МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ
Навчально-науковий інститут кібернетики, інформаційних технологій та інженерії
Кафедра комп'ютерних наук та прикладної математики

"До захисту допущена"
Зав. кафедри комп'ютерних наук
та прикладної математики
д.т н., проф. Ю.В. Турбал
«____» _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

Проєктування та розробка веб-орієнтованої системи моніторингу домашніх пристроїв на основі Arduino, ASP.NET Core та React

Виконала: Мерцалова Ірина Сергіївна _____
(прізвище, ім’я, по батькові) (підпис)
група ІПЗ–41

Керівник: к.т.н., доцент, доцент Жуковський В.В. _____
(науковий ступінь, вчене звання, посада, прізвище, ініціали) (підпис)

Рівне – 2025

Національний університет водного господарства та природокористування

(повне найменування вищого навчального закладу)

Навчально-науковий інститут кібернетики, інформаційних технологій та інженерії

Кафедра **комп'ютерних наук та прикладної математики**

Рівень вищої освіти: **Бакалаврський (перший)**

Галузь знань 12 Інформаційні технології

(шифр і назва)

Спеціальність 121 Інженерія програмного забезпечення (Інтернет речей)

(шифр і назва)

«ЗАТВЕРДЖУЮ»

Завідувач кафедри

д.т н., проф. Ю.В. Турбал

«____» _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачу Мерцаловій Ірині Сергіївні

1. Тема роботи *"Проєктування та розробка веб-орієнтованої системи моніторингу домашніх пристроїв на основі Arduino, ASP.NET Core та React"*
керівник роботи *Жуковський Віктор Володимирович, к.т.н., доцент, д кафедри комп'ютерних наук та прикладної математики.*
затверджені наказом по університету від _____

2. Термін подання роботи студентом _____

3. Вихідні дані до роботи: *сучасні технології ASP.NET Core 8, React, апаратна частина проєкту та перелік функціональних можливостей проєкту*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): *Дослідження аналогів, проєктування архітектури, апаратна*

розробка та реалізація програмного забезпечення.

5. Перелік графічного матеріалу: *Мультимедійна презентація*

6. Консультанти розділів проєкту (роботи):

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ 1	*доцент Жуковський В.В.*		
Розділ 2	*доцент Жуковський В.В., старший викладач Повшенюк А.П.*		
Розділ 3	*доцент Жуковський В.В.*		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	Аналіз наукових джерел та прикладних рішень у сфері моніторингу		
2.	Проєктування загальної архітектури веборієнтованої системи		
3.	Розробка прототипу клієнт-серверного застосунку		
4.	Реалізація основних функцій системи		
5.	Інтеграція Arduino-пристрою з вебсистемою		
6.	Проведення функціонального тестування системи та усунення виявлених недоліків		
7.	Узагальнення результатів розробки та формулювання висновків		
8.	Підготовка пояснювальної записки кваліфікаційної роботи		
9.	Створення мультимедійної презентації та підготовка до захисту проєкту		

Здобувач: _____ **Мерцалова І.С.**

(підпис) (прізвище та ініціали)

Керівник: _____ **Жуковський В.В.**

(підпис) (прізвище та ініціали)

ЗМІСТ

РЕФЕРАТ

Кваліфікаційна робота: 64 с., 22 рисунки, 3 таблиць 18 джерел.

Мета роботи: розробка та впровадження веб-орієнтованої системи моніторингу стану домашнього середовища з використанням мікроконтролера Arduino, індивідуально розробленого API та клієнтської частини React для відображення всіх функціональних можливостей проєкту.

Об'єкт дослідження – процеси та технології реалізації систем моніторингу параметрів середовища у розумному будинку.

Предмет дослідження – архітектура, функціональні можливості та стійкість програмного забезпечення системи, зокрема засоби передачі, збереження, візуалізації та взаємодія даних.

Методи вивчення:

1) аналіз вимог до програмних систем;
2) проєктування архітектури клієнт-серверної системи;
3) моделювання та реалізація програмних компонентів;
4) перевірка роботи системи на базі Arduino.

Проведено дослідження сучасних веб-технологій, засобів взаємодії з мікроконтролерами Arduino та підходів до аналізу стійкості програмного забезпечення в контексті розробки системи моніторингу для розумного будинку.

Розглянуто функціональні можливості кожного з проєктів системи – Arduino, ASP.NET Core та React – а також особливості їх взаємодії. Розроблено прототип платформи, що дозволяє віддалено збирати, передавати й візуалізувати дані про стан будинку та проведено тестування її функціональності, продуктивності та стабільності в умовах використання.

Ключові слова: РОЗУМНИЙ БУДИНОК, СИСТЕМА МОНІТОРИНГУ, ARDUINO, ASP.NET CORE, REACT, SIGNALR, СТІЙКІСТЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.

ВСТУП

Сьогодні, коли стрімко розвиваються цифрові технології, автоматизація охоплює дедалі більше сфер життя: від промисловості та транспорту – до побуту, освітніх процесів і навіть сільського господарства. Саме тому, власний будинок стає одним із пріоритетних об’єктів для цифровізації, адже кожна людина прагне зробити свій дім максимально зручним, безпечним і адаптованим до власних потреб. У зв’язку з цим “розумні” пристрої активно впроваджуються та розвиваються, щоб відповідати вимогам стабільності та надійності в умовах реальної експлуатації. Метою цієї роботи було реалізувати систему для автоматизації процесу контролю стану будинку з можливістю доступу до актуальної інформації через веб-інтерфейс.

Для створення інтелектуальної системи моніторингу домашніх пристроїв необхідне глибоке розуміння принципів розробки програмного забезпечення, зокрема – клієнт-серверної архітектури, роботи з Arduino-пристроями, а також сучасних веб-технологій. Система має не лише забезпечувати збір та обробку даних з датчиків, а й надавати користувачеві зручний та простий інтерфейс для контролю й перегляду інформації у режимі реального часу.

У процесі розробки аналізувалися аспекти стійкості та надійності програмного забезпечення, способи ефективної організації обміну даними між мікроконтролером Arduino та сервером, а також інтеграція з frontend-частиною через сучасні інтерфейси. Особливу увагу приділено дослідженню архітектурних рішень та застосуванню практик, що сприяють підтримуваності та масштабованості системи.

Попри розвиток інструментів для розробки розподілених рішень, все ще залишаються актуальними питання складності інтеграції апаратних та програмних компонентів, обмеження ресурсів вбудованих пристроїв, а також забезпечення стабільної роботи в різних умовах реального застосування. Розв’язання цих задач та питань вимагає поєднання теоретичних знань і практичних навичок у галузях IoT, мережевої взаємодії та веб-програмування.

Сьогодні в багатьох виникає потреба стежити за станом будинку – температурою, вологістю, рівнем газу та іншими показниками. Більшість готових рішень або дорогі, або складні у налаштуванні. Це особливо відчутно вдома, де хочеться мати просту та функціонально розвинену систему, яка буде спрощувати життя для користувачів.

Часто проблеми виникають ще на етапі підключення: потрібно розібратись з різними пристроями, форматами даних, серверною частиною або незрозумілість застосування системи для користувачів без досвіду впровадження схожих рішень.

Цей проєкт має на мені зробити просту у використанні систему для домашнього моніторингу, яку можна адаптувати та масштабувати в залежності від потреб клієнта. Вона використовує недорогі компоненти, функціональне API та мінімалістичний вебсайт для зручного перегляду інформації.

Метою роботи є розробка та впровадження веб-орієнтованої системи моніторингу стану домашнього середовища з використанням мікроконтролера Arduino, індивідуально розробленого API та клієнтської частини React для відображення всіх функціональних можливостей проєкту.

Основними завданнями роботи є розробка архітектури системи моніторингу домашнього середовища, реалізація передачі та обробки даних з мікроконтролера Arduino через API до вебсайту, створення клієнтської частини для візуалізації інформації та впровадження механізму прив’язки пристроїв до користувацьких облікових записів.

Об'єкт дослідження – процеси та технології реалізації систем моніторингу параметрів середовища у розумному будинку.

Предмет дослідження – архітектура, функціональні можливості та стійкість програмного забезпечення системи, зокрема засоби передачі, збереження, візуалізації та взаємодія даних.

РОЗДІЛ 1
АРХІТЕКТУРНІ ТА ТЕХНІЧНІ АСПЕКТИ ПОБУДОВИ ВЕБОРІЄНТОВАНИХ МОНІТОРИНГОВИХ СИСТЕМ

1.1. Основні поняття щодо моніторингу середовища (стану будинку) та архітектури IoT-рішень

Моніторинг стану середовища в умовах житлового простору передбачає регулярне зчитування найбільш важливих параметрів, таких як температура, вологість, рівень газу тощо. Такі дані дозволяють користувачу оперативно відстежувати зміни в стані показників та вчасно реагувати на потенційно небезпечні відхилення від встановлених користувачем норм.

Архітектура Інтернету речей (IoT) – це базова структура, яка дозволяє різним пристроям у системі IoT безперебійно працювати разом [1]. Для реалізації подібних задач активно застосовуються технології Інтернету речей, що передбачають взаємодію мікроконтролерів, сенсорів, програмної логіки та мережевої інфраструктури.

Типова IoT-архітектура таких системи складається з чотирьох рівнів:

1) збір даних (датчики та мікроконтролер Arduino);
2) передача даних (мережеве з’єднання);
3) обробка даних (вебсервіс / API);
4) візуалізація та взаємодія (вебінтерфейс на основі React) [2].

У розробці подібних систем важливим є не лише збір та збереження даних, але й забезпечення надійного з’єднання, гнучкої архітектури, зручного інтерфейсу та можливості масштабування. Саме поєднання апаратної та програмної частин дозволяє створити стійке, доступне та практичне рішення для домашнього користування.

1.2. Аналіз принципів передавання даних у мережевих протоколах

Мережевий протокол – це набір правил та інструкцій, які визначають, як дані повинні передаватися через мережу. Вони функціонують як посередники

між передавачем та приймачем даних, забезпечуючи правильну і безперебійну передачу інформації [3]. Робота програмно-апаратної системи успішна перш за все завдяки надійному та своєчасному обміну даними між трьома логічними модулями:

1. Апаратний модуль (Arduino + Ehernet Shiled + датчики);
2. Серверна частина (ASP.NET Core Web API із SignalR Hub);
3. Клієнтський вебінтерфейс (React).

Щоб система контролю будинку працювала ефективно, з мінімальними затримками та втратами даних, важливо обрати протоколи передавання даних. Щоб обґрунтувати вибір засобів передавання, проаналізовано низку протоколів – TCP, UDP, HTTP та WebSocket – для обґрунтування їх придатності в реальних умовах експлуатації.

1.2.1. Характеристика мережевих протоколів

TCP (Transmission Control Protocol) – це транспортний протокол, який використовується для забезпечення надійної передачі даних та гарантує доставку пакетів без втрат [3].

UDP (User Datagram Protocol) – це транспортний протокол, який забезпечує швидку передачу даних без перевірки цілісності, що підходить для додатків, де швидкість важливіша за надійність (потокове відео або онлайн-ігри) [3].

HTTP (HyperText Transfer Protocol) – це протокол прикладного рівня для звичайних з’єднань, який забезпечує зв’язок між сервером та клієнтом [4]. Його архітектура запит-відповідь добре підходить для REST API.

WebSocket – це двонаправлений повнодуплексний протокол зв’язку між клієнтом та сервером, який працює за принципом постійного з’єднання, де обидві сторони (сервер та клієнт) можуть одночасно надсилати й отримувати дані [5].

1.2.2. Дослідження та порівняння протоколів

Щоб краще зрозуміти специфіку кожного з розглянутих протоколів важливо порівняти їх за основними властивостями – це дозволить чітко окреслити, який із них найбільше підходить для реалізації цілей проєкту.

У табл. 1.1 наведено короткий огляд ключових характеристик. Таким чином, таблиця допомагає зрозуміти сильні і слабкі сторони кожного з протоколів та визначити найкращий вибір в контексті проєктування систем.

Таблиця 1.1

Порівняльна таблиця протоколів

Характеристика	**TCP**	**UDP**	**HTTP**	**WebSocket**
Тип з'єднання	Встановлення з'єднання (connection-oriented)	Без з’єднання	Без стану (stateless), зазвичай поверх TCP	Постійне, двохстороннє з’єднання (persistent, full-duplex)
Надійність	Висока	Низька	Надійна (успадковує від TCP)	Надійна (успадковує від TCP)
Швидкість	Відносно повільний	Дуже швидкий	Відносно повільний	Швидкий
Порядок пакетів	Гарантований	Не гарантований	Гарантований	Гарантований
Контроль потоку	Є	Немає	Є	Є
Контроль перенавантаження	Є	Немає	Є	Є
Використання	Передача файлів (FTP), електронна пошта, вебперегляд	Онлайн-ігри, потокове відео/аудіо, DNS, VolP	Передача вебсторінок, API-виклики (REST)	Чат-додатки, онлайн- ігри, фінансові тікери, push-сповіщення

1.3. Огляд сучасних рішень у сфері “розумного дому”

Технології для “розумного дому” активно розвиваються та знаходять широке застосування в повсякденному житті. Серед основних функцій таких систем: моніторинг стану середовища (температури, вологості, якості повітря), управління пристроями, автоматизація сценаріїв дій та інтеграція з іншими цифровими сервісами. Існує багато готових рішень, які пропонують різний

рівень функціональності, масштабованості та гнучкості до розширення, розглянемо найбільш популярні з них.

1.3.1. HomeAssistant

HomeAssistant – це універсальна операційна система з відкритим вихідним кодом для керування IoT-пристроями різних виробників [6], а також з потужними можливостями інтеграції та гнучкою автоматизацією.

Переваги:

1. Відкрите ПЗ з активною спільнотою;
2. Підтримка понад 2000 пристроїв і інтеграцій;
3. Повний локальний контроль;
4. Гнучка автоматизація (YAML, UI, Node-RED);
5. Можливості кастомізації інтерфейсу.

Недоліки:

1. Вимагає технічних знань для налаштування;
2. Може бути складною для пересічного користувача;
3. Потребує окремого хосту (Rasberry Pi, сервер тощо).

1.3.2. Google Home

Google Home – це додаток, який дозволяє налаштовувати, керувати та контролювати пристрої розумного дому (Google Nest, Wifi, Chromecast, світильники, камери, термостати тощо) з одного додатку [7].

