МІНІСТЕРСТВО ОСВІТИ НАУКИ УКРАЇНИ
ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО «ПРИВАТНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД «ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ»

Кафедра інформаційних технологій

ДО ЗАХИСТУ ДОПУЩЕНА

Завідувач кафедри,
д.е.н., доц.
_____ С.І. Левицький

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

РОЗРОБКА МОБІЛЬНОГО ІНТЕРФЕЙСУ УПРАВЛІННЯ ІОТ ПРИСТРОЯМИ

Виконав

ст. гр. ІПЗ – 139 _____ М.В. Чистяков

Керівник

к.т.н., доц. _____ О.А. Жеребцов

Запоріжжя

2023

ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО «ПРИВАТНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД «ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ»

Кафедра інформаційних технологій

ЗАТВЕРДЖУЮ

Завідувач кафедри,
д.е.н., доц.

_____ С.І. Левицький

___.___._____ р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ

студенту гр. ІПЗ-139 ,

спеціальності 121 - «Інженерія програмного забезпечення»

Чистякову Максиму Вадимовичу

1. Тема: Розробка мобільного інтерфейсу управління IoT пристроями затверджена наказом № 02-09 від 27 січня 2023 р.

2. Термін здачі студентом закінченої роботи: 12 червня 2023 р.

3. Перелік питань, що підлягають розробці:

1. Провести огляд літератури, що присвячена тематиці досліджень.

2. Виконати огляд та порівняння аналогів систем з управління IoT пристроями.

3. Провести огляд та порівняння мобільного додатку та мобільної версії сайту.

4. Здійснити обґрунтований вибір використання стеку технологій для розробки проекту.

5. Здійснити проектування розробляємої системи.

6. Здійснити проектування функціональних можливостей користувачів системи.

7. Здійснити програмування запропонованої системи.

8. Оформити звіт за результатами роботи.

4. Календарний графік підготовки кваліфікаційної бакалаврської роботи.

№ етапу	Зміст	Терміни виконання	Готовність по графіку %, підпис керівника	Підпис керівника про повну готовність етапу, дата
1	Формулювання (корегування) теми кваліфікаційної бакалаврської роботи та збір практичного матеріалу за темою	16.01.23-11.02.23		
2	**І атестація** І розділ кваліфікаційної бакалаврської роботи	27.03.23-31.03.23		
3	**ІІ атестація** ІІ розділ кваліфікаційної бакалаврської роботи	24.04.23-28.04.23		
4	**ІІІ атестація** ІІІ розділ кваліфікаційної бакалаврської роботи, висновки та рекомендації, додатки, реферат	22.05.23-26.05.23		
5	Перевірка кваліфікаційної бакалаврської роботи на оригінальність	15.05.23-12.06.23		
6	Доопрацювання кваліфікаційної бакалаврської роботи, підготовка презентації, отримання відгуку керівника і рецензії	29.05.23-12.06.23		
7	**Попередній захист кваліфікаційної бакалаврської роботи**	**12.06.23-18.06.23**		
8	Подача кваліфікаційної бакалаврської роботи на кафедру	за 3 дні до захисту		
9	**Захист кваліфікаційної бакалаврської роботи**	**19.06.23-24.06.23**		

Дата видачі завдання: ___.___._____ р.

Керівник кваліфікаційної

бакалаврської роботи _____ О.А. Жеребцов

Завдання отримав до виконання _____ М.В. Чистяков

РЕФЕРАТ

Кваліфікаційна бакалаврська робота містить 100 сторінок, 47 рисунків, 20 використаних джерел.

Метою розробки є надання комплексної та зручної платформи для реєстрації пристроїв, моніторингу, контролю та аналізу даних.

Об’єктом дослідження є система для управління пристроями IoT.

Предметом дослідження є розробка інтерфейсу взаємодії користувача із системою.

Здійснено детальний огляд предметної області, та сучасних аналогів, таких як Amazon Web Services IoT Platform, Microsoft Azure IoT та IBM Watson IoT. Проект реалізовано у виді веб-сайту, з використанням мови програмування TypeScript за допомогою фреймворку React та бібліотеки для контролю стану Redux Toolkit.

Отриманий програмний продукт є простим та зручним у використанні, наділений інтуїтивно-зрозумілим дизайном та є гнучким у налаштуванні.

