Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Автоматики і комп'ютеризованих технологій
(повна назва)

Кафедра Комп'ютерно-інтегрованих технологій, автоматизації та робототехніки
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

перший (бакалаврський)
(рівень вищої освіти)

Розроблення системи автоматизації процесу збирання та обробки даних з пристроїв IoT на інтелектуальному виробництві

(тема)

Виконав:
здобувач 4 року навчання,
групи АКТАКІТ-21-3
Антон ШАРЛАЙ
(власне ім'я, прізвище)

Спеціальність 151 Автоматизація та комп'ютерно-інтегровані технології
(код і повна назва спеціальності)

Тип програми освітньо-професійна
Освітня програма Автоматизація та комп'ютерно-інтегровані технології
(повна назва освітньої програми)

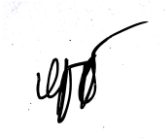
Керівник проф. Сергій НОВОСЕЛОВ
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри _____ _____ Ігор НЕВЛЮДОВ
(підпис) (власне ім'я, прізвище)

2025 р.

Я, Шарлай Антон Юрійович, як здобувач(ка) вищої освіти ХНУРЕ, розумію і підтримую політику закладу із академічної доброчесності. Я не надавав(ла) і не одержував(ла) недозволену допомогу під час підготовки кваліфікаційної роботи. Я не використовував(ла) штучний інтелект для підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

29 травня 2025 р.  Антон ШАРЛАЙ

Харківський національний університет радіоелектроніки

Факультет Автоматики і комп'ютеризованих технологій

Кафедра КІТАР

Рівень вищої освіти перший (бакалаврський)

Спеціальність 151 Автоматизація та комп'ютерно-інтегровані технології

(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Автоматизація та комп'ютерно-інтегровані технології

(повна назва)

ЗАТВЕРДЖУЮ:
Зав. Кафедри _____
(підпис)
«30» квітня 2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Шарлаю Антону Юрійович

(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення системи автоматизації процесу збирання та обробки даних з пристроїв IoT на інтелектуальному виробництві

затверджена наказом університету від 19.05.2025 р. №390 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 11.06.2025 р.

3. Вихідні дані до роботи роботі Мови програмування: C#, JavaScript; СКБД – PostgreSQL; тип інтерфейсу – графічний, операційна система – Windows; програматори: ESP32, Wemos D1 R2; переферійні пристрої: SG90, OV2640, SEN18, HC-SR501.

4. Перелік питань, що потрібно опрацювати в роботі Вступ, аналіз технічного завдання, архітектура та проєктування системи, опис прийнятих програмних та апаратних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Діаграма прецедентів, діаграма розгортання, ER діаграма, функціональна схема SG90, презентація у форматі pptx

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз технічного завдання	10.04.2025	Виконано
2	Архітектура та проєктування системи	18.04.2025	Виконано
3	Опис прийнятих програмних та апаратних рішень	25.04.2025	Виконано
4	Тестування розробленого програмного забезпечення	20.05.2025	Виконано
5	Подання роботи на перевірку Інтернет-сервісом StrikePlagiarism	29.05.2025	Виконано
6	Оформлення пояснювальної записки	30.05.2025	Виконано
7	Подання роботи на рецензію	01.06.2025	Виконано
8	Подання роботи на підпис зав. кафедри	03.06.2025	Виконано
9	Подання кваліфікаційної роботи в ЕК	13.06.2025	Виконано

Дата видачі завдання 30.04.2025 р.

Здобувач _____ Антон ШАРЛАЙ
(підпис) (власне ім'я, прізвище)

Керівник роботи _____ проф. Сергій НОВОСЕЛОВ
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 62 с., 17 табл., 20 рис., 4 дод., 34 джерел

IOT, MQTT, C#, VUE.JS, REST API, VISUAL STUDIO, VISUAL STUDIO CODE

Об’єкт розробки – процес обробки та передачі даних в IoT-інфраструктуру, зокрема процес збору телеметричної інформації від пристроїв, її аналізу та забезпечення обміну цією інформацією між компонентами системи.

Предмет розробки – комплексне програмне рішення, а саме система IoTFlow, яка реалізує процеси збирання, обробки та передачі даних з IoT-пристроїв у рамках інтелектуального виробництва.

Мета роботи – підвищити ефективність збору та обробки даних з IoT-пристроїв, за рахунок створення сценаріїв.

Було проведено аналіз технічного завдання, вивчено сучасні підходи до збору та обробки даних з IoT-пристроїв та розглянуто існуючі аналогічні рішення. На основі цього розроблено архітектуру програмного забезпечення й створено мокапи графічного інтерфейсу, що дозволило чітко окреслити функціональні можливості застосунку та сформувати внутрішню модель об’єктів і зв’язків. Навантажувальне тестування підтвердило здатність платформи одночасно обслуговувати 1100 активних користувачів без зниження продуктивності. У результаті створено програмну платформу для автоматизації збору та обробки даних з IoT-пристроїв із вбудованим інструментом на основі нотації BPMN для побудови індивідуальних сценаріїв.

Робота виконана в рамках цілей сталого розвитку «Інновації та інфраструктура» номер 9.1.

ABSTRACT

Explanatory note: 62 p., 17 table, 20 figures, 4 app., 34 sources.

IOT, MQTT, C#, VUE.JS, REST API, VISUAL STUDIO, VISUAL STUDIO CODE

The object of development is the process of processing and transmitting data to the IoT infrastructure, including the process of collecting telemetry information from devices, analysing it, and ensuring the exchange of this information between system components.

The subject of development is a comprehensive software solution, namely the IoTFlow system, which implements the processes of collecting, processing and transmitting data from IoT devices as part of smart manufacturing.

The goal is to increase the efficiency of collecting and processing data from IoT devices by creating scenarios.

An analysis of the technical requirements was carried out, modern approaches to collecting and processing data from IoT devices were examined, and existing comparable solutions were reviewed. Based on this, the software architecture was developed and mockups of the graphical user interface were created, enabling a clear definition of the application’s functional capabilities and the formation of its internal object–relationship model. Load testing confirmed that the platform can support 1,100 concurrent active users without any performance degradation. As a result, a software platform has been created to automate the collection and processing of data from IoT devices, featuring an integrated tool based on BPMN notation for constructing custom scenarios.

The work was performed within the framework of Sustainable Development Goal 9.1 «Innovation and Infrastructure».

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД – база даних;

КІТАР – комп’ютерно-інтегрованих технологій, автоматизації та робототехніки;

НДР – науково-дослідна робота;

ПЗ – програмне забезпечення;

СКБД – система керування базами даних;

ТВР – технології виробництва радіоапаратури;

ХНУРЕ – Харківський національний університет радіоелектроніки;

BPMN – Business Process Model and Notation;

DI – Dependency Injection;

DTO – Data Transfer Object;

IoT – Internet of Things;

JWT – JSON Web Token;

LED – Light Emitting Diode;

MQTT – message queuing telemetry transport;

QOS – Quality of Service;

SPA – Single Page Application;

UI – User Interface;

UX – User Experience.

ВСТУП

У сучасних умовах розвитку промисловості важливу роль відіграють інтелектуальні системи, здатні автоматично збирати, обробляти та захищати дані, отримані від великої кількості IoT-пристроїв. Сучасні фабрики перетворюються на складні кібер-фізичні системи з тисячами датчиків і виконавчих механізмів. Без автоматизованого збирання, фільтрації та аналізу, ці дані є складними у використанні для інших пристроїв. Тож в умовах індустрії 4.0 зростає потреба в побудові гнучких інформаційних систем, які будуть забезпечують не лише зчитування даних із датчиків, але й обробку даних з пристроїв.

Надійність такої платформи залежить від правильно підібраних протоколів обміну даними та архітектурних рішень. Застосування неефективних протоколів та неадаптованих рішень приводить до втрат даних, затримок у передачі даних чи навіть втрат або порушення цілісності.

Програмна система має забезпечувати реакцію на отримання певних даних на основі визначених користувачем дій. У центрі таких систем знаходиться інфраструктура, побудована на основі IoT, що поєднує фізичні пристрої з цифровими сервісами через мережеві технології.

Для забезпечення безперервної, безпечної та масштабованої роботи таких платформ необхідне застосування сучасних технологій, таких як протоколи для передачі повідомлень, серверна частина для взаємодії між компонентами, шифрування та аутентифікація для захисту даних користувача, а також візуальний інтерфейс для управління пристроями та зручний інструмент для побудови та зміни сценаріїв.

Для розробки серверної частини системи використовуються наступні технології: C#, ASP.NET, MQTTnet, Entity framework, PostgreSQL. Середовище розробки програмного коду – Visual Studio і DataGrip.

Для розробки клієнтської частини системи використовуються наступні технології – TypeScript, Vue.js, Vuex, ESLint, PrimeVue. Середовища розробки програмного коду – Visual Studio Code.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– провести аналіз існуючих архітектур систем обробки даних в IoT-середовищах;

– здійснити вибір технологій і протоколів для побудови взаємодії між компонентами системи;

– реалізувати діаграму прецедентів;

– розробити ER діаграму;

– розробити діаграму розгортання;

– реалізувати серверну частину для збору, фільтрації, обробки та зберігання телеметричних даних;

– розробити візуальний інтерфейс для моніторингу та управління подіями в системі;

– оформити кваліфікаційну роботу згідно ДСТУ 3008:2015 [1], а також з методичними вказівками з підготовки й оформлення кваліфікаційної роботи здобувачами першого (бакалаврського) рівня вищої освіти спеціальності 151 Автоматизація та комп’ютерно-інтегровані технології освітньої програми «Автоматизація та комп’ютерно-інтегровані технології» [2].

1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

1.1 Постановка та зміст технічного завдання

В сучасних умовах розвитку Індустрії 4.0 інтелектуальне виробництво використовує IoT пристрої для автоматизації процесів. Актуальність розробки автоматизації збирання та оброки даних з IoT-пристроїв зумовлена потребою підвищення ефективності виробництва, контролю стану обладнання та можливість побудувати сценарій реагування на подію.

Програмне забезпечення має надавати універсальне підключення до будь-яких пристроїв на різних платформах, гарантуючи зручний веб-інтерфейс і пропонувати конструктор сценаріїв на основі BPMN. Під час створення сценарію один із методів підключеного пристрою виступає початковою подією, а всі інші елементи виконуються послідовно

Реалізація графічного інтерфейсу у вигляді веб-застосунку забезпечує кросплатформену доступність без необхідності встановлення чи оновлення клієнтського ПЗ, що значно спрощує подальшу підтримку та експлуатацію системи.

Система має обробляти помилки та некоректні дії й інформувати користувача про проблеми, що виникли під час користування ПЗ.

Для розробки серверної частини ПЗ варто використовувати мову програмування C# та фреймворк ASP.NET. Для роботи з протоколом MQTT треба використовувати бібліотеку MQTTnet.

Для розробки клієнтської частини ПЗ треба використовувати Vue.js з бібліотекою PrimeVue для надання зручних компонентів.

1.2 Аналіз сучасних підходів до збирання та обробки даних з IoT-пристроїв

Сучасні системи IOT налічують мільярди підключених пристроїв, які генерують величезні обсяги даних у реальному часі. За даними дослідницької компанії IoT Analytics, у 2021 році у світі налічувалося понад 12 млрд пристроїв IOT і експерти прогнозують зростання до 22 млрд до 2025 року [3]. Такий стрімкий ріст підкреслює необхідність ефективних підходів до збору, передачі, обробки та зберігання даних. Для керування настільки величезною кількістю пристроїв вже було вироблено декілька підходів, що забезпечують достатню масштабованість та надійність.

1.2.1 Архітектура IoT-систем та загальні принципи

IoT-системи зазвичай будуються багаторівнево, поєднуючи фізичні пристрої, мережеву інфраструктуру та хмарні сервіси. Незважаючи на різноманітність реалізацій, можна виділити базові компоненти будь-якої IoT-архітектури:

– розумні пристрої з датчиками, що збирають дані;

– мережі та шлюзи, які з’єднують ці пристрої з Інтернетом;

– проміжне програмне забезпечення або IoT-платформи, що забезпечують зберігання даних, обчислення та аналітику;

– застосунки, через які користувачі взаємодіють із системою та отримують результати.

Зазвичай така система формується у декілька шарів. На рівні збору (perception layer) працюють пристрої та датчики. На рівні зв’язку (connectivity layer) дані передаються мережами і через шлюзи до серверів. На рівні обробки (processing layer) дані обробляються платформою IoT. На рівні застосувань (application layer) здійснюється аналітика, візуалізація результатів та управління пристроями.

Для побудови таких систем прийнято використовувати шлюзеву архітектуру (gateway pattern), вона передбачає використання проміжного шлюзу для збору даних від IoT-пристроїв. Шлюз є проміжною ланкою між даними та хмарою, тому часто він виконує попередню обробку даних та відправляє до хмари. Такий підхід є актуальним, коли пристрій має обмежені ресурси.

Інший поширений підхід – це периферійні обчислення (edge computing), коли обробка робиться близько до джерела даних, з основною метою мінімізувати затримки й зменшити обсяг даних до хмари. У таких випадках використовується класична архітектурне рішення як у веб застосунку – подійно-орієнтовану архітектуру (event based architecture), в якій взаємодія компонентів відбувається через передачу подій та реагування на них.

