МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ПЗВО «МІЖНАРОДНИЙ КЛАСИЧНИЙ УНІВЕРСИТЕТ
імені ПИЛИПА ОРЛИКА»
Економіко-технологічний факультет
Кафедра інженерних технологій

Кваліфікаційна робота
на здобуття освітнього ступеня магістра
за освітньою програмою «Комп'ютерна інженерія»
зі спеціальності 123 «Комп'ютерна інженерія»

на тему: **«Веб-застосунок для моніторингу параметрів роботи електромеханічного обладнання з використанням хмарних технологій»**

Виконав:
здобувач ІІ курсу, групи КІ -20-24
Басанець Анатолій Сергійович
Керівник:
к.ф-м.н., доцент кафедри інженерних технологій
АРАМЯН Армен Мартикович

Миколаїв – 2024

АНОТАЦІЯ

Веб-застосунок для моніторингу параметрів роботи електромеханічного обладнання з використанням хмарних технологій. – Рукопис.

Робота складається із вступу, 4 розділів, висновків, переліку використаних джерел, додатків і має загальний обсяг 124 сторінки. Список використаних джерел – 25.

Робота спрямована на вирішення актуальної проблеми на зменшення простоїв та витрат на обслуговування обладнання, поліпшення енергоефективності, реалізацію сучасних методів діагностики, цифровізацію і автоматизацію виробничих процесів.

Ключові слова: моніторинг, електромеханічне обладнання, хмарні технології, Інтернет речей (IoT), діагностика обладнання, гармонічний аналіз.

Об'єктом дослідження є процеси моніторингу та аналізу параметрів роботи асинхронних електродвигунів у промислових системах із використанням сучасних інформаційних технологій, зокрема хмарних обчислень та Інтернету речей (IoT).

Предметом досліджень є методи та засоби моніторингу, аналізу та діагностики роботи електромеханічного обладнання.

Мета та задачі досліджень. Метою даної роботи є розробка та впровадження системи моніторингу та аналізу роботи електромеханічного обладнанням з використанням хмарних технологій.

Методи дослідження. Теоретичні: Аналіз літератури та інформаційних джерел, визначення структури системи, її компонентів і взаємодії між ними, формування вимог до архітектури бази даних, серверної та клієнтської частин.

Експериментальні дослідження базувались на тестуванні системи на змодельованих даних, що імітують роботу двигуна. За допомогою цих даних була проведена перевірка на точність та надійність алгоритмів аналізу.

Наукова новизна роботи полягає в наступному:

- Інтеграція аналізу гармонічного складу з обчисленням миттєвих потужностей для пілвищення точності діагностики.
- Проектування бази даних для збереження даних про кілька двигунів, що дозволяє використовувати її на великих підприємствах.

Поставлена задача була вирішена наступним чином: 1. Аналіз літературних джерел та існуючих рішень. 2. Розробка структури бази даних. 3. Створення серверної частини веб-застосунку. 4. Розробка клієнтської частини веб-застосунку. 5. Експериментальна перевірка розробленої системи.

ABSTRACT

Web-based tool for monitoring the parameters of electromechanical robots using the latest technologies. – Manuscript.

The work consists of an introduction, 4 chapters, conclusions, a list of used sources, applications and has a total volume of 124 pages. List of sources used – 25.

The work is aimed at solving the current problem of reducing downtime and equipment maintenance costs, improving energy efficiency, implementing modern diagnostic methods, digitizing and automating production processes.

Keywords: monitoring, electromechanical equipment, cloud technologies, Internet of Things (IoT), equipment diagnostics, harmonic analysis.

The object of research is the processes of monitoring and analyzing the parameters of asynchronous electric motors in industrial systems using modern information technologies, in particular cloud computing and the Internet of Things (IoT).

The subject of research is the methods and means of monitoring, analyzing and diagnosing the operation of electromechanical equipment.

Purpose and tasks of research. The purpose of this work is the development and implementation of a system for monitoring and analyzing the operation of electromechanical equipment using cloud technologies.

Research methods. Theoretical: Analysis of literature and information sources, determination of the structure of the system, its components and interaction between them, formation of requirements for the architecture of the database, server and client parts.

Experimental studies were based on testing the system on simulated data simulating engine operation. With the help of these data, the accuracy and reliability of the analysis algorithms was checked.

The scientific novelty of the work is as follows:

- Integration of the analysis of the harmonic composition with the calculation of instantaneous powers to increase the diagnostic accuracy by half.

- Designing a database to store data about several engines, which allows it to be used in large enterprises.

The task was solved as follows: 1. Analysis of literary sources and existing solutions. 2. Development of the database structure. 3. Creation of the server part of the web application. 4. Development of the client part of the web application. 5. Experimental verification of the developed system.

ЗМІСТ

ВСТУП

Сучасна промисловість перебуває у стані активної цифрової трансформації, що сприяє підвищенню ефективності виробничих процесів та рівня безпеки експлуатації обладнання. Одним із важливих напрямків цієї трансформації є впровадження систем моніторингу стану електромеханічного обладнання на основі Інтернету речей (IoT) і хмарних технологій [1]. Такі рішення дозволяють у реальному часі отримувати, аналізувати та зберігати дані про роботу обладнання.

Зокрема, для асинхронних електродвигунів, які є ключовими елементами багатьох промислових систем, своєчасне виявлення дефектів може запобігти серйозним аваріям, простою виробництва та підвищеним витратам на ремонт [1]. Проте ефективність таких систем залежить від якості збирання, передачі, обробки і візуалізації даних, а також від їхньої безпеки та надійності.

Актуальність дослідження полягає в тому, що сучасні підходи до моніторингу стану електромеханічного обладнання не завжди враховують можливості інтеграції хмарних технологій, а також ризики, пов’язані із їх використанням. Це створює потребу у розробці веб-застосунків, які не лише забезпечують ефективну взаємодію з обладнанням і зручну візуалізацію даних, але й враховують безпекові аспекти.

Практична проблема, яку вирішує дана робота, полягає у необхідності забезпечення ефективного моніторингу стану асинхронних електродвигунів у промислових умовах [2].

Метою роботи є створення веб-застосунку для моніторингу параметрів роботи асинхронних електродвигунів із використанням хмарних технологій. Застосунок повинен забезпечити збирання даних про струми, потужновсті та напруги, обчислення миттєвих значень потужностей, гармонійного складу параметрів, а також візуалізацію результатів у вигляді графіків і таблиць [3].

Об'єктом дослідження є процеси моніторингу та аналізу параметрів роботи асинхронних електродвигунів у промислових системах із

використанням сучасних інформаційних технологій, зокрема хмарних обчислень та Інтернету речей (IoT).

Предметом дослідження є методи та засоби моніторингу та аналізу роботи електромеханічного обладнання.

Задачами дослідження є:

1. Аналіз літературних джерел та існуючих рішень.
2. Розробка структури бази даних.
3. Створення серверної частини веб-застосунку.
4. Розробка клієнтської частини веб-застосунку.
5. Інтеграція хмарних технологій..
6. Експериментальна перевірка розробленої системи.

Пояснювальна записка складається зі вступу, 4 розділів, висновків та додатків, що складає 124 сторіни тексту, список використаної літератури на 26 назв та 2-х додатків.

У вступі виконується обґрунтування актуальності теми магістерської роботи, визначення об’єкту та предмета дослідження, формування мети дослідження, обґрунтування наукової новизни та визначення практичного значення.

У першому розділі виконується загальна характеристика проблеми моніторингу електромеханічного обладнання, аналіз використання хмарних технологій у задачах моніторингу, огляд і класифікація існуючих систем та платформ для моніторингу, визначення вимог до веб-застосунку та постановка задач.

У другому розділі виконується розробка UseCases для інформаційно-аналітичної системи моніторингу, створення діаграми варіантів використання та діаграм взаємодій.

У третьому розділі виконанується вибір базових технологій, платформ і стандартів розробки, розробка загальної архітектури системи проектованої системи, структури бази даних, взаємодії клієнтської частини з сервером, інтеграції з хмарними сервісами.

У четвертому розділі виконано розробку вимог до інтерфейсу, принципів UX/UI дизайну, опис основних екранів веб-застосунку та тестування та результати роботи.

Наукова новизна роботи полягає у поєднанні методів обробки параметрів електромеханічного обладнання та інтеграції хмарних технологій, що дозволяє підвищити точність і швидкість діагностики дефектів, а також забезпечити безперервний доступ до даних.

- Інтеграція аналізу гармонічного складу з обчисленням миттєвих потужностей для пілвищення точності діагностики.

- Проєктування бази даних для збереження даних про кілька двигунів, що дозволяє використовувати її на великих підприємствах.

Практичне значення дослідження полягає у створенні прототипу веб-застосунку, який може бути інтегрований у промислові системи для підвищення ефективності моніторингу, аналізу та діагностики електромеханічного обладнання.

Таким чином, у роботі вирішується актуальна проблема забезпечення зручності використання інформаційних систем для моніторингу і діагностики промислових електромеханічних пристроїв, що відповідає сучасним вимогам у сфері цифровізації виробництва.

1 АНАЛІТИЧНА ЧАСТИНА

Постійний розвиток промисловості та зростаючі вимоги до надійності обладнання вимагають впровадження нових підходів. Таких, які поєднують можливості Інтернету речей (IoT) та хмарних технологій. Водночас вони стикаються з низкою проблем, серед яких безпека даних, інтеграція з наявними інфраструктурами та забезпечення високої точності діагностики [4].

1.1 Загальна характеристика проблеми моніторингу електромеханічного обладнання

Моніторинг стану електромеханічного обладнання є важливим елементом у забезпеченні ефективної експлуатації промислових систем. Зміни, що відбуваються в електричних та механічних параметрах роботи обладнання, можуть бути показниками несправностей, що можуть призвести до аварій. З огляду на складність і високі вимоги до надійності сучасних виробничих систем, важливою задачею є своєчасне виявлення дефектів і попередження поломок до їх критичного стану.

Електромеханічне обладнання, зокрема асинхронні електродвигуни, є одним із основних компонентів багатьох промислових систем. Втрата їх працездатності може спричинити значні економічні втрати через зупинку виробництва, що призводить до додаткових витрат на ремонт і відновлення роботи. Тому важливою задачею є впровадження систем моніторингу, що дозволяють у реальному часі відслідковувати параметри роботи двигунів для виявляти потенційні проблеми на ранніх етапах їхнього розвитку [5].

Одним із найефективніших підходів до моніторингу є використання технологій Інтернету речей (IoT), які дозволяють підключати обладнання до мережі і здійснювати його контроль через різноманітні пристрої. Це відкриває можливості для збору даних про параметри роботи в режимі реального часу, що є критично важливим для оперативного реагування на будь-які зміни у

функціонуванні обладнання. Проте технології IoT також мають свої обмеження, зокрема, пов'язані з безпекою та надійністю системи передачі даних, що вимагає впровадження додаткових заходів для забезпечення безпеки інформації та мінімізації ризиків [4].

Завдяки інтеграції хмарних технологій з системами моніторингу, можна значно підвищити масштабованість та ефективність обробки даних, що дозволяє здійснювати більш точну діагностику та передбачати можливі дефекти за допомогою аналітики. Хмарні технології забезпечують зручний доступ до даних з будь-якої точки світу, що дозволяє оперативно реагувати на проблеми та покращити процеси обслуговування. Однак безпека передачі та зберігання таких даних є важливою проблемою, яка потребує ретельного аналізу і відповідних технічних рішень [6].

Таким чином, проблема моніторингу електромеханічного обладнання є комплексною і потребує врахування не лише технічних аспектів, а й безпекових вимог, що стосуються збору, обробки та зберігання даних.

1.2 Аналіз використання хмарних технологій у задачах моніторингу

Використання хмарних технологій у задачах моніторингу електромеханічного обладнання значно розширює можливості збору, обробки та аналізу даних у реальному часі. Хмарні обчислення забезпечують масштабованість, доступність даних з будь-якого місця та інтеграцію з іншими інформаційними системами, що робить їх невід'ємною частиною сучасних IoT-рішень [7].

Хмарні платформи дозволяють централізовано зберігати великі обсяги даних, отриманих із сенсорів, розташованих на обладнанні. Наприклад, у промислових умовах це може включати параметри струму, напруги та потужності. Завдяки хмарним сервісам, такі дані легко обробляються і візуалізуються, що значно спрощує процес прийняття рішень для діагностики та прогнозування несправностей [6].

Одним із головних переваг хмарних технологій є можливість використання аналітичних інструментів для аналізу великих обсягів даних (Big Data). Використання алгоритмів машинного навчання у хмарному середовищі дозволяє виявляти складні залежності між параметрами роботи обладнання і на ранніх етапах прогнозувати виникнення несправностей [9]. Зокрема, такі рішення інтегровані у хмарні платформи, як AWS IoT [10], Microsoft Azure IoT Hub [11] та Google Cloud IoT Core [12].

Amazon Web Services (AWS) IoT пропонує численні переваги для розробників і підприємств, які хочуть створювати, впроваджувати та керувати рішеннями Інтернету речей (IoT). AWS IoT має такі преваги:

- масштабованість: вона підтримує великий обсяг даних і дозволяє підключати мільйони пристроїв, забезпечуючи стабільність і продуктивність навіть при зростанні навантаження;

- інтеграція з іншими сервісами AWS: вона може легко підключатись до інших сервісів AWS, таких як AWS Lambda, Amazon S3, Amazon DynamoDB або Amazon RDS та Amazon SageMaker, що створює потужну екосистему для аналізу даних, автоматизації та управління IoT-рішеннями;

- безпека: вона надає комплексні механізми безпеки, включаючи шифрування даних під час передачі та зберігання, управління сертифікатами та авторизацію на основі політик AWS IoT Core, що допомагає захистити пристрої та дані від загроз;

- реальний час: завдяки AWS IoT Core дані від пристроїв можна отримувати й обробляти у реальному часі, що дозволяє оперативно реагувати на зміни;

- легкість у розробці: вона підтримує SDK для різних мов програмування (C, Python, JavaScript та інші), що спрощує підключення пристроїв, а вбудовані бібліотеки полегшують впровадження функціоналу IoT;

- аналітика та моніторинг: сервіси, такі як AWS IoT Analytics і Amazon QuickSight, дозволяють здійснювати аналіз IoT-даних та створювати візуалізації;

- глобальна доступність: вона забезпечує надійну роботу через глобальну мережу дата-центрів, дозволяючи пристроям бути на зв’язку незалежно від місцезнаходження;

- гнучка оплата: вона використовує модель "оплати за використання", що дозволяє оптимізувати витрати в залежності від кількості підключених пристроїв та обсягу даних;

- підтримка різних протоколів: вона підтримує стандартні протоколи, такі як MQTT, HTTP і WebSockets, що робить його сумісним з широким спектром пристроїв;

- можливості управління пристроями: AWS IoT Device Management надає інструменти для, такі як реєстрації, моніторингу та оновлення пристроїв, а також віддаленого управління та усунення несправностей;

- легкість вбудованого машинного навчання: AWS IoT Greengrass дозволяє виконувати моделі машинного навчання локально на пристроях, забезпечуючи автономну роботу в разі втрати з’єднання з хмарою;

- екосистема IoT-партнерів: AWS співпрацює з численними компаніями, що забезпечує підтримку різноманітних пристроїв, датчиків та платформ.

AWS IoT має численні переваги, але існують також деякі недоліки, які варто враховувати при виборі платформи:

- висока складність: вона вимагає часу на вивчення через складність платформи, її інтерфейсів та широкого набору функцій, інтеграція з іншими сервісами AWS потребує хорошого розуміння екосистеми AWS;

- залежність від інтернету: більшість функцій AWS IoT залежить від стабільного підключення до інтернету, особливо для реального часу, хоча AWS IoT Greengrass і пропонує можливості для офлайн-роботи, але це може вимагати додаткових зусиль з налаштування;

- вартість: ціна може швидко зростати через масштабування пристроїв або обсягу даних, що передаються, зберігаються чи обробляються, через що погане управління ресурсами може призвести до непередбачуваних витрат;

- залежність від хмари AWS: використання цієї технології прив’язує вас до екосистеми AWS, через що в разі бажання змінити постачальника хмарних послуг це може бути складним і дорогим;

- обмежена підтримка спеціалізованих протоколів: підтримує основні протоколи (MQTT, HTTP, WebSockets), але якщо пристрої використовують менш поширені протоколи, інтеграція може вимагати додаткових зусиль;

- обмежені можливості кастомізації: вона орієнтована на універсальні рішення, і тому можливості глибокої кастомізації можуть бути обмеженими, а у деяких випадках доведеться створювати окремі мікросервіси для специфічних потреб;

- проблеми сумісності з деякими пристроями: пристрої зі старим апаратним забезпеченням або специфічним програмним забезпеченням можуть не повністю підтримувати AWS IoT SDK або протоколи;

- висока залежність від API: вона значною мірою залежить від API, у разі змін в API від AWS рішення можуть вимагати оновлення або внесення коректив;

- обмежений контроль над інфраструктурою: як і з усіма хмарними сервісами, користувачі не мають прямого доступу до фізичних серверів і обладнання, це може бути проблемою для компаній з суворими вимогами до безпеки або локалізації даних;

- залежність від SLA: продуктивність IoT-рішень залежить від SLA (Service Level Agreement) AWS, а будь-який збій у сервісах AWS може призвести до збоїв у роботі системи;

- можливі проблеми з конфіденційністю: дані зберігаються в хмарі AWS, що може викликати занепокоєння у компаній з чутливими даними або суворими регуляторними вимогами до зберігання та обробки даних;

- нестабільність через регіональні відмінності: деякі функції або сервіси можуть бути недоступними в певних регіонах, що ускладнює глобальне розгортання рішень.

Microsoft Azure IoT Hub пропонує широкий спектр переваг для створення, впровадження та управління IoT-рішеннями. Microsoft Azure IoT Hub має такі переваги:

- масштабованість і продуктивність: вона може обробляти мільйони одночасно підключених пристроїв, забезпечуючи стабільність навіть при великому навантаженні, також вона підтримує високий обсяг обробки даних у реальному часі;

- двосторонній обмін даними: вона забезпечує двосторонній зв’язок між хмарою та пристроями;

- широкий вибір протоколів: підтримка стандартних протоколів: MQTT, AMQP, HTTPS та WebSockets, що дозволяє легко інтегрувати різні пристрої;

- інтеграція з Azure: вона має тісну інтеграцію з іншими сервісами Azure, такими як Azure Stream Analytics для аналізу даних у реальному часі, Azure Functions для автоматизації обробки подій, Azure Machine Learning для впровадження моделей ШІ, Azure Data Lake або Azure Cosmos DB для зберігання даних, що спрощує створення комплексних IoT-рішень;

- безпека: потужні механізми для захисту даних і пристроїв, такі як аутентифікація пристроїв за допомогою сертифікатів або ключів, шифрування даних під час передачі, контроль доступу на основі ролей (RBAC);

- управління пристроями: цифрове відображення фізичних пристроїв, яке дозволяє відстежувати стан пристроїв, оновлювати їхні конфігурації та здійснювати моніторинг та масове управління пристроями для оновлення програмного забезпечення або конфігурацій;

- можливості локальної обробки: Azure IoT Edge дозволяє виконувати обробку даних та моделі машинного навчання локально на пристроях без постійного підключення до хмари;

- глобальна доступність: Azure має розгалужену мережу дата-центрів по всьому світу, що забезпечує низьку затримку для пристроїв, незалежно від їхнього розташування;

- інструменти розробника: SDK для різних мов програмування (C#, Python, Java, JavaScript тощо), що спрощує інтеграцію пристроїв, а також інструменти для симуляції пристроїв під час розробки;

- гнучке ціноутворення: модель «оплата за використання», що дозволяє масштабувати витрати залежно від кількості пристроїв і обсягу переданих даних;

- аналітика та візуалізація: інтеграція з Power BI дозволяє створювати візуалізації IoT-даних;

- екосистема IoT: Microsoft активно співпрацює з партнерами, пропонуючи сертифіковані IoT-пристрої та сумісні платформи;

- швидкий старт: вона має інтуїтивно зрозумілий інтерфейс та готові шаблони для швидкого створення рішень.