TYPESCRIPT, REACT, INTERNET OF THINGS, DEVICE MANAGAMENT, USER INTERFACE, DATA ANALYSIS

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

(Спочатку українські потім англійські терміни - відсортуй)

Скорочення	Повна назва	Пояснення/переклад
БД	База даних	Сукупність даних, організованих відповідно до концепції, яка описує характеристику цих даних і взаємозв'язки між їх елементами
API	Application Programming Interface	Прикладний програмний інтерфейс
IoT	Internet of things	Концепція мережі, що складається із взаємозв'язаних фізичних пристроїв, які мають вбудовані давачі, а також програмне забезпечення, що дозволяє здійснювати передачу і обмін даними між фізичним світом і комп'ютерними системами
ООП	Об’єктно-орієнтоване програмування	Одна з парадигм програмування
СУБД	Система управління базами даних	Поєднання ПЗ та лінгвістичних інструментів для загальних або спеціальних цілей, що забезпечують управління створенням та використанням баз даних
DOM	Document Object Model	Об'єктна модель документа
IT	Information Technology	Сукупність методів і засобів, що використовуються для збору, зберігання, обробки і поширення інформації.
JS	JavaScript	Динамічна, об'єктно-орієнтована прототипна мова програмування; реалізація стандарту ECMAScript
TS	TypeScript	Мова програмування, що компілюється в JavaScript і є більш дисциплінованою та строгою, ніж динамічно типізований JS.
JSON	JavaScript Object Notation	Текстовий формат обміну даними на основі JavaScript
JSX	JavaScript Syntax Extension	Розширення REST до синтаксису мови JavaScript, яке надає спосіб структурування відтворення

		компонентів
HTTP	Hyper Text Transfer Protocol	Протокол передачі гіпертекстових документів – протокол передачі даних, що використовується в комп'ютерних мережах
REST	Representational State Transfer	Архітектурний стиль взаємодії компонентів розподіленого застосування в мережі
REST API	Representational State Transfer та Application programming interface	API, який відповідає принципам дизайну REST або архітектурному стилю передачі репрезентативного стану
SQL	Structured query language	Мова структурованих запитів
UI	User Interface	Користувацький інтерфейс
URI	Uniform Resource Locator	Уніфікований локатор ресурсів або адреса ресурсу
URL	Uniform Resource Locator	Уніфікований локатор ресурсів або адреса ресурсу
Веб	Web	Інтернет-простір
ПЗ	Програмне забезпечення	Сукупність програм системи оброблення інформації та програмних документів
Фреймворк	Framework	Програмне середовище, яке спрощує та прискорює створення програмного забезпечення
CSS	Cascading Style Sheets	Каскадні таблиці стилів
HTML	HyperText Markup Language	Стандартизована мова розмітки документів у Всесвітньому павутинні
SPA	Single Page Application	Програма, яка працює всередині браузера і не потребує перезавантаження сторінок під час його завантаження
SEO	Search Engine Optimization	Комплекс заходів щодо збільшення видимості сайту в пошукових системах за цільовими пошуковими запитами
AOT	Ahead-of-time	Варіант компіляції програми, яка виконується один раз при складанні програми

ВСТУП

(Не бачу посилань – треба додати перед крапкою кінця речення у квадратних дужках [1].)

Швидке зростання Інтернету речей (IoT) революціонізувало спосіб нашої взаємодії з навколишнім середовищем. Пристрої IoT, від розумної побутової техніки до промислових датчиків, генерують величезні обсяги даних, якими потрібно ефективно керувати та контролювати. У контексті цього динамічного ландшафту існує нагальна потреба в ефективних рішеннях для управління, які можуть впоратися зі складністю мереж пристроїв IoT. Цей дипломний проект спрямований на розробку мобільного інтерфейсу для керування пристроями IoT, що забезпечує централізовану платформу для моніторингу, контролю та аналізу.

Він слугуватиме зручним інтерфейсом, що дозволить адміністраторам і кінцевим користувачам взаємодіяти зі своїми пристроями IoT і отримувати доступ до важливої інформації в режимі реального часу. Забезпечуючи безперебійне підключення та візуалізацію даних, веб-сайт дозволить користувачам приймати обґрунтовані рішення, оптимізувати роботу пристрою та забезпечувати безпеку їхньої екосистеми IoT.