1.2.2 Збирання даних з IoT-пристроїв

Джерелом даних у IoT є фізичні пристрої, оснащені датчиками. Вони відслідковують параметри навколишнього середовища або стан об’єктів (температура, вологість, освітленість, тощо) і перетворюють ці вимірювання на цифрові сигнали. Зібрані дані передаються на контролер, що керує пристроєм. Через обмежені обчислювальні потужності та енергоресурси більшість IoT-пристроїв виконує лише мінімальну обробку, а основне, що вони роблять це доставка даних до зовнішніх систем для подальшого аналізу. У деяких випадках пристрій надсилають пакетно, тобто накопичують замість безперервної передачі. Такий підхід відповідає шаблону пакетної передачі даних (bulk data pattern), коли великі обсяги вимірів, що не потребують негайної реакції, буферизуються та відправляються до серверу або хмари порціями. Пакетний збір даних економить енергію автономних датчиків і знижує навантаження на мережу у порівнянні з постійним потоковим режимом.

Часто у IoT-системах присутні спеціальні шлюзи – це проміжні пристрої чи програми, що збирають дані від сенсорів і відправляють далі до серверів, інколи їх використовують для обробки даних. Такий підхід підвищує

ефективність, надійність, масштабованість та розподілення навантаження у системі.

1.2.3 Передача даних до сервера

Після збору на рівні пристрою або шлюзу, дані IoT потребують надійної передачі до серверної інфраструктури. Основні вимоги на цьому етапі – ефективність, масштабованість та безпека. Дані зазвичай передаються у вигляді компактних повідомлень, часто із застосуванням протоколів, оптимізованих для IoT, таких як – MQTT, CoAP та інші. Є можливість передавати через стандартні веб-протоколи (HTTP/HTTPS, WebSocket) при менших обсягах телеметрії чи коли це зручніше для інтеграції, але такий підхід є не рекомендованим.

Сучасні платформи IoT здебільшого підтримують подійно-орієнтований обмін повідомленнями, тобто модель публікація/підписка (publish/subscribe) – пристрої виступають публікаторами, відправляючи дані на певні теми, а сервер підписується на ці теми і отримує повідомлення асинхронно. Така архітектура підвищує гнучкість системи і полегшує масштабування, оскільки додавання нових споживачів даних не потребує змін у коді пристроїв, а лише оформлення відповідної підписки на потрібні канали.

У випадку хмарних технологій, використовують керовані сервіси, що приймають дані від пристроїв та направляють її до інших компонентів системи. Прикладом може слугувати AWS IoT Core від Amazon та Azure IoT Hub від Microsoft. Вони дозволяють безпечно підключити пристрої до хмари та що важливо підтримують двохстороннє з’єднання.

1.2.4 Хмарна та локальна обробка даних

У більш сучасних системах є тренд на перенесення обчислювальних завдань з хмари на периферійні пристрої, щоб пристрій або локальний шлюз самостійно аналізував отримані від датчика дані і приймав оперативні рішення,

не чекаючи відповідь від віддаленого сервера. Цей метод суттєво скорочує затримку, але даний підхід сильно залежить від обчислювальних можливостей пристрою та є не завжди ефективним. До хмари надійдуть вже відфільтровані дані, таким чином досягається, що важлива задача виконується зразу, а хмарна інфраструктура розвантажується, отримуючи менший об’єм даних для обробки. Периферійна обробка є вирішальною для сценаріїв, де потрібна мінімальна затримка або є обмеження зв’язку – зокрема, в автономних транспортних засобах, промисловому контролі чи системах розумного міста. За рахунок децентралізації обчислень IoT-система стає більш стійкою до перебоїв зв’язку – навіть при втраті з’єднання з хмарою, локальні вузли можуть продовжувати виконувати основні функції і накопичувати дані для подальшої передачі.

Хмарна інфраструктура відіграє ключову роль в обробці IoT-даних, зокрема у випадках, коли потрібен поглиблений аналіз, тривале зберігання інформації та виконання складних обчислень. Зазвичай до хмари надходить телеметрія, яка в подальшому оброблюється. Головним плюсом хмарного рішення є майже нескінчений потенціал для масштабування, є можливість використовувати складні алгоритми машинного навчання або обробки сигналів, що були б недоступні для більшості IoT-пристроїв.

1.2.5 Збереження даних та сховища IoT

Телеметрія, відправлена IoT-пристроями, можуть сягати значних обсягів, особливо коли йдеться про великі розгортання (наприклад, промислові IoT на виробництві). Зазвичай інформація від пристроїв представляє собою часові ряди (time-series data) – послідовності вимірювань, прив’язані до часу. Для ефективного зберігання таких даних застосовуються спеціалізовані часорядні бази даних. Приклади: InfluxDB, TimescaleDB, OpenTSDB та інші системи, оптимізовані під високочастотні вставки даних і зберігання їх зі стисканням. На рисунку 1.1 наведено приклад графіку time-series data [4].

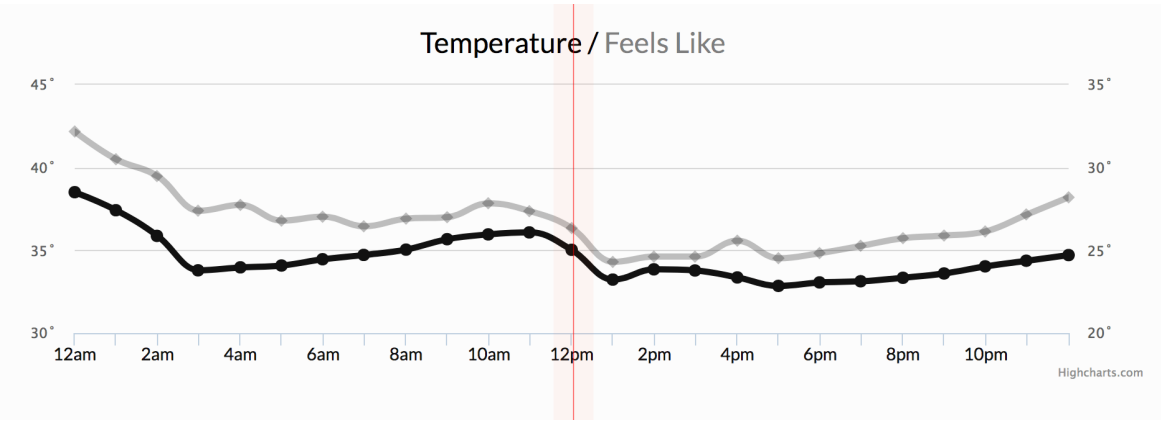


Рисунок 1.1 – Графік погодних умов [4]

Важливо зазначити, що не всі дані мають цінність, деякі дані немає сенсу зберігати на довгий період, тому архітектура IoT часто включає механізми фільтрації. Значущі події чи показники записуються у довготривале сховище, тоді як надлишкова інформація видаляється після використання. Наприклад відео-потік з камер часто аналізуються у реальному часі, але самі відео повністю не зберігаються якщо не зафіксовано нічого важливого, через їх великий розмір. Такий підхід дозволяє оптимізувати використання пам’яті.

1.2.6 Подійно-орієнтовна архітектура та реагування на події

Однією з найважливіших концепцій при побудові розподілених IoT-рішень є подійно-орієнтована архітектура. У цій моделі всі компоненти системи спілкуються між собою через події. На відміну від монолітних або клієнт-сервер архітектур, тут відправник події не очікує негайної відповіді від отримувача [5]. Натомість подія передається в інфраструктуру (наприклад, в брокер повідомлень або шину подій), де її можуть отримати один або кілька підписаних отримувачів. Такий підхід особливо добре підходить для IoT, де пристрої генерують велику кількість дрібних повідомлень, а різні сервіси повинні на них реагувати у режимі реального часу.

Для реалізації реагування на події в IoT часто використовуються правила та тригери. Багато IoT-платформ мають вбудовані механізми правил обробки

даних. Так, AWS IoT Core містить Rules Engine – компонент, який дозволяє визначити певні умови або шаблони у вхідних повідомленнях і автоматично виконати дію, коли ці умови виконуються. Дії можуть бути різними: збереження повідомлення в базу даних, пересилання його до іншого сервісу, виклик безсерверної функції, надсилання оповіщення тощо. Для прикладу, в Azure IoT Hub можна налаштувати потоки подій, коли пристрій надсилає телеметрію, що відповідає заданим критеріям, запускається Azure Function або інший модуль для обробки цієї події.

1.3 Аналіз аналогічних рішень

Для розробки системи для автоматизації процесу збирання та обробки даних з IoT-пристроїв важливо проаналізувати існуючі рішення на ринку. Було проаналізовано найпопулярніші аналогічні застосунки. Серед найпопулярніших застосунків було виділено «AWS IoT Core», «Microsoft Azure IoT Hub» та «IBM Watson IoT Platform».

1.3.1 AWS IoT Core

AWS IoT Core [6] пропонує комплексну систему для підключення й управління великою кількістю пристроїв, також надає можливість використовувати різні протоколи передачі даних та інтеграцію зі службами AWS, що є перевагою для користувачів, які використовують інші інструменти AWS. Вбудований механізм правил (Rules Engine) обробляє вхідні повідомлення за допомогою SQL-подібних запитів, фільтрує їх за заданими умовами та маршрутизує до AWS Lambda, DynamoDB, Kinesis, Amazon S3 [7].

Також надає привабливий інтерфейс (див. рис. 1.2). Але він є не дуже зручний через велику вкладеність розділів.

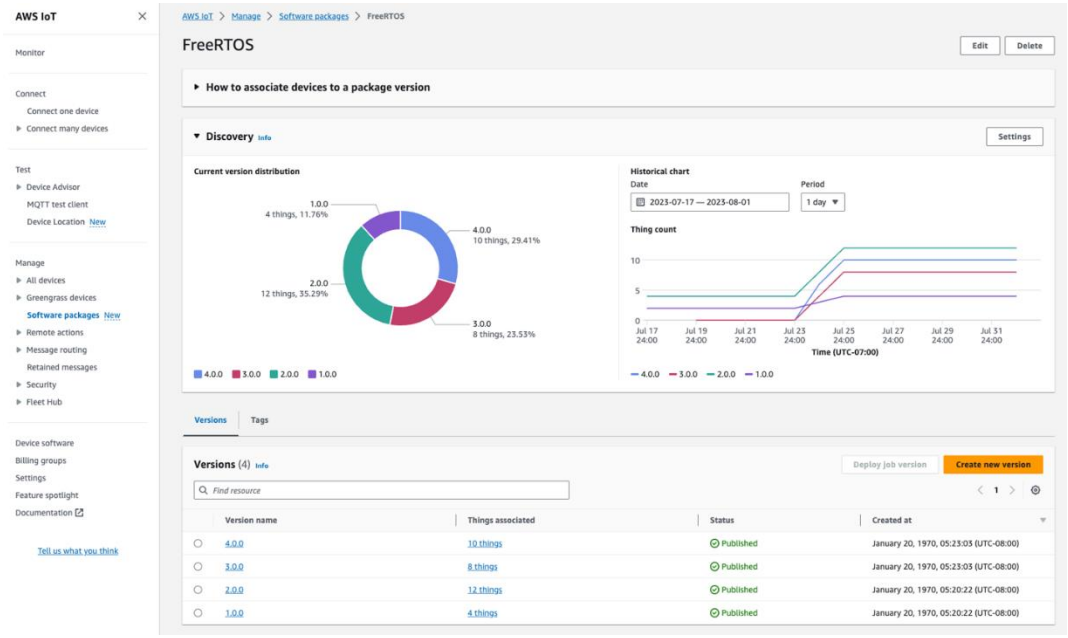


Рисунок 1.2 – Інтерфейс AWS IoT Core

Багатошарова архітектура AWS IoT Core забезпечує горизонтальне масштабування до мільйонів одночасних з’єднань [8]. Інтеграція з AWS Greengrass дозволяє виконувати обробку даних на периферії, зменшуючи затримки та навантаження на канал [9]. Проте налаштування політик IAM (Identity and Access Management), створення правил і налаштування моніторингу через CloudWatch та CloudTrail робить початковий етап більш складним у впровадженні [10]. Крім того, з ростом кількості телеметричних даних витрати на відправку повідомлень і обробку правил можуть вирости. Тож можна зробити висновок, що дане рішення є масштабованим та гнучким, адже здатне оброблювати велику кількість з’єднань та виконувати бізнес-логіку як на хмарі, так і на пристроях. Але є складним у початковому впровадженні.

1.3.2 Microsoft Azure IoT Hub

Microsoft Azure IoT Hub [11] – це керований хмарний сервіс для побудови масштабованих двонаправлених комунікацій між вашими IoT-пристроями та бекендом у Azure [11]. Він забезпечує надійну та захищену передачу телеметрії від пристроїв у хмару, а також відправку команд і повідомлень із хмари на

пристрої [12]. Підтримує протоколи MQTT, AMQP та HTTPS, а завдяки вбудованому механізму Message Routing із SQL-подібними виразами дозволяє фільтрувати й маршрутизувати телеметрію без додаткового коду до Azure Functions, Event Grid, Blob Storage чи Service Bus.

Консоль Azure поєднує в одному вікні віджети для перегляду пристроїв, метрики, журнали, завдяки цьому не потрібно переходити між кількома розділами. Також у Azure є інтерактивні графи, що показують зв'язки між Edge-модулями і залежними сервісами, AWS таких можливостей не надає без додаткових плагінів. На основі цього інтерфейс програми Microsoft Azure IoT Hub (див рис. 1.3) є більш інтуїтивним та зручним ніж інтерфейс AWS.