Microsoft Azure IoT Hub, хоч і має багато переваг, також має певні недоліки, які варто враховувати перед вибором цієї платформи для IoT-рішень:

- висока складність налаштування: початкова конфігурація та інтеграція можуть бути складними, особливо для новачків, необхідне глибоке розуміння екосистеми Azure для створення ефективного рішення;

- залежність від інтернету: більшість функцій системи потребує стабільного з'єднання з інтернетом, через що у разі перебоїв із мережею можуть виникати затримки або збої в роботі пристроїв;

- вартість: хоч ціноутворення і є гнучким, при масштабуванні IoT-рішення витрати можуть значно зростати, через що необхідність використання додаткових сервісів Azure може суттєво збільшити загальні витрати;

- залежність від хмарної платформи Azure: як і з іншими хмарними платформами, використання IoT Hub прив’язує вас до екосистеми Azure, через що перехід на іншу платформу може бути складним і дорогим;

- проблеми з протоколами: хоч вона і підтримує основні протоколи (MQTT, AMQP, HTTPS), інтеграція з пристроями, які використовують менш поширені або кастомні протоколи, може потребувати додаткової розробки;

- затримки у разі великого навантаження: при обробці великих обсягів даних або підключенні великої кількості пристроїв можуть виникати затримки, особливо якщо налаштування не оптимізовані;

- проблеми з локалізацією даних: у деяких регіонах певні функції або сервіси Azure можуть бути недоступними, що ускладнює глобальне розгортання;

- залежність від SLA: продуктивність залежить від Service Level Agreement (SLA), через що у разі збоїв на стороні Microsoft Azure, робота IoT-рішення може постраждати;

- обмеження кастомізація: вона орієнтована на стандартні сценарії, тому для нестандартних рішень можуть знадобитися додаткові налаштування або використання інших сервісів;

- навчання та сертифікація: команда, що працює з цією технологіїю, має мати відповідну підготовку, а навчання та сертифікація можуть бути дорогими і потребувати часу;

- залежність від API: зміни в API або оновлення на стороні Azure можуть вимагати оновлення IoT-рішень, що може спричиняти додаткові витрати;

- обмежений офлайн-функціонал: Azure IoT Edge дозволяє працювати офлайн, але його використання потребує окремого налаштування і може бути складним для новачків;

- безпека даних: незважаючи на потужні механізми безпеки, зберігання даних у хмарі може викликати занепокоєння у компаній, що працюють із чутливими або конфіденційними даними.

Google Cloud IoT Core — це платформа для побудови IoT-рішень, яка забезпечує безпечне підключення, управління пристроями та обробку даних у хмарі. Google Cloud IoT Core переваги:

- інтеграція з Google Cloud Platform (GCP): має тісну інтеграцію з іншими сервісами GCP, такими як BigQuery для аналізу даних, Cloud Functions для обробки подій, Dataflow для потокової обробки даних, AI/ML сервіси для прогнозування та аналітики;

- безпека: підтримка автентифікації на основі OAuth 2.0 та сертифікатів TLS, шифрування даних під час передачі та зберігання, управління доступом через IAM (Identity and Access Management);

- широка підтримка протоколів: підтримка популярних протоколів: MQTT та HTTP, що дозволяє підключати більшість сучасних IoT-пристроїв;

- масштабованість: платформа може обробляти мільйони одночасно підключених пристроїв, а також підходить для проєктів будь-якого масштабу — від малого бізнесу до глобальних корпорацій;

- легке управління пристроями: Device Registry дозволяє централізовано реєструвати та управляти IoT-пристроями;

- потокова обробка даних: дані з пристроїв передаються у Cloud Pub/Sub, що дозволяє виконувати реальну обробку даних, інтегрувати дані з іншими сервісами для аналітики та обробки;

- інтелектуальна аналітика: інтеграція з ML/AI-сервісами Google;

- глобальна інфраструктура: використання розгалуженої інфраструктури Google для забезпечення низької затримки та високої доступності сервісу в будь-якій точці світу;

- економія ресурсів: легкий у використанні інтерфейс та готові інструменти допомагають економити час і зусилля на розробку та управління, а також автоматизація багатьох процесів через скрипти або Cloud Functions;

- ціноутворення «оплата за використання»: оплата лише за використані ресурси, що особливо вигідно для проєктів із динамічним навантаженням;

- гнучкість у виборі операційної системи та мов: можливість використання різних ОС на пристроях (Linux, Windows тощо), а також SDK доступні для різних мов програмування, таких як Python, Java, C тощо;

- моніторинг і діагностика: інтеграція з Cloud Monitoring та Cloud Logging для відстеження стану пристроїв, аналізу помилок і швидкого реагування на збої;

- сучасні можливості роботи з цифровими моделями: підтримка цифрових двійників для моделювання та управління фізичними пристроями через хмару;

- економічна відповідальність: використання Google Cloud допомагає зменшити екологічний вплив завдяки інфраструктурі, що працює на відновлюваних джерелах енергії;

- швидкий старт: простий у налаштуванні сервіс, що дозволяє швидко розгорнути IoT-рішення.

Google Cloud IoT Core має численні переваги, але існують також деякі недоліки, які варто враховувати при виборі платформи:

- закриття Google Cloud IoT Core: Google офіційно припинив підтримку IoT Core у серпні 2023 року, що робить платформу непридатною для нових проектів та викликає труднощі для існуючих користувачів, які змушені переходити на альтернативні сервіси;

- залежність від екосистеми Google Cloud: як і в інших хмарних платформах, використання Google Cloud IoT Core сильно прив'язує вас до екосистеми GCP, що може ускладнити інтеграцію з іншими хмарними платформами або локальними рішеннями;

- відсутність локального офлайн-режиму: сервіс орієнтований на хмарні обчислення, а для офлайн-роботи потрібно додатково налаштовувати інші рішення, наприклад, Google Cloud IoT Edge, що ускладнює розгортання;

- високий рівень складності для новачків: навіть зручний інтерфейс потребує базових знань про Google Cloud Platform, що може бути складним для тих, хто раніше не працював із хмарними технологіями;

- обмеження підтримка протоколів: підтримується лише MQTT та HTTP, що може бути недостатньо для пристроїв, які працюють з іншими протоколами, наприклад, CoAP, AMQP або власними протоколами;

- вартість: попри те, що ціноутворення базується на «оплаті за використання», для великих IoT-рішень витрати можуть суттєво зрости,

наприклад додаткові витрати можуть виникати через інтеграцію з іншими сервісами GCP, такими як BigQuery, Pub/Sub тощо;

- вимоги до технічного рівня команди: розробка складних IoT-рішень вимагає досвіду роботи з хмарними сервісами Google та сучасними інструментами аналітики, що може потребувати навчання та часу;

- затримка у переданні даних: при обробці великих обсягів даних можуть виникати затримки, особливо якщо дані потребують складної обробки в реальному часі;

- відсутність повної кастомізації: IoT Core пропонує готові рішення для типових IoT-сценаріїв, але для унікальних або специфічних проектів можуть знадобитися додаткові налаштування або створення власних рішень;

- обмеження SLA та гарантій: як і у всіх хмарних сервісах, продуктивність залежить від SLA (рівня гарантії обслуговування), що може стати проблемою у випадку збою на стороні Google;

- залежність від API: зміни або оновлення API можуть вимагати доопрацювання вашого IoT-рішення, що спричиняє додаткові витрати;

- питання конфіденційності: зберігання даних у хмарі може викликати занепокоєння у компаній, які працюють із чутливими даними, особливо якщо необхідно дотримуватися суворих нормативів, таких як GDPR.

Ще однією важливою функцією хмарних платформ є безпека даних. Вбудовані механізми шифрування, контроль доступу та інструменти моніторингу знижують ризики несанкціонованого доступу до даних і втрати інформації [13]. Однак, залежність від стороннього постачальника хмарних послуг може створювати додаткові проблеми, зокрема у випадку відмови в роботі або кіберзагроз.

Незважаючи на численні переваги, впровадження хмарних технологій також пов’язане з певними проблемами. Це, зокрема, високі вимоги до пропускної здатності мережі для передачі даних у реальному часі та забезпечення низьких затримок, що є критичним для багатьох задач

моніторингу. Крім того, постійна передача даних у хмару збільшує ризики витоку конфіденційної інформації.

Загалом, хмарні технології є ефективним інструментом для підвищення продуктивності, безпеки і надійності систем моніторингу, однак їх впровадження потребує врахування особливостей конкретного виробничого середовища та відповідного вдосконалення безпеки даних.

1.3 Огляд і класифікація існуючих систем та платформ для моніторингу

Системи для моніторингу електромеханічного обладнання є ключовими для підвищення безпеки та ефективності промислових процесів. Вони дозволяють здійснювати моніторинг параметрів роботи обладнання, таких як струм, потужність і напруга, у реальному часі, з можливістю дистанційного доступу та прогнозування несправностей. З розвитком технологій, зокрема Інтернету речей (IoT), хмарних технологій та великого обсягу даних (Big Data), такі системи значно вдосконалюються, пропонуючи більш точні інструменти для збору та обробки інформації. Нижче наведено класифікацію існуючих систем моніторингу:

- SCADA – програмний пакет, призначений для розробки або забезпечення роботи в реальному часі систем збору, обробки, відображення та архівування інформації про об'єкт моніторингу або управління [14]. Він є одними з найбільш розповсюджених у промисловому моніторингу. Він забезпечує централізоване управління та моніторинг, а також взаємодію з сенсорами для збору даних. Проте, він часто потребують локальних серверів і складної інфраструктури, що може бути дорого. Прикладами системи SCADA є SIMATIC WinCC V7 [15], Schneider Electric EcoStruxure [16].

- Хмарні платформи дозволяють зберігати та обробляти великі обсяги даних, що генеруються з IoT-пристроїв, без необхідності мати власне серверне обладнання. Вони забезпечують доступність даних з будь-якої точки світу, а також дозволяють інтегрувати системи машинного навчання для аналізу

даних. Прикладами таких платформ є Amazon Web Services (AWS) IoT [10], Microsoft Azure IoT Hub [11], Google Cloud IoT Core [12].

- Платформи для моніторингу в реальному часі надають можливість відслідковувати дані про обладнання в режимі реального часу, реагуючи на зміни в параметрах і своєчасно інформуючи про потенційні несправності. Вони зазвичай використовують хмарні технології для зберігання та обробки даних, а також можуть забезпечити віддалений доступ через мобільні додатки або веб-застосунки. Прикладами таких платформ є Uptake [17], Senseye Predictive Maintenance [18].

- Інтегровані IoT платформи для промислового моніторингу об'єднують апаратне та програмне забезпечення для комплексного моніторингу стану різноманітних промислових систем. Вони зазвичай включають в себе різноманітні датчики, мережі зв'язку, а також програмне забезпечення для аналізу даних і візуалізації результатів. Прикладом таких платформ є GE Predix [19].

Сучасні платформи для моніторингу електромеханічного обладнання значно вдосконалюються завдяки інтеграції хмарних технологій та IoT. Вони не тільки забезпечують більш ефективне і безпечне управління процесами, але й сприяють зниженню витрат на обслуговування і ремонти через своєчасне виявлення дефектів.

1.4 Визначення вимог до веб-застосунку

Розробка веб-застосунку для моніторингу електромеханічного обладнання передбачає врахування такі вимог, щоб забезпечити надійність, ефективність та зручність використання:

- Збір даних із пристроїв IoT: Веб-застосунок повинен отримувати дані в реальному часі від IoT-сенсорів, встановлених на електромеханічному обладнанні. Дані включають показники струму, напруги та потужності, а також обчислені параметри (миттєва потужність, гармоніки).

- Обробка та аналіз даних: обчислення миттєвої потужності на основі отриманих даних; аналіз гармонічного складу струму та напруги для виявлення потенційних дефектів.

- Зберігання даних: зберігання отриманих та оброблених даних у базі даних для подальшого аналізу та звітності.

- Візуалізація даних: графіки та таблиці для представлення динаміки показників обладнання.

- Користувацький доступ: аутентифікація користувачів із різними рівнями доступу.

- Продуктивність: здатність обробляти дані від великої кількості IoT-пристроїв; реакція на події не повинна перевищувати 2 секунд.

- Зручність використання: інтуїтивно зрозумілий інтерфейс із детальними інструкціями;

Вимоги до веб-застосунку орієнтовані на забезпечення точності моніторингу, зручності у використанні та високої надійності системи. Дотримання цих вимог дозволить створити ефективний інструмент для моніторингу електромеханічного обладнання, спрямований на своєчасне виявлення дефектів і оптимізацію технічного обслуговування.

1.5 Постановка задач

Тема магістерського проекту – «Веб-застосунок для моніторингу параметрів роботи електромеханічного обладнання з використанням хмарних технологій».

Об’єкт розробки – процеси моніторингу та діагностики параметрів роботи асинхронних електродвигунів у промислових системах із використанням сучасних інформаційних технологій, зокрема хмарних обчислень та Інтернету речей (IoT).

Мета розробки – розробка та впровадження системи моніторингу, аналізу та діагностики роботи електромеханічного обладнанням з використанням хмарних технологій.

Вимоги до бази даних:

- структурованість даних;
- підтримка великих обсягів даних;
- інтеграція з хмарними платформами;
- продуктивність;
- надійність;
- безпека даних.

Вимоги до бекенд-частини:

- обробка даних;
- управління користувачами;
- підтримка хмарних сервісів;
- продуктивність;
- надійність.

Вимоги до фронт-енд частини:

- інтерфейс користувача;
- візуалізація даних;
- інтуїтивність;
- швидкодія.

У межах цього проекту було глибоко досліджено проблематику, вивчено існуючі рішення та сформулювати основні вимоги до системи:

- Виявлено основні труднощі у сфері моніторингу, такі як необхідність отримання оперативних даних, забезпечення надійної діагностики несправностей та інтеграція різнорідних пристроїв. Вирішення цих проблем потребує сучасних технологічних підходів та інтегрованих рішень.

- Хмарні технології продемонстрували значний потенціал у задачах моніторингу завдяки їхній масштабованості, доступності даних у реальному часі, зручності зберігання великих обсягів інформації та можливості

використання аналітичних інструментів. Їх застосування дозволяє підвищити ефективність і точність моніторингу.

- Проведено аналіз доступних рішень, які класифіковані за функціональністю, підтримкою обладнання, гнучкістю інтеграції та вартістю.

- Сформульовано вимоги до веб-застосунку, що включають надійність передачі та зберігання даних, інтуїтивний інтерфейс, підтримку аналітичних функцій, можливість інтеграції з хмарними сервісами та адаптивність для різних пристроїв.

- Визначено основні задачі проекту.

Цей розділ забезпечив глибоке розуміння проблеми та визначила необхідні вимоги для проектування веб-застосунку. Проведений аналіз існуючих рішень та можливостей хмарних технологій підтвердив доцільність створення нового інноваційного рішення, яке відповідатиме специфічним потребам моніторингу електромеханічного обладнання.

2 ФУНКЦІОНАЛЬНА ЧАСТИНА

2.1 Розробка UseCases для інформаційно-аналітичної системи моніторингу

Для розроблюваного проекту було спроектовано чотири UseCases:

- моніторинг параметрів роботи;
- перегляд розрахованих даних;
- перегляд репортів;
- створення репортів.

UseCases «Моніторинг параметрів роботи» занесено до таблиці 2.1. В цій таблиці наведено список дій, які потрібно виконати користувачу щоб отримати візуалізовані дані про моніторинг.

Таблиця 2.1 – UseCases «Моніторинг параметрів роботи»:

Виконавець	Користувач
Ціль	Переглянути параметри роботи
Успішний сценарій	1. Користувач вибирає меню моніторингу. 2. Система відображає графіки та таблиці з поточними значеннями струмів, напруги та потужності.
Результат	Користувач дані моніторингу.
Альтернативні сценарії	
2а	Виникла помилка підключення до обладнання. Результат: система відображає попередження.

UseCases «Перегляд розрахованих даних» занесено до таблиці 2.2. В цій таблиці наведено список дій, які потрібно виконати користувачу щоб отримати розраховані дані моніторингу.

Таблиця 2.2 – UseCases «Перегляд розрахованих даних»:

Виконавець	Користувач
Ціль	Переглянути розраховані дані
Успішний сценарій	1. Користувач вибирає меню перегляду розрахованих даних. 2. Система розраховує миттєві значення потужностей та гармонійний склад струмів, напруг і потужностей даних, що моніторяться. 3. Система відображає графіки та таблиці з розрахованими даними.
Результат	Користувач розраховані дані.
Альтернативні сценарії	
2а	Виникла помилка підключення до обладнання. Результат: система відображає попередження.

UseCases «Перегляд репортів» занесено до таблиці 2.3. В цій таблиці наведено список дій, які потрібно виконати користувачу щоб отримати репорти відносно кожного обладнання.

Таблиця 2.3 – UseCases «Перегляд репортів»:

Виконавець	Користувач
Ціль	Переглянути репорти
Успішний сценарій	1. Користувач вибирає розділ перегляду репортів. 2. Система відображає таблиці з репортами та їх статусами.
Результат	Користувач бачить список репортів.
Альтернативні сценарії	
2а	Виникла помилка підключення до бази даних. Результат: система відображає попередження.

UseCases «Створення репортів» занесено до таблиці 2.4. В цій таблиці наведено список дій, які потрібно виконати користувачу щоб створити репорт для двигуна в якому він виявив дефект.

Таблиця 2.4 – UseCases «Створення репортів»:

Виконавець	Користувач
Ціль	Створити репорт

Продовження таблиці 2.4

Успішний сценарій	1. Користувач вибирає розділ перегляду репортів. 2. Користувач натискає кнопку «Створити репорт». 3. Система відображає форму створення репорту. 4. Користувач заповняє форму створення репорту та підтверджує його створення. 5. Система створює репорт.
Результат	Система створює репорт.
Альтернативні сценарії	
5а	Виникла помилка підключення до бази даних. Результат: система відображає попередження.

2.2 Діаграми варіантів використання

Згідно попереднього пункту даного розділу, було розроблено діаграму варіантів використання, вона наведена на рис. 2.1. На ній зображено взаємодії між користувачем та системою.

