; i++){
    if (webSocket.isConnected()){
      webSocket.sendTXT(delayedMessages[i]);
      delayedMessagesCount--;
    }
    else {
      break;
    }
  }
}

void setup() {
  Serial.begin(9600);
  Wire.begin(D5, D6);
  sht20.initSHT20();
  delay(100);
  sht20.checkSHT20();
  pinMode(LED, OUTPUT);
  pinMode(GER, INPUT_PULLUP);
  pinMode(RELAY, OUTPUT);
  prev_gerkon_read = digitalRead(GER);
  digitalWrite(LED, LOW);
  ledState = false;
  digitalWrite(RELAY, HIGH);
  relayState = true;
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    digitalWrite(LED, LOW);
    delay(250);
    digitalWrite(LED, HIGH);
    delay(250);
    Serial.print(".");
```

```
  }
  Serial.println("");
  Serial.println("WiFi connected.");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  ArduinoOTA.begin();
  rebootFlag = true;
  webSocket.setExtraHeaders(auth_token);
  webSocket.begin("192.168.0.104", 80, "/ws/main/esp");
  webSocket.onEvent(webSocketEvent);
  webSocket.enableHeartbeat(1000, 1000, 1);
  webSocket.setReconnectInterval(10000);
}

void loop() {
  webSocket.loop();
  checkGerkon();
  handleSHT20();
  handleDelayedMessages();
  ArduinoOTA.handle();
  delay(100);
}
```

**Додаток Б**

Лістинг файлу models.py

```
from django.db import models
from django.contrib.auth.models import User


class ESPDevice(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    device_id = models.CharField(max_length=6, unique=True)
    token_key = models.CharField(max_length=40, unique=True, blank=True)
    created_at = models.DateTimeField(auto_now_add=True)
    last_online = models.DateTimeField(blank=True, null=True, default=None)
    is_online = models.BooleanField(blank=True, null=True, default=None)


class DoorLog(models.Model):
    DOOR_STATES = (
        ("opened", "opened"),
        ("closed", "closed")
    )
    log_date = models.DateTimeField()
    door_state = models.CharField(max_length=6, choices=DOOR_STATES)
```

```
    device = models.ForeignKey(ESPDevice, on_delete=models.CASCADE)


class DeviceLog(models.Model):
    MESSAGE_TYPES = (
        ("ERROR", "ERROR"),
        ("INFO", "INFO")
    )
    log_date = models.DateTimeField()
    message_type = models.CharField(max_length=10, choices=MESSAGE_TYPES)
    message_body = models.TextField()
    device = models.ForeignKey(ESPDevice, on_delete=models.CASCADE)
```

**Додаток В**

Лістинг файлу views.py

```
from django.shortcuts import render, redirect
from django.urls import reverse_lazy
from django.views.generic import View, TemplateView
from django.contrib.auth.mixins import LoginRequiredMixin
from django.contrib.auth.views import LoginView
from .models import ESPDevice


class LoginPageView(LoginView):
    template_name = 'main/login.html'
    fields = '__all__'
    redirect_authenticated_user = True

    def get_success_url(self):
        return reverse_lazy('main:select_dev')


class IndexPageView(LoginRequiredMixin, TemplateView):
    template_name = 'main/index.html'

    def get(self, request, *args, **kwargs):
        if 'selected_device' not in request.session:
            return redirect(reverse_lazy('main:select_dev'))
        else:
            return super().get(request, *args, **kwargs)


class DoorLogs(LoginRequiredMixin, TemplateView):
    template_name = 'main/doorlog.html'
```

```
    def get(self, request, *args, **kwargs):
        if 'selected_device' not in request.session:
            return redirect(reverse_lazy('main:select_dev'))
        else:
            return super().get(request, *args, **kwargs)

    def post(self, request, *args, **kwargs):
        if request.POST.get('reset_filter'):
            request.session['doorlogs_from_to'] = None
            return redirect('main:doorlogs')
        from_datetime = request.POST.get('from_datetime')
        to_datetime = request.POST.get('to_datetime')

        request.session['doorlogs_from_to'] = (from_datetime, to_datetime)

        return redirect('main:doorlogs')


class LogsPageView(LoginRequiredMixin, TemplateView):
    template_name = 'main/c_logs.html'

    def get(self, request, *args, **kwargs):
        if 'selected_device' not in request.session:
            return redirect(reverse_lazy('main:select_dev'))
        else:
            return super().get(request, *args, **kwargs)

    def post(self, request, *args, **kwargs):
        if request.POST.get('reset_filter'):
            request.session['c_logs_from_to'] = None
            return redirect('main:c_logs')
        from_datetime = request.POST.get('from_datetime')
        to_datetime = request.POST.get('to_datetime')
        request.session['c_logs_from_to'] = (from_datetime, to_datetime)

        return redirect('main:c_logs')


class ControlDevicePageView(LoginRequiredMixin, TemplateView):
    template_name = 'main/control_device.html'

    def get(self, request, *args, **kwargs):
        if 'selected_device' not in request.session:
            return redirect(reverse_lazy('main:select_dev'))
        else:
```