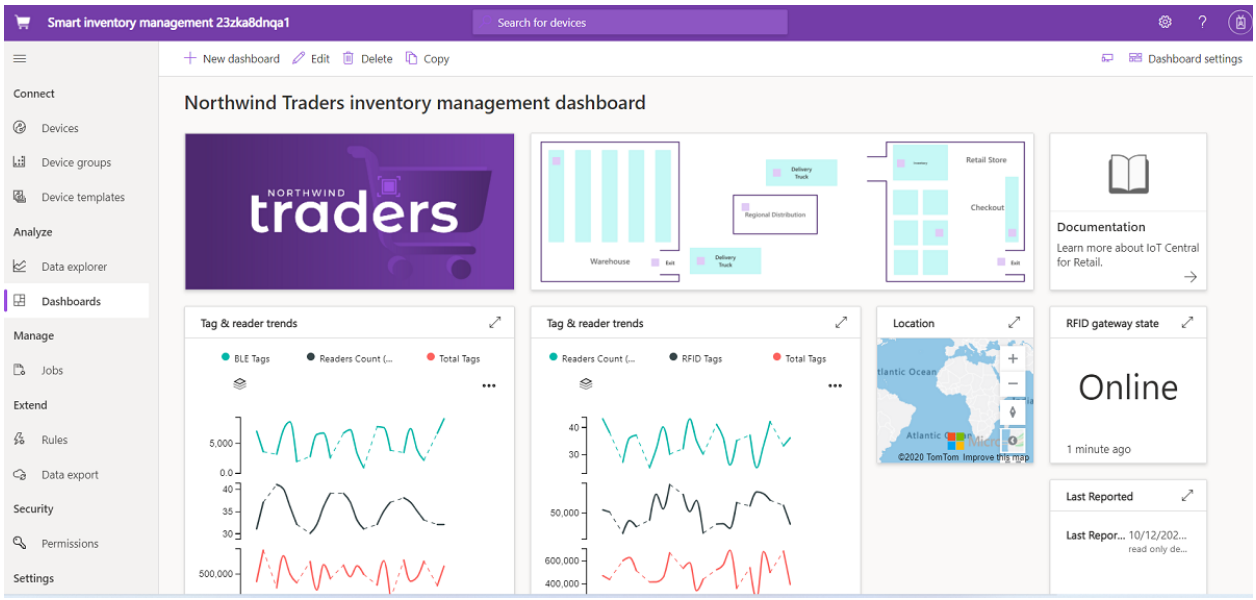


Рисунок 1.3 – Інтерфейс Microsoft Azure IoT Hub

Багатошарова архітектура з рознесеним навантаженням по кільком IoT Hub гарантує підтримку мільйонів одночасних підключень. Інтеграція з Azure IoT Edge дає змогу виконувати обробку телеметрії прямо на пристроях, знижуючи затримки та навантаження на канал. Однак налаштування ролей та політик доступу, конфігурація Device Provisioning Service і правил маршрутизації може ускладнити перший етап розгортання. Отже рішення на базі Microsoft Azure IoT Hub надає високу масштабованість і надійність, як і

AWS, але при цьому пропонує більш зручний інтерфейс з інтегрованими віджетами й інтерактивними графами топології пристроїв. Але важливим недоліком цього рішення є прив'язка до екосистеми Azure, що ускладнює інтеграцію з іншими сервісами, хоча і не є неможливою.

1.3.3 IBM Watson IoT Platform

IBM Watson IoT Platform [13] – це хмарний сервіс для підключення IoT-пристроїв, їх управління та аналіз телеметрії від цих пристроїв. Платформа підтримує протоколи MQTT, HTTP та WebSockets. Також надає зручний RESTful API для різного роду інтеграцій з іншими сервісами. Має можливість групувати пристрої, застосовувати різні політики безпеки на основі сертифікатів або різних токенів [14].

Інтерфейс IBM Watson IoT Platform (див. рис. 1.4) містить панель приладів, для відображення телеметрії у реальному часі, різного роду віджети для моніторингу статусу приладів та різні журнали подій.

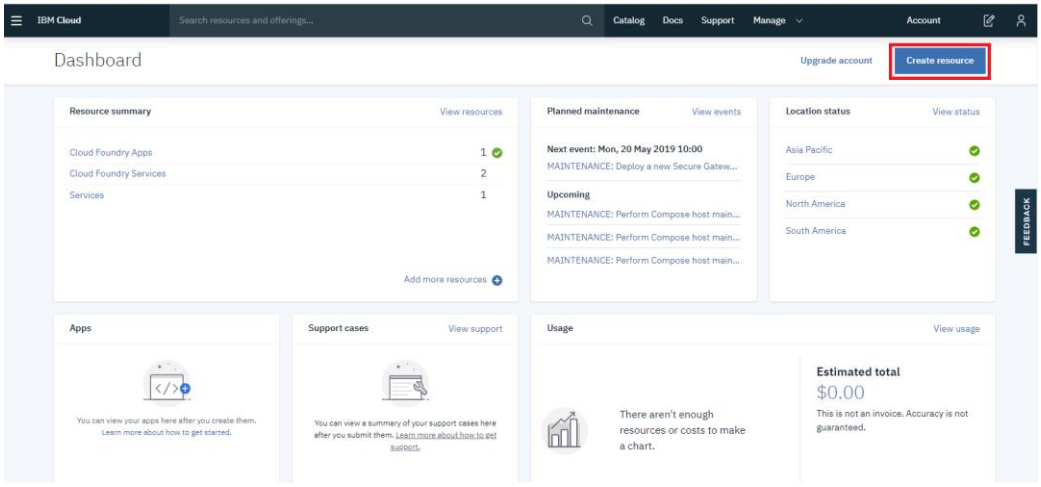


Рисунок 1.4 – Інтерфейс IBM Watson IoT Platform

Отже IBM Watson IoT Platform надає гнучке централізоване управління пристроями з привабливим інтерфейсом, але має трохи менше функціоналу ніж у її конкурентів.

2 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ СИСТЕМИ

2.1 UML проєктування ПЗ

Для моделювання системи та розробки діаграм було використано сервіс Draw.io [15]. Draw.io – інструмент для створення діаграм, блок-схем, інтелект-карт, бізнес-макетів, відносин сутностей, програмних блоків та іншого. Сервіс розповсюджується на безкоштовній основі з відкритим кодом. Draw.io має багатий набір функцій для візуалізації більшості завдань користувача [16].

Для визначення того, як користувач буде взаємодіяти з системою та визначення основних сценаріїв взаємодії, було створено діаграму прецедентів. Діаграма прецедентів показує різні варіанти використання, типи користувачів систем і часто супроводжується іншими типами діаграм [17].

Розроблена діаграма прецедентів (див. рис. 2.1), на якій відображено різні варіанти використання системи.

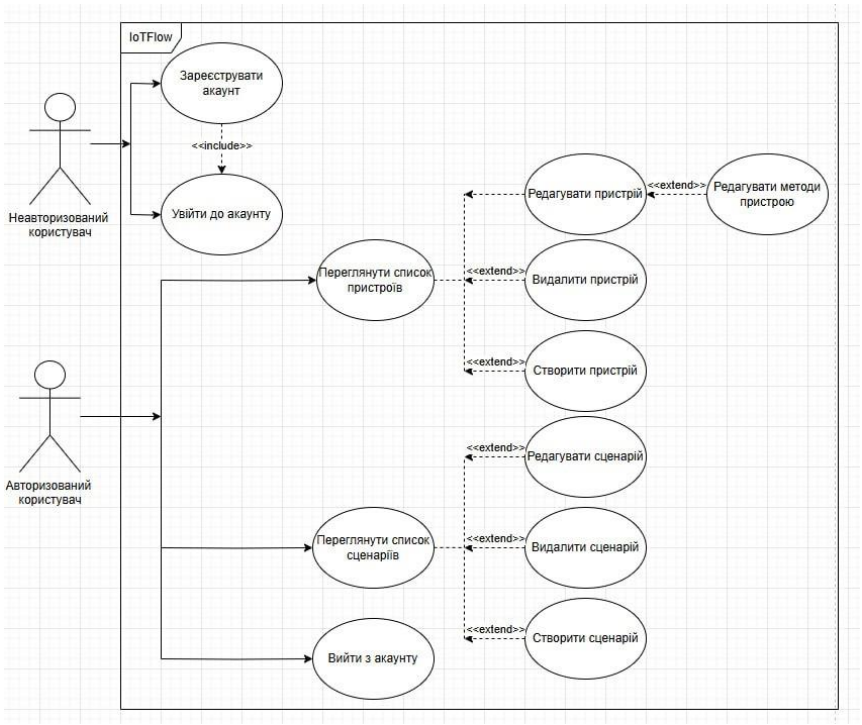


Рисунок 2.1 – Діаграма прецедентів

Неавторизований користувач має можливість реєстрації облікового запису чи входу до свого акаунту. У свою чергу авторизований користувач має доступ до головного інтерфейсу, де можна переглянути список всіх зареєстрованих користувачем пристроїв. Якщо користувач не має пристрою, він може створити пристрій вказавши його назву, до цього пристрою згенерується унікальний індетифікатор. Також йому стають доступні опції по видаленю пристрою чи його редагуванню. У межах редагування пристрою користувач може змінити його назву та задати аргументи команд (якщо команди були вже надані відповідним пристроєм).

Аналогічним чином реалізовано інтерфейс сценаріїв – користувач так само може переглянути список всіх сценаріїв та може з цього ініціювати створення, редагування та видалення. У межах створення чи редагування сценарію користувач може додавати певну послідовність умовних позначень BPMN, після чого в нього з’являється можливість присвоїти певному позначеню конкретну команду пристрою і перевизначити аргументи виклику, що потім будуть зчитуватися і передаватися до пристою. Також важливо зазначити, що користувач в будь-який момент може зайти та змінити передані ним аргументи чи видалити їх за потреби. BPMN – система умовних позначень (нотація) для моделювання бізнес-процесів [18]. BPMN відіграє ключову роль у візуальній частині формування сценаріїв.

Таким чином, вся логіка зосереджена навколо двох базових прецедентів «Переглянути список пристроїв» та «Переглянути список сценаріїв», з яких розгортаються всі інші функції.

Щоб описати обчислювальні вузли, компоненти та об’єкти, що виконуються у цих вузлах, було створено діаграму розгортання (див рис. 2.2).

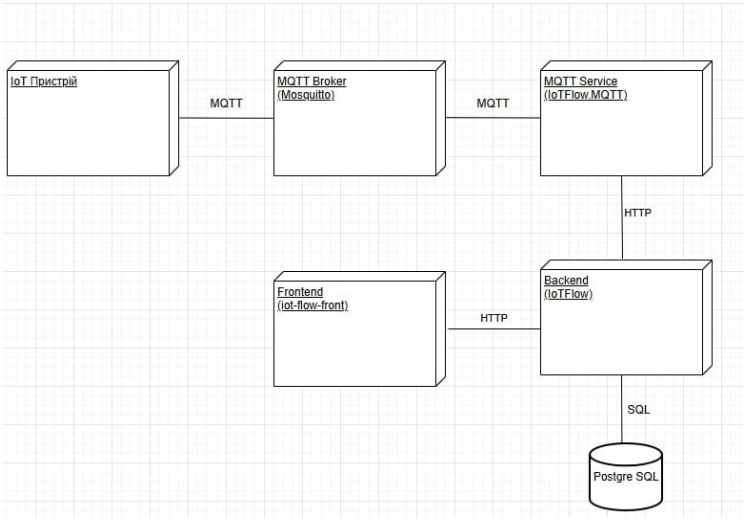


Рисунок 2.2 – Діаграма розгортання

Перший рівень діаграми складають пристрої, що зчитують виробничі параметри (температуру, вологість, рух, тощо) і передають їх за протоколом MQTT до проміжного вузла MQTT Broker (Mosquitto). Сам брокер виконує лише функцію маршрутизації повідомлень. Кожен пристрій публікує свої дані у відповідні теми, а підписники отримують ці повідомлення.

Другий рівень містить MQTT Service, який виступає спеціалізованим клієнтом MQTT, що формує спеціальні підписки на теми. Після підписки він приймає вхідні телеметричні дані, здійснює їх первинну обробку (валідація, нормалізація та парсинг даних) і надсилає результати у форматі HTTP POST на вузол Backend (IoTFlow).

Третій рівень є центральним Backend (IoTFlow). Він приймає HTTP-запити від MQTT сервісу та зберігає дані в реаліційну базу даних PostgreSQL. Окрім цього Backend надає REST-інтерфейс для динамічного веб-клієнта.

Останнім рівнем у цьому ланцюгу є Frontend (iot-flow-front), який розгортається, як односторінковий веб-додаток (SPA) на Vue.js. Він звертається до Backend через HTTPS/REST для отримання пристроїв, сценаріїв та відображає статистику часу виконання на пристроях. Також відправляє команди на пристрій через ті самі HTTP-ендпоінти.

Кожен компонент може розміщуватися на окремому сервері чи контейнері – це дозволяє масштабувати рішення горизонтально і легко адаптувати навнтаження виробничого середовища.

2.2 Проєктування архітектури ПЗ

2.2.1 Архітектура серверної частини

Серверна частина має бути реалізована, як веб-додаток на платформі .NET (C#). Буде представляти собою ASP.NET застосунок, що інкапсулює бізнес-логіку та забезпеує доступ до даних для кілєнтів. Архітектура серверної частини буде побудована за принципом N-tier (багатошарова).

Багатошарова архітектура має кілька основних рис. По-перше, вона забезпечує розділення обов'язків, де кожен шар виконує певні завдання, що дозволяє чітко розділити відповідальність між різними частинами системи. По-друге, вона забезпечує модульність, де шари працюють незалежно один від одного, що дозволяє легко замінювати або оновлювати окремі частини системи без впливу на інші. По-третє, вона забезпечує масштабованість, через що легко додавати нові функціональні можливості, розширюючи існуючі шари або додаючи нові. По-четверте, вона забезпечує легкість у підтримці та чітке розділення логіки додатку на шари, спрощує відлагодження, тестування та підтримку коду [19].