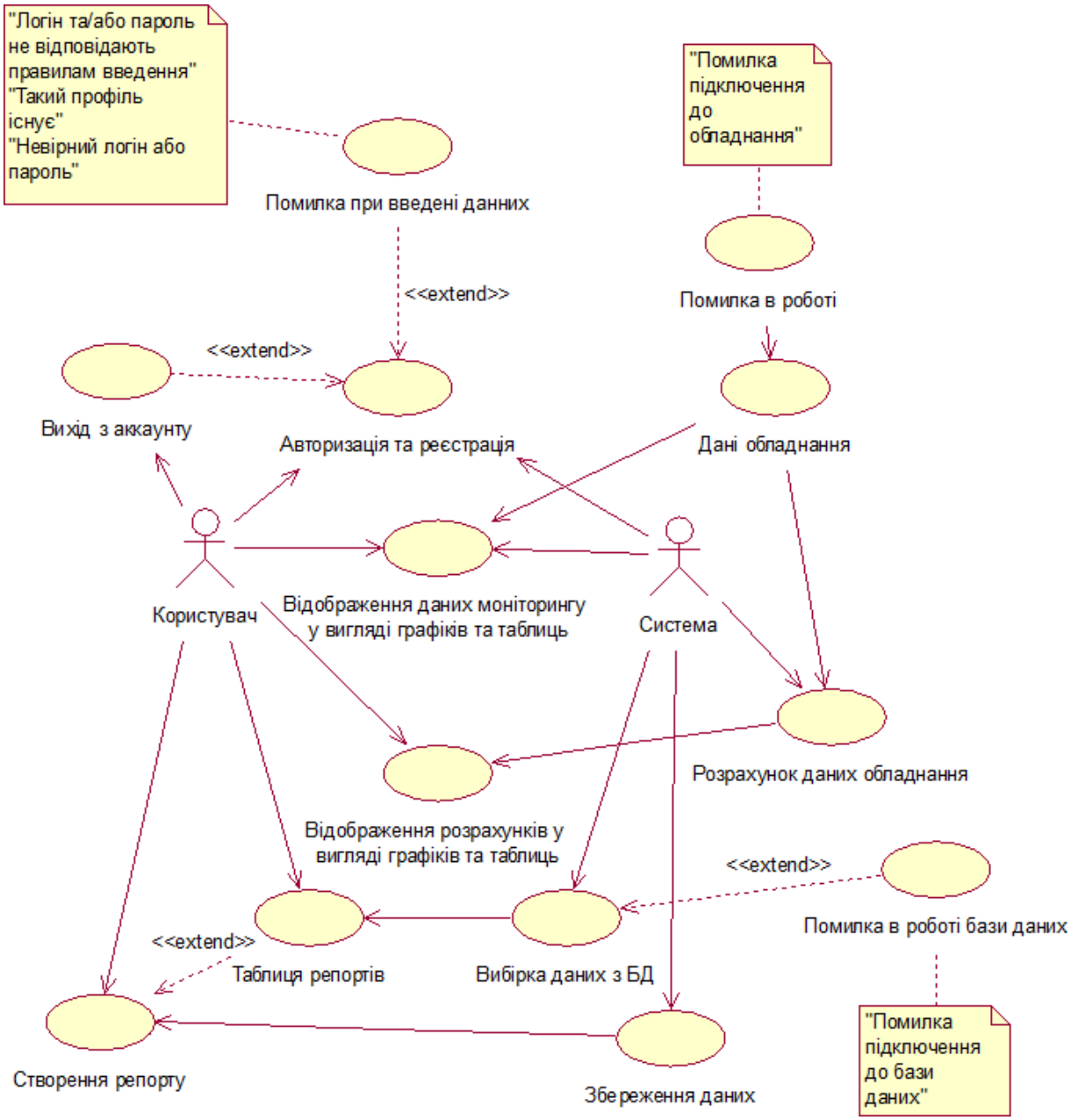


Рисунок 2.1 – Діаграма варіантів використання

2.3 Діаграми взаємодій

Діаграма взаємодії «Моніторинг параметрів роботи» створена згідно однойменного UseCases. Діаграма взаємодії зображена на рисунку 2.3.

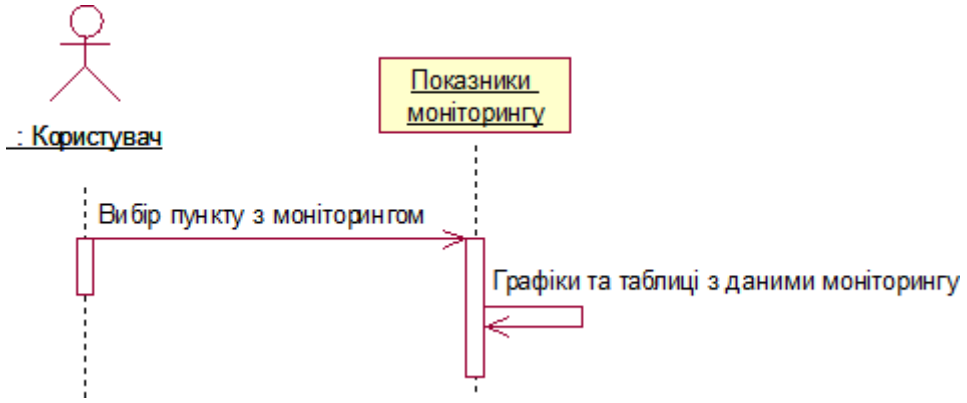


Рисунок 2.2 – діаграма взаємодії «Моніторинг параметрів роботи»

Діаграма взаємодії «Перегляд розрахованих даних» створена згідно однойменного UseCases. Діаграма взаємодії зображена на рисунку 2.2.

Рисунок 2.3 – діаграма взаємодії «Перегляд розрахованих даних»

Діаграма взаємодії «Перегляд репортів» створена згідно однойменного UseCases. Діаграма взаємодії зображена на рисунку 2.4.

Рисунок 2.4 – діаграма взаємодії «Перегляд репортів»

Діаграма взаємодії «Створення репортів» створена згідно однойменного UseCases. Діаграма взаємодії зображена на рисунку 2.5.

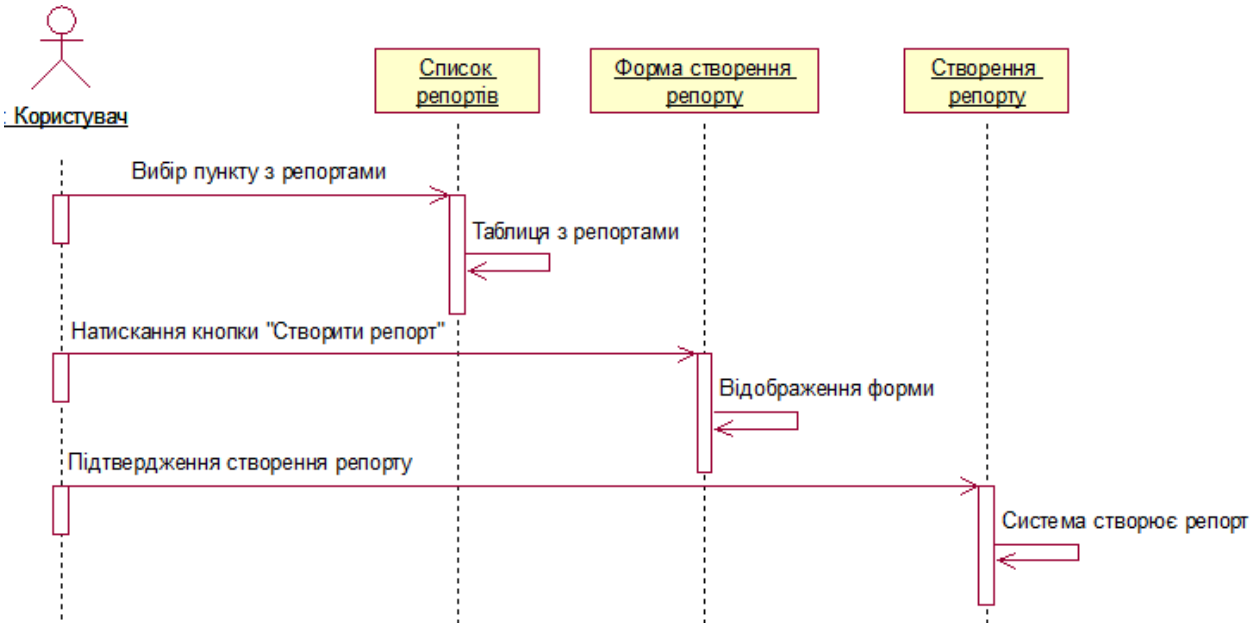


Рисунок 2.5 – діаграма взаємодії «Створення репортів»

У межах цього розділу було розроблено та деталізовано функціональні можливості системи:

- Визначено основні сценарії використання системи, які охоплюють ключові функції, такі як моніторинг параметрів роботи, перегляд розрахованих даних, перегляд репортів та створення репортів;

- Створено діаграми варіантів використання, які візуально відображають взаємодію між користувачами та системою і функціями застосунку;

- Розроблено діаграми взаємодій, що відображають послідовність операцій між компонентами системи для реалізації кожного сценарію використання.

Функціональна частина проекту визначила ключові аспекти взаємодії користувачів із системою та забезпечила детальний опис сценаріїв її використання. Розроблені діаграми сприяли кращому розумінню функціоналу і забезпечили основу для подальшої реалізації та тестування системи. Завдяки структурованому підходу вдалося врахувати всі необхідні варіанти використання, що дозволяє веб-застосунку ефективно виконувати свої задачі.

3 ПРОЄКТНА ЧАСТИНА

3.1 Вибір базових технологій, платформ і стандартів розробки

Для розробки веб-застосунку для моніторингу параметрів роботи електромеханічного обладнання з використанням хмарних технологій потрібно врахувати вимоги до продуктивності, надійності, гнучкості та масштабованості. Нижче обґрунтовується вибір базових технологій, платформ і стандартів розробки для цього проекту.

3.1.1 Базові технології

Вибір мови програмування залежить від потреб розробки та наявних навичок. Для клієнтської частини обрано React.js через його компонентну архітектуру, що спрощує побудову інтерактивних інтерфейсів і забезпечує високу продуктивність.

3.1.2 Платформи розробки

Для серверної частини обрана Java через її високу продуктивність, стабільність, потужні інструменти для інтеграції з різними системами (наприклад, через MQTT), а також для обробки великих обсягів даних в реальному часі. Spring Boot забезпечує швидку розробку REST API, необхідних для збору та надання даних моніторингу.

Для візуальної інтерпретації моніторингу важливо мати потужні візуалізаційні бібліотеки. D3.js є однією з найпопулярніших бібліотек для створення інтерактивних візуалізацій на веб-сторінках. Також D3.js дозволяє створювати складні графіки та діаграми, використовуючи дані, отримані в реальному часі. Його гнучкість дає можливість створювати як базові графіки,

так і складні візуалізації, що підходять для моніторингу технічних параметрів обладнання.

3.1.3 Стандарти

Стандарти обміну даними, такі як JSON, є важливими для забезпечення інтеграції та взаємодії системи моніторингу з іншими частинами системами. JSON легко інтегрується з більшістю мов програмування. Його легко читати та обробляти завдяки своєму компактному формату та легкості обробки. JSON зручний для передачі даних у реальному часі через MQTT.

MQTT є протоколом обміну повідомленнями, який використовується для передачі даних у реальному часі, особливо для IoT пристроїв. JSON, як легкий формат передачі даних, ідеально підходить для використання з MQTT через свою простоту та ефективність.

3.2 Загальна архітектура проектованої системи

Загальна архітектура проектованої системи моніторингу електромеханічного обладнання базується на принципах розподіленої обробки даних, використанні хмарних технологій та інтернету речей (IoT). Основними компонентами системи є:

- IoT-пристрої, що встановлені на електромеханічному обладнанні, збирають дані про роботу обладнання. Ці пристрої постійно моніторять стан обладнання та передають дані через мережу.

- Для збирання та передачі даних між IoT-пристроями та сервером використовується протокол MQTT. Цей легкий протокол ідеально підходить для IoT-додатків завдяки своїй низькій затримці і можливості роботи з великими обсягами даних у реальному часі.

- Сервер обробляє отримані від пристроїв дані. Серверна частина реалізована на Java з використанням фреймворка Spring Boot для розробки REST API.

- Веб-застосунок для візуалізації даних реалізується на основі React.js та D3.js, що дозволяє побудувати динамічний і інтерактивний інтерфейс. Веб-додаток дозволяє користувачам отримувати інформацію про параметри роботи обладнання в реальному часі, переглядати графіки та таблиці, а також проводити діагностику на основі отриманих даних.

Взаємодія компонентів:

- IoT-пристрої передають дані через MQTT на сервер;

- Сервер отримує дані та розраховує миттєві значення потужностей, а також гармонійний склад струмів, напруг і потужностей;

- Дані обробляються та аналізуються для подальшої візуалізації в React.js веб-застосунку;

- Хмарна платформа забезпечує зберігання даних та доступність системи на всіх етапах її роботи;

- Інтерфейс дозволяє користувачам моніторити стан обладнання в режимі реального часу та отримувати сповіщення про можливі проблеми.

Ця архітектура дозволяє створити масштабовану, надійну систему, яка здатна ефективно моніторити електромеханічне обладнання в умовах промислових підприємств.

3.3 Структура бази даних

3.3.1 Вибір СУБД

Система управління базами даних (СУБД) – набір взаємопов'язаних даних (база даних) і програм для доступу до цих даних. Надає можливості створення, збереження, оновлення та пошуку інформації в базах даних з контролем доступу до даних [20].

Вибір системи управління базами даних (СУБД) для проекту залежить від вимог до функціональності, масштабованості, продуктивності та інтеграції з іншими компонентами системи. Для даного проекту було обрано PostgreSQL, оскільки вона відповідає ключовим критеріям.

Критерії вибору СУБД:

- вона є потужною реляційною СУБД, яка підтримує складні запити SQL, функції, тригери, а також розширення, такі як JSON та XML, що дозволяє зберігати структуровані та напівструктуровані дані, що підходить для моніторингу IoT-пристроїв та запису їхніх параметрів.

- вона підтримує розширення, як-от TimescaleDB, яке оптимізоване для роботи з часовими рядами, що корисно для зберігання і аналізу даних сенсорів у контексті часу;

- вона добре інтегрується з хмарними платформами, такими як AWS, Azure та Google Cloud, які можуть використовуватись для реалізації хмарних компонентів системи;

- вона має відкриту ліцензію, що робить її економічно вигідним рішенням.

3.3.2 Опис логічної структури БД

Логічна структура бази даних є основою для організації даних у системі. Вона визначає, як різні елементи даних взаємопов’язані між собою, та включає сутності, їхні атрибути і зв’язки між ними. Нижче наведено опис основних сутностей і взаємозв’язків для системи моніторингу електромеханічного обладнання.

Логічна структура бази даних визначається за допомогою концептуальних моделей, таких як модель сутності-відношення (Entity-Relationship Model) або UML-діаграми. Ці моделі дозволяють візуалізувати структуру даних та відносини між ними, що допомагає розробникам створювати ефективні бази даних.

Логічну структуру бази даних наведено в рисунку 3.1.

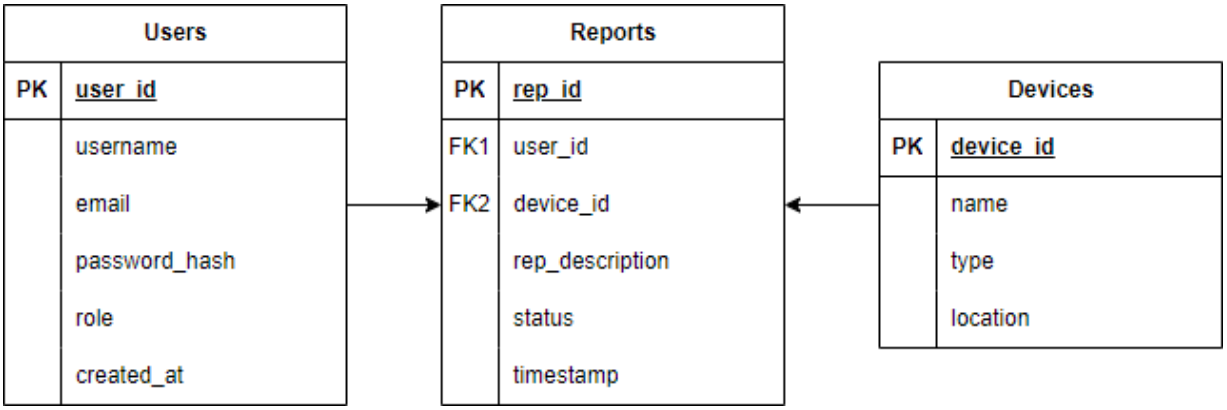


Рисунок 3.1 – Логічна структура бази даних

При проектуванні бази даних було розроблено три основні таблиці, що мають первиний ключ (primary key).

3.3.3 Опис фізичної структури БД

Фізична структура бази даних описує спосіб зберігання та організації даних на рівні файлів і блоків у конкретній технології бази даних. Вона включає у себе розміщення таблиць, індексів, файлів журналів та інших об'єктів бази даних на фізичному носії.

Фізична структура бази даних визначається конкретною технологією бази даних. Кожна технологія має свої власні механізми та правила організації даних на фізичному рівні.

Фізичну структуру бази даних наведено в рисунку 3.2.

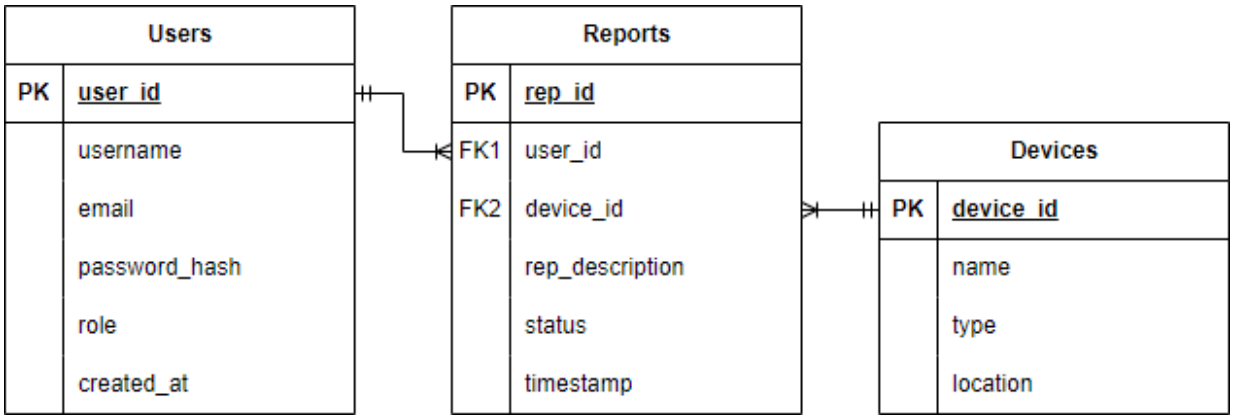


Рисунок 3.2 – Фізична структура бази даних

Розглянемо більш детально таблиці, які зображені в таблицях 3.1-3.3.

Таблиця «devices» (таблиця обладнання) містить в собі наступні дані: назву пристрою, тип пристрою, місцезнаходження пристрою. За допомогою цієї таблиці адміністратор зможе переглядати пристрої, додавати пристрої, оновлювати інформацію про пристрої та видаляти пристрої. Структура зображена в таблиці 3.1.

Таблиця 3.1 – Таблиця «devices»

Найменування поля	Тип даних	Призначення поля
device_id	INT	Ідентифікатор обладнання
name	VARCHAR(255)	Назва обладнання
type	VARCHAR(100)	Тип обладнання
location	VARCHAR(255)	Місце розташування

Таблиця «users» (таблиця користувачів) містить в собі наступні дані: логін, електронну пошту, пароль та роль користувача в системі, а також дату його реєстрації. За допомогою цієї таблиці користувач зможе створювати облікові записи, адміністрація зможе видаляти облікові записи та оновлювати дані користувача. Структура зображена в таблиці 3.2.