```
        return super().get(request, *args, **kwargs)


class SelectDeviceView(LoginRequiredMixin, View):
    template_name = 'main/select_device.html'

    def render_select_device_page(self, request):
        devices = ESPDevice.objects.filter(user=request.user)
        return render(request, self.template_name, {'devices': devices})

    def get(self, request):
        return self.render_select_device_page(request)

    def post(self, request):
        device_id = request.POST.get('device_id')
        if device_id:
            device = ESPDevice.objects.filter(user=request.user,
device_id=device_id).first()
            if device:
                request.session['selected_device'] = device_id
                return redirect(reverse_lazy('main:index'))
            else:
                return self.render_select_device_page(request)
        else:
            return self.render_select_device_page(request)
```

**Додаток Г**

Лістинг файлу login.html

```
{% extends 'main/base.html' %}
{% block content %}
<section class="vh-100">
    <div class="container py-5 h-60">
      <div class="row d-flex justify-content-center align-items-center h-100">
        <div class="col-12 col-md-8 col-lg-6 col-xl-5">
          <div class="card bg-dark text-white" style="border-radius: 1rem;">
            <div class="card-body p-5 text-center">

              <div class="mb-md-2 mt-md-2 pb-2">
                <form method="POST" class="text-center">
                    {% csrf_token %}
                    <h2 class="fw-bold mb-2 text-uppercase">Login</h2>
                    <p class="text-white-50 mb-5">Please enter your login and
password!</p>

                    <div class="form-outline form-white mb-2">
```

```
                <input type="text" id="typeEmailX" class="form-control"
name="username" required />
                <label class="form-label" for="typeEmailX">Username</label>
                </div>

                <div class="form-outline form-white mb-2">
                <input type="password" id="typePasswordX" class="form-control"
name="password" required />
                <label class="form-label" for="typePasswordX">Password</label>
                </div>

                <button class="btn btn-outline-light px-5"
type="submit">Login</button>
              </form>
            </div>
          </div>
        </div>
      </div>
    </div>
  </section>
{% endblock content %}
```

**Додаток Ґ**

Лістинг файлу select_device.html

```
{% extends 'main/base.html' %}
{% load static %}
{% block content %}
  <h2>Available devices</h1>
  <ul class="list-group" style="max-height: 80vh; overflow:scroll;" id="itemsList">
    {% for device in devices %}
    <li class="list-group-item text-light w-25 text-center bg-dark"
onclick="document.getElementById('{{ device.device_id }}').submit();">
       <span style="font-size: 20px;">ID {{ device.device_id }} Status: <span
id="deviceStatus_{{device.device_id}}">Loading...</span></span>
       <form id="{{ device.device_id }}" method="post" action="{% url
'main:select_dev' %}">
         {% csrf_token %}
         <input type="hidden" name="device_id" value="{{ device.device_id }}">
       </form>
    </li>
    {% endfor %}
  </ul>
<script src="{% static 'main/scripts/device_status_select.js' %}"></script>
{% endblock %}
```

**Додаток Д**

Лістинг файлу index.html

```
{% extends 'main/base.html' %}
{% load static %}
{% block style %}
<style>
  .square {
    background-color: #ffffff;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.2);
    padding: 2%;
    text-align: center;
    height: 40vh;
    width: 100%;
    margin: 1%;
    display: flex;
    justify-content: center;
    align-items: center;
    max-width: 100%;
  }
</style>
{% endblock style %}
{% block content %}
<div class="container">
  <div class="row justify-content-center">
    <div class="col-md-6">
      <div class="square">
        <div>
          <h3 id="deviceStatus" style="margin: 0;">Device status is loading...</h3>
          <p class="text-danger" id="lastOnline" style="margin: 0;"></p>
        </div>
      </div>
      <div class="square">
        <div>
          <h3 style="margin: 0;">Humidity: <span id="humid">Loading...</span></h3>
        </div>
      </div>
    </div>
    <div class="col-md-6">
      <div class="square">
        <div>
          <h3 style="margin: 0;" id="doorStatus">Loading...</h3>
        </div>
      </div>
      <div class="square">
```

```
      <div>
        <h3 style="margin: 0;">Temperature: <span
id="temp">Loading...</span></h3>
      </div>
    </div>
  </div>
 </div>
</div>
<script src="{% static 'main/scripts/dashboard.js' %}"></script>
{% endblock content %}
```

**Додаток Е**

Лістинг файлу doorlog.html