Переваги:

1. Зручність для початківців;
2. Голосове управління через Google Assistant;
3. Інтеграція з Android, YouTube, Google Calendar, тощо;
4. Підтримка великої кількості пристроїв.

Недоліки:

1. Обмежена кастомізація;
2. Залежність від хмари (немає локального управління);

3. Погана інтеграція з неофіційними або DIY-пристроями.

1.3.3. Apple HomeKit

Apple HomeKit – це більше, ніж система розумного будинку, це додаток, який можна назвати пультом управління твоїм домашнім космічним кораблем з контролем освітлення, клімату, безпеки тощо, просто кажучи, команди Siri [8].

Переваги:

1. Глибока інтеграція з IOS, macOs, Siri;
2. Високий рівень безпеки та конфіденційності;
3. Простота налаштування для користувачів Apple;
4. Підтримка локального управління (через HomeHub).

Недоліки:

1. Працює лише з сертифікованими HomeKit-пристроями;
2. Дорожча екосистема;
3. Обмежена кількість сумісного обладнання.

1.4. Можливості та переваги використання Arduino в системах моніторингу

У контексті швидкого розвитку систем IoT виникає необхідність у надійних, доступних і простих у використанні апаратних рішеннях, здатних ефективно здійснювати збір, обробку та миттєву передачу даних. Однією з найпоширеніших платформ для реалізації таких систем є Arduino – відкритий апаратно-програмний комплекс, який забезпечує базову функціональність для широкого спектра завдань у галузі моніторингу.

Arduino здобула свою популярність завдяки своїй модульності, простоті програмування, наявності великої кількості бібліотек та широкій спільноті. У системах моніторингу вона виконує роль центрального модуля, який приймає дані з підключених сенсорів, здійснює їх попередню обробку та передає у вебсистему або мобільний інтерфейс користувача.

1.4.1. Можливості Arduino в системах моніторингу

Загальна архітектура системи передбачає використання Arduino як базової апаратної платформи. Саме через свою надійність, модульність та сумісність із широким спектром датчиків вона забезпечує реалізацію ключових функцій:

1. Зчитування даних з сенсорів: температура, вологість, рівень газу та інші показники з навколишнього середовища за допомогою відповідних датчиків;
2. Попередня обробка даних: фільтрація шуму, нормалізація значень та базова валідація;
3. Виведення інформації на дисплей: за допомогою TFT LCD 3.5” користувач може взаємодіяти з пристроєм, переглядати стан або вводити дані (в контексті проєкту, pairing-token);
4. Передача даних у мережу: передавання поточних показників на сервер за допомогою Ethernet Shield, з використанням HTTP-запитів або через WebSocket (SignalR);
5. Стійкість до збоїв: здатність зберігати працездатність навіть при короткочасних проблемах з мережею або живленням.

1.4.2. Основні переваги використання Arduino

Вибір Arduino як апаратної платформи зумовлений її перевагами, які суттєво спрощують розробку рішень. Особливо це актуально для систем, де важлива швидкість розгортання та безпека. Головними факторами, які визначають ефективність Arduino у проєкті, є:

1. Доступність та простота використання: відсутність складного налаштування коду та зрозуміле середовище розробки (Arduino IDE);
2. Гнучкість та модульність: підтримка різних типів сенсорів, модулів, дисплеїв тощо;
3. Надійність у реальному середовищі: стабільність роботи в умовах автономної експлуатації, включаючи температурні коливання, нестабільну мережу або енергопостачання;

4. Енергозбереження: мінімальне споживання енергії завдяки відсутності повноцінної операційної системи, що є критично для багатьох IoT-рішень;
5. Широка підтримка та документація: одна з найбільших мікроконтролерних платформ, яка має одну з найбільших спільнот, що полегшує пошук відкритих рішень та бібліотек.

РОЗДІЛ 2
ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОЄКТУ

2.1. Постановка задачі та методологія розробки

У сучасних умовах цифрової трансформації побуту особливого значення набуває автоматизація моніторингу стану будинку. Інтелектуальні системи в режимі миттєвого відстежування критичних параметрів – температура, вологість, наявність газу чи диму – не лише підвищують рівень комфорту, а й сприяють безпеці житла. Саме ці міркування стали основою для розробки комплексного проєкту, метою якого є розробка та впровадження веб-орієнтованої системи моніторингу стану домашнього середовища з можливістю подальшого масштабування, гнучкої адаптації до нових умов та розширення функціональності.

Розробка системи передбачає розв’язання низки технічних і програмних задач, пов’язаних зі збором, обробкою, передаванням, збереженням і візуалізацією даних.

Архітектурно рішення поділяється на три ключові компоненти:

1. мікроконтролерна частина: Arduino;
2. серверна логіка: ASP.NET Core Web API;
3. клієнтський вебінтерфейс: React.

Взаємодія між цими компонентами організована з урахуванням сучасних підходів до побудови веборієнтованих систем.

2.1.1. Завдання проєкту

Для досягнення поставленої мети було сформовано такі основні завдання:

1. Розробити прошивку для мікроконтролера Arduino, що дозволяє збирати дані з підключених сенсорів (температури, вологості, газу);
2. Організувати передачу показників на сервер за допомогою API-запитів через Ethernet Shield;

3. Реалізувати API на основі ASP.NET Core, яке прийматиме, оброблятиме та зберігатиме інформацію в базу даних, а також надаватиме їх клієнтській стороні через авторизовані запити;
4. Спроєктувати реляційну модель бази даних з підтримкою поточних значень, прив’язки пристроїв до користувачів, а також реєстрації pairing-токенів;
5. Реалізувати вебінтерфейс за допомогою React, який забезпечить реєстрація та авторизацію користувача, управління пристроями, візуалізація даних по них, вивід інформації по датчиках через графіки та базові налаштування системи;
6. Налаштувати механізм прив’язки пристрою до облікового запису користувача за допомогою pairing-коду, включаючи його генерацію, введення та підтвердження з’єднання;
7. Забезпечити псевдореальний режим оновлення даних, щоб дозволити серверу надсилати нові дані до клієнта для відображення без потреби постійного API-запиту;
8. Додати налаштування автоматичних дій для пристрою за заданими користувачем умовами;
9. Реалізація системи сповіщень користувача про критичні зміни у стані будинку;
10. Впровадити локальне тестування усіх можливих компонентів системи для подальшого розгортання та масштабування.

У ході реалізації було обрано інкрементальний підхід до розробки – це методологія, яка передбачає побудову системи невеликими, керованими частинами або інкрементами, де кожен з елементів представляє частину функціональності системи. Такий метод дозволив виявляти і усувати проблеми на ранніх етапах розробки та забезпечити гнучке масштабування функціоналу.

2.2. Архітектура та принципи проєктування системи

Для розробки надійної та ефективної веборієнтованої системи важливим є ґрунтовний архітектурний підхід. Завдяки йому забезпечується зрозумілість структури, стійкість до змін, можливість масштабування та легкість у подальшій підтримці. Продумане проєктування дозволяє чітко розділити відповідальність між компонентами системи, що позитивно впливає на її стабільність та розвиток.

2.2.1. Архітектурна модель системи

Система побудована за принципами клієнт-серверна архітектура, де кожен компонент виконує окрему роль у загальній структурі проєкту.

- *Клієнтська частина* – реалізована за допомогою React і відповідає за візуалізацію доступного функціоналу та взаємодію з користувачем.
- *Серверна частина* – реалізована на ASP.NET Core Web API, включає REST API, а також канал SignalR для обміну даними в режимі реального часу.
- *Апаратна частина* – Arduino-пристрій, який збирає показники (температура, вологість, рівень газу тощо) та передає їх на сервер.
- *База даних* – відповідає за зберігання інформації про користувачів, пристрої, дані показників та pairing токени.

Взаємодія між компонентами відбувається через чітко визначені канали зв’язку: REST API для запитів (авторизація, отримання актуальних даних тощо) та через SignalR для передачі поточних даних в режимі live. Комунікація з базою даних реалізована на рівні серверного застосунку через EntityFrameworkCore (рис. 2.1).

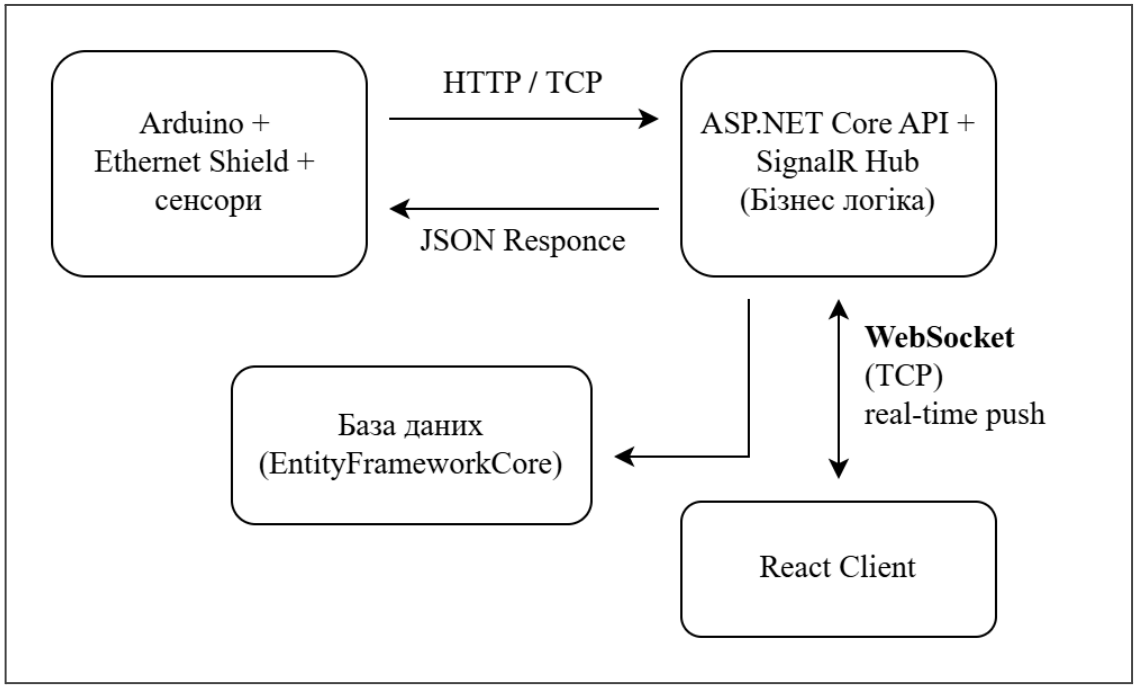

Рис. 2.1. Діаграма компонентів системи та їх взаємодія

2.2.2. Організація коду та архітектурна стійкість

Під час розробки системи особлива увага приділялася дотриманню принципів чистої архітектури та забезпеченню підтримуваності коду. Було реалізовано низку практик, що сприяють масштабованості, модульності та тестованості рішення.

- **Принцип єдиної відповідальності** (Single Responsibility Principle) – це один з основоположних принципів SOLID, який стверджує, що кожен клас, модуль або функція повинні мати лише одну відповідальність. Це означає, що код повинен бути розбитий на логічні частини, кожна з яких виконує чітко окреслену задачу [9]:
  - Контролери (Controllers) відповідають лише за прийом HTTP-запитів і делегування бізнес-логіки.
  - Сервіси (Services) інкапсулюють функції, оперуючи між DTO (Data Transfer Object) та репозиторіями.
  - Контекст бази даних (Data Access Layer) відповідають за роботу з EntityFrameworkCore і доступ до даних.

- Застосоване **шарове розділення** (Layered Architecture) – це підхід до проектування, де проєкт поділяється на окремі шари, якими можна керувати та підтримувати незалежно [10]:
    - Контролери “не мають” знання про реалізацію доступу до бази даних.
    - Сервіси працюють лише з абстракціями, не прив’язані до конкретних технологій зберігання;
    - Це спрощує тестування та повторне використання окремих частин системи.

- Дані між клієнтом (frontend) і сервером (backend) передаються через **DTO-моделі** (Data Transfer Objects), які повністю відокремлені від Entity-класів бази даних. Такий підхід:
    - Знижує ризик витоку важливих полів при серіалізації.
    - Дозволяє точно контролювати структуру API-відповідей.
    - Сприяє адаптації API до змін у бізнес-логіці без порушення внутрішньої структури БД.

- **Dependency Injection** (DI) – фундаментальна частина ASP.NET Core – це патерн проектування, який використовується для створення залежностей між компонентами [11]:
    - Впровадження інтерфейсів сервісів.
    - Реєстрація залежностей в Program.cs файлі через scoped- або singelton-екземплярів на один життєвий цикл.
    - Слабке зв’язування компонентів, покращення тестованості та дозвіл використовувати мок-об’єкти при тестуванні.

Усі ці рішення дозволили створити гнучку, легко підтримувану та розширювальну систему, яка поєднує фізичні пристрої з веборієнтованим інтерфейсом у рамках стійкої архітектури.

2.3. Апаратна платформа IoT-рішення

Для реалізації системи обрано платформу Arduino у поєднанні з Ethernet Shield, що дозволяє організувати передачу даних у локальну мережу або безпосередньо до хмарного API.

Arduino є однією з найбільш популярних платформ для побудови IoT-прототипу завдяки своїй відкритості, широкому асортименту сумісних модулів, низькій вартості та простоті розробки.

Використання Ethernet-модуля, порівняно з Wi-Fi-рішеннями, обумовлено бажанням забезпечити стабільне та передбачуване мережеве з’єднання, що є критичним для систем реального часу.

2.3.1. Вибір середовища реалізації та ресурсів

Для реалізації проєкту було обрано низку апаратних та програмних ресурсів, які забезпечують стабільну роботу, зручну розробку та ефективну інтеграцію всіх компонентів системи. Нижче наведено перелік ключових інструментів та бібліотек, які було використаними у процесі впровадження.