Таблиця 3.2 – Таблиця «users»

Найменування поля	Тип даних	Призначення поля
user_id	INT	Ідентифікатор користувача
username	VARCHAR(100)	Ім'я користувача
email	VARCHAR(255)	Електронна пошта
password_hash	VARCHAR(255)	Хеш паролю
role	VARCHAR(50)	Роль (адмін, користувач)
created_at	TIMESTAMP	Дата реєстрації

Таблиця «reports» (таблиця зі сповіщеннями) містить в собі наступні дані: унікальний ідентифікатор пристрою, у якого виявлена проблема, унікальний ідентифікатор користувача який створив репорт, текст репорту, статус усунення проблеми та дату виникнення цієї проблеми. За допомогою цієї таблиці користувач зможе переглядати та створювати репорти, а адміністратор міняти їх статуси в процесі їх усунення. Структура зображена в таблиці 3.3.

Таблиця 3.3 – Таблиця «reports»

Найменування поля	Тип даних	Призначення поля
rep_id	INT	Ідентифікатор репорту
user_id	INT	Ідентифікатор користувача
device_id	INT	Ідентифікатор обладнання
rep_description	TEXT	Опис події
status	VARCHAR(50)	Рівень важливості
timestamp	TIMESTAMP	Час створення сповіщення

3.3.4 Підтримка цілісності даних

PostgreSQL також надає потужну підтримку для забезпечення цілісності даних у базі даних, яка реалізується за допомогою вбудованих механізмів та

гнучких можливостей конфігурації. Основні аспекти забезпечення цілісності даних у PostgreSQL включають:

- вона підтримує різні типи обмежень на рівні таблиць та полів;

- вона дозволяє визначати складні правила перевірки, використовуючи CHECK constraints або тригери;

- вона повністю підтримує ACID-транзакції, що гарантують;

- вона дозволяє автоматизувати перевірки чи виконання певних дій при додаванні, зміні або видаленні записів у таблицях;

- вона дозволяє створювати власні типи даних з вбудованими правилами перевірки, що спрощує управління складними структурами даних;

- вона дозволяє додавати розширення для більш складних перевірок даних.

PostgreSQL забезпечує потужні механізми для підтримки цілісності даних, використовуючи гнучкі обмеження, транзакції та розширення. Це робить її відмінним вибором для систем, що вимагають високої точності, узгодженості та надійності даних.

3.4 Взаємодія клієнтської частини з сервером

У контексті даного проєкту, клієнтська частина веб-застосунку взаємодіє з сервером для реалізації функцій моніторингу електромеханічного обладнання та візуалізації отриманих даних. Ця взаємодія забезпечується через REST API, розроблений на серверній стороні. Основні аспекти цієї взаємодії описано нижче.

1. Аутентифікація та авторизація:

- під час входу клієнтська частина надсилає запит з обліковими даними користувача на сервер;

- сервер виконує перевірку облікових даних та повертає клієнту токен доступу, який використовується для аутентифікації подальших запитів.

2. Отримання даних моніторингу:

- клієнт запитує поточні дані моніторингу, які відображаються у вигляді графіків та таблиць.

3. Отримання розрахованих даних:

- клієнт запитує розраховані дані, які розраховуються сервером, відправляються клієнту та відображаються у вигляді графіків та таблиць.

4. Обробка помилок:

- у випадку виявлення помилок, користувач може створити репорт про це, після чого клієнтська частина відправить репорт на сервер для обробки.

Така архітектура взаємодії клієнт-сервер дозволяє забезпечити швидкий доступ до даних моніторингу, високий рівень безпеки та масштабованість системи, що відповідає вимогам цього проекту.

3.5 Інтеграція з хмарними сервісами

Інтеграція з хмарними сервісами в даному проекті спрямована на забезпечення надійного збору, обробки та аналізу даних моніторингу електромеханічного обладнання. Використання хмарних технологій дозволяє підвищити масштабованість, забезпечити централізований доступ до даних і оптимізувати витрати на інфраструктуру.

Для проєкту вибрана AWS (Amazon Web Services), яка надає потужний набір інструментів для роботи з даними IoT і моніторингу.

Дані передаються з IoT-пристроїв через MQTT, що забезпечує низьке енергоспоживання і швидкість передачі даних у реальному часі.

Для взаємодії з хмарною платформою використовується HTTPS і REST API. Потік даних з IoT-пристроїв передається в AWS IoT Core, де вони маршрутизуються в Amazon RDS для збереження.

Оброблені дані доступні через API для клієнтської частини, яка будує графіки та таблиці за допомогою D3.js. Для більш складного аналізу і побудови звітів дані інтегруються з Amazon QuickSight.

Переваги інтеграції:

- хмарні сервіси дозволяють легко адаптувати систему до зростання обсягів даних;

- інтегровані механізми шифрування даних і управління доступом;

- легке підключення нових пристроїв та функцій.

Інтеграція з хмарними сервісами забезпечує ефективність і надійність роботи системи моніторингу, спрощує управління даними, а також створює основу для подальшого розвитку проекту.

У межах цього розділу було проведено аналіз та визначення ключових технологічних аспектів, що забезпечують ефективність, надійність і масштабованість системи:

- Проведено обґрунтований вибір сучасних технологій, що забезпечують високу продуктивність, безпеку та сумісність з хмарними сервісами. Було обрано платформу розробки, мову програмування, фреймворки та стандарти, які відповідають вимогам проекту.

- Розроблено архітектуру системи на основі модульного підходу, що забезпечує розділення обов’язків між клієнтською, серверною частинами та базою даних. Використано багаторівневу архітектуру з інтеграцією хмарних сервісів для масштабованого зберігання даних і обчислювальних ресурсів.

- Обрано сучасну реляційну СУБД, яка підтримує роботу з великими обсягами даних і забезпечує високу швидкість обробки запитів.

- Оптимізовано фізичну модель для ефективного зберігання великих масивів даних із можливістю швидкого доступу до них.

- Використано REST API для забезпечення безшовної та безпечної взаємодії клієнтської та серверної частин. Реалізовано ефективну передачу даних у форматі JSON.

- Здійснено інтеграцію із хмарними сервісами для забезпечення масштабованого та надійного зберігання даних, використання обчислювальних ресурсів і резервного копіювання.

Таким чином, проектна частина забезпечила комплексний підхід до створення веб-застосунку, включаючи вибір оптимальних технологій, побудову ефективної архітектури та реалізацію інтеграції з хмарними сервісами. Розроблена структура системи відповідає вимогам надійності, масштабованості та зручності використання.

4 РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОЇ ЧАСТИНИ

4.1 Вимоги до інтерфейсу

Інтерфейс користувача для системи моніторингу електромеханічного обладнання має бути інтуїтивно зрозумілим, доступним та зручним для користувачів різного рівня підготовки. Вимоги до інтерфейсу можна класифікувати за кількома ключовими аспектами:

- він повинен бути таким, щоб користувачі могли без труднощів виконувати основні функції, навіть якщо вони не мають досвіду роботи з подібними системами, що включає прості й логічні меню, зрозумілі кнопки та елементи керування, а також чітке позначення функцій;

- всі його елементи повинні бути добре структуровані, тобто основні функціональні блоки, такі як перегляд даних, мають бути легкодоступними;

- для відображення стану обладнання і його параметрів мають використовуватися графіки, та таблиці;

- Він має бути здатний відображати поточний стан обладнання в реальному часі, без необхідності оновлювати сторінку вручну;

- для доступу до чутливих даних і налаштувань має бути впроваджена система аутентифікації з багаторівневою авторизацією, що означає що користувачі з різними рівнями доступу можуть мати різні можливості.

Інтерфейс користувача має бути розроблений з урахуванням вимог зручності, ефективності та безпеки, що дозволить користувачам швидко орієнтуватися в системі та ефективно виконувати моніторинг і налаштування електромеханічного обладнання.

4.2 Принципи UX/UI дизайну

При проектуванні інтерфейсу користувача для системи моніторингу електромеханічного обладнання важливо дотримуватись принципів UX/UI

дизайну, які забезпечують ефективність, зручність і задоволення користувача. Нижче наведено основні принципи, що застосовуються до цього проекту:

Дизайн має бути таким, щоб користувачі могли швидко зрозуміти, як працювати з системою. Всі елементи інтерфейсу повинні бути чітко позначені, а навігація – логічною та послідовною. Це дозволить користувачам з мінімальним досвідом роботи з подібними системами без зусиль виконувати потрібні дії. Система має не навантажувати користувача зайвою інформацією, а надавати тільки ті дані, що є важливими в поточний момент.

Всі ключові функції, такі як перегляд стану обладнання і перегляд репортів, повинні бути легкодоступними через логічну навігацію, яку можна реалізувати за допомогою меню.

Інтерфейс має підтримувати оновлення даних в реальному часі, що дає змогу користувачам отримувати актуальну інформацію про стан обладнання без необхідності вручну оновлювати сторінку.

Інтерфейс має забезпечувати належну аутентифікацію та авторизацію користувачів для доступу до конфіденційних даних.

Принципи UX/UI дизайну цього проекту зосереджені на забезпеченні зручності, доступності та безпеки користувачів. Врахування інтуїтивно зрозумілого дизайну, чіткої візуальної ієрархії, адаптивності інтерфейсу та інтерактивності дасть змогу створити ефективну систему моніторингу електромеханічного обладнання, що задовольняє потреби різних користувачів.

4.3 Опис основних екранів веб-застосунку

Веб-застосунок, створений для моніторингу електромеханічного обладнання, включатиме кілька основних екранів, які забезпечують зручний доступ до всіх необхідних функцій користувача.

У застосунку є три основні екрани:

- «Авторизація та реєстрація» – забезпечує доступ до системи. Користувачі вводять свої облікові дані для входу в систему.

- «Головна панель» – оперативне відображення стану всього обладнання та поточних показників. Містить графічні елементи, які допомагають користувачам швидко оцінити стан.

- «Панель розрахунків» – оперативне відображення розрахунків миттєвих значень потужностей, а також гармонійних складів струмів, напруг і потужностей. Містить графічні елементи, які допомагають користувачам швидко оцінити стан.

- «Репорти» – відображає репорти та їх стани, а також дає можливість створити репорт.

Описані екрани забезпечують ефективне управління системою моніторингу електромеханічного обладнання. Важливим аспектом є зручність і простота використання, зокрема для користувачів, які не мають спеціалізованих технічних знань. Інтерфейс має бути інтуїтивно зрозумілим, з можливістю швидко отримати потрібну інформацію для прийняття рішень щодо технічного стану обладнання.

4.4 Тестування та результати роботи

Коли користувач заходить на сайт він бачить сторінку входу. Сторінка входу зображено на рисунку 4.1.

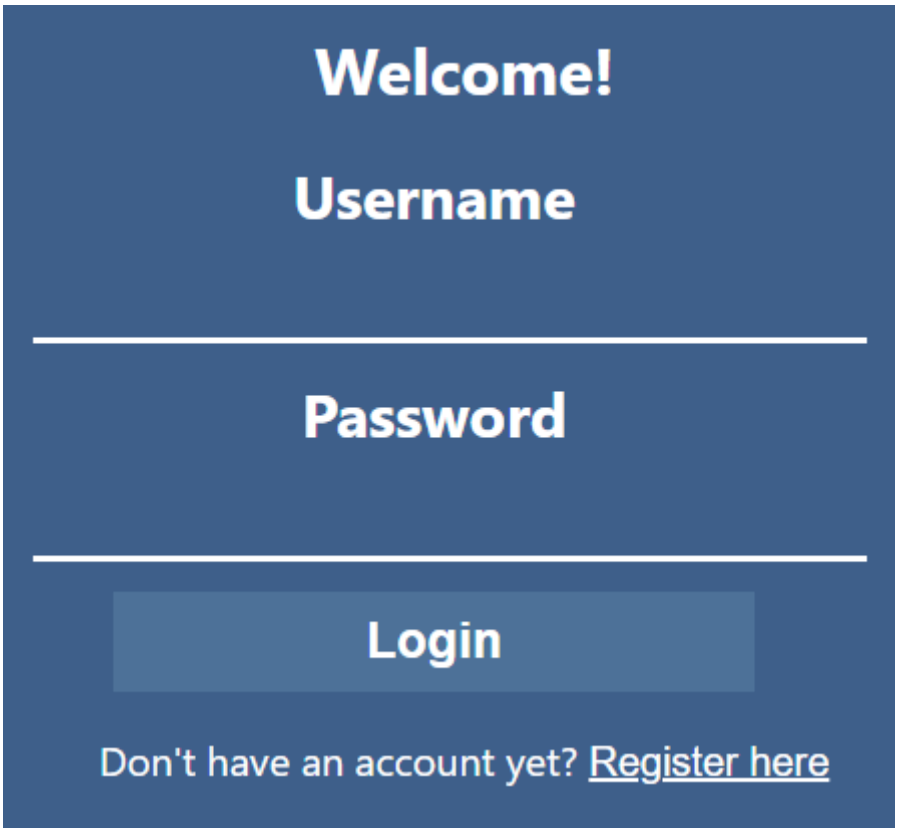


Рисунок 4.1 – Сторінка входу

При введені неправильних даних, користувач побачить повідомлення про невірне введення даних. Сторінка з повідомленням зображено на рисунку 4.2.

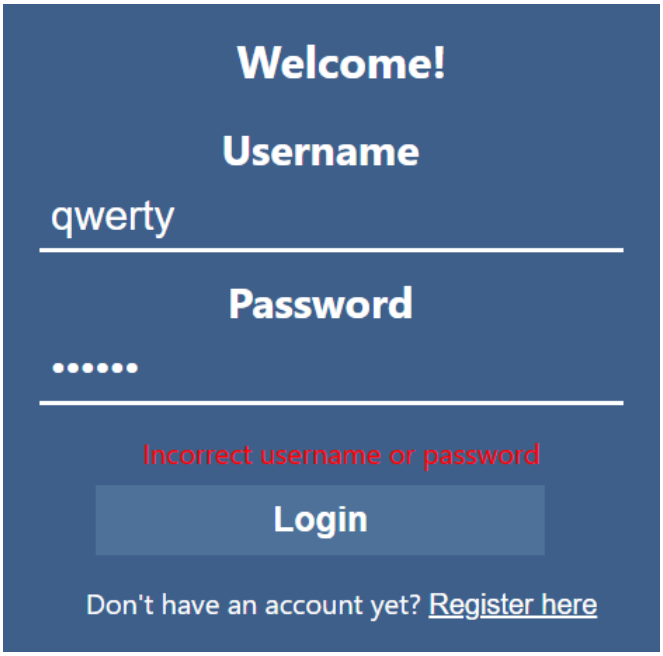


Рисунок 4.2 – Повідомлення про помилку

На сторінці входу є посилання «Register here». Якщо користувач натисне на неї, то він перейде на сторінку реєстрації. Сторінку реєстрації зображено на рисунку 4.3.

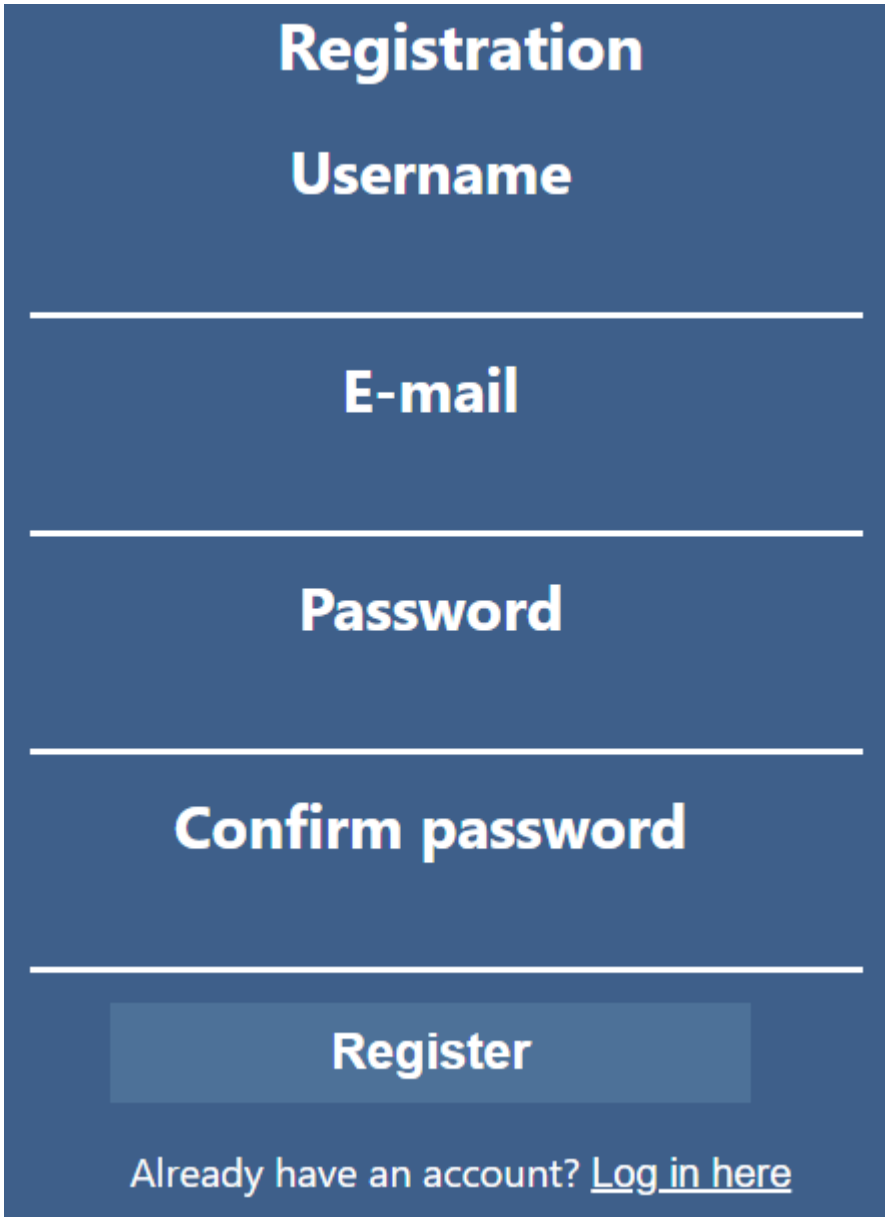


Рисунок 4.3 – Сторінка реєстрації

Якщо користувач порушить правильність введення, то він побачить помилку про це. Одне з повідомлень про помилку зображено на рисунку 4.4.