Основні функції інтерфейсу включатимуть реєстрацію пристроїв, візуалізацію даних за допомогою інтерактивних інформаційних панелей, дистанційне керування та налаштування пристроїв, оповіщення та сповіщення, а також аналіз даних.

Тема розробки інтерфейсу для управління пристроями IoT є дуже актуальною в сучасному технологічному ландшафті. З широким впровадженням пристроїв IoT у різних галузях промисловості потреба в ефективному управлінні та контролі цих пристроїв стала першочерговою. Оскільки екосистеми Інтернету речей стають дедалі складнішими, організації та окремі особи стикаються з проблемами ефективного моніторингу та налаштування своїх пристроїв. Спеціальний інтерфейс для керування пристроями IoT може вирішити ці проблеми та забезпечити централізовану

платформу для безперебійного контролю та аналізу.

Існує кілька критичних проблем, пов’язаних із керуванням пристроями Інтернету речей, на вирішення яких спрямований цей проект. Ці проблеми включають:

- Масштабованість: зі збільшенням кількості пристроїв Інтернету речей у мережі управління та координація їх роботи стає складнішою. Метою веб-сайту є надання масштабованих рішень для широкомасштабного розгортання пристроїв IoT.
- Управління та аналіз даних. Пристрої IoT генерують величезні обсяги даних, і отримання цінної інформації з цих даних має вирішальне значення. Даний інтерфейс запропонує можливості візуалізації та аналітики даних, що дозволить користувачам ефективно контролювати та аналізувати дані пристрою.

Метою цього дослідження є розробка інтерфейсу, який вирішує проблеми та складності, пов’язані з керуванням пристроями IoT. Метою розробки є надання комплексної та зручної платформи для реєстрації пристроїв, моніторингу, контролю та аналізу даних. Дослідження спрямоване на використання сучасних технологій веб-розробки, фреймворків і найкращих практик для створення ефективного та масштабованого рішення.

Об’єктом дослідження є розробка інтерфейсу для управління пристроями IoT. Основна увага буде зосереджена на розробці та впровадженні функцій, які полегшують реєстрацію пристрою, дистанційне керування, візуалізацію даних та аналітику.

Предметом дослідження є сам процес розробки інтерфейсу. Це включає вибір відповідних фреймворків і технологій веб-розробки, розробку інтуїтивно зрозумілого інтерфейсу користувача, та забезпечення масштабованості та продуктивності системи.

РОЗДІЛ 1
ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА АНАЛОГІВ

IoT означає Інтернет речей. Це концепція, яка стосується взаємозв'язку фізичних пристроїв, транспортних засобів, будівель та інших об'єктів із вбудованими датчиками, програмним забезпеченням і мережевим підключенням, що дозволяє їм збирати та обмінюватися даними. Простіше кажучи, пристрої IoT – це повсякденні об'єкти, які підключені до Інтернету та можуть спілкуватися з іншими пристроями для виконання завдань без втручання людини.

Пристрої IoT бувають різних форм, включаючи розумну побутову техніку, як термостати, розумні колонки, камери безпеки та системи освітлення. Інші приклади пристроїв IoT включають носимі пристрої, такі як фітнес-трекери та розумні годинники, промислові машини, пристрої для охорони здоров'я та навіть автомобілі з підключенням до Інтернету. Ці пристрої зазвичай мають датчики та мікропроцесори, які збирають дані, обробляють їх і обмінюються даними з іншими пристроями чи хмарою.

Пристрої IoT розроблені як автономні та вимагають мінімального втручання людини для функціонування. Ними можна дистанційно керувати та контролювати їх за допомогою смартфонів, планшетів або комп'ютерів, що робить їх надзвичайно зручними та зручними для користувача. Дані, зібрані пристроями IoT, можна аналізувати, щоб отримати цінну інформацію, яку можна використовувати для оптимізації операцій, підвищення ефективності та покращення досвіду клієнтів.

Для управління та контролю пристроїв IoT використовуються платформи керування пристроями IoT. Ці платформи забезпечують централізовану інформаційну панель, яка дозволяє користувачам відстежувати та контролювати свої пристрої IoT. Вони також надають інструменти для ініціалізації пристрою, оновлення мікропрограми, керування

безпекою та аналізу даних.