```
{% extends 'main/base.html' %}
{% load static %}
{% block content %}
<div class="d-flex justify-content-between align-items-end border-bottom border-
dark mb-3 pb-1">
 {% if request.session.doorlogs_from_to %}
  <h3 class="text-dark d-inline-block me-5">
   Door's open log journal ({{ request.session.doorlogs_from_to.0}} -
{{request.session.doorlogs_from_to.1}})
  </h3>
 {% else %}
  <h3 class="text-dark d-inline-block me-5">Door's open log journal</h3>
 {% endif %}
 <form method="post" action="{% url 'main:doorlogs' %}">
  {% csrf_token %}
  <div class="d-flex align-items-end">
   <div class="flex-grow-1 me-2">
    <label for="from_datetime" class="form-label">From:</label>
    <input type="datetime-local" class="form-control w-100" id="from_datetime"
name="from_datetime"
           step="1">
   </div>
   <div class="flex-grow-1 me-2">
    <label for="to_datetime" class="form-label">To:</label>
    <input type="datetime-local" class="form-control w-100" id="to_datetime"
name="to_datetime"
           step="1">
   </div>
   <button type="submit" class="btn btn-dark">Filter</button>
   <button type="submit" class="btn btn-dark" name="reset_filter" value="1">Reset
Filter</button>
  </div>
```

```
  </form>
</div>
<ul class="list-group" style="max-height: 80vh; overflow:scroll;" id="itemsList">
</ul>
<div class="loader"></div>
<script src="{% static 'main/scripts/doorlog.js' %}"></script>
{% endblock content %}
```

**Додаток Є**

Лістинг файлу c_logs.html

```
{% extends 'main/base.html' %}
{% load static %}
{% block content %}
<div class="d-flex justify-content-between align-items-end border-bottom border-
dark mb-3 pb-1">
  {% if request.session.c_logs_from_to %}
    <h3 class="text-dark d-inline-block me-5">
      Controller's log journal ({{ request.session.c_logs_from_to.0}} -
{{request.session.c_logs_from_to.1}})
    </h3>
  {% else %}
    <h3 class="text-dark d-inline-block me-5">Controller's log journal</h3>
  {% endif %}
  <form method="post" action="{% url 'main:c_logs' %}">
    {% csrf_token %}
    <div class="d-flex align-items-end">
      <div class="flex-grow-1 me-2">
        <label for="from_datetime" class="form-label">From:</label>
        <input type="datetime-local" class="form-control w-100" id="from_datetime"
name="from_datetime"
               step="1">
      </div>
      <div class="flex-grow-1 me-2">
        <label for="to_datetime" class="form-label">To:</label>
        <input type="datetime-local" class="form-control w-100" id="to_datetime"
name="to_datetime"
               step="1">
      </div>
      <button type="submit" class="btn btn-dark">Filter</button>
      <button type="submit" class="btn btn-dark" name="reset_filter" value="1">Reset
Filter</button>
    </div>
  </form>
</div>
<ul class="list-group" style="max-height: 80vh; overflow:scroll;" id="itemsList">
```

```
</ul>
<div class="loader"></div>
<script src="{% static 'main/scripts/devicelog.js' %}"></script>
{% endblock content %}
```

**Додаток Ж**

Лістинг файлу control_device.html

```
{% extends 'main/base.html' %}
{% load static %}
{% block content %}
<ul class="list-group" id="itemsList">
   <li class="list-group-item" id="led_cell">
      <h3 class="text-dark d-inline-block me-3" style="width:5%;">Led</h2>
      <button class="btn btn-secondary" id="led_button">Loading...</button>
   </li>
   <li class="list-group-item">
      <h3 class="text-dark d-inline-block me-3" style="width:5%;">Relay</h2>
      <button class="btn btn-secondary" id="relay_button">Loading...</button>
   </li>
   <li class="list-group-item">
      <button class="btn btn-primary d-inline-block" id="reset_button">Reset
device</button>
   </li>
</ul>
<script src="{% static 'main/scripts/device_control.js' %}"></script>
{% endblock content %}
```

**Додаток З**

Лістинг файлу base.html

```
{% load static %}
<!doctype html>
<html lang="en">
  <head>
   <meta charset="utf-8">
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <title>ESP8266</title>
   <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
Zenh87qX5JnK2Jl0vWa8Ck2rdkQ2Bzep5IDxbcnCeuOxjzrPF/et3URy9Bv1WTRi"
crossorigin="anonymous">
   <link rel="stylesheet" type="text/css" href="{% static 'main/header_style.css'
%}">
   <link rel="shortcut icon" type="image/png" href="{% static 'main/favicon.ico'
%}"/>
```