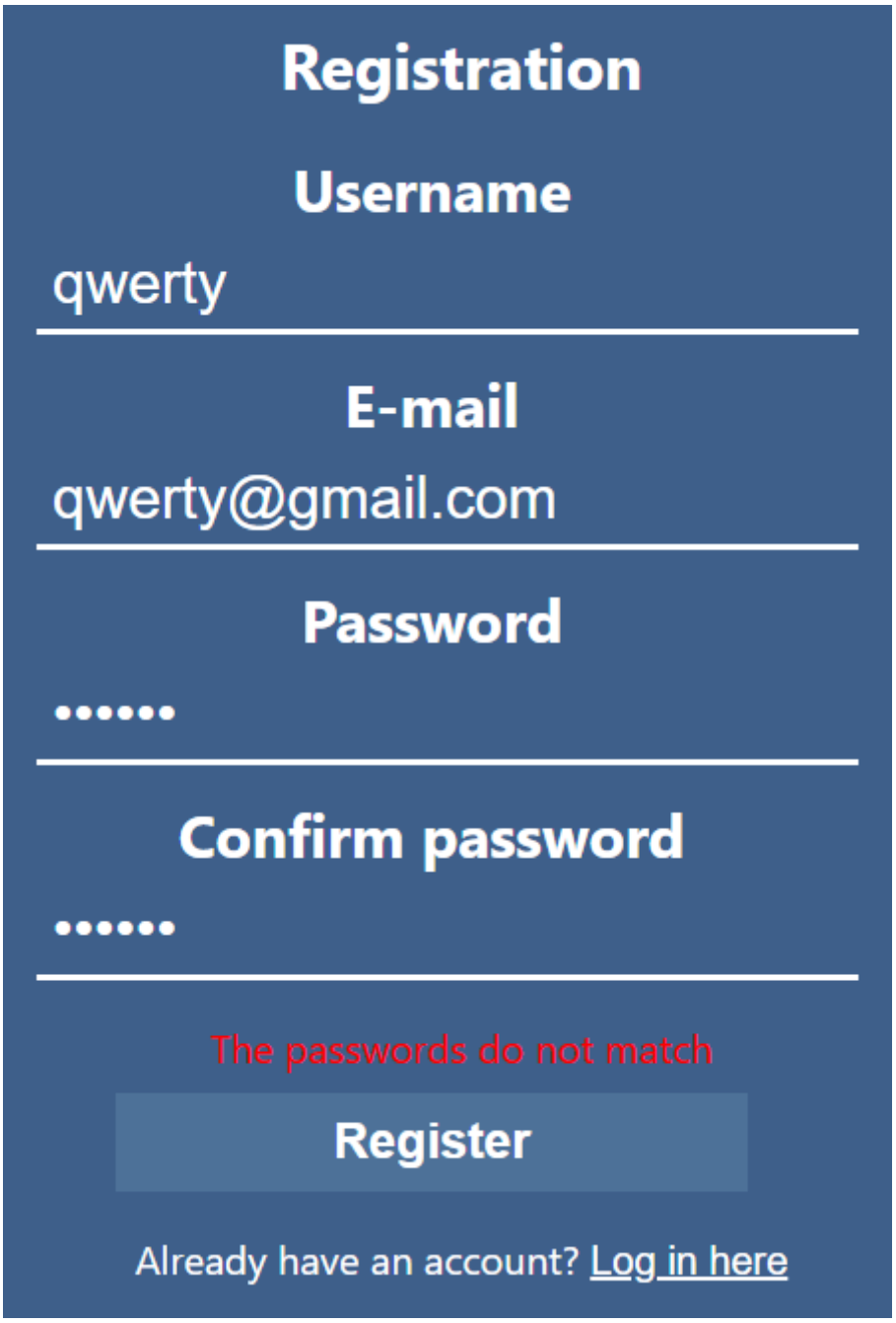


Рисунок 4.4 – Повідомлення про помилку

Після того як користувач зареєструється, його перекине на сторінку авторизації. Після введення своїх даних на сторінці авторизації користувач потрапляє на сторінку свого профілю. Сторінка профілю зображена на рисунку 4.5.

На сторінці профілю є кнопка «Exit from Profile». При натисканні на неї користувач вийде з профілю та опиниться на сторінці авторизації.

Зверху кожної сторінки є меню, за допомогою якого можна переходити на різні сторінки сайту, такі як: моніторинг («Monitoring»), розраховані дані («Calculating»), сторінку перегляду репортів («Reports») та сторінку профілю («Profile»). Меню зображено на рисунку 4.6.

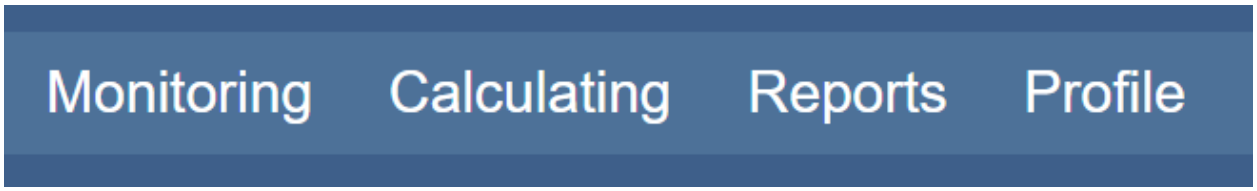
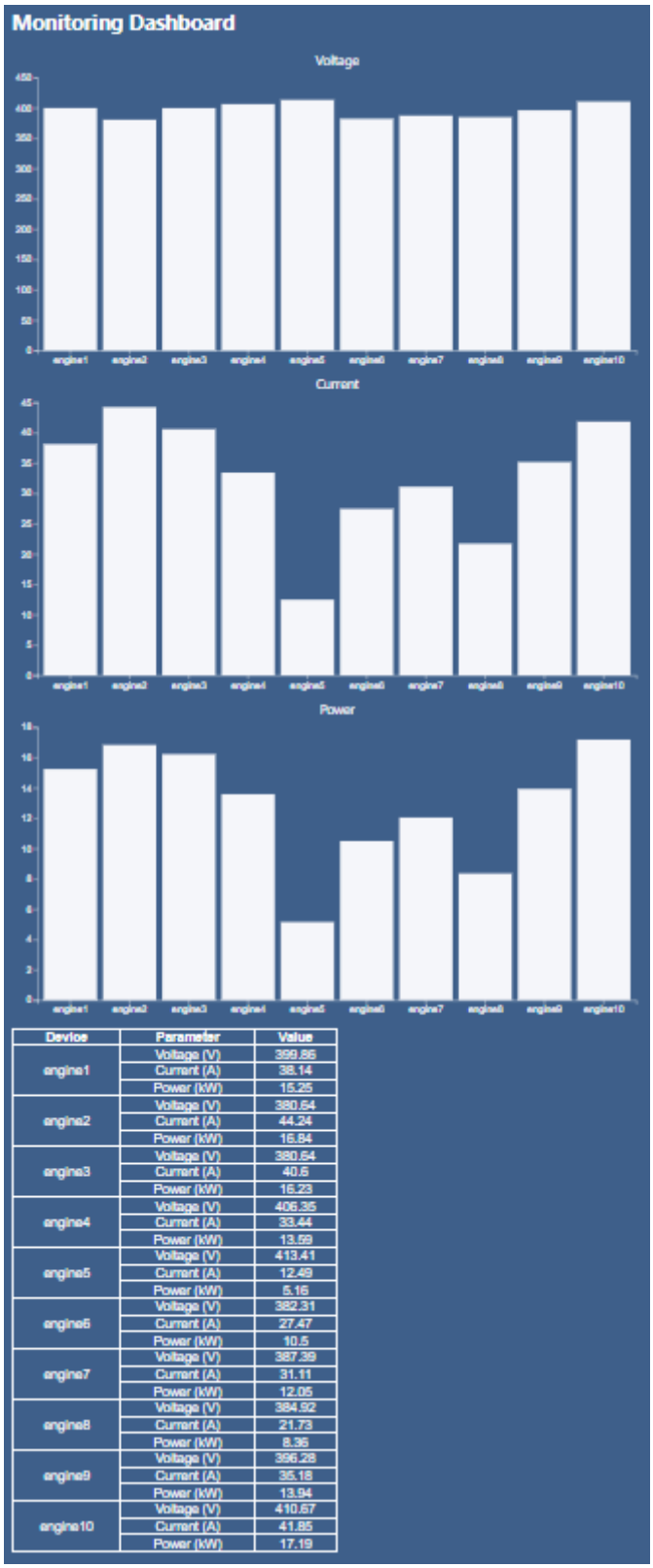


Рисунок 4.6 – Меню веб-застосунку.

Коли користувач натискає кнопку «Monitoring» в меню, він потрапляє на сторінку моніторингу, де бачить графіки та таблиці з даними про напругу, силу струму та потужність кожного обладнання в базі. Сторінка моніторингу зображена на рисунку 4.7.

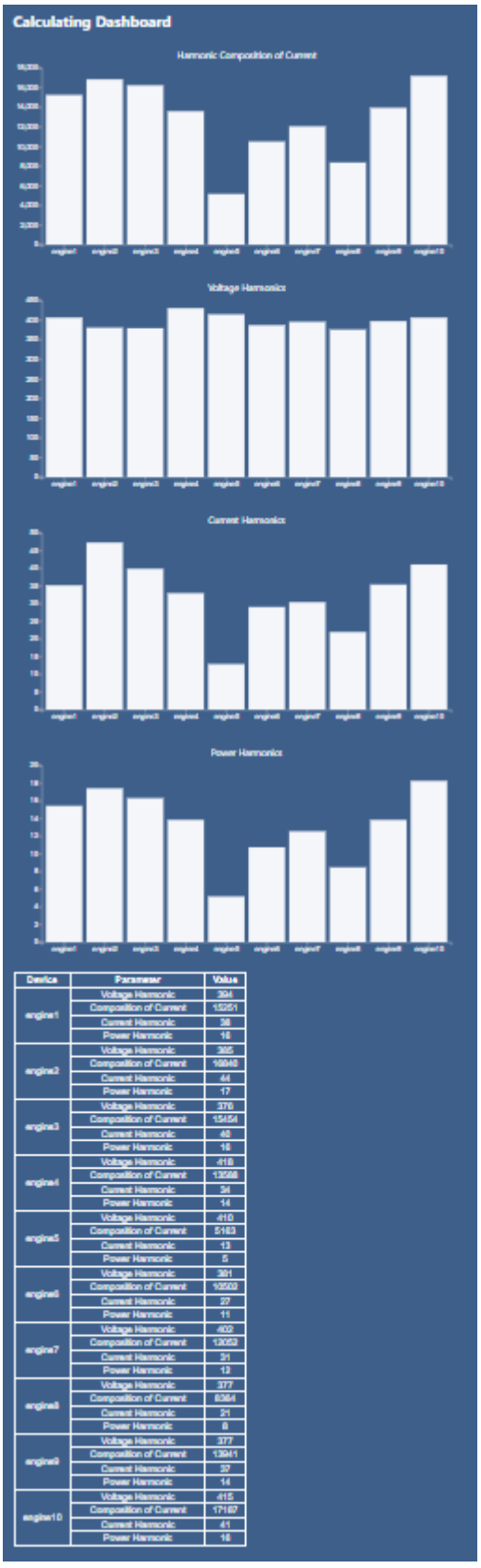


Device	Parameter	Value
engine1	Voltage (V)	399.86
	Current (A)	38.14
	Power (kW)	15.25
engine2	Voltage (V)	380.64
	Current (A)	44.24
	Power (kW)	16.84
engine3	Voltage (V)	380.64
	Current (A)	40.6
	Power (kW)	16.23
engine4	Voltage (V)	406.35
	Current (A)	33.44
	Power (kW)	13.59
engine5	Voltage (V)	413.41
	Current (A)	12.49
	Power (kW)	5.16
engine6	Voltage (V)	382.31
	Current (A)	27.47
	Power (kW)	10.5
engine7	Voltage (V)	387.39
	Current (A)	31.11
	Power (kW)	12.05
engine8	Voltage (V)	384.92
	Current (A)	21.73
	Power (kW)	8.36
engine9	Voltage (V)	396.28
	Current (A)	35.18
	Power (kW)	13.94
engine10	Voltage (V)	410.67
	Current (A)	41.85
	Power (kW)	17.19

Рисунок 4.7 – Сторінка моніторингу

Коли користувач натискає кнопку «Calculating» в меню, він потрапляє на сторінку з розрахованими даними, де бачить графіки та таблиці з миттєвими значеннями потужностей, а також гармонійними складами струмів, напруг і

потужностей кожного обладнання в базі. Сторінка з розрахованими даними зображена на рисунку 4.8.



Device	Parameter	Value
engine1	Voltage Harmonic	394
	Composition of Current	15251
	Current Harmonic	38
	Power Harmonic	16
engine2	Voltage Harmonic	395
	Composition of Current	16948
	Current Harmonic	44
	Power Harmonic	17
engine3	Voltage Harmonic	378
	Composition of Current	15454
	Current Harmonic	40
	Power Harmonic	16
engine4	Voltage Harmonic	418
	Composition of Current	13568
	Current Harmonic	34
	Power Harmonic	14
engine5	Voltage Harmonic	410
	Composition of Current	5183
	Current Harmonic	13
	Power Harmonic	5
engine6	Voltage Harmonic	391
	Composition of Current	10559
	Current Harmonic	27
	Power Harmonic	11
engine7	Voltage Harmonic	402
	Composition of Current	13053
	Current Harmonic	31
	Power Harmonic	12
engine8	Voltage Harmonic	377
	Composition of Current	8364
	Current Harmonic	21
	Power Harmonic	8
engine9	Voltage Harmonic	377
	Composition of Current	13941
	Current Harmonic	37
	Power Harmonic	14
engine10	Voltage Harmonic	415
	Composition of Current	17167
	Current Harmonic	41
	Power Harmonic	16

Рисунок 4.8 – Сторінка з розрахованими даними

Коли користувач натискає кнопку «Reports» в меню, він потрапляє на сторінку з репортами, де бачить таблицю з інформацією про репорти, та кнопку створення нового репорту («Create New Report»). Сторінка з репортами зображена на рисунку 4.9.

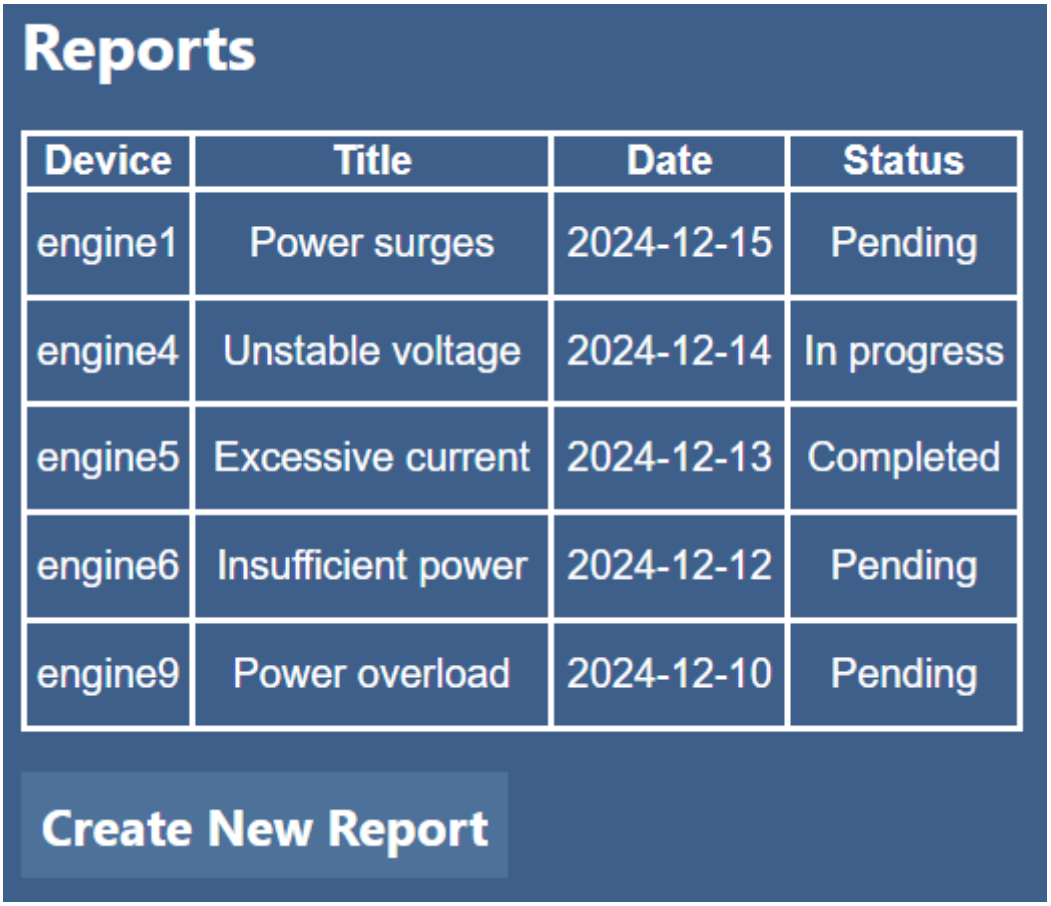


Рисунок 4.9 – Сторінка з репортами

Натиснувши кнопку «Create New Report» на сторінці репортів, користувач опиняється в меню створення репорту, де він обирає обладнання та його помилку, після чого підтверджує налаштування та відправляє репорт на сервер, і переходить на сторінку з репортами. Меню створення репорту зображено на рисунку 4.10.

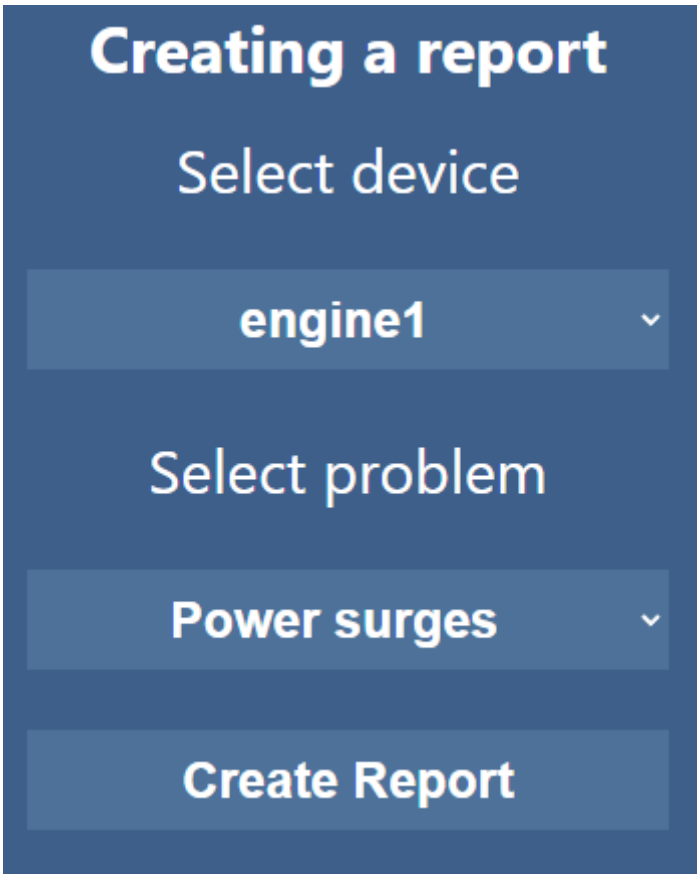


Рисунок 4.10. – Меню створення репорту

В рамках цього розділу були реалізовані наступні ключові аспекти:

- були визначені чіткі вимоги до інтерфейсу користувача;

- Застосовано сучасні принципи UX/UI дизайну;

- Створено детальний опис основних екранів, зокрема Авторизація та реєстрація, Головної панелі моніторингу, Панель розрахунків та Панель репортів;

- Проведено функціональне тестування веб-застосунку. Результати тестування показали стабільну роботу системи за умов навантаження, коректну обробку даних і швидкий відгук інтерфейсу.

Таким чином, розробка програмної частини успішно завершена, а результати тестування підтвердили відповідність реалізованого рішення поставленим вимогам. Застосунок готовий до впровадження та використання для моніторингу електромеханічного обладнання в реальних умовах.

ВИСНОВКИ

У результаті виконання роботи було створено систему моніторингу електромеханічного обладнання на основі хмарних технологій. Результати досліджень та розробок у рамках цієї роботи свідчать про її актуальність та практичну значущість.

Основні результати роботи полягають у наступному.

1. Розроблено клієнтську частину з використанням React.js, що гарантує зручний і зрозумілий інтерфейс.

2. Серверна частина побудована на Java та Spring Boot, що забезпечує стабільність і високу продуктивність при обробці великих обсягів даних.

3. Спроєктовано ефективну базу даних на основі PostgreSQL, яка забезпечує цілісність та збереження великих обсягів інформації.

4. Реалізовано інтеграцію з MQTT для збору даних від IoT-пристроїв.

5. Використання хмарного сховища забезпечило додаткові можливості для масштабованості та захисту даних.