Типові шари в N-Tier архітектурі складаються з: презентаційного шару, логічного шару, шару доступу до даних та шару зберігання даних. Презентаційний шар відповідає за взаємодію з користувачем і відображення даних. Логічний шар відповідає за обробку бізнес-логіки додатку. Шар доступу до даних забезпечує з’єднання з базою даних та іншими джерелами даних. Шар зберігання даних містить бази даних та інші системи зберігання даних [19]. Приклад класичної реалізації, зображено на рисунку 2.3.

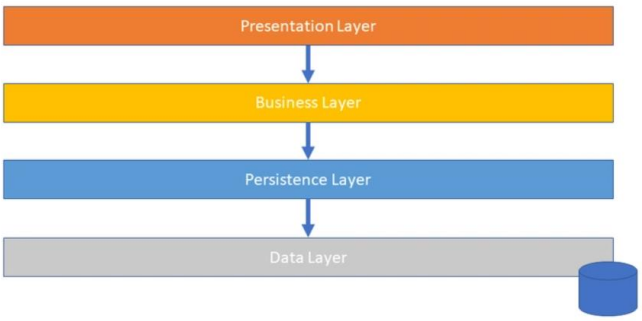


Рисунок 2.3 – Класична реалізація багатошарової архітектури [19]

Основна функціональна частина буде реалізована у шарі бізнес-логіки, де і має відбуватися обробка даних, їх валідування, виконання алгоритмів відпрацювання сценаріїв, тощо. Для розділення відповідальності бізнес-логіки буде реалізована за шаблоном Service Layer. Цей шаблон буде гарантом, що рівень представлення не має жодного уявлення про існування рівня доступу до даних [20].

2.2.2 Архітектура MQTT-брокера

Цей програмний засіб буде виконувати роль брокера, тобто він буде займатися маршрутизацією повідомлень за допомогою протоколу MQTT. MQTT спрямований на максимізацію використання пропускної здатності, є заміною традиційної споживчої архітектури, яка безпосередньо взаємодіє з кінцевою точкою [21].

MQTT надає спосіб створення ієрархії каналів зв’язку – свого роду гілка з листям. Щоразу, коли видавець має нові дані для поширення серед клієнтів, повідомлення супроводжується приміткою контролю доставки. Клієнти вищого рівня можуть отримувати кожне повідомлення, тоді як клієнти нижчого рівня можуть отримувати повідомлення, які стосуються лише одному чи двом базовим каналам, «відгалуженим» у нижній частині ієрархії. Це полегшує обмін інформацією розміром від двох байт до 256 Мб [21].

Важливо визначити клієнта, це може бути будь-який пристрій з доступом до мережі і можливістю виконувати програмний код. Бібліотеки підключення до MQTT є на багатьох мовах програмування таких як – Android, Arduino, C, C++, C#, Go, iOS, Java, JavaScript, NET [21].

Брокер має постійно підтримувати TCP-з'єднання з клієнтами та оброблювати їх повідомлення за допомогою спеціальних тем. Після отримання повідомлення він має їх обробити та відправити до основного сервера.

Також це ПЗ не лише має можливість отримувати повідомлення, а й відправляти команди з певними аргументами до пристроїв. У свою чергу пристрої мають забезпечити MQTT-брокер повідомленнями при створенні з'єднання з усіма доступними командами та аргументами, що вони приймають.

Як результат, можна буде забезпечити зворотній зв'язок і виконання певних команд як на пристроях, так і на серверній частині.

Архітектура MQTT-брокера побудована таким чином, щоб ізолювати роботу з брокером від іншої частини системи і щоб мати більше можливостей для оброки пристроїв.

2.2.3 Архітектура клієнтської частини

Клієнтська частина буде представляти собою класичний веб-додаток (SPA – Single Page Application) розроблений з використанням фреймоворку Vue.js [22]. Архітектура клієнтської частини спрямована на забезпечення інтерактивного інтерфейсу, що відображає дані з пристроїв у зручному форматі та дозволяє користувачам взаємодіяти із системою.

SPA застосунок завантажується, як єдина веб-сторінка, яка динамічно оновлюється при взаємодії користувача [23]. Компонування інтерфейсу буде реалізовано засобами Vue компонентів. Для збереження стану користувача буде використано Vuex [24] – це дозволить синхронізувати дані між собою і забезпечити реактивне оновлення при змінах стану.

Клієнтська частина взаємодіє із сервером через запити до REST API. Для виконання HTTP-запитів буде використано бібліотеку Axios [25]. За допомогою цієї бібліотеки, при завантаженні сторінки, веб-додаток буде надсилати запит до сервера із запитом на отримання потрібних даних для клієнта чи навпаки, клієнт буде надсилати дані серверу.

2.3 Проєктування структури бази даних

Для зберігання даних було обрано СКБД PostgreSQL. PostgreSQL – це потужна об'єктно-реляційна система баз даних з відкритим вихідним кодом, яка використовує і розширює мову SQL у поєднанні з багатьма функціями, що дозволяють безпечно зберігати і масштабувати найскладніші робочі навантаження даних [26]. Через такі переваги, як підтримка ACID-транзакцій, підтримання складних запитів, аналітики та відкритий програмний код. Під час проєктування структури зберігання даних була створена ER діаграма (рис. 2.4).

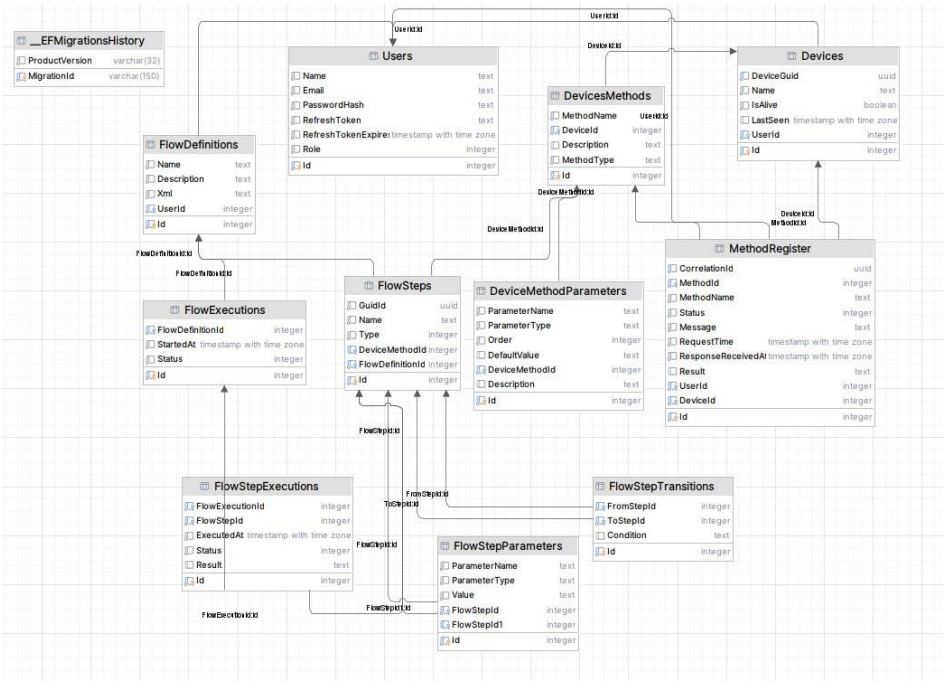


Рисунок 2.4 – ER діаграма

Таблиця «Users» (табл. 2.1) є ключовою для зберігання даних про всіх користувачів системи. Вона дозволяє відслідковувати інформацію про кожного користувача, включаючи його ім'я, електронну пошту та роль у системі. Важливим аспектом є наявність поля для зберігання токенів і хешу пароля, що забезпечує безпеку для оновлення сеансу користувача. Це гарантує, що кожен користувач має відповідний рівень доступу, наприклад, адміністратор чи оператор. Також таблиця включає інформацію про термін дії токенів, що важливо для контролю безпеки доступу.

Таблиця 2.1 – Опис сутності «Users»

Назва	Опис
Id	Унікальний ідентифікатор користувача.
Name	Ім'я користувача.
Email	Електронна пошта користувача.
PasswordHash	Хеш пароля користувача для забезпечення безпеки.
RefreshToken	Токен оновлення для сесії користувача.
RefreshTokenExpires	Термін дії токену оновлення.
Role	Роль користувача

Таблиця «Devices» (табл. 2.2) зберігає інформацію про всі IoT-пристрої, підключені до системи. Вона має унікальний ідентифікатор кожного пристрою, що дозволяє чітко відрізняти один пристрій від іншого, а також його назву для легшого сприйняття. Статус пристрою визначається полем «IsAlive», яке вказує на активність пристрою в системі. Час останньої активності («LastSeen») допомагає відслідковувати, коли пристрій востаннє взаємодіяв із системою.

Також в таблиці є посилання на користувача, якому належить пристрій, що забезпечує організацію і контроль за підключеними пристроями.

Таблиця 2.2 – Опис сутності «Devices»

Назва	Опис
Id	Ідентифікатор пристрою.
DeviceGuid	Унікальний ідентифікатор пристрою.
Name	Назва пристрою.
IsAlive	Статус активності пристрою (активний/неактивний).
LastSeen	Час останньої активності пристрою.
UserId	Посилання на користувача, якому належить пристрій.

Таблиця «FlowDefinitions» (табл. 2.3) зберігає основну інформацію про сценарії виконання, які використовуються в системі. Вона має унікальний ідентифікатор потоку, назву та опис, що дозволяють зрозуміти, який саме процес описується. Поле «Xml» містить структуру XML, що дозволяє зберігати інформацію про BPMN модель у вигляді структурованих даних. Також є посилання на користувача, який створив цей потік, що дозволяє відслідковувати авторство і контролювати зміни в процесах.

Таблиця 2.3 – Опис сутності «FlowDefinitions»

Назва	Опис
Id	Ідентифікатор потоку.
Name	Назва потоку.

Продовження таблиці 2.3

Назва	Опис
Description	Опис потоку.
Xml	XML-структура для зберігання BPMN моделі.
UserId	Посилання на користувача, який створив цей потік.

Таблиця «FlowExecutions» (табл. 2.4) фіксує інформацію про кожен запуск бізнес-процесу. Вона включає унікальний ідентифікатор виконання, що дозволяє відстежувати кожен запуск потоку. Поле «FlowDefinitionId» містить посилання на потік, що був виконаний, що дозволяє зв'язати виконання з конкретним визначенням потоку. Час старту вказує, коли саме почався процес, а статус допомагає визначити поточний стан виконання, наприклад, чи було завершено виконання, чи виникла помилка.

Таблиця 2.4 – Опис сутності «FlowExecutions»

Назва	Опис
Id	Ідентифікатор виконання потоку.
FlowDefinitionId	Посилання на визначення потоку.
StartedAt	Час початку виконання потоку.
Status	Поточний статус виконання потоку

Таблиця «FlowSteps» (табл. 2.5) описує окремі етапи в межах потоку виконання. Кожен крок в потоці має унікальний ідентифікатор (Guid), назву та тип, що дозволяє чітко визначити його функцію в загальному процесі. Також є посилання на метод пристрою, який виконується на цьому кроці. Це важливо для інтеграції різних пристроїв та їх функцій у рамках виконання потоку.

Таблиця 2.5 – Опис сутності «FlowSteps»

Назва	Опис
Guid	Унікальний ідентифікатор кроку.
Name	Назва кроку.
Type	Тип кроку
DeviceMethodId	Посилання на метод пристрою, який виконується на цьому кроці.
FlowDefinitionld	Збереження ідентифікатора визначення потоку.

Таблиця «FlowStepParameters» (табл. 2.6) містить змінні, які використовуються в конкретному кроці потоку. Кожен параметр має унікальний ідентифікатор та опис, що дозволяє чітко зрозуміти його значення та роль в процесі. Тип параметра та його значення також зберігаються в таблиці, що дає змогу динамічно змінювати або налаштовувати параметри кроків потоку відповідно до потреб користувача чи системи.

Таблиця 2.6 – Опис сутності «FlowStepParameters»

Назва	Опис
Id	Унікальний ідентифікатор параметра.
ParameterName	Назва параметра.
ParameterType	Тип параметра (наприклад, число, рядок, тощо).
Value	Значення параметра для цього кроку.
FlowStepId	Посилання на крок потоку, з яким пов'язаний параметр.

Таблиця «FlowStepTransitions» (табл. 2.7) описує логіку переходу між кроками потоку. Вона дозволяє визначити, як кроки взаємодіють між собою, залежно від умов переходу. Кожен запис містить посилання на початковий та кінцевий крок, а також умову, за якою відбувається перехід. Це забезпечує гнучкість і можливість динамічного налаштування алгоритмів виконання потоків.

Таблиця 2.7 – Опис сутності «FlowStepTransitions»

Назва	Опис
Id	Ідентифікатор переходу між кроками.
FromStepId	Ідентифікатор початкового кроку.
ToStepId	Ідентифікатор кінцевого кроку.
Condition	Умова, яка визначає перехід від одного кроку до іншого.

Таблиця «DeviceMethodParameters» (табл. 2.8) зберігає параметри, які використовуються в методах пристроїв. Вона містить інформацію про кожен параметр, його тип, значення за замовчуванням та опис. Це дозволяє налаштовувати методи пристроїв на основі параметрів, що вказуються користувачем чи системою, забезпечуючи точність і гнучкість в управлінні IoT-пристроями.