```
    {% block style %}{% endblock style %}
  </head>
  <body class="bg-light">

    <nav class="navbar navbar-expand-lg navbar-dark bg-dark px-3">
      <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-
label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbarNav">
        <ul class="navbar-nav w-100">
          <li class="nav-item">
            {% if request.path == '/' %}
            <a class="nav-link hoverLink active rounded" href="{% url 'main:index'
%}">Dashboard</a>
            {% else %}
            <a class="nav-link hoverLink rounded" href="{% url 'main:index'
%}">Dashboard</a>
            {% endif %}
          </li>
          <li class="nav-item">
            {% if request.path == '/doorlogs' %}
            <a class="nav-link hoverLink active rounded" href="{% url 'main:doorlogs'
%}">Door's Logs</a>
            {% else %}
            <a class="nav-link hoverLink rounded" href="{% url 'main:doorlogs'
%}">Door's Logs</a>
            {% endif %}
          </li>
          <li class="nav-item">
            {% if request.path == '/clogs' %}
            <a class="nav-link hoverLink active rounded" href="{% url 'main:c_logs'
%}">Controller's Logs</a>
            {% else %}
            <a class="nav-link hoverLink rounded" href="{% url 'main:c_logs'
%}">Controller's Logs</a>
            {% endif %}
          </li>
          <li class="nav-item">
            {% if request.path == '/control' %}
            <a class="nav-link hoverLink active rounded" href="{% url
'main:control_dev' %}">Control Device</a>
            {% else %}
```

```
            <a class="nav-link hoverLink rounded" href="{% url 'main:control_dev'
%}">Control Device</a>
            {% endif %}
          </li>
          <li class="nav-item me-auto">
            {% if request.path == '/select_device' %}
            <a class="nav-link hoverLink active rounded" href="{% url
'main:select_dev' %}">Select Device</a>
            {% else %}
            <a class="nav-link hoverLink rounded" href="{% url 'main:select_dev'
%}">Select Device</a>
            {% endif %}
          </li>
          {% if user.is_authenticated %}
          <li class="nav-item navbar-text text-light me-3">
             <span>User: "{{request.user}}"</span>
          </li>
          <li class="nav-item navbar-text text-light me-3">
            <span>Selected device id: "{{request.session.selected_device}}"</span>
          </li>
          <li class="nav-item">
             <a class="navbar-nav nav-link link-light hoverLink rounded" href="{% url
'main:logout' %}">Logout</a>
          </li>
          {% endif %}
        </ul>
      </div>
    </nav>
  <div class="p-4">
  {% block content %}

  {% endblock content %}
  </div>

  <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-
q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
crossorigin="anonymous"></script>
  <script
src="https://cdn.jsdelivr.net/npm/popper.js@1.14.3/dist/umd/popper.min.js"
integrity="sha384-
ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm
49" crossorigin="anonymous"></script>
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.1.3/dist/js/bootstrap.min.js"
integrity="sha384-
```

```
ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE241rYiqJxyMiZ6OW/JmZQ5stwEU
LTy" crossorigin="anonymous"></script>
   </body>
</html>
```

**Додаток І**

Лістинг файлу consumers.py

```
import datetime as dt
import json
import pytz

from django.utils import timezone
from channels.generic.websocket import AsyncWebsocketConsumer
from channels.db import database_sync_to_async
from django.db.models import Model
from django.core import serializers
from django.core.exceptions import ObjectDoesNotExist

from .models import DoorLog, DeviceLog, ESPDevice


ACTION_TYPES = {
   '0': 'INFO',
   '1': 'ERROR'
}


class MainConsumer(AsyncWebsocketConsumer):

   def format_datetime(self, dt):
      return dt.strftime('%Y-%m-%d %H:%M:%S')

   def model_to_json(self, obj):
      if isinstance(obj, Model):
         obj = [obj, ]
      return json.dumps(json.loads(serializers.serialize('json', obj)))

   def get_doorlogs(self):
      if from_to := self.scope['session'].get('doorlogs_from_to'):
         from_datetime = dt.datetime.fromisoformat(from_to[0])
         to_datetime = dt.datetime.fromisoformat(from_to[1])
         return DoorLog.objects.filter(
            device__device_id=self.device_id,
            log_date__range=(from_datetime, to_datetime)
         )
```