Розроблена система може слугувати базою для подальших досліджень у галузі моніторингу електромеханічного обладнання, інтеграції IoT-пристроїв та впровадження хмарних рішень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Cloud-enabled prognosis for manufacturing [Електронний ресурс]. URL: https://www.sciencedirect.com/science/article/abs/pii/S000785061500150X?via%3Dihub

2. B. Furht, A. Escalante. «Handbook of Cloud Computing.» [Електронний ресурс]. URL: https://hero.lecturer.pens.ac.id/datahero/kuliah/cloud_computing/Handbook_of_Cloud_Computing.pdf

3. An environment safety monitoring system for agricultural production based on artificial intelligence, cloud computing and big data networks [Електронний ресурс]. URL: https://link.springer.com/article/10.1186/s13677-023- 00463-1

4. Хмарні рішення для IoT та IIoT. [Електронний ресурс]. URL: https://pupenasan.github.io/TI40/%D0%9B%D0%B5%D0%BA%D1%86/cloudiot.html

5. Induction motor fault diagnostic and monitoring methods [Електронний ресурс]. URL: https://www.researchgate.net/publication/243055807_Induction_motor_fault_diagnostic_and_monitoring_methods

6. MindSphere [Електронний ресурс]. URL: https://www.siemens.com/ua/uk/produkty/prohramne-zabezpechennya/mindsphere.html

7. Розбираємо роль хмарних технологій в IoT пристроях: в чому їх переваги? [Електронний ресурс]. URL: https://gigacloud.ua/blog/navchannja/rozbiraemo-rol-hmarnih-tehnologij-v-iot-pristrojah-v-chomu-ih-perevagi

8. Cloud intelligence deployed locally on IoT edge devices [Електронний ресурс]. URL: https://azure.microsoft.com/en-us/products/iot-edge/

9. Machine learning for IoT network monitoring [Електронний ресурс]. URL: https://hal.science/hal-02438733/file/final_RESSI_2019_ML_for_IoT_network_monitoring.pdf

10. AWS IoT [Електронний ресурс]. URL: https://aws.amazon.com/ru/iot/

11. Build your IoT application with two-way communication [Електронний ресурс]. URL: https://azure.microsoft.com/en-us/products/iot-hub

12. What is Cloud IoT Core? [Електронний ресурс]. Точка доступу: https://cloud.google.com/blog/topics/developers-practitioners/what-cloud-iot-core

13. Безпека хмарних обчислень [Електронний ресурс]. URL: https://uk.wikipedia.org/wiki/%D0%91%D0%B5%D0%B7%D0%BF%D0%B5%D0%BA%D0%B0_%D1%85%D0%BC%D0%B0%D1%80%D0%BD%D0%B8%D1%85_%D0%BE%D0%B1%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D1%8C

14. SCADA [Електронний ресурс]. URL: https://uk.wikipedia.org/wiki/SCADA

15. SIMATIC WinCC V7 [Електронний ресурс]. URL: https://www.siemens.com/ua/uk/produkty/avtomatyzatsiya-promyslovosti/prohramne-zabezpechennya-dlya-promyslovosti/prohramne-zabezpechennya-dlya-avtomatyzatsiyi/scada/simatic-wincc-v7.html

16. EcoStruxure Platform [Електронний ресурс]. URL: https://www.se.com/ua/uk/work/campaign/innovation/platform.jsp

17. Uptake Technologies [Електронний ресурс]. URL: https://www.uptake.com/

18. Senseye Predictive Maintenance [Електронний ресурс]. URL: https://www.siemens.com/global/en/products/services/digital-enterprise-services/analytics-artificial-intelligence-services/predictive-services/senseye-predictive-maintenance.html

19. Predix (software) [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/Predix_(software)

20. Система управління базами даних. [Електронний ресурс]. URL: https://uk.wikipedia.org/wiki/Система_управління_базами_даних

21. Кібербезпека: актуальні загрози та методи захисту [Електронний ресурс]. URL: https://lemon.school/blog/kiberbezpeka-aktualni-zagrozy-ta-metody-zahystu

22. Найкращі практики для безпеки веб- та мобільних додатків [Електронний ресурс]. URL: https://wezom.com.ua/ua/blog/naykraschi-praktiki-dlya-bezpeki-veb-ta-mobilnih-dodatkiv

23. Про моніторинг безпеки в інформаційній системі [Електронний ресурс]. URL: https://duikt.edu.ua/uploads/n_7836_42482763.pdf?file=n_7836_42482763.pdf

24. Правила безпечної експлуатації електроустановок споживачів - Основні вимоги безпеки під час обслуговування електроустановок [Електронний ресурс]. Точка доступу: https://leg.co.ua/knigi/pravila/pravila-bezpechnoyi-ekspluataciyi-elektroustanovok-spozhivachiv-2.html

25. Правила безпечної експлуатації електроустановок споживачів - Технічні заходи [Електронний ресурс]. URL: https://leg.co.ua/knigi/pravila/pravila-bezpechnoyi-ekspluataciyi-elektroustanovok-spozhivachiv-4.html

Додаток А

Охорона праці та безпека життєдіяльності

А.1. Оцінка ризиків для працівників при використанні веб-застосунків для моніторингу електромеханічного обладнання

З основних ризиків для працівників при використанні веб-застосунків для моніторингу електромеханічного обладнання можна виділити:

- Технічні ризики. До них входять: некоректні або помилкові дані; відмова в роботі веб-застосунку; кібератаки [21, 22].

Виведення помилкових даних через збої в сенсорах, мережах або алгоритмах обробки може призвести до неправильного розуміння стану обладнання, через що працівники можуть залишатись в зоні небезпеки, або не вжити своєчасних заходів. Також, помилкові дані можуть призвести до несподіваного виходу з ладу обладнання, що створить небезпеку для працівників, що знаходяться поруч. Ще постійні помилкові дані можуть змусити працівників ігнорувати попередження системи.

Недоступність системи через технічні збої (проблеми з сервером, мережею або апаратним забезпеченням), що може призвести до відсутності інформації в реальному часі, через що працівники не зможуть оцінити стан обладнання, що спричинить затримку виявлення небезпечної ситуації.

Хакери можуть отримати контроль над системою та спричинити небезпечну ситуацію за допомогою сфальсифікованих показників, через які працівники не зможуть виявити реальну небезпеку. DDoS-атаки або блокування серверів може призвести до зупинки роботи веб-застосунку, що залишить працівників без даних.

Для того щоб зменшити ризики показу некоректних або помилкових даних можна використати такі способи мінімізації ризиків: регулярне калібрування сенсорів; валідація даних на рівні програмного забезпечення [15]; вбудовані механізми сигналізації при виявленні аномалій [14].

Для зменшення відмов в роботі веб-застосунку можна такі способи мінімізації: використання хмарних технологій з високою доступністю; налаштування резервних копій та дублюючих серверів [14].

Щоб зменшити ризик кібератак можна застосувати: використання безпечних протоколів передачі даних (HTTPS, MQTT із TLS); системи аутентифікації користувачів; [22] регулярні оновлення програмного забезпечення для усунення вразливостей [22].

- Організаційні ризики. До них входять: недостатня кваліфікація працівників [22]; залежність від оператора.

Недостатня кваліфікація працівників може призвести до некоректного інтерпретування даних, або здійснення неправильних дій, що може призвести до ситуації, коли обладнання залишиться в небезпечному для працівників стані. Через невміння швидко знайти потрібну інформацію у застосунку або правильно використати його функції, зростає ризик затримки в ухваленні рішень у критичних умовах. Також неправильні дії можуть поставити працівників у ситуацію, коли вони опиняються поруч із несправним або небезпечним обладнанням.

Залежність від оператора при відсутності оператора може призвести до того, що інші працівники не зможуть знайти потрібні параметри, або зрозуміти попередження системи, що створює ризик для всього підприємства. Якщо ж оператор доступний, то він може помилитися, і в разі цієї помилки може статися аварійна ситуація. Вірогідність помилок може зрости через тиск відповідальності оператора, через який у оператора може зрости стрес та перевтома.

Щоб зменшити недостатню кваліфікацію працівників можна проводити навчання по використанню веб-застосунку. Також для облегшення використання застосунку можна забезпечити його зрозумілим та інтуїтивним інтерфейсом. Також може допомогти наявність технічної документації.

Щоб зменшити залежність від оператора можна організувати колективний доступ для кількох працівників. Також допоможе наявність чітких протоколів дій у разі збою системи.

- Психологічні ризики. До них входять: перевантаження інформацією; залежність від застосунку.

Небезпеку для працівників може спричините перевантаження інформацією, через що працівники можуть не звертати увагу на критично важливі сигнали, наприклад критичне перевантаження двигуна. Стрес через велику складність аналізу великої кількості даних, також може призвести до помилки і аварії. Надлишок інформації уповільнює процес ухвалення рішень і виконання завдань. Також надмірна кількість даних змушує працівників приймати рішення на основі припущень, що підвищує ризики.

Якщо система залежить від застосунку, то його збій унеможливить моніторинг стану обладнання, через що працівник не зможе оцінити стан двигуна без цифрових показників. Якщо застосунок помилиться, а працівник повністю на нього покладається, то вони можуть не помітити проблему. Також через відключення електроенергії, застосунок не працює, через що небезпечні стани залишаються непоміченими.

Для зменшення перевантаження інформацією можна використовувати інфографіку для візуалізації основних параметрів, а також налаштувати системи сповіщень лише для критичних подій.

Щоб зменшити залежність системи від застосунку, можна забезпечити можливість ручного втручання та підтвердження автоматизованих рішень. Також допоможе розробка процедур для прийняття рішень у випадку сумнівних результатів.

- Фізичні ризики. Перебування біля обладнання під час аварійної ситуації.

Через неправильну оцінку стану обладнання, працівники можуть перебувати у небезпечній зоні.

Для зменшення цього ризику потрібно створити систему попереджень для сигналізації небезпечного стану та розробити безпечну процедуру обслуговування обладнання.

А.2. Безпека при інтеграції хмарних технологій для моніторингу електромеханічного обладнання

Використання хмарних технологій значно розширює можливості моніторингу, зберігання та обробки даних, але водночас створює ризики, пов’язані із забезпеченням безпеки, таких як:

- Кіберзагрози: атаки на дані, під час їх передачі між обладнанням, сервером і хмарою або несанкціонований доступ; віруси та шкідливі ПЗ в хмарній інфраструктурі [21].

При атаці на дані, під час їх передачі або несанкціонованому доступі зловмисники можуть підмінити або підробити дані, що може призвести неправильного прийняття рішень, що натомість може призвести до аварії та зростання стресу у працівників через недовіру до системи моніторингу [22].

Віруси та шкідливі ПЗ в хмарній інфраструктурі можуть вивести з ладу систему моніторингу або маніпулювати даними, через що несправність обладнання залишиться непоміченою, що може призвести до аварії через неправильні дії. Також через інфікування хмарної інфраструктури може заразитися обладнання підприємства, що насамперед може викликати збої в його управлінні, що спричинить ризик для працівників.

- Залежність від провайдера хмарних послуг.

Якщо сервіси провайдера стануть недоступними через збій або помилки в роботі (наприклад через кібератаку), то це спричинить недоступність до даних або їх втрату, що може унеможливить своєчасно виявити несправності обладнання, та збільшить ризик для здоров’я працівників через аварію. Також провайдер може не забезпечувати необхідну пропускну здатність, що призведе до затримки у відображені даних, що призведе до запізнілого реагування на критичні неполадки обладнання.

- Інтеграційні ризики.

Інтеграція нових систем моніторингу, зокрема з використанням хмарних технологій, може створювати низку ризиків для працівників. Основні

небезпеки пов’язані з технічними, організаційними та інформаційними аспектами впровадження.

Нові системи можуть не інтегруватися належним чином із існуючим обладнанням, що може призвести до неповного відображення даних, через що зросте ризик аварійної ситуації. Також некоректні конфігурації під час інтеграції можуть порушити роботу системи, що може підвищити ризик аварійної несправності обладнання через невчасне виявлення. Ще інтеграція може призвести до раптових збоїв у роботі обладнання чи моніторингових систем, що також підвищить ризик аварійної ситуації.

Працівники можуть не мати достатніх знань чи навичок для роботи з новими системами, що призведе до зростання ймовірності помилок через недостатнє розуміння функціонування системи.

У процесі переходу на нову систему можуть бути втрачені історичні дані або поточні показники, що ускладнить аналіз стану обладнання та підвищить ризик помилки у діагностиці та прийнятті рішень. Ще може зрости кількість вразливостей системи під час інтеграції, що може бути використано зловмисниками для несанкціонованого доступу та підміни даних. Також працівники можуть бути перевантажені інформацією, що знизить продуктивність обробки великої кількості даних, та призведе до втрати концентрації, що може призвести до помилки.

Щоб збільшити безпеку при інтеграції хмарних технологій для моніторингу електромеханічного обладнання можна виконати наступні заходи:

- Захист даних: використання протоколів шифрування (наприклад, TLS/SSL) при передачі даних; шифрувати дані в хмарі [21].

- Контроль доступу: використання двофакторної аутентифікації для доступу до хмарних ресурсів; розподілити права доступу для користувачів залежно від їхніх ролей [22]; журнулювання та моніторинг усіх спроб доступу до системи [21].

- Вибір надійного провайдера хмарних технологій: оцінення репутацію та сертифікацію провайдера; забезпечення можливість відновлення даних за допомогою резервного копіювання [21].

- Захист від атак: використання фаєрволів, VPN, та інших засобів захисту для з'єднань між хмарою і локальними пристроями; регулярна перевірка системи на наявність вразливостей за допомогою спеціалізованого ПЗ [21].

- Забезпечення безперебійної роботи: впровадження планів на випадок збоїв; використання розподілених хмарних ресурсів для мінімізації впливу регіональних проблем.

- Захист IoT-пристроїв: перевірка, що пристрої IoT використовують офіційне програмне забезпечення без вразливостей [23]; регулярне оновлювання прошивки та застосовувати патчів безпеки [22]; забезпечення захист IoT-пристроїв від фізичного втручання.

- Виявлення аномалій: використання інструментів штучного інтелекту для моніторингу підозрілої активності [21].

- Інтеграція з локальними системами: використання шлюзів з обмеженим доступом для підключення обладнання до хмари; перевірка надійності протоколів, які використовуються для обміну даними (наприклад, MQTT) [21].

- Навчання персоналу з питань безпеки та роботи із хмарними технологіями [21-23].

- Регулярне проведення аудитів безпеки хмарної інфраструктури [21, 23].

- Розробка плану дій у разі інцидентів, що включає як технічні, так і організаційні заходи.

Інтеграція хмарних технологій для моніторингу електромеханічного обладнання може бути безпечною за умови грамотної реалізації заходів захисту і регулярного вдосконалення системи безпеки.

А.3. Технічні заходи безпеки під час обслуговування та ремонту електромеханічного обладнання, яке моніториться за допомогою веб-застосунку

Обслуговування та ремонт електромеханічного обладнання, що використовує веб-застосунки для моніторингу, потребують спеціальних технічних заходів для забезпечення безпеки працівників. Такі заходи спрямовані на зниження ризиків, пов’язаних із роботою під напругою, неправильним відображенням даних, збоєм у системі моніторингу чи людським фактором.

1. Забезпечення електробезпеки [25]:

- відключення напруги перед початком робіт перед обслуговуванням чи ремонтом обладнання [24];

- використання блокувальних пристроїв для запобігання випадковому ввімкненню обладнання;

- використання сертифікованих пристроїв для перевірки відсутності напруги на всіх фазах перед початком робіт;

- установка переносного заземлення на частини обладнання, де можливе накопичення залишкової напруги.

2. Контроль роботи веб-застосунку під час обслуговування:

- використання ручних вимірювальних приладів для підтвердження інформації, отриманої від веб-застосунку;

- перевірка показників у режимі реального часу для виявлення розбіжностей;

- налаштування сповіщень про критичні помилки чи потенційні ризики у веб-застосунку для оперативного реагування.

3. Фізичний контроль доступу:

- розміщення устаткування у приміщеннях із обмеженим доступом, оснащених системами контролю доступу [24];

- впровадження систем авторизації, які дозволяють лише кваліфікованим спеціалістам вносити зміни чи вмикати обладнання;

- установка датчиків відкривання кришок шаф управління для запобігання несанкціонованому доступу.

4. Використання засобів індивідуального захисту [24]:

- діелектричні рукавички, боти та ізолювальні килимки [25];

- захисні окуляри для запобігання травмам від електричних дуг [25];

- каски та робочий одяг із вогнетривкими властивостями;

- ремені безпеки при роботі на висоті або у важкодоступних місцях.

5. Контроль за функціонуванням системи моніторингу:

- регулярне технічне обслуговування веб-застосунку, серверів, датчиків та інших компонентів системи моніторингу;

- наявність резервного доступу до даних моніторингу;

- регулярне проведення навчань для персоналу щодо дій у разі збоїв у роботі системи.

6. Захист інформаційної безпеки [21]:

- використання сучасних протоколів шифрування для передачі даних між обладнанням, сервером і веб-застосунком.

- ведення журналів доступу до даних та дій працівників у веб-застосунку.

- налаштування регулярного створення резервних копій даних для збереження інформації у разі кібератак чи збою.

7. Організаційні заходи:

- регулярні тренінги з безпечного виконання робіт та роботи із системою моніторингу;

- ознайомлення з особливостями функціонування обладнання та веб-застосунку;

- створення чітких інструкцій, щодо виконання робіт із зазначенням потенційних небезпек та заходів безпеки [24].