Таблиця 2.8 – Опис сутності «DeviceMethodParameters»

Назва	Опис
Id	Ідентифікатор параметра методу пристрою.
ParameterName	Назва параметра.

Продовження таблиці 2.8

Назва	Опис
ParameterType	Тип параметра.
Order	Порядковий номер параметра в методі.
DefaultValue	Значення параметра за замовчуванням.

Таблиця «DevicesMethods» (табл. 2.9) містить список функцій, які можуть виконувати IoT-пристрої в системі. Кожен метод має унікальний ідентифікатор і опис, що пояснює його призначення. Поле «DeviceId» дозволяє зв'язати метод з конкретним пристроєм, а тип методу («MethodType») вказує на його функціональність, наприклад, чи це команда для ввімкнення пристрою або зчитування даних.

Таблиця 2.9 – Опис сутності «DevicesMethods»

Назва	Опис
Id	Ідентифікатор методу пристрою.
MethodName	Назва методу пристрою.
DeviceId	Посилання на пристрій, до якого належить метод.
Description	Опис методу пристрою.
MethodType	Тип методу (наприклад, виконання, отримання даних).

Таблиця «MethodRegister» (табл. 2.10) фіксує інформацію про виклики методів пристроїв. Вона має ідентифікатор, що дозволяє відслідковувати послідовність викликів, а також статус виконання методу. Час виклику та

отримання відповіді зберігаються для контролю за тривалістю виконання та надійністю роботи пристроїв.

Таблиця 2.10 – Опис сутності «MethodRegister»

Назва	Опис
Id	Унікальний ідентифікатор запису реєстрації.
CorrelationId	Кореляційний ідентифікатор для відслідковування викликів.
MethodId	Посилання на метод пристрою, що виконується.
MethodName	Посилання на назву методу пристрою, що виконується.
Status	Статус виконання методу (наприклад, успішно, помилка).
Message	Повідомлення про результат виконання методу.
RequestTime	Час виклику методу.
ResponseReceivedAt	Час отримання відповіді на виклик методу.

Таблиця «_EFMigrationHistory» (табл. 2.11) є системною таблицею, що відстежує зміни в структурі бази даних. Вона містить історію міграцій, що дозволяє відновлювати попередні версії бази даних або стежити за її розвитком. Це важливо для підтримки стабільності і можливості відкату змін у випадку необхідності.

Таблиця 2.11 – Опис сутності «_EFMigrationHistory»

Назва	Опис
MigrationId	Ідентифікатор міграції.
ProductVersion	Версія продукту, для якого була виконана міграція.

2.4 Створення UI/UX дизайну системи

Останнім кроком перед розробкою програмної системи є проєктування зовнішнього вигляду та дизайну. Це необхідно, щоб надати зручний інструмент для користувача, він має бути простим і зрозумілим у керуванні.

Першим кроком у створенні інтуїтивного інтерфейсу стануть мокапи. Для цього було використано інструмент Balsamiq [27].

Щоб користувач міг зареєструватися у системі, йому необхідно вести всі необхідні поля, для цього було створено мокап сторінки реєстрації (рис 2.5), де користувач має ввести своє ім’я (логін), пошту та пароль.

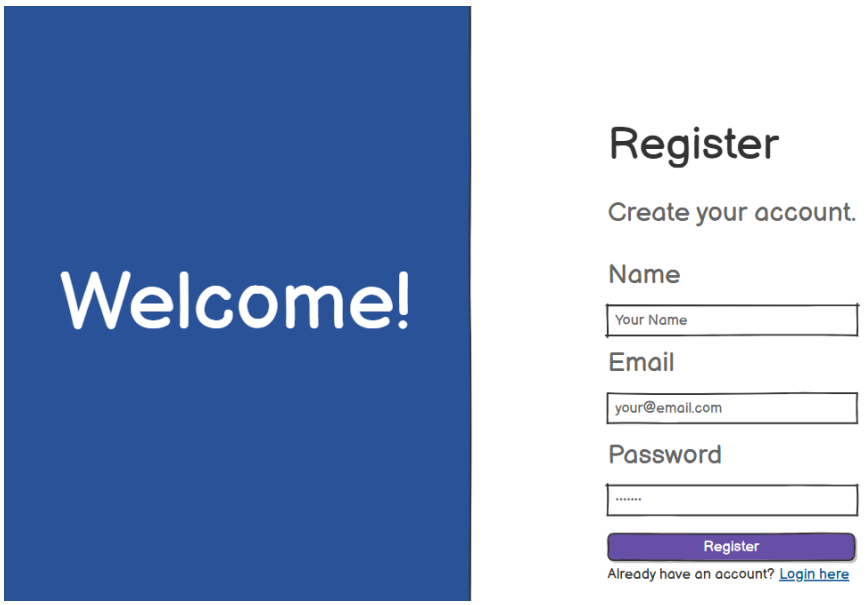


Рисунок 2.5 – Форма реєстрації

На рисунку 2.6 наведено мокап сторінки входу до облікового акаунту. На цій формі користувач може обрати, метод входу, а саме – він буде входити за допомогою імені чи за допомогою пошти. По дизайну сторінка дуже схожа на форму реєстрації з основною відмінністю у кольорі. Також ця форма містить в собі галочку посилання для користувача, якщо він забув свій пароль, це посилання буде переадресовувати користувача на іншу сторінку, де користувач зможе змінити свій пароль.

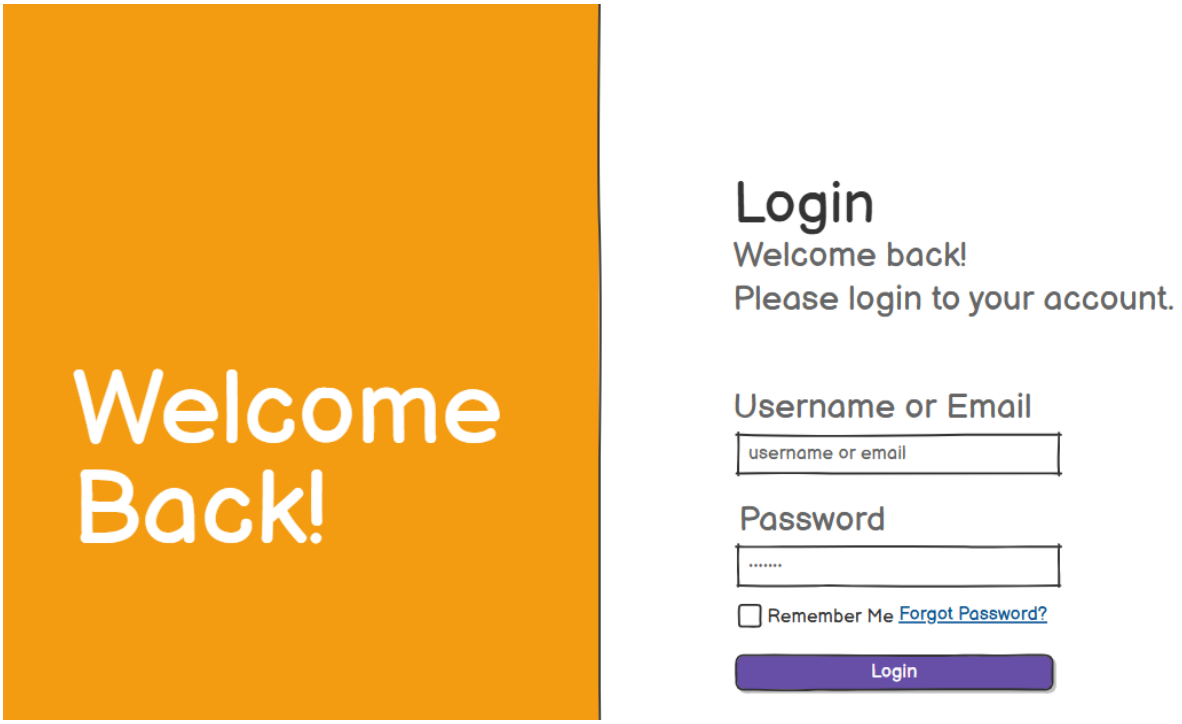


Рисунок 2.6 – Форма входу до акаунта

На рисунку 2.7 наведено сторінку перегляду пристроїв. На цій сторінці можна побачити дві частини. Ліва частина відповідає за меню, щоб користувач міг легко переміщатися між сторінками. У нижній частині відображається текст з ім’ям користувача та кнопкою «Log out», при натисканні на яку буде здійснено вихід з облікового запису та переадресація на форму логіну. Права частина показує пункт меню «Devices», яка містить в собі таблицю з полями: ім’я, ідентифікатор пристрою, час коли пристрій останній раз був у мережі та галочку, що показує чи в мережі цей пристрій прямо зараз.

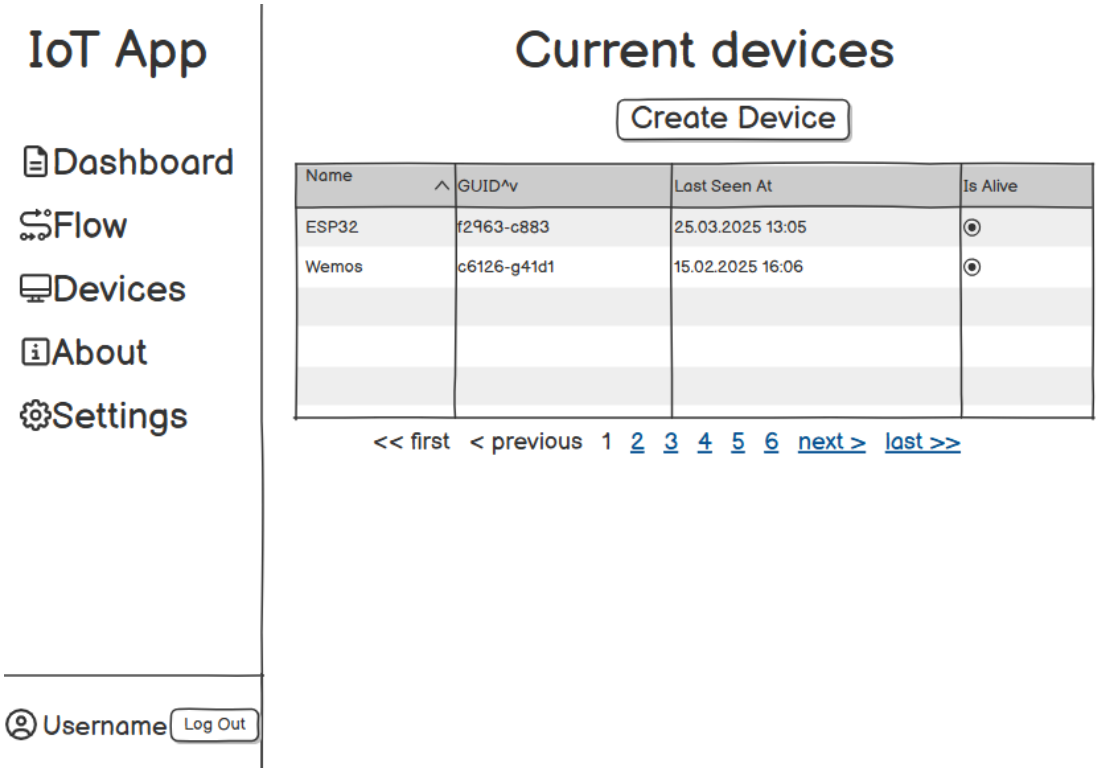


Рисунок 2.7 – Форма перегляду таблиці пристроїв

Мокап містить в собі кнопку «Create Device», що має відкривати модальну форму створення пристрою. При подвійному натисканні на рядок у таблиці має відкриватися редагування пристрою. Знизу під таблицею міститься компонент пагінатор – це потрібно для зміни поточної сторінки. Також на мокапі зображено кнопку, що має відкривати модальну форма для створення пристрою.

3 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ ТА АПАРАТНИХ РІШЕНЬ

3.1 Серверна частина

Для забезпечення слабкого зв’язування компонентів, полегшення тестування та гнучкої конфігурації проєкту було застосовано шаблон Dependency Injection (DI) та використано готову бібліотеку AutoFac [28].

DI має на меті розділити проблеми побудови об’єктів та їх використання, що призводить до слабозв’язаних компонентів. Шаблон гарантує, що об’єкт або функція, яка хоче використовувати певний сервіс, не повинна знати, як створити ці сервіси, натомість отримує свої залежності за допомогою зовнішнього коду, про який він не знає [29].

Нижче наведено приклад DI:

```
public static class DependencyInjector
    {
        public static void Load(ContainerBuilder containerBuilder)
        {
            containerBuilder
.RegisterAssemblyTypes(typeof(DependencyInjector).Assembly)
                    .AsSelf()
                    .AsImplementedInterfaces();
        }
    }
```

Щоб упорядкувати та оптимізувати обмін даними між різними шарами чи сервісами, гарантуючи, що передаються лише ті поля, що дійсно необхідні, без зайвих властивостей, використано шаблон Data Transfer Object (DTO). Головна різниця між звичайним об’єктом і DTO – це те, що такий об’єкт немає містити ніякої бізнес логіки [30].

Щоб полегшити роботу з DTO, використовується бібліотека AutoMapper [31]. AutoMapper допомагає з копіюваннями властивостей між об’єктами. Таким чином ми можемо прибрати дублікати та централізувати всю логіку

перетворень між моделями. При зміні об’єкту не потрібно буде всюди змінювати створення об’єкта, буде достатньо змінити лише в одному місці.