```
        return DoorLog.objects.filter(device__device_id=self.device_id)

    def get_devicelogs(self):
        if from_to := self.scope['session'].get('c_logs_from_to'):
            from_datetime = dt.datetime.fromisoformat(from_to[0])
            to_datetime = dt.datetime.fromisoformat(from_to[1])
            return DeviceLog.objects.filter(
                device__device_id=self.device_id,
                log_date__range=(from_datetime, to_datetime)
            )
        return DeviceLog.objects.filter(device__device_id=self.device_id)

    def get_device(self, device_id):
        device = ESPDevice.objects.get(device_id=device_id)
        return device

    async def get_latest_door_status(self):
        door_logs = [log async for log in self.get_doorlogs()]
        if door_logs:
            return door_logs[-1].door_state

    async def send_esp_get_state(self):
        await self.send_group_msg('esp', 'getState')

    async def send_group_msg(self, group, msg):
        group_name = f'{group}_{self.device_id}' if group != 'status_select' else group
        await self.channel_layer.group_send(
            group_name, {
                'type': 'chat_message',
                'message': msg
            }
        )

    async def get_device_ids_for_current_user(self):
        device_ids = [obj.device_id async for obj in
ESPDevice.objects.filter(user=self.scope['user'])]
        return device_ids

    async def handle_logs_intro(self, database_method):
        logs = [logs async for logs in database_method()]
        for log in logs:
            log.log_date = log.log_date.astimezone(timezone.get_default_timezone())
        await self.send(text_data=self.model_to_json(logs))

    async def send_device_status(self):
```

```
        device_status, device_last_online = await self.get_device_status(self.device_id)
        await self.send_group_msg(
            'dashboard',
            json.dumps({
                'infoType': 'device_status',
                '1': device_status,
                '2':
self.format_datetime(device_last_online.astimezone(timezone.get_default_timezone(
))),
            })
        )

    async def send_door_status(self):
        door_status = await self.get_latest_door_status()
        await self.send_group_msg(
            'dashboard',
            json.dumps({
                'infoType': 'door_status',
                '1': door_status
            })
        )

    @staticmethod
    def convert_date(msg):
        time_string, date_string = msg['3'], msg['2']
        date_time_string = f'{date_string} {time_string}'
        if date_string[:4] == '1970':
            return dt.datetime.now()
        date_time_string = f'{date_string} {time_string}'
        return pytz.utc.localize(dt.datetime.strptime(date_time_string, "%Y-%m-%d
%H:%M:%S"))

    async def esp_token_auth(self):
        try:
            auth_header = self.scope.get('headers', [])
            auth_token = next((value for name, value in auth_header if name ==
b'authorization'), None)
            device_id = next((value for name, value in auth_header if name == b'device'),
None)
            if auth_token and device_id:
                token_key = auth_token.decode('utf-8')
                self.device_id = device_id.decode('utf-8')
            else:
                return False
```

```
            self.device_obj = await
database_sync_to_async(self.get_device)(self.device_id)
            token = self.device_obj.token_key
            if token != token_key:
                return False
            await self.set_device_status(True)
            await self.send_device_status()
        except ObjectDoesNotExist:
            return False
        return True

    @database_sync_to_async
    def set_device_status(self, is_online):
        device = ESPDevice.objects.get(device_id=self.device_id)
        device.is_online = is_online
        device.last_online = timezone.now()
        device.save()

    @database_sync_to_async
    def get_device_status(self, device_id):
        device = ESPDevice.objects.get(device_id=device_id)
        return ('online' if device.is_online else 'offline', device.last_online)

    async def connect(self):
        self.workspace = self.scope['url_route']['kwargs']['type']

        if self.workspace == 'esp':
            if not await self.esp_token_auth():
                await self.close()
                return
        else:
            # authenticate web browsers via session id
            if not self.scope['session'].get('_auth_user_id'):
                await self.close()
                return
            self.user = self.scope["user"]
            self.device_id = self.scope['session'].get('selected_device')
        group_name = f'{self.workspace}_{self.device_id}' if self.workspace !=
'status_select' else self.workspace
        await self.channel_layer.group_add(group_name, self.channel_name)
        await self.accept()

        if self.workspace == 'doorlog':
            await self.handle_logs_intro(self.get_doorlogs)
```

```
        elif self.workspace == 'devicelog':
            await self.handle_logs_intro(self.get_devicelogs)

        elif self.workspace == 'device_control':
            await self.send_esp_get_state()

        elif self.workspace == 'dashboard':
            await self.send_device_status()
            await self.send_door_status()

        elif self.workspace == 'status_select':
            device_ids = await self.get_device_ids_for_current_user()
            msg = {}
            for device_id in device_ids:
                device_status, _ = await self.get_device_status(device_id)
                msg[device_id] = device_status
            await self.send_group_msg('status_select', json.dumps(msg))

    async def disconnect(self, close_code):
        if self.workspace == 'esp':
            await self.set_device_status(True)
        await self.channel_layer.group_discard(self.workspace, self.channel_name)