Додаток Б

Код, таблиці, графіки

src/App.js:

```
import React from 'react';
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import Navbar from './components/Navbar';
import Home from './pages/Home';
import Monitoring from './pages/Monitoring';
import Reports from './pages/Reports';
import Profile from './pages/Profile';
import Registration from './pages/Registration';
import Calculating from './pages/Calculating';
import RepCreate from './pages/RepCreate';


const App = () => (
  <BrowserRouter>
    <Navbar />
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/monitoring" element={<Monitoring />} />
      <Route path="/calculating" element={<Calculating />} />
      <Route path="/reports" element={<Reports />} />
      <Route path="/profile" element={<Profile />} />
      <Route path="/register" element={<Registration />} />
      <Route path="/report-create" element={<RepCreate />} />
    </Routes>
  </BrowserRouter>
);

export default App;
```

src/index.js:

```
body {
  margin: 0;
  font-family:  -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto',
'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  background-color: #3E5F8A;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}

td, th {
  border: 3px solid white;
  color: white;
}

table {
  border-collapse: collapse;
  width: 500px;
  height: 300px;
  font-family: Arial;
        font-size: 20px;
  text-align: center;
```

```
  margin-left: 30px;
  margin-bottom: 30px;
}

ul {
  display: flex;
  background-color: #4D7198;
  list-style-type: none;
}

li, .nav {
  padding: 15px 10px;
  font: 30px Arial;
  text-decoration: none;
  color: white;
}

a:hover {
  background-color: #4285B4;
}

svg, p, h1 {
  color: white;
  margin-left: 30px;
}

svg {
  font-size: 20px;
}
```

```
p {
        font-size: 20px;
}

.home-container, .register-container {
  max-width: 400px;
  margin: 0 auto;
  padding: 20px;
  text-align: center;
}

.auth-form .form-group {
  margin-bottom: 15px;
}

.auth-form label {
  display: block;
  margin-bottom: 5px;
  font-weight: bold;
}

.auth-form input {
  width: 100%;
  height: 30px;
  font-size: 30px;
  padding: 8px;
  background: none;
  border: none;
  border-bottom: 3px solid white;
  color: white;
```

```
}

.error-message {
  color: red;
  margin-bottom: 10px;
}

.login-button, .register-button, .generate-report-button, .exit-btn {
  width: 320px;
  height: 50px;
  font-size: 25px;
  font-weight: bold;
  padding: 10px;
  background-color: #4D7198;
  color: white;
  border: none;
  cursor: pointer;
}

.login-button:hover, .register-button:hover, .generate-report-button:hover, .exit-btn:hover {
  background-color: #4285B4;
}

label {
  color: white;
  font-size: 30px;
}

.reg, .log {
```

```
  font: 20px Arial;
  color: white;
}

.repCr {
  width: 320px;
  height: 50px;
  font-size: 25px;
  font-weight: bold;
  padding: 10px;
  background-color: #4D7198;
  color: white;
  cursor: pointer;
  text-decoration: none;
  margin-left: 30px;
}

.create-report {
  max-width: 400px;
  margin: 0 auto;
  padding: 20px;
  text-align: center;
}

h1.create-report {
  margin-left: 0px;
}

select {
  width: 320px;
```

```
  height: 50px;
  font-size: 25px;
  font-weight: bold;
  padding: 10px;
  background-color: #4D7198;
  color: white;
  cursor: pointer;
  text-decoration: none;
  text-align: center;
  margin: 30px 0;
  border: none;
}

#start-date {
  width: 320px;
  height: 50px;
  font-size: 25px;
  font-weight: bold;
  padding: 10px;
  background-color: #4D7198;
  color: white;
  cursor: pointer;
  text-decoration: none;
  text-align: center;
  margin: 30px 0;
  border: none;
}
.exit-btn {
  margin-left: 30px;
}
```

src/componets/CalChart.js:

```
import React, { useEffect, useRef } from 'react';
import * as d3 from 'd3';

const Chart = () => {
  const svgRef = useRef();

  useEffect(() => {
    const voltage = [
      { device: 'engine1', value: 399.86 },
      { device: 'engine2', value: 380.64 },
      { device: 'engine3', value: 399.77 },
      { device: 'engine4', value: 406.35 },
      { device: 'engine5', value: 413.41 },
      { device: 'engine6', value: 382.31 },
      { device: 'engine7', value: 387.39 },
      { device: 'engine8', value: 384.92 },
      { device: 'engine9', value: 396.28 },
      { device: 'engine10', value: 410.67 }
    ];

    const current = [
      { device: 'engine1', value: 38.14 },
      { device: 'engine2', value: 44.24 },
      { device: 'engine3', value: 40.6 },
      { device: 'engine4', value: 33.44 },
      { device: 'engine5', value: 12.49 },
      { device: 'engine6', value: 27.47 },
      { device: 'engine7', value: 31.11 },
```

```
  { device: 'engine8', value: 21.73 },
  { device: 'engine9', value: 35.18 },
  { device: 'engine10', value: 41.85 }
];

const power = [
  { device: 'engine1', value: 15.25 },
  { device: 'engine2', value: 16.84 },
  { device: 'engine3', value: 16.23 },
  { device: 'engine4', value: 13.59 },
  { device: 'engine5', value: 5.16 },
  { device: 'engine6', value: 10.50 },
  { device: 'engine7', value: 12.05 },
  { device: 'engine8', value: 8.36 },
  { device: 'engine9', value: 13.94 },
  { device: 'engine10', value: 17.19 }
];

const hcoc = voltage.map((volt, index) => {
    const currentValue = current.find(item => item.device === volt.device);
    return {
      device: volt.device,
      value: volt.value * currentValue.value
    };
});

// Функція для генерації гармонійного складу та обчислення середнього значення
function generateHarmonics(baseValue) {
    const harmonics = [];
```

```
        const numHarmonics = 5; // Кількість гармонік (основна частота + перші кілька гармонік)

        for (let i = 0; i < numHarmonics; i++) {
          // Генерація гармонік: основна частота + варіації для кожної гармоніки
          harmonics.push(baseValue * (Math.random() * 0.2 + 0.9)); // Скалярний множник для варіації значень гармонік
        }

        // Обчислення середнього значення гармонік
        const averageHarmonic = harmonics.reduce((sum, value) => sum + value, 0) / harmonics.length;

        return averageHarmonic;
      }

      // Обчислення гармонійного складу для напруг, струмів і потужностей
      const voltageHarmonics = voltage.map(v => ({
        device: v.device,
        value: generateHarmonics(v.value)
      }));

      const currentHarmonics = current.map(c => ({
        device: c.device,
        value: generateHarmonics(c.value)
      }));

      const powerHarmonics = power.map(p => ({
        device: p.device,
```

```
  value: generateHarmonics(p.value)
}));

  const width = 1000;
  const height = 500;
  const margin = { top: 60, right: 60, bottom: 60, left: 60 };

  const x = d3.scaleBand()
    .domain(hcoc.map(d => d.device))
    .range([margin.left, width - margin.right])
    .padding(0.1);

  const y1 = d3.scaleLinear()
    .domain([0, d3.max(hcoc, d => d.value)])
    .nice()
    .range([height - margin.bottom, margin.top]);

  const y2 = d3.scaleLinear()
    .domain([0, d3.max(voltageHarmonics, d => d.value)])
    .nice()
    .range([height - margin.bottom, margin.top]);

  const y3 = d3.scaleLinear()
    .domain([0, d3.max(currentHarmonics, d => d.value)])
    .nice()
    .range([height - margin.bottom, margin.top]);

  const y4 = d3.scaleLinear()
    .domain([0, d3.max(powerHarmonics, d => d.value)])
    .nice()
```

```
  .range([height - margin.bottom, margin.top]);

const svg = d3.select(svgRef.current)
  .attr('width', width)
  .attr('height', height * 4);

svg.selectAll('*').remove(); // Очистка перед рендером

svg.append('g')
  .selectAll('.bar')
  .data(hcoc)
  .enter()
  .append('rect')
  .attr('class', 'bar')
  .attr('x', d => x(d.device))
  .attr('y', d => y1(d.value))
  .attr('width', x.bandwidth())
  .attr('height', d => y1(0) - y1(d.value))
  .attr('fill', '#F5F6FA');

svg.append('g')
  .attr('transform', `translate(0,${height - margin.bottom})`)
  .call(d3.axisBottom(x))
  .selectAll('text')
  .style('font-size', '15px');

svg.append('g')
  .attr('transform', `translate(${margin.left}, 0)`)
  .call(d3.axisLeft(y1))
  .selectAll('text')
```

```
  .style('font-size', '15px');

svg.append('text')
  .attr('x', width / 2)
  .attr('y', margin.top - 20)
  .attr('text-anchor', 'middle')
  .attr('fill', 'white')
  .text('Harmonic Composition of Current');

svg.append('g')
  .selectAll('.bar')
  .data(voltageHarmonics)
  .enter()
  .append('rect')
  .attr('class', 'bar')
  .attr('x', d => x(d.device))
  .attr('y', d => y2(d.value) + height)
  .attr('width', x.bandwidth())
  .attr('height', d => y2(0) - y2(d.value))
  .attr('fill', '#F5F6FA');

svg.append('g')
  .attr('transform', `translate(0,${height + margin.bottom + 380})`)
  .call(d3.axisBottom(x))
  .selectAll('text')
  .style('font-size', '15px');

svg.append('g')
  .attr('transform', `translate(${margin.left}, ${height})`)
  .call(d3.axisLeft(y2))
```

```
  .selectAll('text')
  .style('font-size', '15px');

svg.append('text')
  .attr('x', width / 2)
  .attr('y', height + margin.top - 20)
  .attr('text-anchor', 'middle')
  .attr('fill', 'white')
  .text('Voltage Harmonics');

svg.append('g')
  .selectAll('.bar')
  .data(currentHarmonics)
  .enter()
  .append('rect')
  .attr('class', 'bar')
  .attr('x', d => x(d.device))
  .attr('y', d => y3(d.value) + height * 2)
  .attr('width', x.bandwidth())
  .attr('height', d => y3(0) - y3(d.value))
  .attr('fill', '#F5F6FA');

svg.append('g')
  .attr('transform', `translate(0,${height * 2 + margin.bottom + 380})`)
  .call(d3.axisBottom(x))
  .selectAll('text')
  .style('font-size', '15px');

svg.append('g')
  .attr('transform', `translate(${margin.left}, ${height * 2})`)
```

```
  .call(d3.axisLeft(y3))
  .selectAll('text')
  .style('font-size', '15px');

svg.append('text')
  .attr('x', width / 2)
  .attr('y', height * 2 + margin.top - 20)
  .attr('text-anchor', 'middle')
  .attr('fill', 'white')
  .text('Current Harmonics');

  svg.append('g')
  .selectAll('.bar')
  .data(powerHarmonics)
  .enter()
  .append('rect')
  .attr('class', 'bar')
  .attr('x', d => x(d.device))
  .attr('y', d => y4(d.value) + height * 3)
  .attr('width', x.bandwidth())
  .attr('height', d => y4(0) - y4(d.value))
  .attr('fill', '#F5F6FA');

svg.append('g')
  .attr('transform', `translate(0,${height * 3 + margin.bottom + 380})`)
  .call(d3.axisBottom(x))
  .selectAll('text')
  .style('font-size', '15px');

svg.append('g')
```

```
      .attr('transform', `translate(${margin.left}, ${height * 3})`)
      .call(d3.axisLeft(y4))
      .selectAll('text')
      .style('font-size', '15px');

    svg.append('text')
      .attr('x', width / 2)
      .attr('y', height * 3 + margin.top - 20)
      .attr('text-anchor', 'middle')
      .attr('fill', 'white')
      .text('Power Harmonics');
  }, []);

  return (
    <svg ref={svgRef}></svg>
  );
};

export default Chart;
```

src/components/CalDashboard

```
import React from 'react';
import CalChart from './CalChart';
import CalDataTable from './CalDataTable';

const CalDashboard = () => (
  <div>
    <h1>Calculating Dashboard</h1>
    <CalChart />
```

```
    <CalDataTable />
  </div>
);

export default CalDashboard;
```

src/components/CalDataTables.js:

```
import React from 'react';

const DataTable = () => {
  const data = [
    { device: 'engine1', parameter: 'Voltage (V)', value: 399.86 },
    { device: 'engine1', parameter: 'Current (A)', value: 38.14 },
    { device: 'engine1', parameter: 'Power (kW)', value: 15.25 },

    { device: 'engine2', parameter: 'Voltage (V)', value: 380.64 },
    { device: 'engine2', parameter: 'Current (A)', value: 44.24 },
    { device: 'engine2', parameter: 'Power (kW)', value: 16.84 },

    { device: 'engine3', parameter: 'Voltage (V)', value: 380.64 },
    { device: 'engine3', parameter: 'Current (A)', value: 40.6 },
    { device: 'engine3', parameter: 'Power (kW)', value: 16.23 },

    { device: 'engine4', parameter: 'Voltage (V)', value: 406.35 },
    { device: 'engine4', parameter: 'Current (A)', value: 33.44 },
    { device: 'engine4', parameter: 'Power (kW)', value: 13.59 },

    { device: 'engine5', parameter: 'Voltage (V)', value: 413.41 },
    { device: 'engine5', parameter: 'Current (A)', value: 12.49 },
```

```
        { device: 'engine5', parameter: 'Power (kW)', value: 5.16 },

        { device: 'engine6', parameter: 'Voltage (V)', value: 382.31 },
        { device: 'engine6', parameter: 'Current (A)', value: 27.47 },
        { device: 'engine6', parameter: 'Power (kW)', value: 10.50 },

        { device: 'engine7', parameter: 'Voltage (V)', value: 387.39 },
        { device: 'engine7', parameter: 'Current (A)', value: 31.11 },
        { device: 'engine7', parameter: 'Power (kW)', value: 12.05 },

        { device: 'engine8', parameter: 'Voltage (V)', value: 384.92 },
        { device: 'engine8', parameter: 'Current (A)', value: 21.73 },
        { device: 'engine8', parameter: 'Power (kW)', value: 8.36 },

        { device: 'engine9', parameter: 'Voltage (V)', value: 396.28 },
        { device: 'engine9', parameter: 'Current (A)', value: 35.18 },
        { device: 'engine9', parameter: 'Power (kW)', value: 13.94 },

        { device: 'engine10', parameter: 'Voltage (V)', value: 410.67 },
        { device: 'engine10', parameter: 'Current (A)', value: 41.85 },
        { device: 'engine10', parameter: 'Power (kW)', value: 17.19 }
      ];

    // Функція для генерації гармонійного складу та обчислення середнього значення
    function generateHarmonics(baseValue) {
      const harmonics = [];
      const numHarmonics = 5; // Кількість гармонік (основна частота + перші кілька гармонік)
```

```
        for (let i = 0; i < numHarmonics; i++) {
            // Генерація гармонік: основна частота + варіації для кожної гармоніки
            harmonics.push(baseValue * (Math.random() * 0.2 + 0.9)); // Скалярний множник для варіації значень гармонік
        }

        // Обчислення середнього значення гармонік
        const averageHarmonic = harmonics.reduce((sum, value) => sum + value, 0) / harmonics.length;

        return averageHarmonic;
    }

    // Об'єднання результатів в один масив
    const results = data.reduce((acc, item) => {
        // Генерація гармоніків для кожного параметра
        const harmonicValue = generateHarmonics(item.value);

        // Додавання результатів для гармонік
        acc.push({
            device: item.device,
            parameter: `${item.parameter.replace(/ \(.*\)/, '')} Harmonic`, // Наприклад, "Voltage Harmonic"
            value: Math.round(harmonicValue)
        });

        // Додавання складу струму (скалярний добуток напруги на струм)
        if (item.parameter === 'Voltage (V)') {
```

```
            const currentItem = data.find(d => d.device === item.device && d.parameter === 'Current (A)');
            if (currentItem) {
              acc.push({
                device: item.device,
                parameter: 'Composition of Current',
                value: Math.round(item.value * currentItem.value) // Добуток напруги та струму
              });
            }
          }

          return acc;
        }, []);

        console.log(results);

        // Групуємо дані за device
        const groupedData = results.reduce((acc, item) => {
          if (!acc[item.device]) {
            acc[item.device] = [];
          }
          acc[item.device].push(item);
          return acc;
        }, {});

        return (
          <table>
            <thead>
              <tr>
```

```
            <th>Device</th>
            <th>Parameter</th>
            <th>Value</th>
          </tr>
        </thead>
        <tbody>
          {Object.entries(groupedData).map(([device, rows]) => (
            <React.Fragment key={device}>
              <tr>
                <td rowSpan={rows.length + 1}>{device}</td>
              </tr>
              {rows.map((row, index) => (
                <tr key={`${device}-${index}`}>
                  <td>{row.parameter}</td>
                  <td>{row.value}</td>
                </tr>
              ))}
            </React.Fragment>
          ))}
        </tbody>
      </table>
    );
};

export default DataTable;
```

src/components/CalChart.js:

```
import React, { useEffect, useRef } from 'react';
import * as d3 from 'd3';
```

```
const Chart = () => {
  const svgRef = useRef();

  useEffect(() => {
    const voltage = [
      { device: 'engine1', value: 399.86 },
      { device: 'engine2', value: 380.64 },
      { device: 'engine3', value: 399.77 },
      { device: 'engine4', value: 406.35 },
      { device: 'engine5', value: 413.41 },
      { device: 'engine6', value: 382.31 },
      { device: 'engine7', value: 387.39 },
      { device: 'engine8', value: 384.92 },
      { device: 'engine9', value: 396.28 },
      { device: 'engine10', value: 410.67 }
    ];

    const current = [
      { device: 'engine1', value: 38.14 },
      { device: 'engine2', value: 44.24 },
      { device: 'engine3', value: 40.6 },
      { device: 'engine4', value: 33.44 },
      { device: 'engine5', value: 12.49 },
      { device: 'engine6', value: 27.47 },
      { device: 'engine7', value: 31.11 },
      { device: 'engine8', value: 21.73 },
      { device: 'engine9', value: 35.18 },
      { device: 'engine10', value: 41.85 }
    ];
```

```
const power = [
  { device: 'engine1', value: 15.25 },
  { device: 'engine2', value: 16.84 },
  { device: 'engine3', value: 16.23 },
  { device: 'engine4', value: 13.59 },
  { device: 'engine5', value: 5.16 },
  { device: 'engine6', value: 10.50 },
  { device: 'engine7', value: 12.05 },
  { device: 'engine8', value: 8.36 },
  { device: 'engine9', value: 13.94 },
  { device: 'engine10', value: 17.19 }
];

const width = 1000;
const height = 500;
const margin = { top: 40, right: 40, bottom: 40, left: 40 };

const x = d3.scaleBand()
  .domain(voltage.map(d => d.device))
  .range([margin.left, width - margin.right])
  .padding(0.1);

const y1 = d3.scaleLinear()
  .domain([0, d3.max(voltage, d => d.value)])
  .nice()
  .range([height - margin.bottom, margin.top]);

const y2 = d3.scaleLinear()
  .domain([0, d3.max(current, d => d.value)])
```

```
  .nice()
  .range([height - margin.bottom, margin.top]);

const y3 = d3.scaleLinear()
  .domain([0, d3.max(power, d => d.value)])
  .nice()
  .range([height - margin.bottom, margin.top]);

const svg = d3.select(svgRef.current)
  .attr('width', width)
  .attr('height', height * 3);

svg.selectAll('*').remove(); // Очистка перед рендером

svg.append('g')
  .selectAll('.bar')
  .data(voltage)
  .enter()
  .append('rect')
  .attr('class', 'bar')
  .attr('x', d => x(d.device))
  .attr('y', d => y1(d.value))
  .attr('width', x.bandwidth())
  .attr('height', d => y1(0) - y1(d.value))
  .attr('fill', '#F5F6FA');

svg.append('g')
  .attr('transform', `translate(0,${height - margin.bottom})`)
  .call(d3.axisBottom(x))
  .selectAll('text')
```

```
  .style('font-size', '15px');

svg.append('g')
  .attr('transform', `translate(${margin.left}, 0)`)
  .call(d3.axisLeft(y1))
  .selectAll('text')
  .style('font-size', '15px');

svg.append('text')
  .attr('x', width / 2)
  .attr('y', margin.top - 20)
  .attr('text-anchor', 'middle')
  .attr('fill', 'white')
  .text('Voltage');

svg.append('g')
  .selectAll('.bar')
  .data(current)
  .enter()
  .append('rect')
  .attr('class', 'bar')
  .attr('x', d => x(d.device))
  .attr('y', d => y2(d.value) + height)
  .attr('width', x.bandwidth())
  .attr('height', d => y2(0) - y2(d.value))
  .attr('fill', '#F5F6FA');

svg.append('g')
  .attr('transform', `translate(0,${height + margin.bottom + 420})`)
  .call(d3.axisBottom(x))
```

```
  .selectAll('text')
  .style('font-size', '15px');

svg.append('g')
  .attr('transform', `translate(${margin.left}, ${height})`)
  .call(d3.axisLeft(y2))
  .selectAll('text')
  .style('font-size', '15px');

svg.append('text')
  .attr('x', width / 2)
  .attr('y', height + margin.top - 20)
  .attr('text-anchor', 'middle')
  .attr('fill', 'white')
  .text('Current');

svg.append('g')
  .selectAll('.bar')
  .data(power)
  .enter()
  .append('rect')
  .attr('class', 'bar')
  .attr('x', d => x(d.device))
  .attr('y', d => y3(d.value) + height * 2) // Переміщаємо третій графік
вниз
  .attr('width', x.bandwidth())
  .attr('height', d => y3(0) - y3(d.value))
  .attr('fill', '#F5F6FA');

svg.append('g')
```

```
      .attr('transform', `translate(0,${height * 2 + margin.bottom + 420})`)
      .call(d3.axisBottom(x))
      .selectAll('text')
      .style('font-size', '15px');

    svg.append('g')
      .attr('transform', `translate(${margin.left}, ${height * 2})`)
      .call(d3.axisLeft(y3))
      .selectAll('text')
      .style('font-size', '15px');

    svg.append('text')
      .attr('x', width / 2)
      .attr('y', height * 2 + margin.top - 20)
      .attr('fill', 'white')
      .attr('text-anchor', 'middle')
      .text('Power');

  }, []);

  return (
    <svg ref={svgRef}></svg>
  );
};

export default Chart;
```