Щоб надати гарний інтерфейс і не визначати всюди AutoMapper, було використано шаблон адаптер. Його суть полягає у реалізації інтерфейсу одного об’єкта і обгортання його у інший, або у забезпеченні єдиного інтерфейсу. Завдяки такому підходу код залишається зрозумілим, бо при виклику не вдаємося до деталей. У додатку Б показана реалізація адаптера.

Важливим аспектом є безпека. Безпека користувача забезпечено механізмом JWT-токен. В момент входу в обліковий акаунт чи реєстрації після того, як була перевірена коректність вхідних даних, пароль користувача хешується у SHA-256, тому у БД потрапляє не просто текст. Далі створюються два окремі токени: AccessSecretCode та RefreshSecretCode. Підпис створюється симетричним ключем HMAC-SHA256, тому без цього секретного коду неможливо підробити дійсний код. Також кожен код має свій строк дії. У додатку В наведено інструмент, що створює дані токени.

Для реалізації MQTT-сервера використовується бібліотека MQTTnet, яка надає повну реалізацію MQTT-протоколу версії 3.1.1 і 5.0, включно з підтримкою різних рівнів QOS та зберігання повідомлень. Поточна реалізація дозволяє запустити окремий екземпляр на стандартному порту 1883, який слухає з’єднання від IoT-пристроїв. Ця бібліотека позбавляє необхідності запускати сторонні брокери, а усе працює і так у середовищі .NET.

Всі сервіси у проєкті мають власний інтерфейс – це потрібно, щоб шари залежали не від конкретних реалізацій, що зменшує залежність. Також це забезпечує зручність підтримки, бо зміни у реалізації не обов'язково вимагають змін у залежних шарах, якщо інтерфейс залишився незмінним.

3.2 База даних

Робота з БД організована за допомогою Entity Framework Core, таким чином ми взаємодіємо з БД за допомогою класів-сутностей. Зміни структури автоматично застосовуються за допомогою механізму міграції, що зменшує час на розробку SQL-скриптів.

Усі сутності, що зберігаються у БД мають успадковувати клас BaseEntity, який задає єдину структуру для основного ідентифікатора. Це приводить сутності до єдиного виду та спряє узгодженню даних та полегшує підтримку моделей.

Нижче наведено реалізацію BaseEntity:

```
public abstract class BaseEntity
    {
        [Key]
        public int Id { get; set; }
    }
```

Щоб відокремити шар доступу до даних від основної логіки, застосовано шаблон Repository та Unit of Work. Репозиторії містять у собі роботу з DbContext, від додавання до видалення запису, тоді як Unit Of Work забезпечує зберігання єдиним викликом метода SaveChangesAsync, забезпечуючи атомарність операції. У разі будь-якої помилки всі накопичені зміни відкотяться. Також UnitOfWork містить в собі метод по визначенню репозиторію. У додатку Г наведено реалізацію патерну Unit of Work та декілька прикладів репозиторіїв.

3.3 Створення апаратних макетів та їх системне налаштування

Щоб перевірити працездатність системи, було розроблено два макети. Перший макет зображено на рисунку 3.1. Даний макет представляє собою фото-пастку, що складається з двох мікроконтролерів та ноутбуку, моделі

мікроконтролерів: ESP32 та ESP8266 WEMOS D1 R2. Для розпізнання руху використано датчик HC-SR501 – це пасивний інфрачервоний датчик руху, який використовується для виявлення руху ESP32 має вбудовано камеру OV2640, яка активується лише після тригеру. До пристрою ESP8266 під’єднано датчик HC-SR501. У додатку А показано технічні характеристики використаних пристроїв та датчиків.

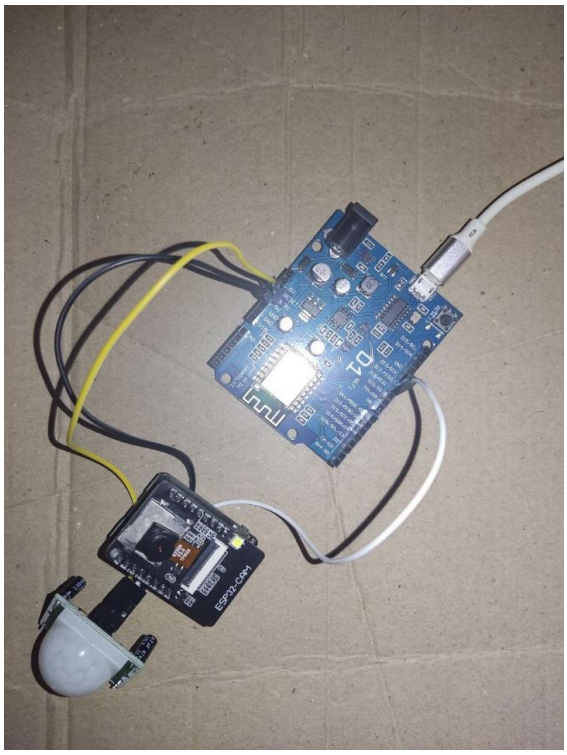


Рисунок 3.1 – Макет фото-пастки

Ідея даного макету полягає у тому, що пристрій WEMOS D1 R2 самостійно фіксує появу об'єкта в зоні спостереження за допомогою датчика, надсилає дані про появу об'єкта через MQTT до сервера, сервер надсилає команду до пристрою ESP32 і той робить фото, а після фото відображається на ноутбуку, що є пристроєм виведення. Налаштування даного макету можна побачити на рисунку 3.2 у форматі BPMN-процесу.

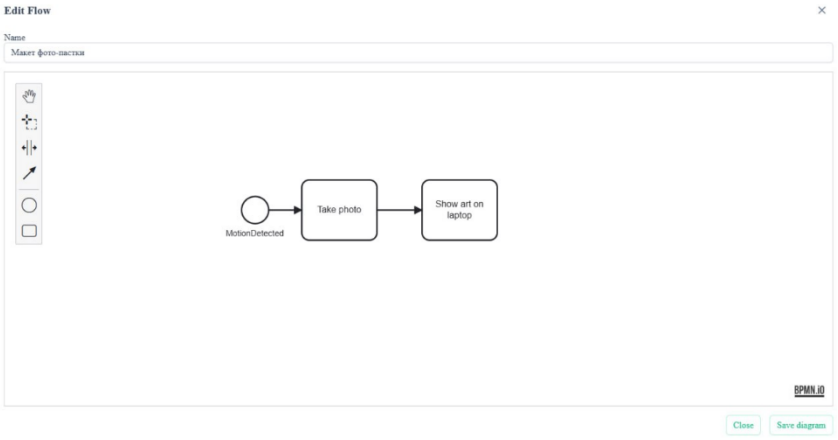


Рисунок 3.2 – Налаштування макета фото-пастка

Першим елементом у цій моделі є «StartEvent» – це початкова подія, яка має відбутися, щоб сценарій спрацював. Подія зображується колом і показує, де починається процес [18]. Далі стрілка веде на «Task» – це наша дія, яку ми хочемо зробити. Таких задач може бути багато, вони йдуть послідовно один за одним і у такому ж порядку виконуються. В нашому випадку ми маємо дві послідовні задачі, спочатку буде зроблено фото, а після воно буде показано на ноутбуці. Таким чином, ми з’єднали три пристрої, що не знають один про одного, але ми можемо використовувати їх як складові однієї системи.

Для другого прикладу було побудовано макет системи затоплення. Було використано WEMOS D1 R2, датчик затоплення SEN18, зуммер та червоний LED (рис. 3.3).

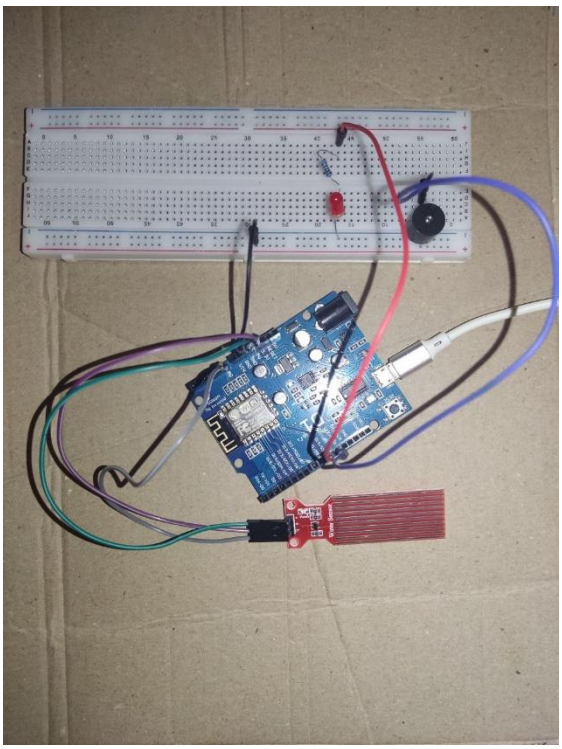


Рисунок 3.3 – Макет системи затоплення

Ідея даного макету полягає у реагуванні на затоплення за допомогою датчику затоплення. Після того як рівень води збільшиться заграє тривога, за допомогою зуммера і буде надіслано повідомлення до ноутбука. Налаштування цього макету зображено на рисунку 3.4.

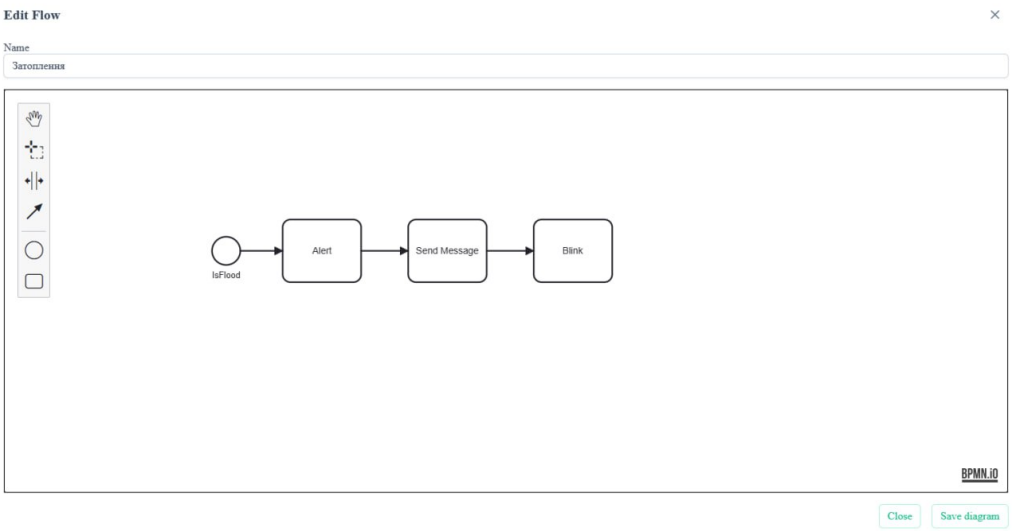


Рисунок 3.4 – Налаштування системи затоплення

Налаштування системи затоплення – є лінійним процесом реагування на затоплення. Як тільки датчик фіксує перевищення рівня води, одразу запускається послідовність трьох завдань. Спершу пристрій видає звуковий сигнал через зумер, щоб привернути увагу до небезпечної ситуації. Після цього мікроконтролер формує і відправляє повідомлення на підключений ноутбук, інформуючи користувача про початок потопу. Для відображення ситуації вмикається миготіння світлодіодом, щоб візуально підтвердити факт спрацювання.

Така система затоплення може бути корисна, коли потрібна швидко реагувати на витоки рідини, саме ця проста, але наочна схема дозволяє з мінімальними витратами виявити потоп. Схема є спрощена, через обмеження наявних пристроїв.

Третім макетом було зібрано систему сейфу. Цей макет складається з двох Wemos D1 R2 матричної клавіатури, лазера та серводвигуна (див рис. 3.5).

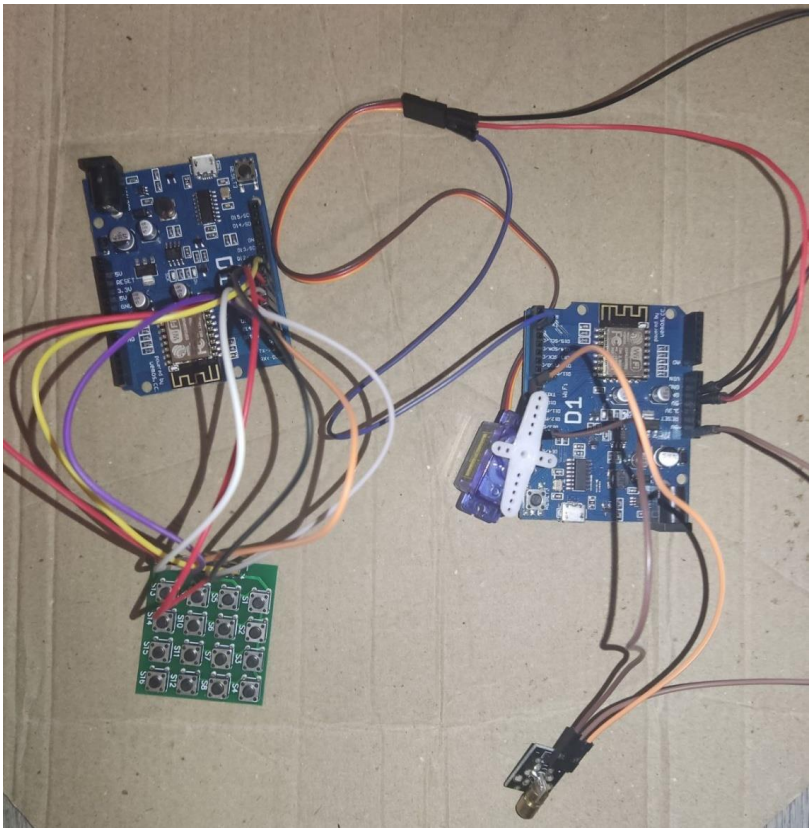


Рисунок 3.5 – Макет системи сейфа

Основна ідея даного макету у відкриванні/закриванні дверей за допомогою серводвигуна. У цієї системи також є механізм безпеки, щоб відкрились двері треба вести спеціальний код за допомогою матричної клавіатури. Користувач може задати цей пароль самостійно на цьому пристрої. Але для тестування системи я вирішив піти іншим шляхом, створив генерацію паролю, а саме пароль змінюється кожного разу, коли двері сейфа зачиняються. Налаштування генерації паролю можна побачити на рисунку 3.6.