    async def handle_esp_messages(self, text_data_json):
        if text_data_json['0'] == '2':  # if type == gerkon
            door_state = 'opened' if int(text_data_json['8']) else 'closed'
            doorlog = DoorLog(
                log_date=self.convert_date(text_data_json),
                door_state=door_state,
                device=self.device_obj
            )
            doorlog.log_date =
doorlog.log_date.astimezone(timezone.get_default_timezone())
            await database_sync_to_async(doorlog.save)()
            await self.send_group_msg('doorlog', self.model_to_json(doorlog))
            await self.send_door_status()

        elif text_data_json['0'] == '0':  # if type == device
            devicelog = DeviceLog(
                log_date=self.convert_date(text_data_json),
                message_type=ACTION_TYPES[text_data_json['5']],
                message_body=text_data_json['4'],
                device=self.device_obj
            )
```

```
            devicelog.log_date =
devicelog.log_date.astimezone(timezone.get_default_timezone())
            await database_sync_to_async(devicelog.save)()
            await self.send_group_msg('devicelog', self.model_to_json(devicelog))

        elif text_data_json['0'] == '1':  # if type == state
            await self.send_group_msg(
                'device_control',
                json.dumps({
                    '1': text_data_json['6'],
                    '2': text_data_json['7']
                })
            )
        elif text_data_json['0'] == '3':  # if type == temperature_and_humidity
            await self.send_group_msg(
                'dashboard',
                json.dumps({
                    'infoType': 'temp_and_humid',
                    'temp': text_data_json['9'],
                    'humid': text_data_json['10']
                })
            )

    async def receive(self, text_data):
        if self.workspace == 'esp':
            if not await self.esp_token_auth():
                await self.close()
                return
            text_data_json = json.loads(text_data)
            await self.handle_esp_messages(text_data_json)

        elif self.workspace == 'device_control':
            # check session authentication
            if not self.scope['session'].get('_auth_user_id'):
                await self.close()
                return
            await self.send_group_msg('esp', text_data)
            await self.send_esp_get_state()

    async def chat_message(self, event):
        message = event['message']
        await self.send(text_data=message)
```

**Додаток Ї**

Лістинг файлу routing.py

```
from django.urls import path

from . import consumers

websocket_urlpatterns = [
    path('ws/main/<str:type>', consumers.MainConsumer.as_asgi()),
]
```

**Додаток К**

Лістинг файлу urls.py

```
from django.urls import path
from .views import (
    IndexPageView, ControlDevicePageView, LogsPageView,
    LoginPageView, SelectDeviceView, DoorLogs
)
from django.contrib.auth.views import LogoutView

app_name = 'main'
urlpatterns = [
    path('', IndexPageView.as_view(), name='index'),
    path('control', ControlDevicePageView.as_view(), name='control_dev'),
    path('login', LoginPageView.as_view(), name='login'),
    path('logout/', LogoutView.as_view(next_page='main:login'), name='logout'),
    path('clogs', LogsPageView.as_view(), name='c_logs'),
    path('select_device', SelectDeviceView.as_view(), name='select_dev'),
    path('doorlogs', DoorLogs.as_view(), name='doorlogs')
]
```

**Додаток Л**

Лістинг файлу settings.py

```
from pathlib import Path

BASE_DIR = Path(__file__).resolve().parent.parent

SECRET_KEY = 'django-insecure-
#lu64ej&k&3gie(xh_b^6ds%^^bjeq(ou6n#5t$hn8_*2c-c5)'

DEBUG = True

ALLOWED_HOSTS = ['*']

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
```

```
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'main',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'control.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'control.wsgi.application'

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}


AUTH_PASSWORD_VALIDATORS = [
```

```
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'Europe/Kyiv'

USE_I18N = True

USE_TZ = True

LOGIN_URL = 'main:login'

STATIC_URL = '/staticfiles/'

STATIC_ROOT = '/var/www/'

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

ASGI_APPLICATION = "control.asgi.application"
CHANNEL_LAYERS = {
    "default": {
        "BACKEND": "channels_redis.core.RedisChannelLayer",
        "CONFIG": {
            "hosts": [("127.0.0.1", 6379)],
        },
    },
}

LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
```

```
        'file': {
            'level': 'DEBUG',
            'class': 'logging.FileHandler',
            'filename': '/home/kstamax/debug.log',
        },
    },
    'loggers': {
        'django': {
            'handlers': ['file'],
            'level': 'DEBUG',
            'propagate': True,
        },
    },
}
```

**Додаток М**

Лістинг файлу asgi.py