- Середовище розробки**:** Arduino IDE (2.2.1)
- Мова програмування: C++ 10.0
- Плата з мікроконтролером: Arduino Mega 2560 [12]
- Використані компоненти:
    - Arduino Ethernet Shield 2
    - DHT22
    - MQ2
    - 3.5” TFT LCD Shield
    - Світлодіоди (LED)
    - Raspberry Pi Compatible Cooling Fan
    - Макетна плата
    - Перемички
- Бібліотеки:
    - PI

- Ethernet.
- ArduinoHttpClient
- ArduinoJson
- DHT
- MCUFRIEND_kbv
- TouchScreen
- Adafruit_GFX

2.3.2. Склад апаратної частини

Arduino Mega 2560 – це покращена плата мікроконтролера Arduino, яка реалізована на мікроконтролері ATmega2560, яка використовується для обробки сигналів від сенсорів та вирізняється підвищеною надійністю, великою кількістю входів/виходів та розширеними додатковими параметрами:

- Аналогові входи: 16 шт.;
- Цифрові виходи/входи: 54 шт. (15 з яких забезпечують вихід PWM/ШІМ);
- Флеш-пам'ять: 256 кб, з яких 8 КБ використовуються завантажувачем;
- Частота тактова: 16 МГц;
- Робоча напруга плати Arduino Mega 2560: 5V – 12V.

Мікроконтролер дуже добре себе зарекомендував у великих проєктах автоматизації й побудові розумних будинків, завдяки збільшеній пам'яті, в яку можна записати багато коду для виконання [12].

Ця платформа є найбільш популярною серед розробників IoT-систем та автоматизованих рішень та підтримує численні бібліотеки і модулі (датчиків, дисплеїв та виконавчі механізми). Саме тому дана плата була обрана для реалізації розробленої системи моніторингу (рис. 2.2).

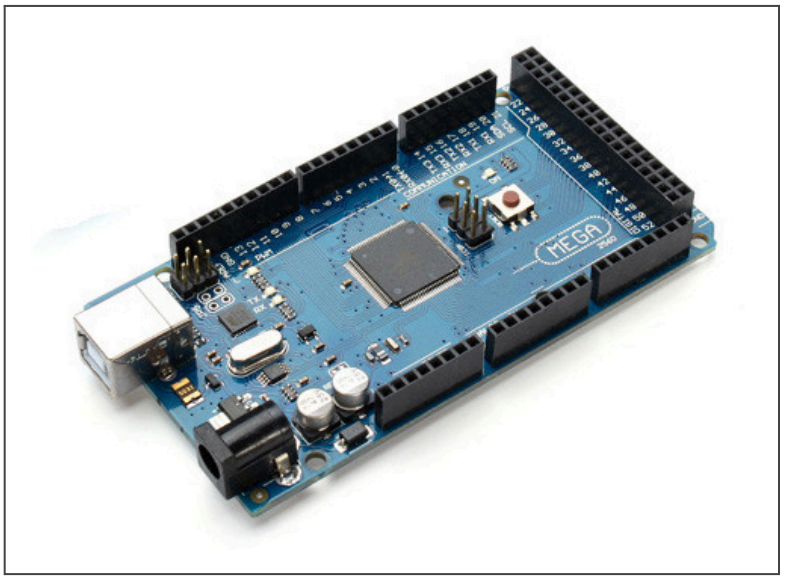

Рис. 2.2. Мікроконтролер Arduino Mega

Arduino Ethernet Shield 2 – це модуль розширення, який дозволяє Arduino-платі підключитися до Інтернету за допомогою кабелю RJ45. Шилд побудований на базі чипі Ethernet Wiznet W5500, який реалізує повноцінний IP-стек з підтримкою TCP та UDP протоколів.

Основні технічні характеристики:

- Робоча напруга: 5В (подається від плати Arduino);
- Ethernet-контролер: W5500 з вбудованим буфером об’ємом 32 КБ;
- Швидкість передавання даних: 10/100 Мбіт/с;
- Інтерфейс з’єднання з Arduino: через порт SPI (Serial Peripheral Interface) [13].

Для реалізації програмної логіки з використанням цього модуля були застосовані такі бібліотеки: SPI, Ethernet, ArduinoHttpClient, ArduinoJson – вони забезпечують роботу з мережею, створення HTTP-запитів та обробку JSON-даних (рис. 2.3).

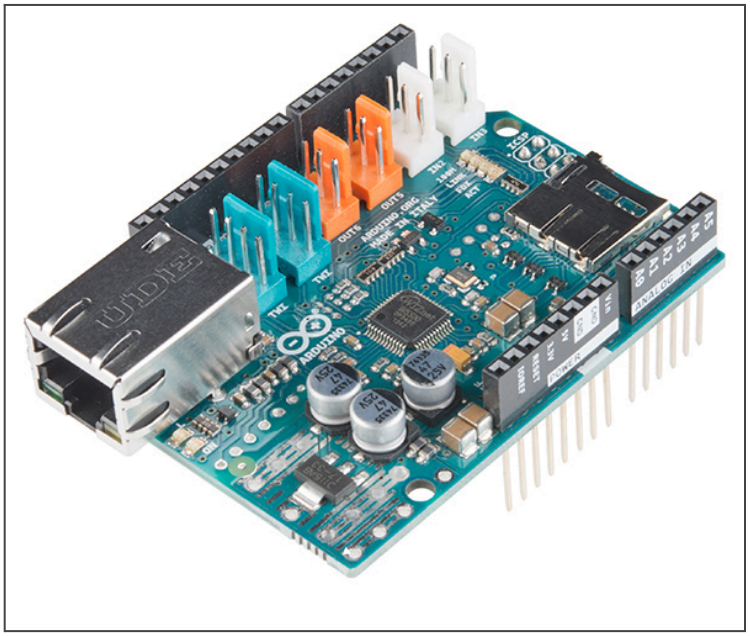

Рис. 2.3. Модуль Arduino Ethernet

Датчики:

DHT22 – це цифровий датчик температури та вологості виробника ASAIR підвищеної точності та чутливості, який має заводське калібрування і характеризується низьким енергоспоживанням (рис. 2.4).

Основні технічні характеристики:

- тип: AM2302 цифровий;
- точність: 0.1℃;
- діапазон вимірювання вологості: 0–100%;
- діапазон виміру температури: -40~80℃;
- точність вимірювання вологості: ± 2% RH;
- точність вимірювання температури: ± 0.5℃;
- напруга живлення: 3.5–5.5 В;
- кількість виводів: 4 [14].

У межах реалізації системи цей датчик використовується для періодичного зчитування показників температури та вологості у приміщенні. Значення зчитуються безпосередньо мікроконтролером використовуючи бібліотеку DHT.h, яка забезпечує простоту взаємодії з датчиком.

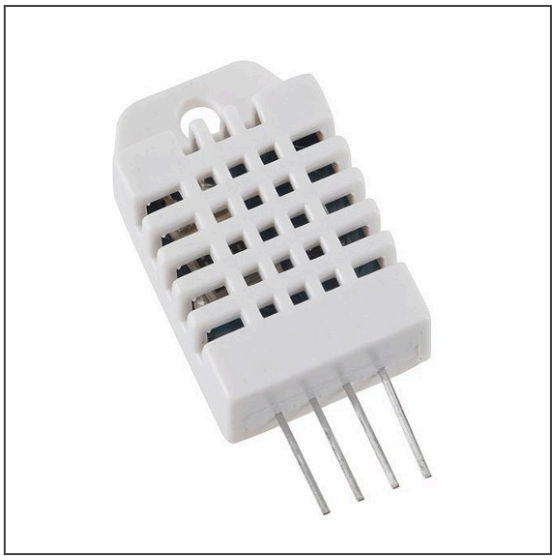

Рис. 2.4. Датчик вологості та температури DHT22

MQ-2 – це модуль, який являє собою датчик диму, розташований на платі з потенціометром і 4 виводами, який широко застосовується в системах безпеки, розумних будинках та автоматизації. Його основна функція – виявлення горючих газів і даму в приміщенні (рис. 2.5).

Основні технічні характеристики:

- Газ, що детектується: горючий газ, дим;
- Діапазон чутливості: 300–10000ppm;
- Rs (опір чутливого елемента): 1 ... 20кОм (при 50ppm толуолу);
- Газ, для якого нормується датчик: ізобутан, 1000ppm;
- Час відгуку: ≤ 10с;
- Чутливість (R в повітрі) / (R в присутності характерного газу): ≥ 5с;
- Rh (опір нагрівача): 31Ω ± 3Ω;
- Ih (струм нагрівача): ≤ 180мА;
- Vh (напруга нагрівача): 5В ± 0.2 В;
- Ph (потужність нагрівача): ≤ 900мВт;
- Vc (напруга схеми): ≤ 24В;
- Стандартні робочі умови: температура: -10 ~ +50 ° C, вологість: ≤ 95% RH, концентрація кисню: 21%;
- Умови зберігання: температура: -20 ~ +70℃, вологість: ≤ 70% RH;

- Конфігурація A або B (металевий або пластиковий корпус) [15].

У межах проєкту модуль використовується для контролю якості повітря в середовищі. Значення рівня диму або наявності горючих газів зчитуються через аналоговий вхід плати, а результати надсилаються через API-запит.

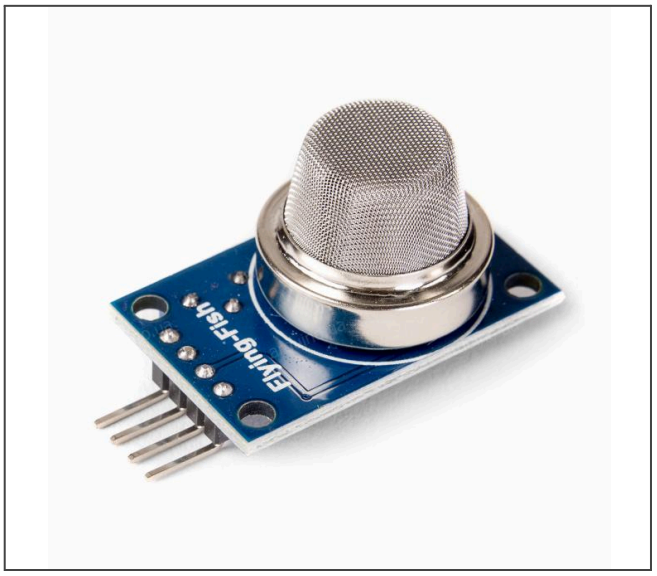

Рис. 2.5. Модуль датчика диму MQ-2

3.5” TFT LCD Shield – це модуль розширення для Arduino, який забезпечує виведення графічної інформації, тексту та зображень. TFT драйвер побудований на базі чіпсета ILI9486 з 8-бітовим інтерфейсом керування. Шилд містить сенсорну панель (тачскрін), що дозволяє не лише відображати дані, а й взаємодіяти з користувачем, також є з підтримка SD-карти. Підключення здійснюється безпосередньо через роз’єм Shield-формату, що дуже спрощує інтеграцію компонента в прототип (рис. 2.6) [16].

Основні технічні характеристики:

- Діагональ дисплея: 3.5 дюйма;
- Роздільна здатність: 320x480 пікселів;
- Тип дисплея: TFT LCD, кольоровий;
- Сенсорна панель: резистивна, 4-провідна;
- Контроль дисплея: ILI9486;
- Інтерфейс з’єднання: паралельний 8-бітний (через цифрові порти);
- Робоча напруга: 5В;

- Слот для карти пам’яті: для microSD.

У межах проєкту цей компонент було використано для реалізації віртуальної клавіатури для введення pairing-токена, щоб користувач міг “прив’язати” пристрій до свого облікового запису та отримати результат: чи успішно встановлений зв’язок між користувачем та пристроєм, чи доданий пристрій, чи пристрій не був доданий раніше і він не належить іншому користувачеві.

Для відображення клавіатури для вводу pairing-токену у проєкті були використані такі бібліотеки:

- MCUFRIEND_kbv – ініціалізація дисплея та роботи з графікою;
- TouchScreen – обробка натискань на сенсорній панелі (ввід коду);
- Adafruit_GTX – базові графічні операції (текст, лінії, фігури тощо).

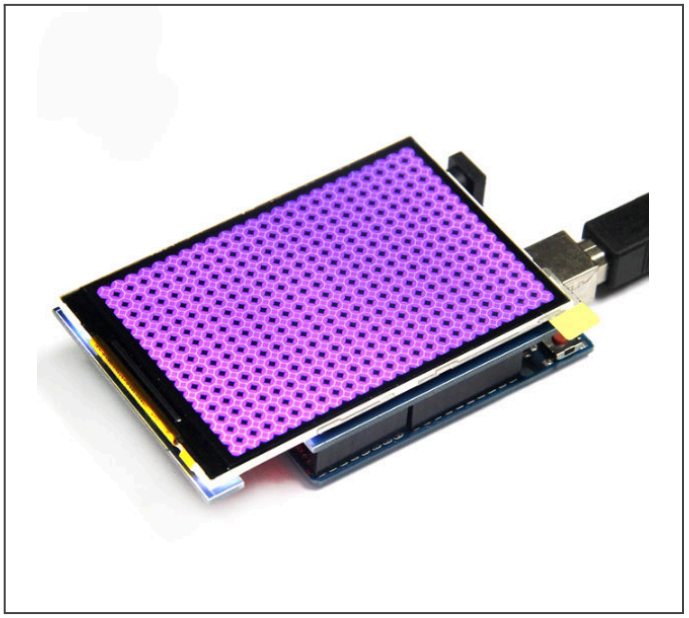

Рис. 2.6. Шилд TFT дисплея 3.5" 320x480 ILI9486 з тачскрином для Arduino

Raspberry Pi Compatible Cooling Fan – це компактний вентилятор охолодження, який був розроблений спеціально для використання мікрокомп’ютерами. Його основне призначення – забезпечення додаткового охолодження центрального процесора та інших компонентів пристрою. Живлення вентилятор здійснюється безпосередньо від GPIO-пінів (пін – 5V, пін

6 – GND), що забезпечує простоту підключення без потреби у зовнішньому живленні [17].

Основні технічні характеристики:

- Розміри: 30x30x8 мм;
- Напруга: 5V;
- Тип підключення: перемички типу female;
- Рівень шуму: низький (тиха робота);

У межах проєкту цей модуль виконує роль демонстрації роботи вентилятора при високій температурі для зниження ризику надзвичайних ситуацій.

Рис. 2.7. Raspberry Pi Compatible Cooling Fan

Світлодіоди (LED) – це прості електронні компоненти, які використовуються для візуальної індикації станів системи, наприклад, сигналізації про підключення пристрою (рис. 2.8).