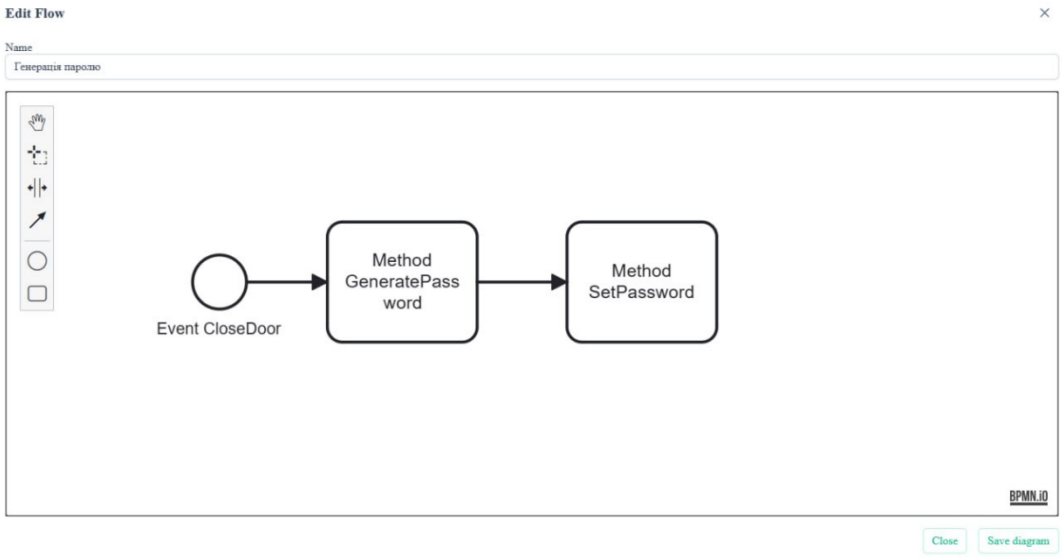


Рисунок 3.6 – Налаштування генерації паролю

Цей приклад, можна було б доповнити, що пароль надсилається у месенджер користувачу чи будь-які інший варіант сповіщення. Але в інших макетах це вже неодноразово використовувалось, тому тут розрахунок на те, що користувач зможе подивитися пароль через платформу, за допомогою перегляду результату виконання метода на генерацію пароля. Налаштування відкривання дверей за допомогою паролю зображено на рисунку 3.7.

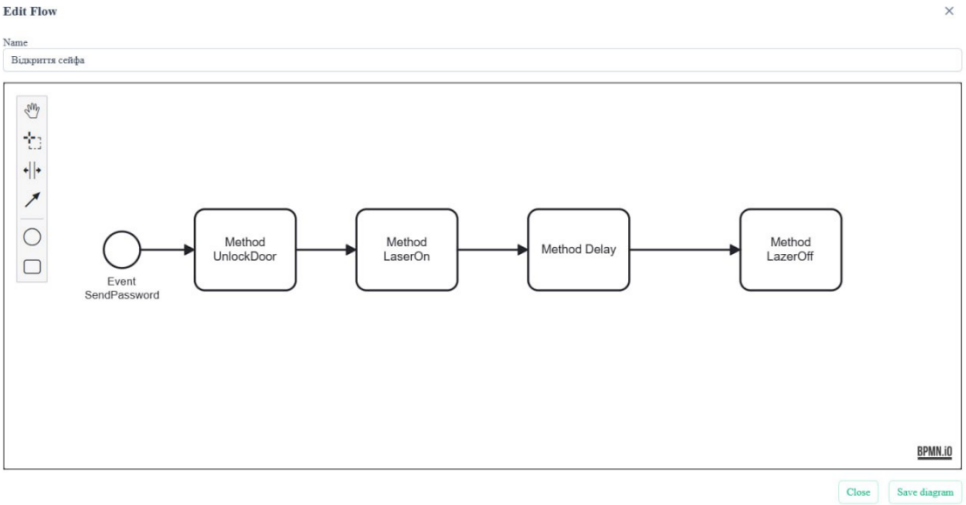


Рисунок 3.7 – Налаштування відкривання дверей

Коли система отримує будь-яке значення, з матричної клавіатури вона запускає сценарій, переходить до методу відкривання дверей, цей же метод перевіряє пароль, якщо він співпадає то двері відкриваються, якщо ні то нічого не відбувається. Далі після цього по сценарію йде включення лазеру він включається на секнуду лише як сигнал про відкриття дверей.

Пристрій Wemos D1 R2 до якого підключено матричну клавіатуру, він завжди зчитує значення, що натискається на клавіатурі та конкатенує ці значення. Якщо натиснуто клавішу «*» то значення відправляються на сервер, якщо ж натиснуто на «#» то значення, що були стираються. Таким чином, користувачу буде зручніше вводити значення, якщо він помилився у якомусь знаку. Також, що важливо матрична клавіатура має всього лише 16 знаків, що означає, що не всі літери можуть бути написані, тому при генерації паролю, зроблено так, що пароль завжди складається з символів, що є на матричній клавіатурі.

3.4 Опис роботи серводвигуна SG90 з точки зору ТАУ

Серводвигун SG90 – це привід, здатний здійснювати обертальний рух від 0° до 180°. Його діапазон обмежений, оскільки до кінцевої шестерні

прикріплена механічна перешкода, яка не дає змоги вихідному валу здійснити повний оберт [32].

Серводвигун є замкнутою системою, що містить електродвигун постійного струму з редуктором та датчиком положення (див. рис. 3.8).

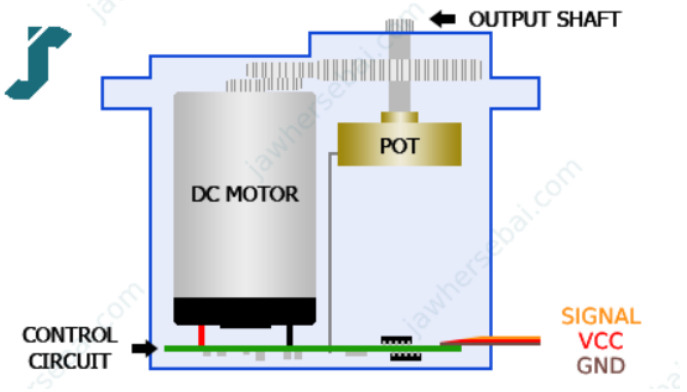


Рисунок 3.8 – Внутрішні компоненти серводвигуна

На рисунку 3.9, зображено функціональну схему замкненої системи позиційного керування серводвигуна SG90.

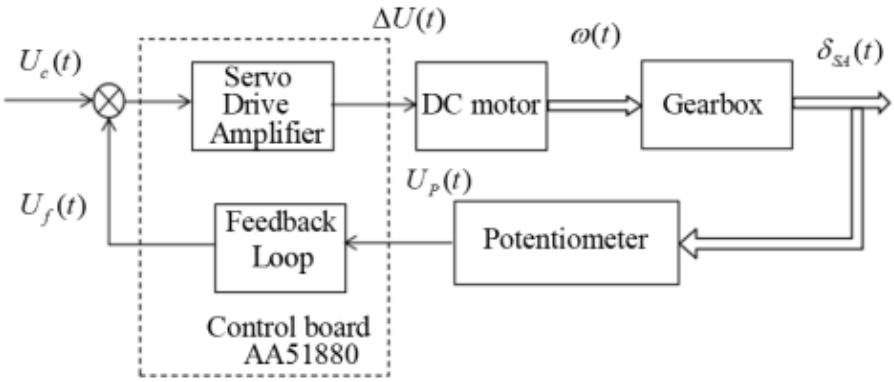


Рисунок 3.9 – Функціональна схема SG90

У якості вхідного сигналу система отримує напругу $U_c(t)$, що формується на вході ШІМ-контролера (У нашому випадку Wemos D1 R2) і змінює на бажаний кут обороту валу.

Серводрайв (Servo Drive Amplifier) — це вбудований у контролер AA51880 аналоговий підсилювач помилки, який перетворює ΔU(t) у керуючу напругу $U_p(t)$ для двигуна.

Далі $U_p(t)$ живить двигун, який через редуктор створює механічний момент і забезпечує кутову швидкість w(t) та кутове положення $\delta_{sd}(t)$ вихідного валу.

Потенціометр, що закріплений на вихідному валу, перетворює кут $\delta_{sd}(t)$ на електричний сигнал $U_f(t)$, який через ланцюг обробки повертається на вхід. Завдяки цій схемі система автоматично коригує положення вала.

Для розрахунку передавальної функції звернимось до електричного рівняння якоря, його записано за (3.1):

$$U(s) = RI(s) + L_s I_s + Es => RI(s) + L_s I_s + K_e \Omega_s, \qquad (3.1)$$

де U(s) – напруга на якір двигуна (В);

I(s) – струм якоря (А);

R ≈ 7.4 (Ом);

L ≈ 1 (мГн);

K_e ≈ 0.27 (В*с/рад);

Запишемо механічне рівняння (3.2):

$$J \cdot s \cdot \Omega(s) + T_{load}(s) = K_t I(s) \qquad (3.2)$$

де J – момент інерції якоря;

$T_{load}(s)$ – момент навантаження;

Kt – коефіцієнт крутного моменту;

Ω(s) – кутова швидкість валу;

Підставимо I(s) із електричного рівняння в механічне й виразимо Ω(s)/U(s):

$$\frac{\Omega(s)}{U(s)} = \frac{K_t}{(Ls+R)(Js+T_{load}(s)+K_tK_e}, \quad (3.3)$$

де U(s) – лапласів образ прикладеної напруги на обмотку;

Ω(s) – лапласів образ кутової швидкості якоря;

s – змінна Лапласа

L – індуктивність обмотки ротора

J – момент інерції ротора із редуктором

$T_{load}(s)$ – момент навантаження;

K_t – коефіцієнт крутного моменту;

K_e – коефіцієнт зворотньої ЕРС;

У результаті маємо передаточну функцію серводвигуна SG90 від поданої напруги до кутової швидкості, наведену у формулі (3.3). Це дозволяє оцінити динаміку руху ротора за різних значень вхідної напруги.

4 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА МАКЕТІВ

4.1 Тестування навантаження

Навантажувальне тестування – це підмножина тестування продуктивності, яка використовується для програмного забезпечення, веб-сайтів, програм і пов’язаних систем. Це нефункціональний тест, який імітує поведінку кількох користувачів, які одночасно отримують доступ до системи. Навантажувальне тестування, яке також називають «об’ємним тестуванням», відтворює продуктивність, стабільність і функціональність веб-системи в реальних умовах, тому це один із останніх і найважливіших типів тестування, який виконується перед розгортанням [33].

Для тестування було обрано програму Jmeter. Треба обрати найскладніший фрагмент коду для цього. Через те, що при отримані всіх пристроїв, ми отримуємо ще всі методи кожного пристрою, тому було обрано саме цей фрагмент.

На рисунку 4.1 зображено створено Thread Group, в якій кількість потоків встановлена в 1100, період відправки повідомлення встановлено в 1 секунду, а кількість повторів встановлено в 10 разів.

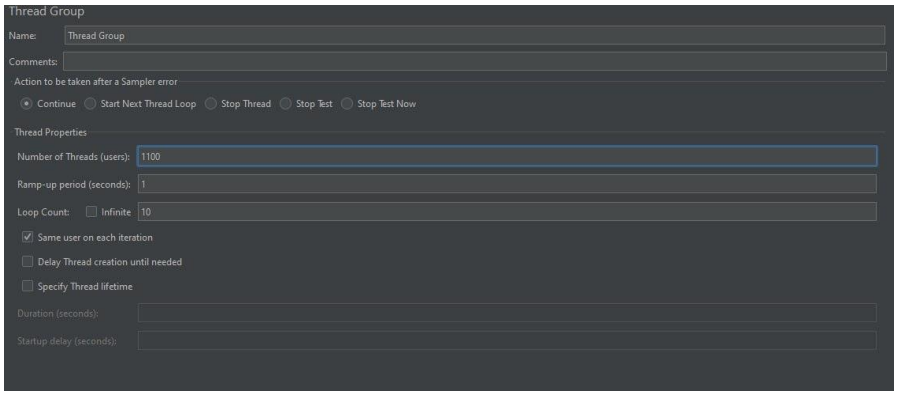


Рисунок 4.1 – Налаштування Thread Group

Отримані результати тестування можна побачити у таблиці 4.1.