1.1 Важливість IoT і мобільних технологій у сучасному цифровому ландшафті

У сучасному цифровому ландшафті Інтернет речей і мобільні технології стали невід'ємною частиною нашого повсякденного життя. Вони революціонізували спосіб нашої взаємодії з технологіями та навколишнім світом. IoT дозволив нам підключатися та спілкуватися з широким спектром пристроїв, від побутової техніки до промислового обладнання, забезпечуючи більшу ефективність, автоматизацію та контроль.

Подібним чином мобільні технології змінили спосіб, у який ми отримуємо доступ до інформації та споживаємо її, спілкуємося з іншими та виконуємо завдання в дорозі. Від смартфонів і планшетів до пристроїв, що носяться, мобільні технології стали повсюдними у нашому житті, надаючи нам миттєвий доступ до інформації, розваг і послуг.

Вплив IoT і мобільних технологій не обмежується додатками для споживачів. Вони також трансформували різні галузі та сектори, зокрема охорону здоров'я, транспорт, виробництво та сільське господарство. Ці технології дозволили розробити нові рішення та послуги, такі як віддалений моніторинг пацієнтів, розумні транспортні системи, прогнозне технічне обслуговування та точне землеробство.

Заглядаючи вперед, потенціал IoT і мобільних технологій величезний. Оскільки впровадження цих технологій продовжує зростати, ми можемо очікувати подальшого прогресу в таких сферах, як штучний інтелект, машинне навчання та периферійні обчислення. Поєднання цих технологій має потенціал для трансформації галузей і секторів у новий і захоплюючий спосіб, створюючи нові можливості для інновацій, зростання та розвитку.

Доступно багато платформ керування пристроями IoT, кожна зі своїми унікальними функціями та можливостями. Ці платформи розроблені для

роботи з різними пристроями, датчиками та шлюзами Інтернету речей і пропонують низку функцій безпеки та відповідності, щоб гарантувати безпечну передачу та зберігання даних.

Тож давайте більш детально розберемо існуючі платформи з управління IoT пристроями.

1.2 Огляд існуючих платформ з управління IoT пристроями

Оскільки Інтернет речей (IoT) продовжує розвиватися, кількість пристроїв і датчиків IoT стрімко зростає. Керування цими пристроями може бути складним завданням, особливо якщо вони розподілені в кількох місцях і мережах. Платформи керування пристроями IoT забезпечують вирішення цієї проблеми, дозволяючи користувачам відстежувати та контролювати свої пристрої IoT з єдиної централізованої інформаційної панелі.

Ці платформи пропонують низку інструментів і послуг, зокрема ініціалізацію пристроїв, керування безпекою, оновлення вбудованого програмного забезпечення та аналітику даних. Вони також надають ряд функцій, які допомагають забезпечити надійність, масштабованість і безпеку систем IoT.

Уснує велика кількість доступних наразі платформ керування пристроями Інтернету речей, їх усі оглядати немає сенсу, тож давайте розглянемо найпопулярніші. За інтернет-ресурсом «euristiq», найпоширенішими платформами є:

- Amazon Web Services IoT Platform;
- Microsoft Azure IoT;
- IBM Watson IoT [1].

1.2.1 Amazon Web Services IoT Platform

Платформа Інтернету речей Amazon Web Services (AWS) – це хмарна

платформа, яка надає набір інструментів і послуг для підключення, керування та захисту пристроїв Інтернету речей у масштабі. Платформа AWS IoT дозволяє користувачам безпечно та легко підключати мільярди пристроїв і керувати ними, а також збирати, зберігати та аналізувати дані пристроїв.

Платформа AWS IoT підтримує кілька протоколів зв'язку, таких як MQTT, HTTP та WebSockets, і надає розробникам SDK пристрою для створення додатків IoT, які можуть спілкуватися з хмарою. Платформа також підтримує інтеграцію з іншими службами AWS, такими як AWS Lambda, Amazon S3 і Amazon Kinesis, щоб забезпечити подальшу обробку, зберігання та аналіз зібраних даних. [5]

Платформа AWS IoT надає функцію керування пристроями, яка дозволяє користувачам керувати життєвим циклом своїх пристроїв, включаючи реєстрацію, оновлення та моніторинг стану пристроїв. Платформа також підтримує бездротові оновлення (OTA) для мікропрограми пристроїв, що дозволяє безперешкодно оновлювати та обслуговувати пристрої в польових умовах.