```
import os

from channels.auth import AuthMiddlewareStack
from channels.routing import ProtocolTypeRouter, URLRouter
from channels.security.websocket import AllowedHostsOriginValidator
from django.core.asgi import get_asgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'control.settings')
django_asgi_app = get_asgi_application()


import main.routing

application = ProtocolTypeRouter(
    {
        "http": django_asgi_app,
        "websocket": AllowedHostsOriginValidator(
            AuthMiddlewareStack(URLRouter(main.routing.websocket_urlpatterns))
        ),
    }
)
```

**Додаток Н**

Налаштування сайту nginx

```
server {
 listen 80;
 server_name _;
```

```
 client_max_body_size 20M;
 charset     utf-8;

location /staticfiles/ {
        root /var/www/;
    }

 location / {
            proxy_pass http://unix:/run/gunicorn.sock;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";

            proxy_redirect     off;
            proxy_set_header   Host $host;
            proxy_set_header   X-Real-IP $remote_addr;
            proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header   X-Forwarded-Host $server_name;
        }
}
```

**Додаток О**

Налаштування gunicorn Unix сокету

```
[Unit]
Description=gunicorn socket

[Socket]
ListenStream=/run/gunicorn.sock

[Install]
WantedBy=sockets.target
```

**Додаток П**

Налаштування сервісу gunicorn

```
[Unit]
Description=gunicorn daemon
Requires=gunicorn.socket
After=network.target

[Service]
User=kstamax
Group=kstamax
WorkingDirectory=/home/kstamax/dyplom_http_server/control
```

```
ExecStart=/home/kstamax/dyplom_http_server/venv/bin/gunicorn
control.asgi:application -w 4 -k uvicorn.workers.UvicornWorker --bind
unix:/run/gunicorn.sock
```

**Додаток Р**

Лістинг файлу dashboard.js

```
var device_status = document.getElementById("deviceStatus");
var last_online = document.getElementById("lastOnline");
var last_known_door_status = document.getElementById("doorStatus");
var humid = document.getElementById("humid");
var temp = document.getElementById("temp");
const chatSocket = new WebSocket(
    'ws://' +
    window.location.host +
    '/ws/main/dashboard'
);
chatSocket.onmessage = function(e) {
    var data = JSON.parse(e.data);
    if (data['infoType']=='device_status'){
        if (data[1]=="offline"){
            device_status.classList.remove('text-success');
            device_status.classList.add('text-danger');
            device_status.textContent = 'Device is offline';
            last_online.textContent = data[2];
        }
        else {
            last_online.textContent = '';
            device_status.classList.remove('text-danger');
            device_status.classList.add('text-success');
            device_status.textContent = 'Device is online';
        }
    }
    else if (data['infoType']=='door_status'){
        if (data[1]=="opened"){
            last_known_door_status.classList.remove('text-danger');
            last_known_door_status.classList.add('text-success');
            last_known_door_status.textContent = 'Door is opened';
        }
        else if (data[1]=="closed"){
            last_known_door_status.classList.remove('text-success');
            last_known_door_status.classList.add('text-danger');
            last_known_door_status.textContent = 'Door is closed';
        }
    }
    else if (data['infoType']=='temp_and_humid'){
```

```
        temp.textContent = data['temp'] + '°C';
        humid.textContent = data['humid'] + '%';
    }
};

chatSocket.onclose = function(e) {
    console.error('Chat socket closed unexpectedly');
};
```

**Додаток С**

Лістинг файлу device_contorl.js

```
var relay_button = document.getElementById("relay_button");
var led_button = document.getElementById("led_button");
var reset_button = document.getElementById("reset_button")
const chatSocket = new WebSocket(
    'ws://' +
    window.location.host +
    '/ws/main/device_control'
);
chatSocket.onmessage = function(e) {
    var data = JSON.parse(e.data);
    console.log(data);
    if (data[1]=="0"){
        led_button.classList.remove('btn-secondary');
        led_button.classList.remove('btn-success');
        led_button.classList.add('btn-danger');
        led_button.textContent = 'Turn Off';
    }
    else if (data[1]=="1"){
        led_button.classList.remove('btn-secondary');
        led_button.classList.remove('btn-danger');
        led_button.classList.add('btn-success');
        led_button.textContent = 'Turn On';
    }

    if (data[2]=="1"){
        relay_button.classList.remove('btn-secondary');
        relay_button.classList.remove('btn-success');
        relay_button.classList.add('btn-danger');
        relay_button.textContent = 'Close';
    }
    else if(data[2]=="0"){
        relay_button.classList.remove('btn-secondary');
        relay_button.classList.remove('btn-danger');
        relay_button.classList.add('btn-success');
```