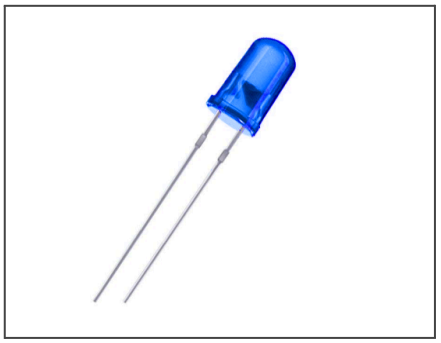

Рис. 2.8. Синій світлодіод

Макетна плата – це зручна платформа для прототипування електронних схем без пайки, яка дозволяє швидко тестувати з’єднання між компонентами (сенсорами, модулями тощо) перед їхньою остаточною інтеграцією в систему.

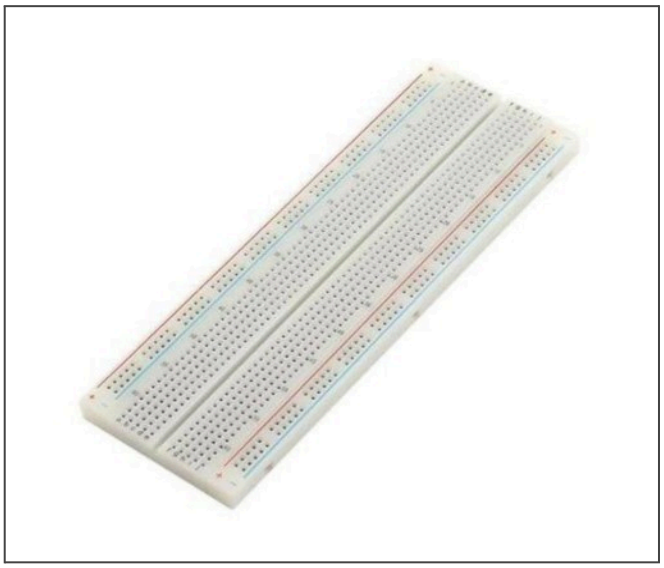

Рис. 2.9. Макетна плата

Перемички (jumper wires) – це гнучкі дроти з конекторами, які використовуються для з‘єднання компонентів на макетній платі або між модулями Arduino і завдяки їм забезпечується швидке та зручне компонування елементів схеми, що корисно на етапі розробки та налагодження.

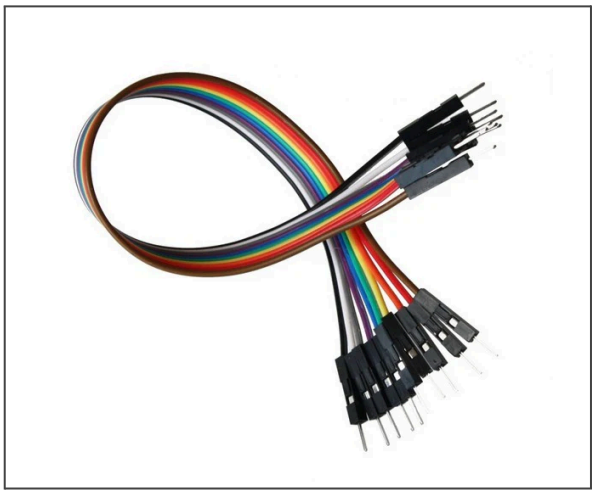

Рис. 2.10. Перемички

2.3.3. Підключення та структурна схема

Arduino-компоненти з’єднано відповідно до технічних характеристик, з урахуванням конфліктів пінів. Нижче на рис. 2.11 наводиться базова схема з’єднання деталей, яка забезпечує читання сенсорів, обробку pairing-коду та передачу даних.

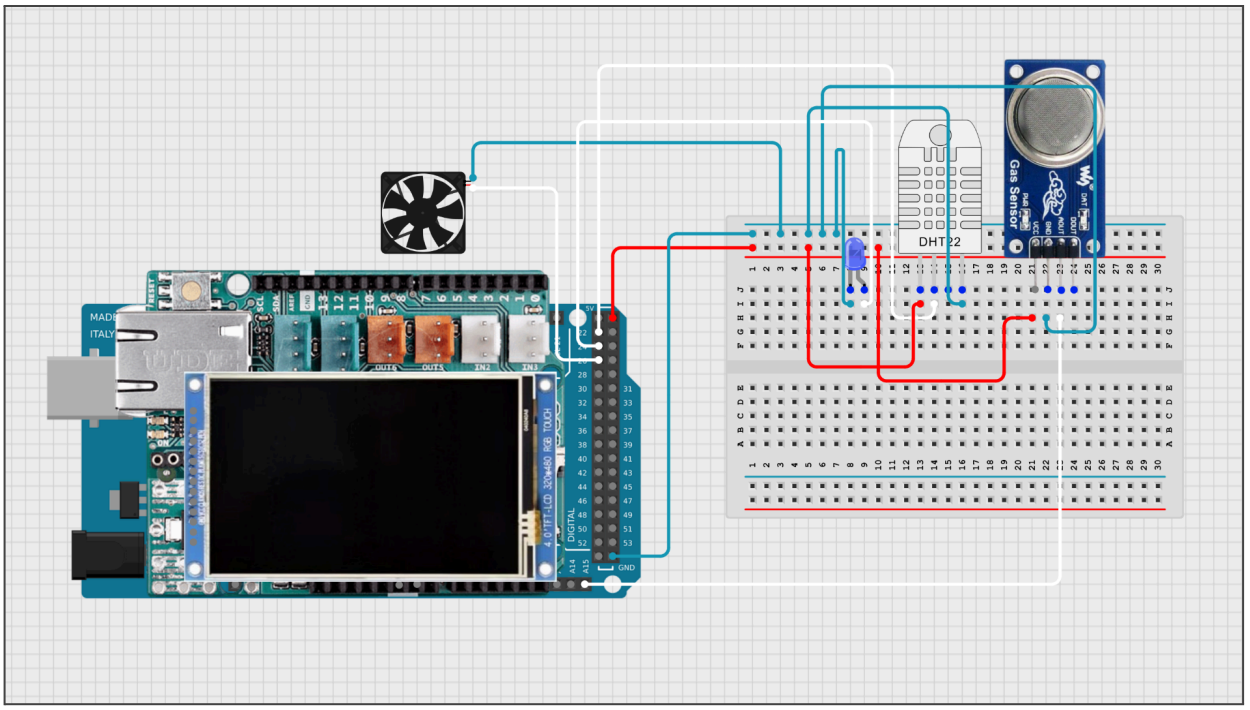

Рис. 2.11. Модель пристрою

2.4. Моделювання бази даних для системи моніторингу

Ефективне функціонування веборієнтованої системи моніторингу домашніх пристроїв вимагає надійної, логічно структурованої та масштабованої бази даних. На цьому етапі було здійснено моделювання бази даних із урахуванням вимог до функціональності системи, а також принципів нормалізації, підтримки зв’язків між об’єктами та забезпечення консистентності даних.

У межах проєкту було обрано реляційну базу даних (SQL) – це система зберігання та організації даних, де дані структуровані у вигляді таблиці та пов’язані між собою за допомогою ключів – це дає змогу встановити зв’язки між таблицями, що дозволяє об’єднувати дані з різних таблиць для виконання

складних запитів [18]. Реляційна модель забезпечує чітку схему та сувору типізацію, що є критично важливим для гарантування цілісності та аналітичної обробки даних у системі моніторингу.

2.4.1. Основні сутності та їх зв'язки

Моделювання бази даних є критичним етапом в проєктуванні, оскільки забезпечує логічну основу роботи всієї системи. Саме тому на основі аналізу функціональних потреб було виокремлено такі ключові сутності, які представлені в табл. 2.1:

Таблиця 2.1

Таблиці бази даних та їх призначення

Таблиця	**Назва**	**Пояснення**
User	Користувач	Розширює стандартну модель IndentityUser платформи ASP.NET Core, зберігає базову інформацію про користувача та забезпечує аутентифікацію через JWT-токени.
Device	Пристрій	Містить назву пристрою та пов'язується з конкретним користувачем. Один користувач може мати декілька пристроїв.
HouseState	Стан будинку	Зберігає показники датчиків, отримані з пристрою: температуру, вологість та рівень газу. Конкретний пристрій має свої стани.
DevicePairingToken	Токен прив'язки пристрою	Зберігає pairing-коди для підключення нового пристрою до облікового запису і запам'ятовує уже підключені для інформаційного наповнення. Має статус (активний/використаний/недійсний) і дату створення, щоб обраховувати різні параметри.
Archive	Архів	Архівна копія HouseState для умовного розділення станів.

Для візуалізації структури бази даних була розроблена діаграма зв’язків між сутностями – ER-діаграма (Entity Relationship Model), яка демонструє основні зв’язки (рис. 2.12).

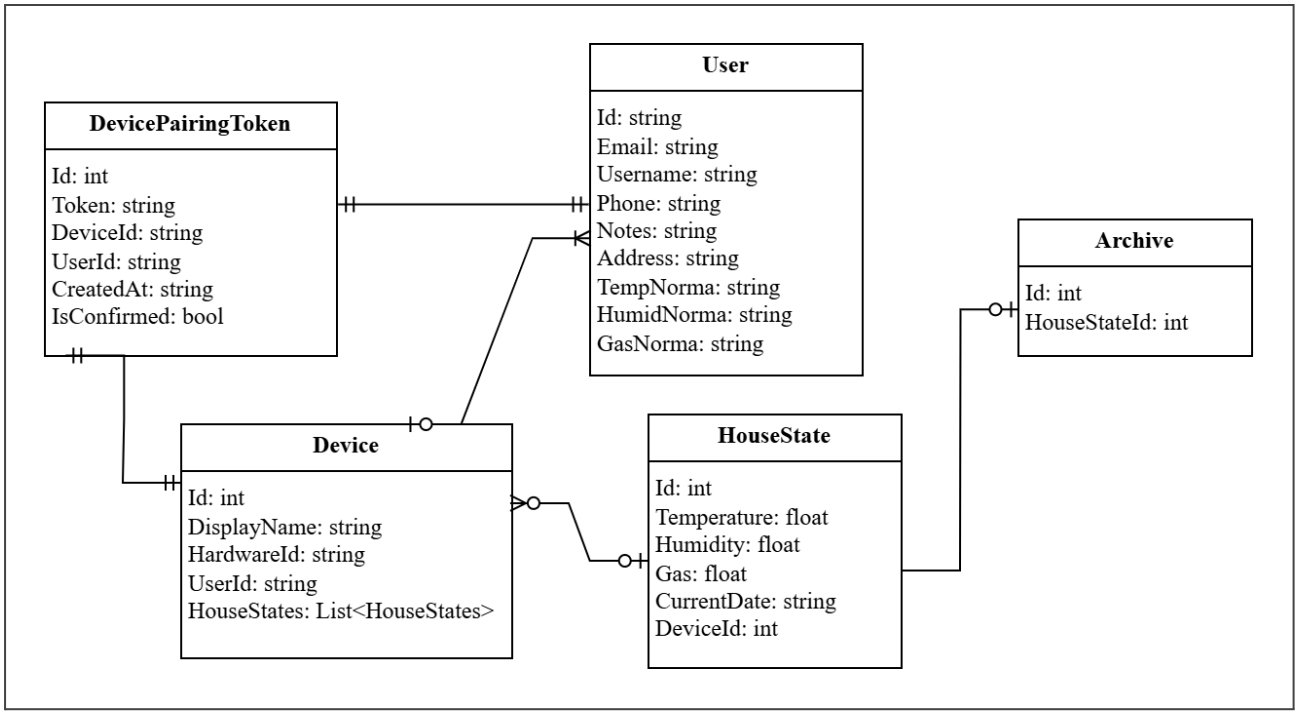

Рис. 2.12. ER-діаграма бази даних

Така структура забезпечує:

- Гнучкість: кожен пристрій можу мати власну історію показників, бо є збереження окремо активних та заархівованих даних.
- Безпеку: токени прив’язки можуть використовуватись тільки один раз для однієї прив’язки, таким чином це унеможливлює несанкціоноване підключення пристроїв.
- Масштабованість: база легко розширюється новими сенсорами або параметрами без кардинальних змін в структурах.
- Простота аналізу даних: окрема таблиця для заархівованих даних дозволяє виконувати аналітику без впливу на продуктивність основної системи.

- Сумісність: вибір SQL-бази даних забезпечує тісну інтеграцію з ORM-технологією EntityFrameworkCore – це суттєво спрощує розробку, підтримку, роботу та міграції даних.

Запропонована модель дозволяє ефективно зберігати, обробляти та аналізувати отриману інформацію, а також гарантує цілісність та надійність даних протягом усього життєвого циклу системи.

2.5. Вибір технологій серверної та клієнтської частин

Успішне створення веборієнтованої системи потребує ретельного вибору технологій, які відповідають вимогам до продуктивності, масштабованості, безпеки та зручності розробки. У цьому проєкті застосовано сучасні технології як для серверної, так і для клієнтської частин, які забезпечують узгоджену взаємодію між Arduino-проєктом, серверною та клієнтською частинами.

2.5.1. Серверна частина

- Середовище розробки**:** Visual Studio Community 2022 (17.8.5)
- Мова програмування: C# 10.0
- Платформа: ASP.NET Core Web API
- Framework: Microsoft .NET 8.0
- Спосіб розробки бази даних: Entity Framework Core з підходом Code First та використання міграцій
- Бібліотеки для API та СУБД:
    - EntityFramework (6.5.1) – підтримка базової функціональності;
    - Microsoft.AspNetCore.Authentication.JwtBearer (8.0.10) – реалізація JWT-аутентифікації для захисту API;
    - Microsoft.EntityFrameworkCore (8.0.10) – базова бібліотека для роботи з EF Core;
    - Microsoft.EntityFrameworkCore.Design(8.0.10) та Microsoft.EntityFrameworkCore.Tools(8.0.10) – створення та застосування міграцій;

- Microsoft.EntityFrameworkCore.Analyzers (8.0.10) – покращення якості коду під час компіляції;
- Microsoft.EntityFrameworkCore.SqlServer (8.0.10) – постачальник для підключення до MS SQL Server;
- Swashbuckle.AspNetCore (6.6.2) – генерація та документування API за допомогою Swagger UI;
- Microsoft.AspNetCore.Identity.EntityFrameworkCore (8.0.10) – інтеграція системи автентифікації ASP.NET Core Identity з EF Core;
- Microsoft.AspNetCore.Mvc.Abstractions (2.1.38) – підтримка побудови MVC/Web API архітектури.