Крім того, платформа AWS IoT пропонує такі функції безпеки, як автентифікація, авторизація та шифрування, щоб гарантувати безпеку пристроїв і захист даних.

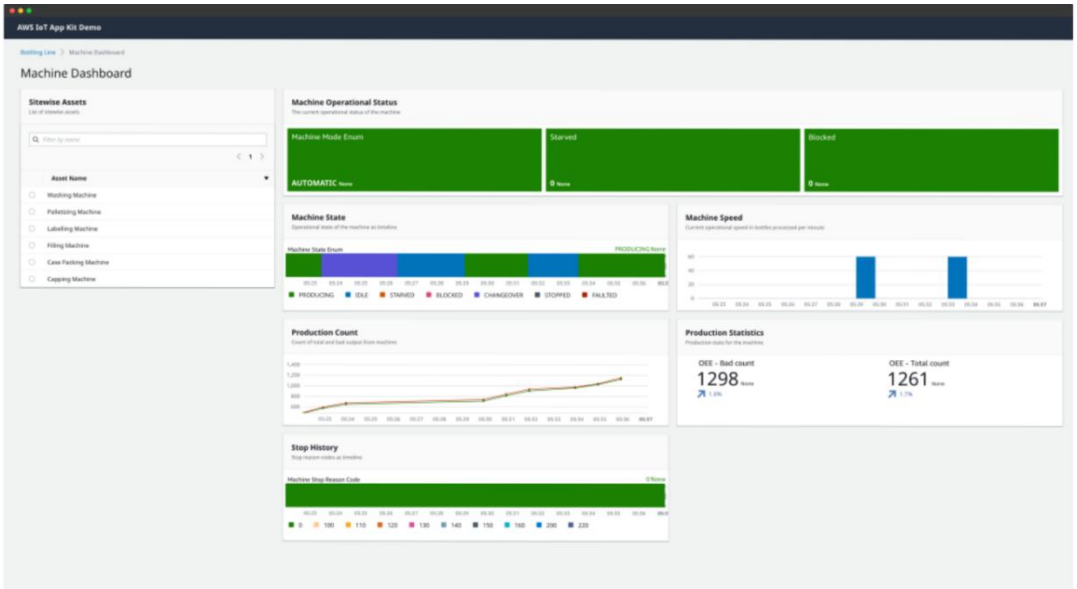

Рисунок 1.1 – Зовнішній вигляд статистичних компонентів AWS [2]

Використання сервісу AWS IoT має такі переваги:

- Масштабованість: Платформа AWS IoT має високу масштабованість, що означає, що вона може підтримувати мільйони пристроїв і трильйони повідомлень.
- Безпека: Платформа AWS IoT пропонує потужні функції безпеки, такі як шифрування, автентифікація пристрою та безпечний контроль доступу, щоб забезпечити безпеку пристроїв і даних IoT.
- Інтеграція: платформа інтегрується з іншими службами AWS, такими як Lambda, DynamoDB і S3, що полегшує створення додатків IoT і керування ними.
- Обробка даних у режимі реального часу: Платформа AWS IoT забезпечує обробку та аналітику даних у режимі реального часу, що може допомогти компаніям приймати зважені рішення та оптимізувати свою діяльність.
- Рентабельність: Платформа AWS IoT пропонує платіжну модель ціноутворення, що означає, що компанії платять лише за послуги, якими вони користуються, що робить її економічно ефективним рішенням.

До недоліків у використанні даного сервісу можна віднести:

- Складність: Платформа AWS IoT може бути складною для налаштування та використання, особливо для компаній, які не мають попереднього досвіду роботи з послугами AWS.
- Технічні знання: Платформа AWS IoT потребує технічних знань для впровадження та підтримки, що може бути проблемою для компаній з обмеженими IT-ресурсами.
- Прив’язка до постачальника: Платформа AWS IoT є власною платформою, що означає, що підприємства, які її використовують, можуть бути обмежені у використанні послуг AWS і можуть мати обмежену гнучкість переходу до інших постачальників.
- Обмежена підтримка пристроїв, що не належать до AWS: в основному підтримує пристрої AWS, що може обмежити корисність платформи для

компаній, які використовують пристрої, що не належать до AWS.

- Залежність від підключення до Інтернету: для роботи платформи AWS IoT потрібне підключення до Інтернету, що може бути недоліком для компаній, які працюють у регіонах із поганим підключенням.