src/components/Dashboard.js:

```
import React from 'react';
```

```
import Chart from './Chart';
import DataTable from './DataTable';

const Dashboard = () => (
  <div>
    <h1>Monitoring Dashboard</h1>
    <Chart />
    <DataTable />
  </div>
);

export default Dashboard;
```

src/components/DataTable.js:

```
import React from 'react';

const DataTable = () => {
  const data = [
    { device: 'engine1', parameter: 'Voltage (V)', value: 399.86 },
    { device: 'engine1', parameter: 'Current (A)', value: 38.14 },
    { device: 'engine1', parameter: 'Power (kW)', value: 15.25 },

    { device: 'engine2', parameter: 'Voltage (V)', value: 380.64 },
    { device: 'engine2', parameter: 'Current (A)', value: 44.24 },
    { device: 'engine2', parameter: 'Power (kW)', value: 16.84 },

    { device: 'engine3', parameter: 'Voltage (V)', value: 380.64 },
    { device: 'engine3', parameter: 'Current (A)', value: 40.6 },
    { device: 'engine3', parameter: 'Power (kW)', value: 16.23 },
```

```
  { device: 'engine4', parameter: 'Voltage (V)', value: 406.35 },
  { device: 'engine4', parameter: 'Current (A)', value: 33.44 },
  { device: 'engine4', parameter: 'Power (kW)', value: 13.59 },

  { device: 'engine5', parameter: 'Voltage (V)', value: 413.41 },
  { device: 'engine5', parameter: 'Current (A)', value: 12.49 },
  { device: 'engine5', parameter: 'Power (kW)', value: 5.16 },

  { device: 'engine6', parameter: 'Voltage (V)', value: 382.31 },
  { device: 'engine6', parameter: 'Current (A)', value: 27.47 },
  { device: 'engine6', parameter: 'Power (kW)', value: 10.50 },

  { device: 'engine7', parameter: 'Voltage (V)', value: 387.39 },
  { device: 'engine7', parameter: 'Current (A)', value: 31.11 },
  { device: 'engine7', parameter: 'Power (kW)', value: 12.05 },

  { device: 'engine8', parameter: 'Voltage (V)', value: 384.92 },
  { device: 'engine8', parameter: 'Current (A)', value: 21.73 },
  { device: 'engine8', parameter: 'Power (kW)', value: 8.36 },

  { device: 'engine9', parameter: 'Voltage (V)', value: 396.28 },
  { device: 'engine9', parameter: 'Current (A)', value: 35.18 },
  { device: 'engine9', parameter: 'Power (kW)', value: 13.94 },

  { device: 'engine10', parameter: 'Voltage (V)', value: 410.67 },
  { device: 'engine10', parameter: 'Current (A)', value: 41.85 },
  { device: 'engine10', parameter: 'Power (kW)', value: 17.19 }
];
```

```
// Групуємо дані за device
const groupedData = data.reduce((acc, item) => {
  if (!acc[item.device]) {
    acc[item.device] = [];
  }
  acc[item.device].push(item);
  return acc;
}, {});

return (
  <table>
    <thead>
      <tr>
        <th>Device</th>
        <th>Parameter</th>
        <th>Value</th>
      </tr>
    </thead>
    <tbody>
      {Object.entries(groupedData).map(([device, rows]) => (
        <React.Fragment key={device}>
          <tr>
            <td rowSpan={rows.length + 1}>{device}</td>
          </tr>
          {rows.map((row, index) => (
            <tr key={`${device}-${index}`}>
              <td>{row.parameter}</td>
              <td>{row.value}</td>
            </tr>
          ))}
```

```
          </React.Fragment>
        ))}
      </tbody>
    </table>
  );
};

export default DataTable;
```

src/components/Navbar.js:

```
import React from 'react';
import { Link } from 'react-router-dom';

const Navbar = () => (
  <nav>
    <ul>
      <li><Link to="/" class="nav">Home</Link></li>
      <li><Link to="/monitoring" class="nav">Monitoring</Link></li>
      <li><Link to="/calculating" class="nav">Calculating</Link></li>
      <li><Link to="/reports" class="nav">Reports</Link></li>
      <li><Link to="/profile" class="nav">Profile</Link></li>
    </ul>
  </nav>
);

export default Navbar;
```

src/pages/Calculating.js:

```
import React from 'react';
import CalDashboard from '../components/CalDashboard';

const Calculating = () => <CalDashboard />;

export default Calculating;
```

src/pages/Home.js:

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';

const Home = () => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [errorMessage, setErrorMessage] = useState('');
  const navigate = useNavigate();

  const handleLogin = (e) => {
    e.preventDefault();

    // Простая проверка для демонстрации
    if (username === 'admin' && password === 'password') {
      navigate('/monitoring');
    } else {
      setErrorMessage('Incorrect username or password');
    }
  };
```

```
        return (
          <div className="home-container">
            <h1>Welcome!</h1>
            <form className="auth-form" onSubmit={handleLogin}>
              <div className="form-group">
                <label htmlFor="username">Username</label>
                <input
                  type="text"
                  id="username"
                  value={username}
                  onChange={(e) => setUsername(e.target.value)}
                  required
                />
              </div>
              <div className="form-group">
                <label htmlFor="password">Password</label>
                <input
                  type="password"
                  id="password"
                  value={password}
                  onChange={(e) => setPassword(e.target.value)}
                  required
                />
              </div>
              {errorMessage && <p className="error-
message">{errorMessage}</p>}
              <button type="submit" className="login-button">Login</button>
            </form>
            <p>
```

```
            Don't have an account yet? <a class="reg" href="/register">Register here</a>
              </p>
            </div>
          );
        };

        export default Home;
```

src/pages/Monitoring.js:

```
        import React from 'react';
        import Dashboard from '../components/Dashboard';

        const Monitoring = () => <Dashboard />;

        export default Monitoring;
```

src/pages/Profile.js

```
        import React from 'react';

        const Profile = () => (
          <div>
            <h1>User Profile</h1>
            <div>
              <p><strong>Name:</strong> Hryhorenko Yaroslav</p>
              <p><strong>Email:</strong> valsoray228@gmail.com</p>
              <p><strong>Registered:</strong>  2024-12-19</p>
              <button className="exit-btn">Exit from Profile</button>
```

```
    </div>
  </div>
);

export default Profile;
```

src/pages/Registration.js:

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';

const Register = () => {
  const [username, setUsername] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [confirmPassword, setConfirmPassword] = useState('');
  const [errorMessage, setErrorMessage] = useState('');
  const navigate = useNavigate();

  const handleRegister = (e) => {
    e.preventDefault();

    // Перевірка підтвердження паролю
    if (password !== confirmPassword) {
      setErrorMessage('The passwords do not match');
      return;
    }

    // Приклад простої перевірки для демонстрації
    if (username && email && password) {
```

```
    // Тут можно додати логіку для реєстрації
    console.log('Registration successful');
    navigate('/'); // Перенаправлення на сторінку авторизації
  } else {
    setErrorMessage('Please fill in all fields');
  }
};

return (
  <div className="register-container">
    <h1>Registration</h1>
    <form className="auth-form" onSubmit={handleRegister}>
      <div className="form-group">
        <label htmlFor="username">Username</label>
        <input
          type="text"
          id="username"
          value={username}
          onChange={(e) => setUsername(e.target.value)}
          required
        />
      </div>
      <div className="form-group">
        <label htmlFor="email">E-mail</label>
        <input
          type="email"
          id="email"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
          required
```

```
          />
        </div>
        <div className="form-group">
          <label htmlFor="password">Password</label>
          <input
            type="password"
            id="password"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            required
          />
        </div>
        <div className="form-group">
          <label htmlFor="confirmPassword">Confirm password</label>
          <input
            type="password"
            id="confirmPassword"
            value={confirmPassword}
            onChange={(e) => setConfirmPassword(e.target.value)}
            required
          />
        </div>
        {errorMessage && <p className="error-message">{errorMessage}</p>}
        <button type="submit" className="register-button">Register</button>
      </form>
      <p>
        Already have an account? <a class="log" href="/">Log in here</a>
      </p>
    </div>
```

```
  );
};

export default Register;
```

src/pages/RepCreate.js:

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';

const CreateReport = () => {
  const [devSelect, setDevSelect] = useState('summary');
  const [statSelect, setStatSelect] = useState('summary');
  const navigate = useNavigate();

  const handleSubmit = (e) => {
    e.preventDefault();
    const reportData = {
      statSelect,
      devSelect,
    };
    console.log('Generate a report with data:', reportData);
    // Тут мождна додати логіку відправки даних на сервер
    navigate('/reports');
  };

  return (
    <div className="create-report">
      <h1 className="create-report">Creating a report</h1>
      <form onSubmit={handleSubmit} className="report-form">
```

```
<div className="form-group">
  <label>Select device </label>
  <select
    value={devSelect}
    onChange={(e) => setDevSelect(e.target.value)}
  >
    <option value="engine1">engine1</option>
    <option value="engine2">engine2</option>
    <option value="engine3">engine3</option>
    <option value="engine4">engine4</option>
    <option value="engine5">engine5</option>
    <option value="engine6">engine6</option>
    <option value="engine7">engine7</option>
    <option value="engine8">engine8</option>
    <option value="engine9">engine9</option>
    <option value="engine10">engine10</option>
  </select>
</div>

<div className="form-group">
  <label>Select problem </label>
  <select
    value={devSelect}
    onChange={(e) => setStatSelect(e.target.value)}
  >
    <option value="Power surges">Power surges</option>
    <option value="Low current">Low current</option>
    <option value="Power overload">Power overload</option>
    <option value="Unstable voltage">Unstable voltage</option>
    <option value="Excessive current">Excessive current</option>
```

```
                <option value="Insufficient power">Insufficient power</option>
                <option value="Voltage fluctuations">Voltage fluctuations</option>
                <option value="Short circuit current">Short circuit current</option>
                <option value="Low voltage">Low voltage</option>
              </select>
            </div>

            <button type="submit" className="generate-report-button">
              Create Report
            </button>
          </form>
        </div>
      );
    };

    export default CreateReport;
```

src/pages/Reports.js:

```
    import React from 'react';

    const Reports = () => {
      const reports = [
        { device: 'engine1', title: 'Power surges', date: '2024-12-15', status: 'Pending'
},
        { device: 'engine4', title: 'Unstable voltage', date: '2024-12-14', status: 'In
progress' },
        { device: 'engine5', title: 'Excessive current', date: '2024-12-13', status:
'Completed' },
```

```
        { device: 'engine6', title: 'Insufficient power', date: '2024-12-12', status: 'Pending' },
        { device: 'engine9', title: 'Power overload', date: '2024-12-10', status: 'Pending' },
        ];

        return (
          <div>
            <h1>Reports</h1>
            <table>
              <thead>
                <tr>
                  <th>Device</th>
                  <th>Title</th>
                  <th>Date</th>
                  <th>Status</th>
                </tr>
              </thead>
              <tbody>
                {reports.map((report) => (
                  <tr key={report.device}>
                    <td>{report.device}</td>
                    <td>{report.title}</td>
                    <td>{report.date}</td>
                    <td>{report.status}</td>
                  </tr>
                ))}
              </tbody>
            </table>
            <a href="/report-create" class="repCr">Create New Report</a>
```

```
    </div>
  );
};

export default Reports;
```

pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>3.1.0</version>
                <relativePath/>
        </parent>
        <groupId>com.example</groupId>
        <artifactId>monitoring</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>monitoring</name>
        <description>Demo project for Spring Boot</description>
        <url/>
        <licenses>
                <license/>
        </licenses>
        <developers>
```

```
        <developer/>
</developers>
<scm>
        <connection/>
        <developerConnection/>
        <tag/>
        <url/>
</scm>
<properties>
        <java.version>21</java.version>
        <spring-boot.version>2.7.10</spring-boot.version>
<maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
</properties>
<dependencies>
        <!-- Spring Data JPA for database access -->
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
        <!-- Spring Boot Web for REST API -->
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <!-- Spring Boot DevTools for development -->
```

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>
        <!-- PostgreSQL Driver -->
        <dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-test</artifactId>
            <scope>test</scope>
        </dependency>
        <!-- Paho MQTT for MQTT integration -->
<dependency>
<groupId>org.eclipse.paho</groupId>
      <artifactId>org.eclipse.paho.client.mqttv3</artifactId>
   <version>1.2.5</version>
   </dependency>
        <!-- Hibernate Core (JPA Provider) -->
        <dependency>
```

```
            <groupId>org.hibernate</groupId>
                <artifactId>hibernate-core</artifactId>
                    <version>5.6.10.Final</version>
            </dependency>
        </dependencies>

        <build>
            <plugins>
                <plugin>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-maven-plugin</artifactId>
                </plugin>
            </plugins>
        </build>

</project>
```

src/main/resources/application.properties:

```
spring.application.name=monitoring

# PostgreSQL
spring.datasource.url=jdbc:postgresql://localhost:5432/monitoring_db
spring.datasource.username=postgres
spring.datasource.password=122122aab29kl
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.datasource.driver-class-name=org.postgresql.Driver

# MQTT Broker Configuration
```

```
mqtt.broker.url=tcp://localhost:1883
mqtt.client.id=equipment-monitoring
mqtt.topic=sensor/data
mqtt.username=mqtt_username
mqtt.password=mqtt_password

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.properties.hibernate.format_sql=true

server.port=8080
spring.mvc.throw-exception-if-no-handler-found=true
spring.web.resources.add-mappings=true

management.endpoints.web.exposure.include=*
```

src/main/java/com/exemple/monitoring/MonitoringApplication.java:

```
package com.example.monitoring;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.autoconfigure.domain.EntityScan;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

@SpringBootApplication
@EnableJpaRepositories(basePackages = "com.example.monitoring.repository")
@EntityScan(basePackages = "com.example.monitoring.model")
```

```
public class MonitoringApplication {
    public static void main(String[] args) {
        SpringApplication.run(MonitoringApplication.class,  args);
    }
}
```

src/main/java/com/exemple/monitoring/controller/DeviceController.java:

```
package com.example.monitoring.controller;

import com.example.monitoring.model.Device;
import com.example.monitoring.service.DeviceService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/devices")
public class DeviceController {

    @Autowired
    private DeviceService deviceService;

    @GetMapping
    public List<Device> getDevices() {
        return deviceService.getAllDevices();
    }

    @PostMapping
```

```
    public Device addDevice(@RequestBody Device device) {
        return deviceService.saveDevice(device);
    }
}
```

src/main/java/com/exemple/monitoring/controller/MeasurementController.java:

```
package com.example.monitoring.controller;

import com.example.monitoring.model.Measurement;
import com.example.monitoring.service.MeasurementService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/measurements")
public class MeasurementController {

    @Autowired
    private MeasurementService measurementService;

    @PostMapping
    public Measurement addMeasurement(@RequestBody Measurement measurement) {
        return measurementService.saveMeasurement(measurement);
    }
}
```

src/main/java/com/example/monitorin/model/Device.java:

```
package com.example.monitoring.model;

import javax.persistence.*;
import java.time.LocalDateTime;

@Entity
@Table(name = "device")
public class Device {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long deviceId;
    private String name;
    private String type;
    private String location;
    private LocalDateTime createdAt;

    // Геттери і сеттери
    public Long getDeviceId() {
        return deviceId;
    }

    public void setDeviceId(Long deviceId) {
        this.deviceId = deviceId;
    }

    public String getName() {
        return name;
    }
```

```
    public void setName(String name) {
        this.name = name;
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    public String getLocation() {
        return location;
    }

    public void setLocation(String location) {
        this.location = location;
    }

    public LocalDateTime getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(LocalDateTime createdAt) {
        this.createdAt = createdAt;
    }
}
```

src/main/java/com/example/monitorin/mqtt/MqttConfig.java:

```
package com.example.monitoring.mqtt;

import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class MqttConfig {

    @Value("${mqtt.broker.url}")
    private String brokerUrl;

    @Value("${mqtt.client.id}")
    private String clientId;

    @Value("${mqtt.username}")
    private String mqttUsername;

    @Value("${mqtt.password}")
    private String mqttPassword;

    @Bean
    public MqttClient mqttClient() throws MqttException {
        MqttClient client = new MqttClient(brokerUrl, clientId);
```

```
        MqttConnectOptions options = new MqttConnectOptions();
        options.setUserName(mqttUsername);
        options.setPassword(mqttPassword.toCharArray());

        client.connect(options);
        return client;
    }

    // Для відправки повідомлень
    @Bean
    public MqttMessage mqttMessage() {
        return new MqttMessage();
    }
}
```

src/main/java/com/example/monitorin/mqtt/MqttService.java:

```
package com.example.monitoring.mqtt;

import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttMessage;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class MqttService {

    private final MqttClient mqttClient;
```

```
    @Autowired
    public MqttService(MqttClient mqttClient) {
        this.mqttClient = mqttClient;
    }

    public void subscribeToTopic(String topic) throws Exception {
        mqttClient.subscribe(topic, (topic1, msg) -> {
            // Обробка отриманого повідомлення
            String message = new String(msg.getPayload());
            System.out.println("Received message: " + message);

            // Тут можна викликати логіку для обробки даних та збереження в базу даних
        });
    }

    public void sendMessage(String topic, String message) throws Exception {
        MqttMessage mqttMessage = new MqttMessage(message.getBytes());
        mqttClient.publish(topic, mqttMessage);
    }
}
```

src/main/java/com/example/monitorin/mqtt/MqttSubscriber.java:

```
package com.example.monitoring;

import com.example.monitoring.mqtt.MqttService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;
```

```
@Component
public class MqttSubscriber implements CommandLineRunner {

    @Autowired
    private MqttService mqttService;

    @Override
    public void run(String... args) throws Exception {
        // Підписка на тему "sensor/data"
        mqttService.subscribeToTopic("sensor/data");
    }
}
```

src/main/java/com/example/monitoring/repository/DeviceRepository.java:

```
package com.example.monitoring.repository;

import com.example.monitoring.model.Device;
import org.springframework.data.jpa.repository.JpaRepository;

public interface DeviceRepository extends JpaRepository<Device, Long> {
}
```

src/main/java/com/example/monitoring/repository/MeasurementRepository.java:

```
package com.example.monitoring.repository;

import com.example.monitoring.model.Measurement;
```

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface MeasurementRepository extends
JpaRepository<Measurement, Integer> {
}
```

src/main/java/com/example/monitoring/repository/DebiceService.java

```
package com.example.monitoring.service;

import com.example.monitoring.model.Device;
import com.example.monitoring.repository.DeviceRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class DeviceService {

    @Autowired
    private DeviceRepository deviceRepository;

    public List<Device> getAllDevices() {
        return deviceRepository.findAll();
    }
    public Device saveDevice(Device device) {
        return deviceRepository.save(device);
    }
}
```

src/main/java/com/example/monitoring/repository/MeasurementService.java

```
package com.example.monitoring.service;

import com.example.monitoring.model.Measurement;
import com.example.monitoring.repository.MeasurementRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class MeasurementService {

    @Autowired
    private MeasurementRepository measurementRepository;

    public Measurement saveMeasurement(Measurement measurement) {
        return measurementRepository.save(measurement);
    }
}
```