```
        relay_button.textContent = 'Open';
    }
};

chatSocket.onclose = function(e) {
    console.error('Chat socket closed unexpectedly');
};

led_button.onclick = function() {
    if (led_button.textContent == 'Turn Off'){
        chatSocket.send('01');
    } else if (led_button.textContent == 'Turn On'){
        chatSocket.send('00');
    }
};

relay_button.onclick = function() {
    if (relay_button.textContent == 'Close'){
        chatSocket.send('10');
    } else if (relay_button.textContent == 'Open'){
        chatSocket.send('11');
    }
};
reset_button.onclick = function() {
    chatSocket.send('2')
}
```

**Додаток Т**

Лістинг файлу device_status_select.js

```
const chatSocket = new WebSocket(
    'ws://' +
    window.location.host +
    '/ws/main/status_select'
);
chatSocket.onmessage = function(e) {
    var data = JSON.parse(e.data);
    for (let row in data){
        var device_status = document.getElementById("deviceStatus_"+row);
        if (data[row]=="offline"){
            device_status.classList.remove('text-success');
            device_status.classList.add('text-danger');
            device_status.textContent = 'Device is offline';
        }
        else {
            device_status.classList.remove('text-danger');
```

```
            device_status.classList.add('text-success');
            device_status.textContent = 'Device is online';
        }
    }
};

chatSocket.onclose = function(e) {
    console.error('Chat socket closed unexpectedly');
};
```

**Додаток У**

Лістинг файлу devicelog.js

```
const lst = document.getElementById("itemsList");
const loader = document.getElementsByClassName("loader")[0]
const chatSocket = new WebSocket(
    'ws://' +
    window.location.host +
    '/ws/main/devicelog'
);
chatSocket.onmessage = function(e) {
    var list = JSON.parse(e.data);
    for (var i in list) {
        text_color = ''
        if (list[i]['fields']['message_type'] === 'INFO') {
            text_color = 'text-primary'
        } else {
            text_color = 'text-danger'
        }
        var item = `
                <li class="list-group-item
${text_color}">${list[i]['fields']['message_type']}
${isoToDateTime(list[i]['fields']['log_date'])}
${list[i]['fields']['message_body']}</li>
                `;
        lst.innerHTML = item + lst.innerHTML;
    }
    loader.remove()
};

chatSocket.onclose = function(e) {
    console.error('Chat socket closed unexpectedly');
};

function isoToDateTime(isoDate) {
    const date = new Date(isoDate);
```

```
    const year = date.getFullYear();
    const month = ("0" + (date.getMonth() + 1)).slice(-2);
    const day = ("0" + date.getDate()).slice(-2);
    const hours = ("0" + date.getHours()).slice(-2);
    const minutes = ("0" + date.getMinutes()).slice(-2);
    const seconds = ("0" + date.getSeconds()).slice(-2);
    return `${year}-${month}-${day} ${hours}:${minutes}:${seconds}`;
  }
```

**Додаток Ф**

Лістинг файлу doorlog.js

```
const lst = document.getElementById("itemsList");
const loader = document.getElementsByClassName("loader")[0]
const chatSocket = new WebSocket(
    'ws://' +
    window.location.host +
    '/ws/main/doorlog'
);
chatSocket.onmessage = function(e) {
    var list = JSON.parse(e.data);
    for (var i in list) {
        text_color = ''
        if (list[i]['fields']['door_state'] === 'opened') {
            text_color = 'text-success'
        } else {
            text_color = 'text-danger'
        }
        var item = `
                <li class="list-group-item
${text_color}">${isoToDateTime(list[i]['fields']['log_date'])}
${list[i]['fields']['door_state']}</li>
                `;
        lst.innerHTML = item + lst.innerHTML;
    }
    loader.remove()
};

chatSocket.onclose = function(e) {
    console.error('Chat socket closed unexpectedly');
};

function isoToDateTime(isoDate) {
    const date = new Date(isoDate);
    const year = date.getFullYear();
    const month = ("0" + (date.getMonth() + 1)).slice(-2);
```

```
    const day = ("0" + date.getDate()).slice(-2);
    const hours = ("0" + date.getHours()).slice(-2);
    const minutes = ("0" + date.getMinutes()).slice(-2);
    const seconds = ("0" + date.getSeconds()).slice(-2);
    return `${year}-${month}-${day} ${hours}:${minutes}:${seconds}`;
  }
```

**Додаток Х**

Лістинг файлу header_style.css

```
.hoverLink:hover {
    font-size: 1.05em;
    background-color: rgba(255, 255, 255, 0.1)
}

select {
    -webkit-appearance: listbox !important;
}

.loader {
    border: 6px solid #f3f3f3; /* Light grey */
    border-top: 6px solid red;
    border-radius: 50%;
    width: 30px;
    height: 30px;
    animation: spin 2s linear infinite;
  }

@keyframes spin {
  0% { transform: rotate(0deg); }
  100% { transform: rotate(360deg); }
}
```