Загалом платформа AWS IoT забезпечує масштабоване та надійне рішення для керування пристроями та даними IoT і використовується в багатьох галузях промисловості, включаючи виробництво, транспорт, охорону здоров'я тощо.

1.2.2 Microsoft Azure IoT

Microsoft Azure IoT – це хмарна платформа, яка надає низку послуг для розробки та керування рішеннями IoT. Платформа пропонує набір послуг, які дозволяють розробляти програми, аналізувати дані та керувати пристроями. Azure IoT надає масштабовану та безпечну інфраструктуру для керування широкомасштабними розгортаннями IoT у різних галузях, таких як виробництво, транспорт і охорона здоров'я. [6]

Однією з ключових особливостей Azure IoT є його здатність обробляти й аналізувати великі обсяги даних, створених пристроями IoT. Платформа використовує вдосконалені інструменти аналітики, щоб отримати інформацію з цих даних і надати зворотний зв'язок у реальному часі для покращення бізнес-операцій. Azure IoT також пропонує ряд функцій безпеки, включаючи автентифікацію пристрою, керування ідентифікацією та виявлення загроз, щоб забезпечити безпеку та конфіденційність даних, що передаються між пристроями та хмарою.

Azure IoT підтримує різноманітні мови програмування, протоколи та операційні системи, що робить його гнучкою платформою для розробки рішень IoT. Він також надає низку готових рішень і шаблонів для прискорення процесу розробки та скорочення часу виходу на ринок. [6]

Однак одним недоліком Azure IoT є його складність. Платформа має круту криву навчання та може потребувати значного досвіду для ефективного використання. Крім того, деякі користувачі можуть вважати структуру ціноутворення складною та важкою для розуміння.

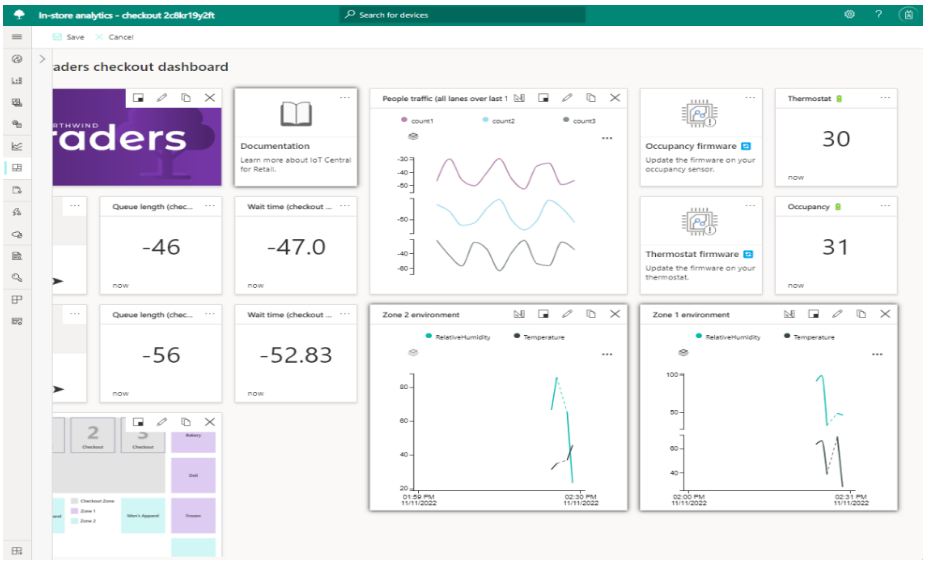

Рисунок 1.2 – Вигляд статистичних компонентів Azure IoT [3]

Переваги використання платформи Microsoft Azure IoT;

- Інтеграція з іншими службами Microsoft, такими як хмара Azure, машинне навчання та аналітика;
- Високий рівень безпеки та відповідність галузевим стандартам;
- Підтримує кілька протоколів і пристроїв, у тому числі з обмеженою обчислювальною потужністю;
- Пропонує інструменти аналізу та візуалізації даних у реальному часі;
- Має гнучкі параметри ціноутворення залежно від обсягу оброблених і збережених даних.

Недоліки використання платформи Microsoft Azure IoT:

- Потрібні попередні знання про екосистему Microsoft Azure та її служби;
- Обмежена підтримка пристроїв і протоколів сторонніх виробників порівняно з іншими платформами IoT;

- Вища вартість порівняно з деякими іншими платформами IoT.