Для реалізації серверної логіки було обрано ASP.NET Core Web API 8.0 – це сучасний та кросплатформний фреймворк з відкритим кодом для створення RESTfull сервісів.

Основні переваги цього вибору:

- Висока продуктивність: .NET Core відомий своєю швидкодією в порівнянні з іншими серверними платформами.
- Інтеграція з EntityFrameworkCore: зручний ORM (Object-relational mapping) для взаємодії з реляційною базою даних.
- Підтримка токен-авторизації: реалізовано захищену авторизацію за допомогою JWT.
- Підтримка WebSocket-технологій через SignalR: забезпечено обмін даними у реальному часі між клієнтом і сервером.
- Гнучка структура: підтримка модульності, розділення відповідальностей через сервіси, контролери та middleware.

Сервер виступає посередником між Arduino-пристроями, які надсилають показники, та frontend-проєктом, який візуалізує функціонал. Також, виконується обробка pairing-токенів, зберігання станів дома та реалізація прав доступу користувачів.

Серверна частина системи реалізовувана за допомогою архітектурного шаблону MVC (Model-View-Controller), метою якого є розбити складні системи на підсистеми для кращого розуміння та організації роботи в проекті.

В основі розробки бази даних застосовано підхід Core First, при якому структура бази даних автоматично генерується на основі Entity-класів. Для керування змінами структури БД використовуються міграції, які забезпечують контроль за розвитком бази даних у процесі розробки.

2.5.2. Клієнтська частина

- Середовище розробки**:** Visual Studio Code (1.95.3)
- Мова програмування: TypeScript 5.0
- Framework: React 18.3.1
- Бібліотеки:
  - React-Spinners – бібліотеки компонентів для відображення завантаження (loading indicators);
  - Tailwind CSS – утилітарний CSS-фреймворк, який дозволяє створювати адаптивні та сучасні інтерфейси без написання власних CSS-класів, що значно пришвидшує розробку;
  - styled-component – бібліотека для стилізації компонентів, яка використовує підхід CSS-in-JS.
  - recharts – бібліотека для побудови інтерактивних графіків та діаграм в React, що базується на бібліотеці D3.js., але надає простий і зрозумілий API та спеціально адаптований під React-компоненти.

Frontend-частину системи реалізовано з використанням *React* – бібліотеки JavaScript, орієнтований на компонентний підхід та створення інтерактивних інтерфейсів.

Причини вибору React:

- Швидке оновлення інтерфейсу: завдяки віртуальному DOM, швидко переписуються лише змінені компоненти;

- Гнучкість і масштабованість: легко організувати складні UI, які реагують на зміну стану системи;
- Розвиток екосистеми: велика кількість бібліотек для роботи з графіками, таблицями, WebSocket тощо;
- Підтримка односторінкових застосунків (SPA) – мінімізує навантаження на сервер і забезпечує краще UX.

Frontend та backend взаємодіють через RESTful API (для авторизації, отримання списку пристроїв тощо) і через WebSocket-з’єднання (SignalR) – для миттєвого оновлення показників в режимі реального часу на сторінці вебсайту.

Таке поєднання технологій дозволяє реалізувати сучасну та надійну систему з ефективною обробкою і візуалізацією даних у режимі real-time. Обрані компоненти гармонійно поєднуються між собою, забезпечуючи стабільну роботу всіх частин системи та зручність для кінцевого користувача.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ СИСТЕМИ ТА ТЕСТУВАННЯ

3.1. Розробка мікропрограмного забезпечення для Arduino

Мікропрограмне забезпечення (скетч Arduino) є критично важливою частиною системи моніторингу, оскільки саме воно відповідає за безпосередню взаємодію з фізичними пристроями, збиранням даних при стан будинку, обробку введених користувачем параметрів та передачу зібраної інформації на сервер.

3.1.1. Ініціалізація пристрою та підключення

Перед початком роботи безпосередньо з сенсорів важливо забезпечити коректну ініціалізацію всіх компонентів пристрою. Зокрема, необхідно налаштувати з’єднання із сервером за допомогою Ethernet-модуля, ініціалізувати сенсори та інші елементи, а також підготувати TFT LCD-екран до активної роботи з користувачем.

Під час запуску пристрою відбувається низка підготовчих кроків, які забезпечують готовність усіх складових до подальшої роботи – від екрана до мережевого з’єднання.

Етапи ініціалізації:

1. Ініціалізація серійного порту

   Встановлюється базове з’єднання для дебагу:

   ```
   Serial.begin(9600);
   ```

2. Запуск TFT LCD-екрану

   Визначається ID дисплея, налаштовується горизонтальна орієнтація та виводиться клавіатура:

   ```
   uint16_t ID = tft.readID();
   tft.begin(ID);
   tft.setRotation(1);  // горизонтальна орієнтація
   drawFullKeyboard();
   ```

Функція *drawFullKeyboard()* написана, щоб генерувати кастомізовану клавіатуру з обраних символів: 2, 3, 4, 6, 7, 9, A, B, C, E, F, X, G, H, J, K, L, U, <, N, P, T, S, >. Спеціальні символи були обрані за людинозчитуваною вимогою до pairing-токена та стійкістю до помилок: цифри або великі букви – без плутаних символів (наприклад, O (буква) ≈ 0 (нуль), l ≈ 1, I ≈ 1).

```
void drawFullKeyboard() {
 tft.fillScreen(BLACK);
 for (int row = 0; row < row_c; row++) {
  for (int col = 0; col < col_c; col++) {
   int x = keyX + col * (keyW + spacing);
   int y = keyY + row * (keyH + spacing);
   if (strcmp(keys[row][col], ">") == 0 || strcmp(keys[row][col], "<")
== 0) {
    tft.fillRect(x, y, keyW, keyH, BLACK);
    tft.setTextColor(WHITE);
   } else {
    tft.fillRect(x, y, keyW, keyH, WHITE);
    tft.setTextColor(BLACK);
   }
   tft.drawRect(x, y, keyW, keyH, WHITE);
   tft.setCursor(x + 22, y + 20);
   tft.setTextSize(3);
   tft.print(keys[row][col]);
  }
 }
```

3. Ініціалізація датчика DHT22 та налаштування режимів пінів

   Запускається модуль DHT та встановлюються пін-моди для реле та світлодіодів:

```
dht.begin();
```

pinMode(relayPin, OUTPUT);

pinMode(ledPin, OUTPUT);

4. Налаштування Ethetnet-з’єднання

   Спочатку пристрій намагається отримати IP-адресу через DHCP:

   if (Ethernet.begin(mac) == 0) {

   Якщо не вдалося – перевіряється наявність модулю та кабелю

   Переходить до ручного задання IP-адреси

   Ethernet.begin(mac, ip, dns, gateway, subnet);

   Якщо отримання IP-адреси було успішним – це фіксується у серійному моніторі

   Serial.println(Ethernet.localIP());

   }

5. Підготовка до з’єднання з сервером

   Виводиться інформація про підключення до серверної частини

   Serial.print("Connecting to ");

   Serial.println(server);

При стабільному інтернет-з’єднанні та справному API – у серійному моніторі повинна бути інформація про підключення та ініціалізацію всіх згаданих компонентів (рис. 3.1.).

У разі відсутнього інтернет-з’єднання, від’єднаного Ethernet-кабелю або інших подібних проблем користувач отримає відповідне повідомлення про помилку (рис. 3.2).

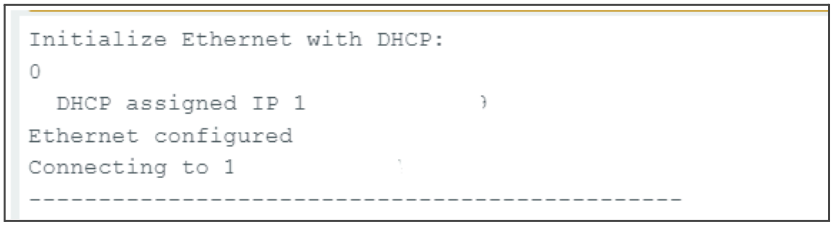

Рис. 3.1. Успішна ініціалізація всіх компонентів

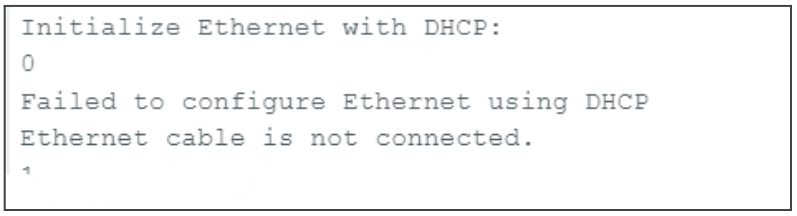

Рис. 3.2. Повідомлення про помилку при відсутності з'єднання

3.1.2. Збір та попередня обробка даних

Одним із ключових етапів реалізації логіки є організація ефективного зчитування та попередньої обробки даних, отримуваних із сенсорів, підключених до мікроконтролера Arduino. Саме ці дані формують основу для подальшої обробки, візуалізації та аналітики станів.

У рамках функціонування пристрою було реалізовано процес зчитування даних із сенсорів DHT22 та MQ-2. Було важливо не тільки отримати значення, а й виконати базову валідацію і попередню обробку даних – це дозволяє зменшити навантаження на серверну частину системи, підвищити точність вимірювання та зберегти надійність системи.

У табл. 3.1 подано покроковий процес виконання з коротким поясненням кожного етапу.

На рис. 3.3 наведено графічну схему, яка ілюструє ці етапи.

Таблиця 3.1

Покрокова обробка даних у системі

Етап	**Компонент**	**Опис**
1.	Ethernet Shield	Отримання норм користувача по температурі, вологості та газу по get-запиту
2.	DHT22	Зчитування температура та вологості
3.	MQ-2	Визначення концентрації газу
4.	Arduino	Перевірка коректності значень у функції *conditionFunc()*
5.	Arduino	Формування JSON для насилання даних через HTTP

Продовження таблиці 3.1

6.	Ethernet Shield	Надсилання даних на сервер API
7.	API	Отримання відповіді про результат виконання запиту

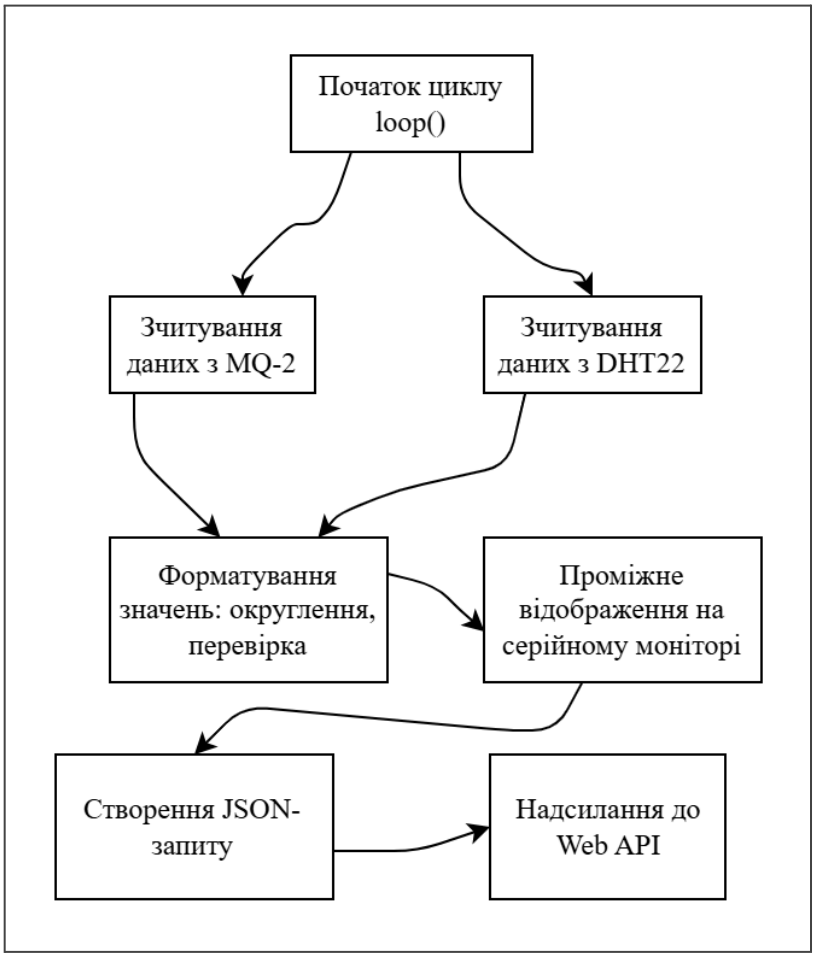

Рис. 3.3. Діаграма послідовності для збору та обробки даних

3.2. Розробка серверної частини

У цьому розділі наведено опис архітектури серверної частини проєкту, основних функціональних компонентів, а також реалізація API для обміну інформацією з пристроями Arduino та клієнтською частиною.

Сервер виконує функції аутентифікації користувачів, обробляє та зберігає отримані дані в базі, а також забезпечує передачу оновлень у реальному часі за допомогою технології SignalR.

Додатково створено допоміжні сервіси для полегшення роботи та функціональної масштабованості API:

- JWTTokenService – для аутентифікації та авторизації користувача в систему.
- SeederDatabase – для заповнення БД тестовими даними.
- CustomValidator – для обробки помилок, які виникають при надсиланні запитів.
- DTO (Data Transfer Object) – моделі об’єктів, які використовуються в функціоналі запитів, для коректної передачі даних між всіма компонентами. Демонстрація моделі на прикладі UserDTO:

```
public class UserDTO
{
    public string Id { get; set; } = string.Empty;
    public string UserName { get; set; } = string.Empty;
    public string Email { get; set; } = string.Empty;
    public string Address { get; set; } = string.Empty;
    public string Phone { get; set; } = string.Empty;
    public string Notes { get; set; } = string.Empty;
    public string TempNorma { get; set; } = string.Empty;
    public string HumidNorma { get; set; } = string.Empty;
    public string GasNorma { get; set; } = string.Empty;
}
```

- hubs – центральна точка зв’язку між клієнтами та серверами, сприяючи взаємодії в реальному часі (рис. 3.4).

Рис. 3.4. Приклад hubs в архітектурі backend проєкту

Для реалізації запитів є AccountController та HouseController. Обидва отримують запити відповідно до того, який запит був надісланий.