Таблиця 4.1 – Результати тестування навантаженням

Показник	Значення
Кількість запитів	11000 шт.
Середні час	149 мс.
Медіана	70 мс.
90-й перцентиль	187 мс.
95-й перцентиль	1061 мс.
99-й перцентиль	1381 мс.
Мінімум	1 мс.
Максимум	1528 мс.
Відсоток помилок	0,02 % (~2 помилки)
Пропускна здатність	2988.3 запитів/с

Під даним сценарієм навантаження ми бачимо високу пропускну здатність, низький рівень помилок, що свідчить, що система витримала обрану інтенсивність та комфортні значення відповіді сервера для кінцевого користувача. Але якщо подивитися на 95-99 перцентилі то можна побачити, що в піках можуть бути підвисання інтерфейсу чи таймауту у частини користувачів. Щоб уникнути таких підвисання, можна використати кешування за допомогою Redis, CDN чи можливо вбудованих бібліотек кешування.

Якщо пікове навантаження може бути вищою за визначене у таблиці 4.1 то треба запроваджувати горизонтальне масштабування, а саме шаблону Load Balancer.

4.2 Тестування макетів

4.2.1 Макет фото-пастка

Цей макет призначений для фотографування об’єкта, що потрапив на детектор руху та відображення фотографії на ноутбуці. Якщо розбирати цей макет на BPMN-модель то він є дуже простим, у нас є початкова подія, а саме, що детектор руху виявив рух, а після цього йдуть послідовно створення і відправка фотографії, а коли фотографії вже була відправлена вона відображається у месенджері Telegram. Макет фото-пастки зображено на рисунку 3.1, а його налаштування на рисунку 3.2.

На пристрої з датчиком руху встановлено обмеження, щоб не кожну секунду надсилати повідомлення про рух, а з періодичністю, щоб штучно обмежити кількість запитів до камери, бо вона сильно обмежена у свої характеристиках. Пристрій з камерою налаштований на передачу JPEG-формату, з роздільною здатністю 1280 на 1024 пікселя. Також назначено високу якість стиску, для кращої якості та більшого розміру. Через такі налаштування ми отримуємо деталізовану фотографію, але через великий об’єм даних це займає деякий час на обробку та відправку. Нижче надано частину коду налаштування камери.

```
bool configInitCamera() {
camera_config_t config;
config.pixel_format = PIXFORMAT_JPEG;
if (psramFound()) {
config.frame_size = FRAMESIZE_SXGA;
config.jpeg_quality = 4;
config.fb_count = 2;
config.grab_mode = CAMERA_GRAB_LATEST;
config.fb_location = CAMERA_FB_IN_PSRAM;
```

```
}
else {
config.frame_size = FRAMESIZE_SVGA;
config.jpeg_quality = 1;
config.fb_count = 1;
config.fb_location = CAMERA_FB_IN_DRAM;
}
esp_err_t err = esp_camera_init(&config);
return true;
}
```

4.2.2 Макет системи затоплення

Макет системи затоплення призначений для швидкої реакції на підвищення рівня води у заданій області. Він, як і минулий макет є простим у своїй реалізації BPMN налаштувань (див. рис 3.4). На цьому макеті початковою подією стає сигнал від пристрою про затоплення, система відправляє команду «Alert» до відповідного пристрою із зумером, щоб він видав відповідний сигнал та показує інформацію про рівень води на пристрої виведення. Нижче надано фрагмент коду з відправкою сигналу про затоплення.

```
void updateFloodSensor() {
int sensorValue = analogRead(A0);
bool flooded = sensorValue > floodThreshold;
if (flooded && !floodSent) {
processFlood("0");
floodSent = true;
}
else if (!flooded) {
```

```
floodSent = false;
}
}
```

4.2.3 Макет системи сейфа

На відміну від інших макетів, цей має декілька налаштованих BPMN-сценаріїв (див. рис. 3.6 - 3.7). Основне його призначення відкривати двері при правильному паролі. Пароль користувач надсилає з матричної клавіатури.

Перший сценарій (див. рис. 3.6) відповідає за створення паролю. Воно відбувається після того як двері зачиняються. Надсилається повідомлення до пристрою про генерацію та встановлення паролю, так як на клавіатурі всього лише шістнадцять символів, пароль обмежений цією кількістю символів, але довільною довжиною, що визначає користувач. Метод для генерації паролю наведено нижче.

```
String generatePassword(int len) {
const char pool[] = "0123456789ABCD";
String pw;
randomSeed(micros());
for(int i=0; i<len; i++){
pw += pool[random(0, sizeof(pool)-1)];
}
return pw;
}
```

Другим сценарієм (див. рис. 3.7) є процес відкривання дверей за допомогою паролю, якщо пароль співпадає то двері відкриваються, якщо ні то залишаються зачиненими, для сигналу використано лазер, який вмикається на секунду, завжди. Метод для відкривання дверей за паролем надано нижче.

```
    void processUnlockDoor(const String &correlationId, const String
&pwd) {
    if (pwd == currentPassword && pwd.length() > 0) {
    openDoor();
    doorOpen = true;
    sendResponse(correlationId, true, "UnlockDoor", "Correct");
    } else {
    sendResponse(correlationId, false, "UnlockDoor", "Incorrect");
    }
    }
```

4.3 Заходи безпеки при тестуванні

Під час апаратного тестування IoT-пристроїв першочерговим є забезпечення фізичної безпеки середовища виконання. Перш за все, слід дотримуватися стандартних правил безпеки при роботі з електрообладнанням, таких як:

– знеструмлювати пристрій перед будь-яким втручанням у його схему;
– уникати оголених проводів;
– уникати перевантаження системи;
– працювати лише з ізольованими інструментами;

Тестовий макет чи пристрій рекомендується закріплювати чи надійно розмістити у місці, де він не зможе впасти. Якщо будь-яка частина макету має рухомі складові, необхідно забезпечити, щоб навколо не було сторонніх предметів або людей у зоні дії цих об’єктів. Крім того, не варто залишати тестовий не випробуваний макет без нагляду, бо може статися ситуація яку ви не врахували, для прикладу перегрів пристрою або ураження струмом [34].

Якщо виконувати задані правила, шанс на небезпечну ситуацію сильно зменшується, але ніколи не дорівнює нулю. Реалізація наведених заходів безпеки при апаратному та програмному тестуванні IoT-пристроїв є обов’язковою умовою успішного впровадження таких пристроїв. Безпека вимагає комплексного підходу для уникнення фізичних травм.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було розроблено серверну та клієнтську частину програмної системи для автоматизації процесу збирання та обробки даних з пристроїв IoT.

В ході виконання роботи було проведено аналіз сучасних підходів до збирання та обробки даних, на основі цього була обрана подійно-орієнтовна архітектура. На підставі аналізу аналогічних рішень було виявлено недоліки та переваги різних систем, що дозволило чітко сформувати вимоги до системи.

Перед початком розробки системи, було створено різні діаграми, такі як: діаграма прецедентів, діаграма розгортання та ER діаграма. Виконана робота дозволила чітко визначити функціональні можлиовсті майбутнього застосунку та його внутрішню модель об’єктів і зв’язків. Крім того, відповідно до стандартів UI-розробки було створено макети користувацького інтерфейсу.

Результатом кваліфікаційної роботи стала система для автоматизації та обробки даних з IoT пристроїв. Програма надає не лише можливість отримувати та відправляти дані між IoT-пристроєм та сервером, а і можливість створювати сценарії реагування на події за допомогою внутрішнього інструмент на основі BPMN-моделей.

Для розробки програми були використанні мови програмування, такі як: C#, фреймворк ASP.NET, JavaScript, фреймворк Vue.js, ORM Entity Framework Core, PostgreSQL.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ДСТУ 3008: 2015. Інформація та документація. Звіти у сфері науки і техніки. Структура і правила оформлення. К.: ДП «УкрНДНЦ». 2016. 30 с.

2. Методичні вказівки з підготовки кваліфікаційної роботи бакалавра для студентів усіх форм навчання спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології» освітньої програми «Автоматизація та комп'ютерно-інтегровані технології» / Упоряд.: І.Ш. Невлюдов, А.О. Андрусевич, О.В. Токарєва, С.П. Новоселов, О.В Сичова. Харків: ХНУРЕ, 2022. – 55 с.

3. IoT. Базові технології від Інтернету людей до Інтернету речей: навч. посібник / В.П. Немченко, С.В. Чумаченко. – Харків, 2020. – 144 с.

4. Дані часових рядів та їх аналіз [Електронний ресурс] : Режим доступу: https://www.influxdata.com/what-is-time-series-data/ (дата звернення: 13.04.2025).

5. Lea P. IoT and Edge Computing for Architects: Implementing Edge and IoT Systems from Sensors to Clouds with Communication Systems, Analytics and Security (2nd ed.) / Perry Lea. – Birmingham : Packt Publishing, 2020. – 632 p.

6. Документація AWS IoT Core [Електронний ресурс] : Режим доступу: https://docs.aws.amazon.com/iot/ (дата звернення: 16.04.2025).

7. Rules for AWS IoT Core [Електронний ресурс] : Режим доступу: https://docs.aws.amazon.com/iot/ (дата звернення: 16.04.2025).

8. Визначення AWS IoT [Електронний ресурс] : Режим доступу: https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html (дата звернення: 17.04.2025).

9. Що таке AWS IoT Greengrass [Електронний ресурс] : Режим доступу: https://docs.aws.amazon.com/greengrass/v2/developerguide/what-is-iot-greengrass.html (дата звернення: 17.04.2025).

10. Безпека в AWS IoT [Електронний ресурс] : Режим доступу: https://docs.aws.amazon.com/iot/latest/developerguide/security.html (дата звернення: 17.04.2025).

11. Що таке Azure IoT Hub [Електронний ресурс] : Режим доступу: https://learn.microsoft.com/en-us/azure/iot-hub/iot-concepts-and-iot-hub (дата звернення: 16.04.2025).

12. Azure IoT Hub [Електронний ресурс] : Режим доступу: https://azure.microsoft.com/en-us/products/iot-hub (дата звернення: 16.04.2025).

13. Головна сторінка IBM IoT Solutions [Електронний ресурс] : Режим доступу: https://www.ibm.com/cloud/internet-of-things (дата звернення: 16.04.2025).

14. Огляд IBM Watson IoT [Електронний ресурс] : Режим доступу: https://www.codeguru.com/iot/iot-development-platforms-ibm-watson-iot-overview/ (дата звернення: 16.04.2025).

15. Draw io [Електронний ресурс] : Режим доступу: https://www.drawio.com/ (дата звернення: 06.05.2025).

16. Опис Draw.io [Електронний ресурс] : Режим доступу: https://vchymo.com/app/application/Drawio (дата звернення: 06.05.2025).

17. Діаграма прецедентів [Електронний ресурс] : Режим доступу: https://shorturl.at/HyIYt (дата звернення: 06.05.2025).

18. BPMN вікіпедія [Електронний ресурс] : Режим доступу: https://shorturl.at/7Sq6W (дата звернення: 06.05.2025).

19. Багатошарова (N-Tier) архітектура в .NET [Електронний ресурс] : Режим доступу: https://dev.to/dotnetfullstackdev/layered-n-tier-architecture-in-net-core-51ic (дата звернення: 06.05.2025).

20. Шаблон сервісного рівня [Електронний ресурс] : Режим доступу: https://studysection.com/blog/service-layer-pattern-in-c/ (дата звернення: 07.05.2025).

21. Node-RED та технологія промислового Інтернету речей : Навчальний посібник / І. Ш. Невлюдов, С. П. Новоселов, О. В. Сичова. – Харків: Видавництво Іванченка І. С., 2024. – 207 с.

22. Vue.js [Електронний ресурс] : Режим доступу: https://vuejs.org (дата звернення: 07.05.2025).

23. Односторікновий застосунок [Електронний ресурс] : Режим доступу: https://goo.su/6MViWM (дата звернення: 07.05.2025).

24. VUEX документація [Електронний ресурс] : Режим доступу: https://vuex.vuejs.org/ (дата звернення: 07.05.2025).

25. Вступ до Axios [Електронний ресурс] : Режим доступу: https://axios-http.com/uk/docs/intro (дата звернення: 07.05.2025).

26. PostgreSQL: introduction and concepts [Електронний ресурс] / Momjian , Bruce . – Upper Saddle River : Addison-Wesley, 2001. – 490 p. : іл. – ISBN 0-201-70331-9 .

27. Balsamiq [Електронний ресурс] : Режим доступу: https://balsamiq.com/ (дата звернення: 07.05.2025).

28. Autofac документація [Електронний ресурс] : Режим доступу: https://autofac.readthedocs.io/en/latest/getting-started/index.html (дата звернення: 12.05.2025).

29. Dependency Injection Principles, Practices, and Patterns: монографія / Mark Seemann, Steven van Deursen. – Shelter Island (NY), 2020. – 620 с.

30. Шаблон DTO [Електронний ресурс] : Режим доступу: https://en.wikipedia.org/wiki/Data_transfer_object (дата звернення: 12.05.2025).

31. AutoMapper документація [Електронний ресурс] : Режим доступу: https://automapper.org/ (дата звернення: 12.05.2025).

32. Серводвигун SG90 [Електронний ресурс] : Режим доступу: https://jawhersebai.com/tutorials/how-to-use-the-sg90-servo-motor. (дата звернення: 20.05.2025).

33. Тестування навнтаженням [Електронний ресурс] : Режим доступу: https://www.zaptest.com/uk/що-таке-тестування-навантаження-глиб. (дата звернення: 20.05.2025).

34. Вимоги до безпеки Інтернету речей (IoT) [Електронний ресурс] : Режим доступу: https://www.intertek.com/blog/2018/09-11-iot. (дата звернення: 20.05.2025).