1.2.3 IBM Watson IoT

IBM Watson IoT Platform – це платформа IoT, яка надає хмарне рішення для підключення та керування пристроями IoT. Це дозволяє користувачам безпечно підключатися, керувати та аналізувати дані з пристроїв IoT у режимі реального часу.

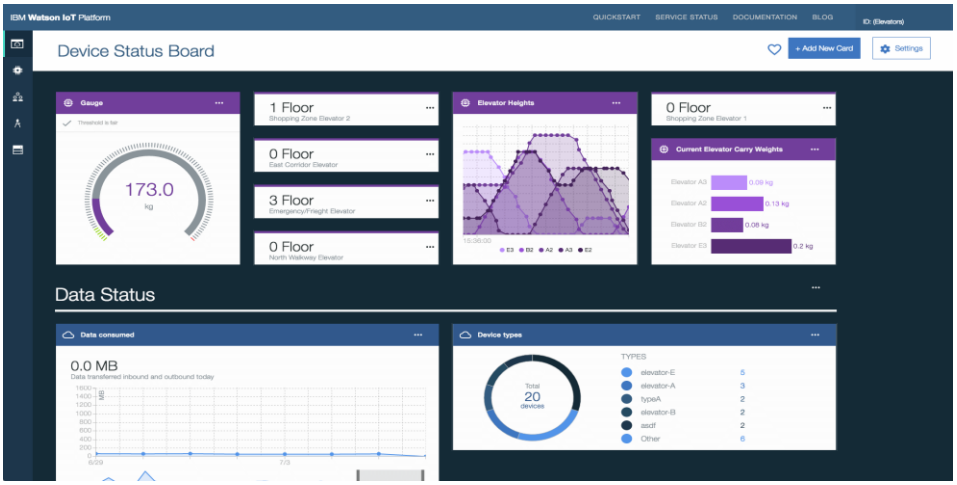

Рисунок 1.3 – Зовнішній вигляд статистичних компонентів IBM IoT [4]

Платформа пропонує ряд функцій, таких як керування пристроями, аналіз даних і можливості машинного навчання. Він також підтримує широкий спектр пристроїв, протоколів і операційних систем, дозволяючи користувачам підключати та керувати пристроями від різних виробників і операційних систем.

IBM Watson IoT Platform також пропонує ряд інструментів і послуг для розробки та розгортання додатків IoT. Він надає розробникам доступ до набору API та SDK, які дозволяють їм швидко розробляти та розгортати програми IoT.

Платформа також включає такі функції безпеки, як автентифікація та контроль доступу, що гарантує, що пристрої та дані IoT захищені від кіберзагроз. Крім того, він підтримує розгортання з кількома клієнтами,

дозволяючи користувачам створювати та керувати кількома додатками та клієнтами IoT.

Серед переваг платформи IBM Watson IoT:

- Масштабованість: платформа розроблена для масштабування, щоб задовольнити потреби підприємств будь-якого розміру.
- Безпека: платформа забезпечує наскрізну безпеку від пристрою до хмари.
- Аналітика: IBM Watson IoT надає потужні аналітичні можливості, що дозволяє користувачам отримувати аналіз даних, створених пристроями IoT.
- Інтеграція: платформа легко інтегрується з іншими продуктами IBM, а також сторонніми рішеннями.
- Гнучкість: IBM Watson IoT можна використовувати в різних галузях і сферах застосування, включаючи виробництво, транспорт, охорону здоров'я та розумні міста.

До недоліків платформи IBM Watson IoT можна віднести:

- Складність: платформу може бути складною для налаштування та конфігурації, що вимагає технічних знань.
- Вартість: платформа може бути дорогою, особливо для малих підприємств або організацій з обмеженим бюджетом.
- Крива навчання: користувачам може знадобитися витратити час на вивчення платформи та її можливостей, перш ніж вони зможуть повністю використовувати її функції.

Підсумовуючи, усі три платформи, Microsoft Azure IoT, IBM Watson IoT і Amazon Web Services (AWS) IoT, пропонують повний набір інструментів і послуг для керування та аналізу пристроїв і даних IoT. Кожна платформа має свої сильні та слабкі сторони, і найкращий вибір для конкретного проекту може залежати від таких факторів, як розмір і складність розгортання, необхідний рівень безпеки та наявний стек технологій в організації.