AccountController – клас, який відповідає за обробку запитів, які стосуються користувача і будь-якої його інформації.

Приклад функції: обробка запиту за маршрутом */Account/get-devices-by-user*, який повертає список пристроїв, що належать авторизованому користувачу. У даній функції за допомогою авторизаційних даних користувача (userId) виконується пошук пристроїв і після цього формується список DTO-об’єктів із основною інформацією про пристрої, включно з останньою активністю, що отримується з таблиці станів пристроїв.

```
[Authorize]
[HttpGet("get-devices-by-user")]
public async Task<List<DeviceDTO>> GetDevicesByUser()
{
    var userId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
    if (userId == null)
    {
        return null;
    }

    var devices = await context.Devices
        .Where(d => d.UserId == userId)
        .Select(d => new DeviceDTO
        {
            Id = d.Id,
            DisplayName = d.DisplayName,
            HardwareId = d.HardwareId,
            Email = context.Users.FirstOrDefault(u => u.Id ==
userId).Email,
            LastActivity = context.HouseStates
                .Where(x => x.DeviceId == d.Id)
                .OrderByDescending(x => x.CurrentDate)
                .Select(x => (DateTime?)x.CurrentDate)
                .FirstOrDefault()
        })
        .ToListAsync();
    return devices;
}
```

HouseController – контролер, який обробляє запити, пов’язані з даними про будинок. Він відповідає за отримання актуальної інформації з датчиків, збереження станів будинку, доступ до архівних даних та ін.

Приклад функції: обробка запиту за маршрутом */House/add-new-state*. У цій функції відбувається прийом даних із запиту у вигляді DTO (HouseStateDTO). Дані конвертуються у внутрішню модель HouseState та доповнюються поточним часом. Після цього відбувається додавання нового запису в базу даних. У разі успішного збереження інформації виконується відправка повідомлення конкретному користувачу через SignalR, що власне дозволяє оновити інтерфейс в реальному часі. У випадку помилки – повертається відповідь з кодом помилки та повідомленням.

```
[HttpPost("add-new-state")]
public async Task<ResultDTO> AddNewState([FromBody] HouseStateDTO data)
{
   HouseState houseState = new HouseState()
   {
      Temperature = GetFloat(data.Temperature),
      Humidity = GetFloat(data.Humidity),
      CurrentDate = DateTime.Now,
      Gas = GetFloat(data.Gas),
            DeviceId = context.Devices.FirstOrDefault(x => x.HardwareId == data.DeviceName).Id
   };
   try
   {
      context.HouseStates.Add(houseState);
      await context.SaveChangesAsync();

     await hubContext.Clients.User(context.PairingTokens
         .FirstOrDefault(x=>x.DeviceId == houseState.DeviceId).UserId)
         .SendAsync("ReceiveNewState", new
               {
                  deviceId = houseState.DeviceId,
                  timestamp = houseState.CurrentDate
               });

      return new ResultDTO()
      {
         Status = 200,
         Message = "Дані додано."
      };
   }
   catch (Exception ex)
   {
```

```
            HttpContext.Response.StatusCode = 500;
            return new ResultDTO()
            {
                Status = 500,
                Message = "Сталась помилка. Додавання скасовано."
            };
        }
    }
```

3.2.1. Аутентифікація користувачів

Для безпечного доступу до ресурсів системи реалізовано механізм аутентифікації на основі *ASP.NET Core Identity* з використанням JWT. Така комбінація дозволяє ефективно керувати користувачами, ролями та правами доступу, а також підтверджувати особу при кожному запиті до сервера.

ASP.NET Core Identity – це потужна бібліотека для управління обліковими записами користувачів. В проєкті було використано кастомізовану модель користувача (*ApplicationUser*), яка розширює стандартний клас IdentityUsert додатковими властивостями.

JWT (JSON Web Tokens) – це цифровий безпечний формат токенів, який містить в собі інформацію про користувача (ID, роль, email тощо) у зашифрованому вигляді.

При успішній аутентифікації сервер генерує токен за допомогою *JWTTokenService*, який клієнт надалі використовує для авторизованих запитів, що перевіряється сервером автоматично в Authorize. При наступних запитах (наприклад, отримати свої пристрої) клієнт додає токен у заголовок запиту:

Authorization: Bearer [токен користувача]

Сервер читає токен – розшифровує його і самостійно визначає хто робить запит, а тобто, це суттєво спрощує процес пошуку активного користувача та зменшує об’єм коду в контролері, бо тепер не потрібно кожного разу передавати пошту чи ID вручну через поля в запитах.

Приклад покрокової авторизації користувача за допомогою Swagger UI:

1. Відкриваємо Swagger та шукаємо потрібний нам post-запит */Account/login* (рис. 3.5)
2. Вводимо коректні дані вже існуючого користувача (рис. 3.6)
3. Надсилаємо запит на сервер
4. Отримуємо успішний результат (рис. 3.7)
5. Якщо ввести неправильні дані, то отримаємо результат з повідомленням про це (рис. 3.8)

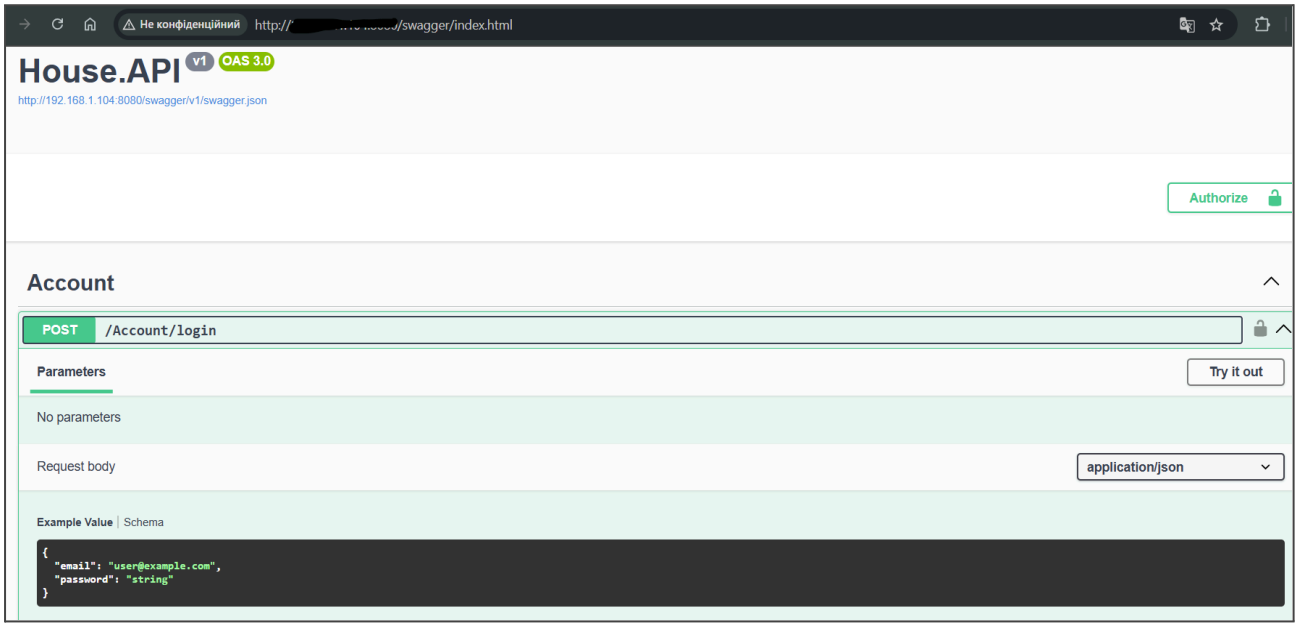

Рис. 3.5. POST-запит для авторизації користувача на Swagger

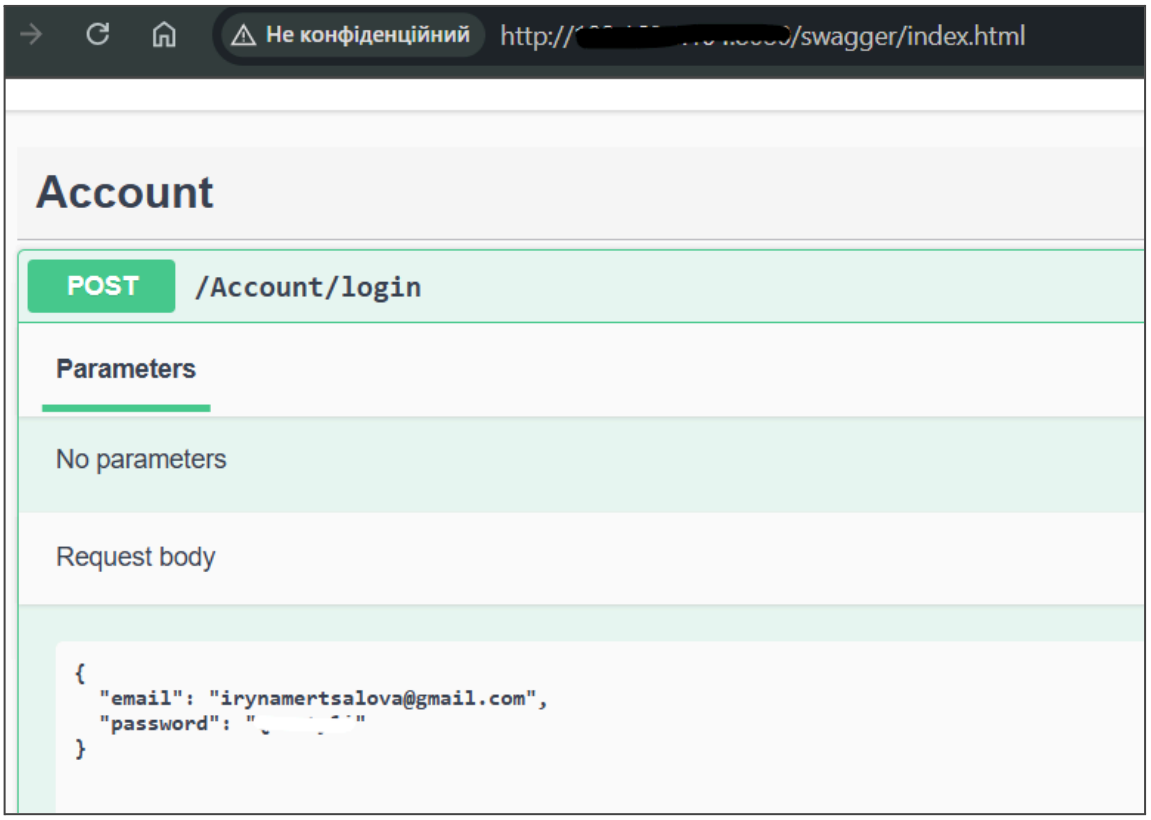

Рис. 3.6. Ввід даних зареєстрованого користувача для логіну

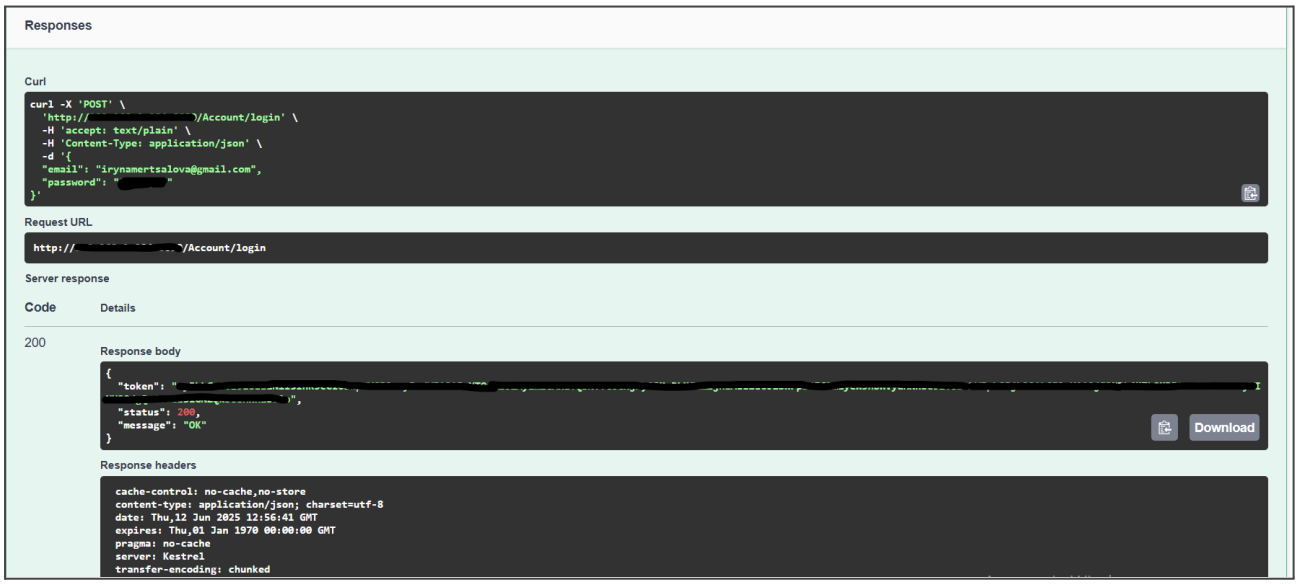

Рис. 3.7. Успішний результат логіну

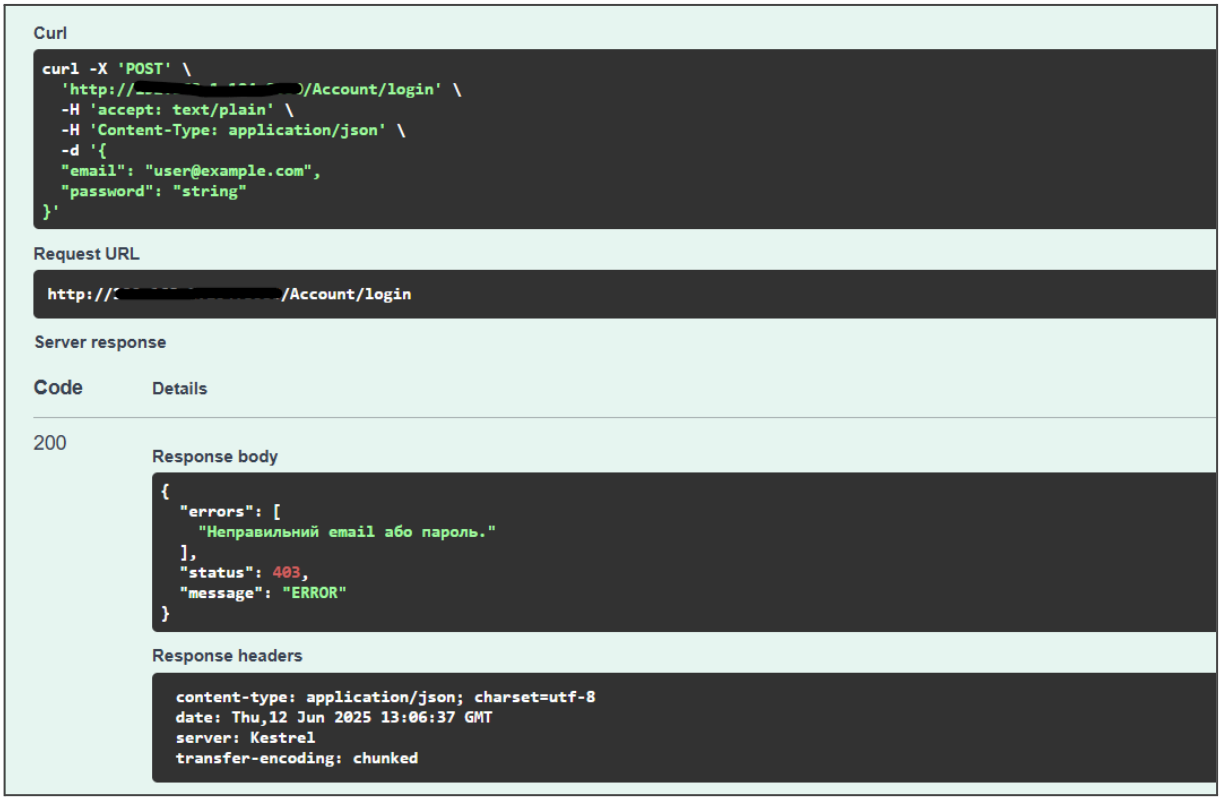

Рис. 3.8. Результат запиту з некоректними даними

3.2.2. Реалізація SignalR-хабу для передачі даних у реальному часі

Для забезпечення миттєвого оновлення даних у клієнтському застосунку використовується *SignalR* – це найкраща і найсучасніша бібліотека для задач з real-time, коли потрібно реалізувати реальні сценарії асинхронної інтеграції в IoT, коли веб відкриває постійне з'єднання з сервером – це стандартна практика для IoT, чату та нотифікації.

З технічної точки зору, було створено окремий Hub-клас (див. рис. 3.4), який обробляє push-повідомлення від сервера до підписаних клієнтів. Кожен клієнт підписується на канал, пов'язаний із конкретним пристроєм, що дозволяє уникнути зайвої передачі даних та зберігає масштабованість рішення.

Після того як сервер отримує нові дані від Arduino, у методі *AddNewState* виконується збереження в базу, а також трансляція даних через хаб.

Приклад коду хаба:

```
public class DeviceStateHub : Hub
{
    public override Task OnConnectedAsync()
```

```
    {
        var userId = Context.UserIdentifier;
        Console.WriteLine($"User {userId} connected with connectionId:
{Context.ConnectionId}");
        return base.OnConnectedAsync();
    }
    public override Task OnDisconnectedAsync(Exception? exception)
    {
        var userId = Context.UserIdentifier;
        Console.WriteLine($"User {userId} disconnected");
        return base.OnDisconnectedAsync(exception);
    }
}
```

Приклад надсилання сповіщення в функції *AddNewState*:

```
await hubContext.Clients.User(context.PairingTokens
.FirstOrDefault(x=>x.DeviceId == houseState.DeviceId).UserId)
.SendAsync("ReceiveNewState", new
{
    deviceId = houseState.DeviceId,
    timestamp = houseState.CurrentDate
});
```

У файлі *Program.cs* було додано конфігурацію та мапінг маршруту, що дозволяє клієнтам підключитися до */hubs/device-state* та працювати з відповідними методами:

```
builder.Services.AddSignalR();
app.MapHub<DeviceStateHub>("/hubs/device-state");
```

3.3. Розробка інтерфейсу користувача на React

Інтерфейс користувача було реалізовано з використанням JavaScript-бібліотеки React, яка забезпечує гнучкий та динамічний підхід до побудови вебдодатків завдяки компонентній архітектруі та можливості реактивного оновлення даних. Такий вибір дозволив створити інтуїтивно зрозумілий, швидкий та масштабований інтефейс для керування та моніторингу домашнії пристроїв, підключених через Arduino.

Основною метою розробки клієнтської частини було забезпечити зручну, адаптовану взаємодію користувача з функціоналом системи: реєстрацією облікового запису, прив'язкою пристроїв до акаунту, переглядом поточних показників з сенсорів та історичних даних, а також взаємодію у режимі реального часу.

3.3.1. Прототипування інтерфейсу в Figma

На початковому етапі розробки інтерфейсу було створено прототипи в Figma (рис. 3.9).

Figma – це інструмент для створення макетів, дизайну інтерфейсів і прототипування. Вона дозволяє UX/UI дизайнерам і розробникам спільно працювати над UI-дизайном у реальному часу прямо в браузері.

Переглянути готові макети сторінок можна за посиланням на Figma Project [19].

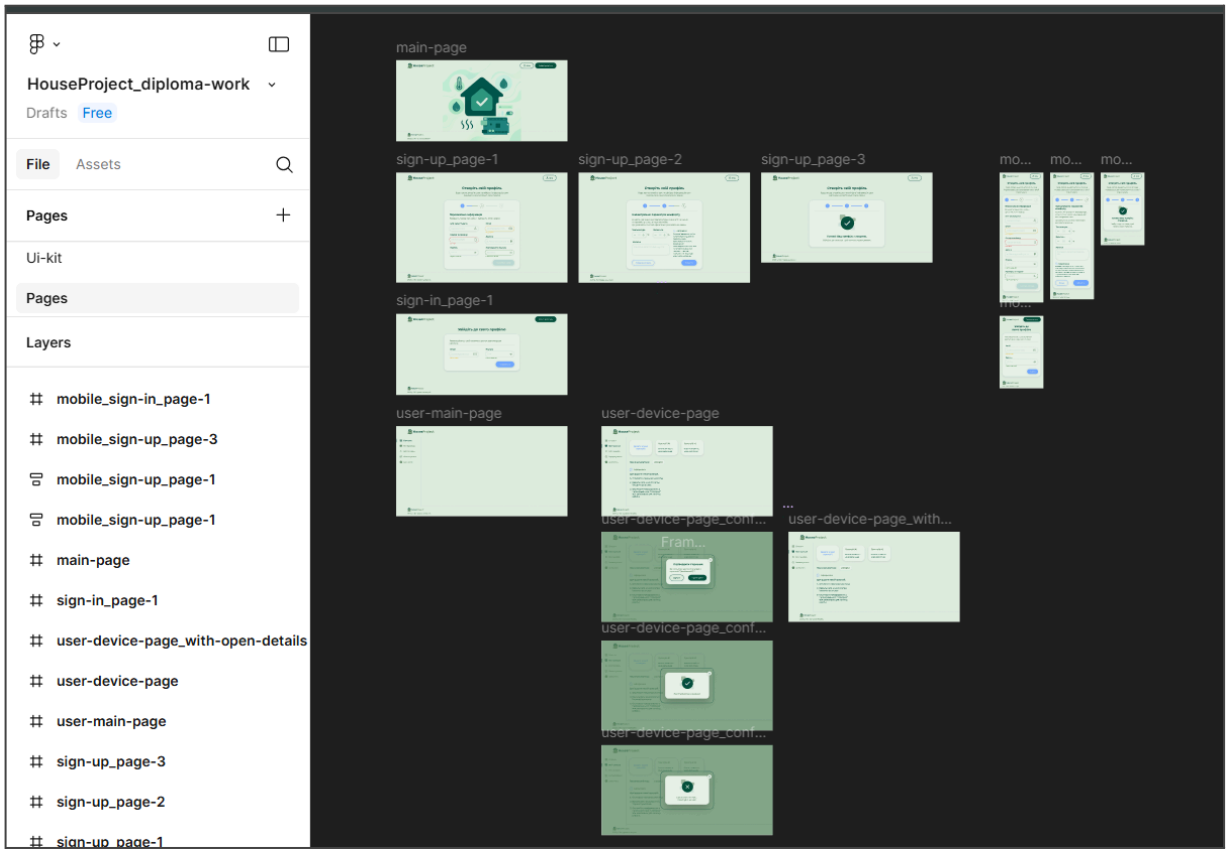

Рис. 3.9. Огляд на макети в Figma

3.3.2. Структура вебсайту

Інтерфейс було структуровано у вигляді окремих логічних сторінок:

- Сторінка авторизації – перша сторінка, яку зустрічає користувач, коли заходить перший раз на вебсайт, для авторизації користувача в систему за допомогою JWT-аутентифікації;
- Сторінка реєстрації – сторінка зі зручним функціоналом для введення всіх потрібних полів при реєстрації користувача в систему;
- Сторінка “Мої пристрої” – після успішного логіну це друга сторінка, яку бачить користувач, відображення всіх пристроїв з можливістю додати новий та іншими додатковими інформаційними блоками.
- Головна сторінка користувача (Dashboard) – сторінка, завдяки якій можна побачити інформаційні графіки показників по конкретному пристрої з динамічним оновленням даних.
- Сторінка “Деталі пристрою” – перегляд максимально зібраної інформації по конкретному пристрої.
- Різноманітні модальні вікна – для оповіщення користувача про важливі дії

3.3.3. Візуалізація отриманих показників у реальному часі

Для того, щоб отримати актуальні дані для відображення інформаційного графіка для конкретного пристрою було використано:

- SignalR – для отримання даних відразу після додавання їх в базу даних.
- Recharts – це бібліотеки для побудови інтерактивних графіків та діаграм, адаптований та зрозумілий в використанні інструмент для легкої візуалізації даних будь-яких типів.

Для того, щоб “ловити” щойно додані показники в базу даних було створено *useDeviceStateHub* – спеціальний хук, який забезпечує інтеграцію з SignalR-хабом для отримання даних від серверної частини в режимі реального часу. Цей хук підключається до хабу device-state, який працює на ASP.NET Core з використанням бібліотеки SignalR. Таким чином, useDeviceStateHub дозволяє

зручно та ефективно реалізувати механізм підписки на зміни стану пристрою і забезпечує взаємодію між сервером і клієнтом без необхідності ручного оновлення сторінки чи періодичного опитування сервера.

Приклад реалізації хука:

```
export default function useDeviceStateHub(onStateReceived) {
  const connectionRef = useRef(null);
  useEffect(() => {
    const token = localStorage.getItem("token");

    const connection = new HubConnectionBuilder()
      .withUrl("http://192.168.1.104:8080/hubs/device-state", {
        accessTokenFactory: () => token,
      })
      .withAutomaticReconnect()
      .configureLogging(LogLevel.Information)
      .build();
    const startConnection = async () => {
      if (connection.state === "Disconnected") {
        try {
          await connection.start();
          console.log("SignalR connected");
        } catch (err) {
          console.error("SignalR Connection Error: ", err);
        }
      }
    };
    const init = async () => {
      await startConnection();
      console.log("Connection ID:", connection.connectionId);
    };
    init();
    const handler = (data) => {
      console.log(data);
    };
    connection.on("ReceiveNewState", (data) => {
      console.log("New state received:", data);
      if (onStateReceived) onStateReceived(data);
    });
    connectionRef.current = connection;
    connection.on("ReceiveNewState", handler);
    return () => {
      connection.off("ReceiveNewState", handler);
    };
  }, [onStateReceived]);
}
```

Для візуалізації отриманих даних було проаналізовано і протестовано декілька React-бібліотек, таких як: React ApexCharts, canvasjs, react-chartjs-2 та recharts. Recharts це одна з найпопулярніших та найзручніших бібліотек для відображення даних у різноманітних формах, видах та анімаціях (рис. 3.10).

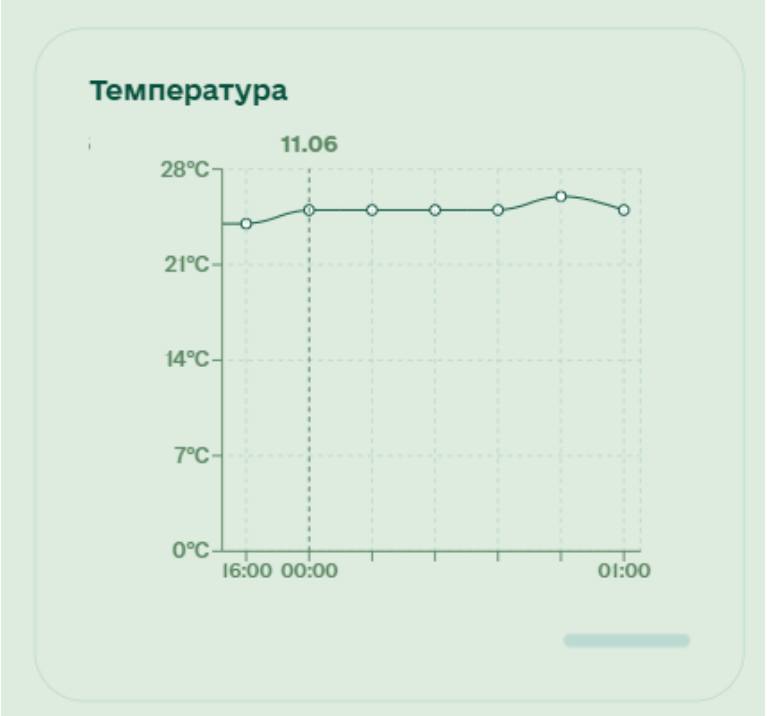

Рис. 3.10. Приклад застосування бібліотеки на практиці

3.4. Механізми аутентифікації та прив’язки пристроїв

У системі реалізовано повноцінний механізм аутентифікації користувачів та безпечної прив’язки пристроїв до їхніх облікових записів. Такий підхід дозволяє забезпечити персоналізацію доступу, запобігти несанкціонованому підключенню сторонніх пристроїв та гарантує захищений обмін даними між сервером і контролерами.

Переваги обраного підходу:

- Можливість динамічного додавання пристроїв без перенавантаження прошивки.
- Надійна прив’язка до користувача через токени.

- Захист від підключення сторонніх або повторно використаних пристроїв.
- Прозорий та контрольований процес підтвердження.

3.4.1. Аутентифікація користувача

Після входу в систему користувач проходить процес аутентифікації, реалізований на основі технології JWT. При успішному логіні користувачеві генерується токен, який містить його ID та email, і який використовується в подальших запитах як доказ авторизації.

Цей токен автоматично “прикріплюється” до HTTP-запитів і обробляється сервером через атрибути *[Authorize]*, що дозволяє обмежити доступ до захищених маршрутів. Таким чином, тільки авторизований користувач може створювати pairing-токени, додавати пристрої та переглядати стан підключених контролерів.

3.4.2. Генерація pairing-токена

На сторінці “Мої пристрої” користувач може ініціювати процес додавання нового пристрою. При натисканні кнопки “Додати новий пристрій” відправляється запит на маршрут */Account/get-pairing-token*, який відповідає за створення або повернення pairing-токена.

Особливості генерації токена:

- Генерується випадковий 6-значний код з літер та цифр.
- Якщо у користувача немає активного токена в базі – створюється новий із полями: *Id, Token, UserId, CreatedAt, IsConfirmed (false)*.
- Якщо токен вже існує, але був створений у попередній день – він оновлюється.
- Якщо токен уже створений сьогодні – системи повертає вже існуючий.
- Якщо користувач уже має токен, але він *IsConfirmed = true* – створюється новий запис в базі з новим токеном для прив’язки нового пристрою.

Цей pairing-токен слугує єдиним способом встановлення зв’язку між пристроєм користувача та його обліковим записом в системі.

На рис 3.11 показано приклад двох записів з бази даних з підтвердженим та не підтвердженим токенами.

	10	XK62KJ	fa427e63-fdc1-...	2	20.05.2025 16:3...	True
	15	L6FAHE	fa427e63-fdc1-...	*NULL*	28.05.2025 22:4...	False

Рис. 3.11. Записи з таблиці PairingTokens з різними станами

3.4.3. Введення pairing-токена з пристрою

На пристрої Arduino, обладнаному сенсорним екраном, реалізовано інтерфейс введення pairing-токена. Користувач вручну вводить отриманий код за допомогою вбудованої екранної клавіатури, після чого натискає кнопку “>” для надсилання інформації на сервер.

Запит з Arduino надсилається на маршрут */Account/is-correct-token* і містить:

- pairing-токен
- унікальний ідентифікатор пристрою (HardwareId)

3.4.4. Обробка pairing-запиту на сервері

Серверна логіка обробки запиту включає кілька етапів перевірки:

1. Валідація токена

   Якщо pairing-токен не існує або вже використаний – користувач отримує повідомлення про помилку.
2. Перевірка пристрою

   Якщо пристрій з таким HardwareId вже зареєстрований у системі – надсилається відповідне повідомлення.
3. Створення нового пристрою

   Якщо перевірки пройдено, у базі даних створюється новий запис пристрою з полями:

   a. *HardwareId*

b. *DisplayName* (початкове згенероване за замовчуванням ім’я)

c. *UserId* (не встановлюється одразу)

Після цього *DeviceId* записується в pairing-токен запис в базі даних, а система очікує на підтвердження з боку користувача.

3.4.5. Підтвердження підключення користувачем

На стороні вебінтерфейсу користувач отримує повідомлення про спробу підключення нового пристрою з зазначеним номером. У нього є можливість:

- Підтвердити додавання – тоді пристрій остаточно закріплюється за обліковим записом користувача, а pairing-токен позначається як *IsConfirmed = true*.
- Відхилити запит – у такому випадку запис про пристрій видаляється з бази, а pairing-токен залишається не підтвердженим і процес можна повторити.

Такий механізм дає змогу користувачам контролювати процес підключення, а системі – гарантувати захищений канал прив’язки без використання заздалегідь прошитих ідентифікаторів.

3.5. Інструкція користувача

Система HouseProject призначена для зручного підключення домашніх пристроїв до вебінтерфейсу з метою віддаленого моніторингу показників середовища – таких як температура, вологість та наявність газу. Завдяки простому механізму прив’язки через pairing-токен та зручному інтерфейсу користувачі можуть легко додати свій пристрій Arduino до облікового запису та стежити за його станом у реальному часі.

Нижче наведено покрокову інструкцію для комфортного користування системою.

1. Реєстрація та вхід до системи

1.1. Якщо ви новий користувач, натисніть “Зареєструватися”, заповніть форму та підтвердьте реєстрацію.

1.2. Якщо у вас вже є обліковий запис, натисніть “Увійти”, введіть свої облікові дані та натисніть “Продовжити”.

1.3. Після успішного входу в систему ви отримуєте JWT-токен, який буде автоматично використовуватись для всіх запитів до сервера, щоб ідентифікувати вас як користувача.

2. Прив’язка нового пристрою

2.1. Перейдіть на сторінку “Мої пристрої”.

2.2. Натисніть кнопку “Додати новий пристрій”.

2.3. Ви отримаєте унікальний 6-значний код прив’язки та всю необхідну інформацію для дій далі.

2.4. Увімкніть ваш пристрій Arduino та введіть на екрані отриманий код та натисніть кнопку “>” для відправки.

2.5. Пристрій здійсниться запит до сервера для перевірки введеного вами коду.

2.6. У разі успіху пристрій з’явиться у списку на сторінці “Мої пристрої”, де ви зможете підтвердити або відхилити прив’язку.

3. Перегляд інформації про пристрій

3.1. У списку пристроїв натисніть на потрібний пристрій, щоб перейти до деталей.

3.2. Тут відображаються: назва та унікальний ідентифікатор пристрою, дата додавання та останнього оновлення, статус (онлайн/офлайн), email власника та останні отримані показники.

3.3. Якщо пристрій активний – показники оновлюються автоматично в режимі реального часу.

4. Статус онлайн/офлайн

4.1. Система визначає, чи пристрій онлайн аналізуючи час останнього оновлення.

4.2. Якщо дані не надходять протягом 5 хвилин – пристрій автоматично переходить у статус “Офлайн”.

5. Підтримувані пристрої

Платформа підтримує підключення Arduino з Ethernet Shield та сенсорами (DHT22, MQ-2, TFT LCD Shield).

6. Тестування та безпека

6.1. Усі взаємодії з пристроями проходять автентифікацію через JWT.

6.2. Pairing-токен доступний лише для поточного дня, що запобігає повторному або несанкціонованому підключенню.

У разі виникнення помилок або проблем – переконайтеся, що ваш пристрій підключено до інтернету та правильно введено pairing-токен. За потреби перезавантажте Arduino або повторно згенеруйте токен.

Завантажити готове рішення можна за посиланням на GitHub [20].

ВИСНОВКИ

У результаті виконаної роботи було розроблено функціональну, масштабовану та адаптивну систему моніторингу домашніх пристроїв, яка поєднує апаратні засоби на базі Arduino з сучасними веб-технологіями ASP.NET Core та React. Проєкт реалізує повноцінну клієнт-серверну архітектуру, що дозволяє забезпечити стабільний збір, обробку та візуалізацію даних про стан домашнього середовища у режимі реального часу.

По-перше, було здійснено глибокий аналіз актуальних проблем у сфері інтеграції IoT-пристроїв у побутові умови. Проаналізовано архітектурні підходи до створення стійкого програмного забезпечення, зокрема методи оптимізації обміну даними між Arduino та сервером, а також способи забезпечення масштабованості системи у майбутньому. Це дозволило сформувати гнучку структуру, яку легко розширювати або адаптувати під конкретні побутові потреби.

По-друге, реалізовано повний цикл передачі та обробки даних: зчитування параметрів із сенсорів температури, вологості й газу на мікроконтролері Arduino; надсилання показників через HTTP-запити до API-сервера; збереження в базі даних і передача у frontend через Web API. Особливу увагу приділено підтримці надійного каналу зв’язку навіть в умовах мережевих збоїв, що підтверджує практичну життєздатність запропонованого рішення.

По-третє, була розроблена інтерактивна веб-частина на основі React, яка дозволяє користувачеві переглядати дані про стан будинку у зрозумілому та лаконічному інтерфейсі. Реалізовано функціональність прив’язки пристроїв до користувацьких облікових записів за допомогою pairing токена, який вводиться через сенсорний TFT-дисплей на пристрої. Цей механізм забезпечує безпечну автентифікацію й асоціацію між пристроєм та обліковим записом без необхідності додаткового налаштування вручну.

У результаті розроблений прототип не лише підтвердив працездатність ідеї створення доступної та стабільної домашньої системи моніторингу, але й

продемонстрував практичну реалізацію ефективної взаємодії між мікроконтролером і веб-застосунком. Важливо, що система залишилася доступною для недосвідченого користувача, зберігши інтуїтивний інтерфейс і мінімальні вимоги до налаштування.

У підсумку, проведена робота підтвердила актуальність створення простих, гнучких і масштабованих рішень для моніторингу домашнього середовища. Вона демонструє приклад успішного об'єднання веб-технологій і IoT-пристроїв у реальному проєкті, відкриваючи перспективи для подальших досліджень у напрямках оптимізації енергоспоживання, розширення спектра сенсорів та інтеграції зі сторонніми платформами типу Home Assistant чи Google Home. Таким чином, ця система може стати основою для подальших розробок у галузі smart home та цифрової автоматизації побуту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Електронні ресурси віддаленого доступу

1. Understanding IoT Architecture: Key Layers and Core Technologies Explained. Welolver. URL: https://www.wevolver.com/article/what-is-architecture-of-internet-of-things-iot (дата звернення: 11.05.2025)
2. Architecture of Internet of Things (IoT). GeeksforGeeks. URL: https://www.geeksforgeeks.org/architecture-of-internet-of-things-iot (дата звернення: 11.05.2025)
3. Що таке мережевий протокол. Optimize: IL. URL: https://optimize-il.com/shho-take-merezhevij-protokol/ (дата звернення: 11.05.2025)
4. TCP/IP як мова спілкування для комп'ютерів у мережі. FoxmindEd. URL: https://foxminded.ua/tcp-ip/ (дата звернення: 15.05.2025)
5. WebSockets: навіщо потрібні та як з ними працювати. ProIT. URL: https://proit.ua/websockets-navishcho-potribni-ta-iak-z-nimi-pratsiuvati-2/ (дата звернення: 15.05.2025)
6. Home Assistant 101. Посібник для початківців. DOU. URL: https://dou.ua/forums/topic/38947/ (дата звернення: 17.05.2025)
7. Google Home – Додатки в Google Play. Google Play. URL: https://play.google.com/store/apps/details?id=com.google.android.apps.chromecast.app&hl=uk (дата звернення: 17.05.2025)
8. HomeKit і розумний дім: інтеграція техніки Apple у розумні системи. Техно Їжак. URL: https://ti.ua/ua/news/homekit_i_umnyy_dom_integratsiya_tekhniki_apple_v_umnye_sistemy/?srsltid=AfmBOooCmLHjBowwfeqX_GytUB42RYvpKQah0jLvcIpB7-8mm4aLzysY (дата звернення: 17.05.2025)
9. Принцип єдиної відповідальності (Single Responsibility Principle) в ООП. IT Blog. URL:

https://it-blog.in.ua/pryntsyp-yedynoyi-vidpovidalnosti-single-responsibility/ (дата звернення: 18.05.2025)
10. Layered Architecture. ScienceDirect. URL: https://www.sciencedirect.com/topics/computer-science/layered-architecture (дата звернення: 18.05.2025)
11. Dependency Injection and Services in ASP.NET Core: A Comprehensive Guide. Medium. Pavi Patel. URL: https://medium.com/@ravipatel.it/dependency-injection-and-services-in-asp-net-core-a-comprehensive-guide-dd69858c1eab#transient (дата звернення: 21.05.2025)
12. Arduino Mega 2560 R3 (CH340) плата мікроконтролера. ArduinoKit. URL: https://arduinokit.com.ua/ua/p1080109370-arduino-mega-2560.html?srsltid=AfmBOoqNtuaoLknxHhk4-fshaLrX0yf2vbPCVbCh-_HmE46aR_RqB6Jc (дата звернення: 21.05.2025)
13. Arduino Ethernet Shield 2 на W5500. Arduino.UA. URL: https://arduino.ua/prod7108-arduino-ethernet-shield-2 (дата звернення: 22.05.2025)
14. Датчик вологості та температури DHT22. Arduino.UA. URL: https://arduino.ua/prod301-datchik-vlajnosti-i-temperatyri-dht22 (дата звернення: 22.05.2025)
15. Модуль датчика диму MQ-2. Arduino.UA. URL: https://arduino.ua/prod298-modyl-datchika-dima-mq-2 (дата звернення: 22.05.2025)
16. Шилд TFT дисплея 3.5" 320x480 ILI9486 з тачскрином для Arduino. Arduino.UA. URL: https://arduino.ua/prod2642-shild-tft-displeya-3-5-320h480-ili9486-s-tachskrinom-dlya-arduino?srsltid=AfmBOoqObXmZRceN-o-BxyDhhHjpcrRTZjANqbyIHktDDQTU_96zXnZD (дата звернення: 22.05.2025)
17. Raspberry Pi Compatible 5V 30x30mm Cooling Fan. Motorobit. URL: https://www.motorobit.com/raspberry-pi-compatible-5v-30x30mm-cooling-fan?sr

sltid=AfmBOoqfqCSAwZlU0Y3eNbYtGcgqcdK-douIJ-1-kvqe8Nbp7sXtrOzL (дата звернення: 22.05.2025)

18. Реляційні бази даних: структура та застосування у практиці. FoxmindEd. URL: https://foxminded.ua/reliatsiini-bazy-danykh/ (дата звернення: 25.05.2025)
19. Figma – HouseProject_diploma-work. Figma. URL: https://www.figma.com/proto/FRPBV3bYKUAlEYDs3AsAzr/HouseProject_diploma-work?node-id=78-906&t=Qszk5PiLQNFQb8Jk-1 (дата звернення: 25.05.2025).
20. GitHub – mertsalovaa/house_course-work. GitHub. URL: https://github.com/mertsalovaa/house_course-work.git (дата звернення: 25.05.2025).