Microsoft Azure IoT пропонує потужну інтеграцію з іншими продуктами та службами Microsoft, що робить його привабливим варіантом для організацій, які вже використовують технології Microsoft. Його надійні функції безпеки та підтримка відкритих стандартів роблять його надійною та гнучкою платформою для керування пристроями IoT.

IBM Watson IoT відома своїми розширеними можливостями аналітики завдяки інтеграції з платформою IBM Watson AI. Це робить його сильним вибором для організацій, які прагнуть використовувати AI та машинне навчання у своїх розгортаннях IoT. Однак його складна модель ціноутворення та відносно висока вартість можуть зробити його менш доступним для невеликих організацій.

Amazon Web Services (AWS) IoT – це масштабована та надійна платформа з широким спектром послуг та інтеграцій. Це популярний вибір серед розробників і організацій, які шукають недороге та гнучке рішення для керування пристроями IoT. Однак його складність може вимагати крутішої кривої навчання для новачків у AWS.

1.3 Порівняльний аналіз мобільної версії сайту та мобільного додатку

Мобільні додатки та мобільні веб-сайти – це два різні підходи доступу до вмісту на мобільному пристрої. Хоча обидва надають користувачам доступ до інформації, вони мають чіткі відмінності, які можуть вплинути на взаємодію з користувачем і функціональність. У цьому порівняльному аналізі ми дослідимо відмінності між мобільними додатками та мобільними веб-сайтами.

Мобільні веб-сайти створюються з головною метою створення адаптивного дизайну, який відповідає екрану мобільного пристрою. Мобільні веб-сайти – це, по суті, веб-сайти, оптимізовані для мобільних пристроїв. Доступ до них здійснюється через мобільний браузер і для роботи потрібне підключення до Інтернету. Мобільні веб-сайти пропонують перевагу

доступності, оскільки користувачі можуть отримати доступ до сайту з будь-якого мобільного пристрою з підключенням до Інтернету. Однак мобільні веб-сайти можуть мати обмежені функції та функції порівняно з мобільними програмами.

Мобільні додатки розроблені спеціально для мобільних пристроїв і встановлюються на пристрої користувача. Вони розробляються з використанням рідного коду або за допомогою кросплатформних фреймворків розробки. Доступ до мобільних програм здійснюється за допомогою піктограми на головному екрані пристрою, і для роботи не потрібне підключення до Інтернету. Мобільні програми забезпечують більш захоплюючий і привабливий досвід користувача та можуть запропонувати більше можливостей і функцій порівняно з мобільними веб-сайтами. Однак мобільні програми можуть бути дорожчими для розробки та обслуговування порівняно з мобільними веб-сайтами. [7]

Порівняльна характеристика мобільної версії сайту та мобільного додатку:

- Доступність: мобільні веб-сайти доступніші, ніж мобільні програми, оскільки до них можна отримати доступ із будь-якого пристрою, підключеного до Інтернету, тоді як мобільні програми потрібно завантажити та встановити на пристрій, перш ніж ними можна буде користуватися.
- Функціональність: мобільні програми можуть запропонувати більше функціональних можливостей, ніж мобільні веб-сайти, оскільки вони мають доступ до специфічних для пристрою функцій, таких як GPS, камера, мікрофон і push-сповіщення.
- Взаємодія з користувачем: Мобільні програми можуть забезпечувати більш захоплюючий і привабливий досвід, ніж мобільні веб-сайти, завдяки їхній здатності використовувати власні функції пристрою та надавати персоналізований вміст.

- Час і вартість розробки: розробка мобільного веб-сайту зазвичай є

швидшою та дешевшою, ніж розробка мобільного додатку, оскільки мобільні веб-сайти можна створювати за допомогою технологій веб-розробки, таких як HTML, CSS і JavaScript, тоді як мобільні додатки потребують вбудованої розробки для кожної платформи.

- Обслуговування: мобільні веб-сайти легше обслуговувати, ніж мобільні додатки, оскільки зміни, внесені до веб-сайту, одразу видно всім користувачам, тоді як оновлення мобільних додатків вимагають від користувачів завантажити та встановити останню версію.
- Монетизація: мобільні програми пропонують більше варіантів монетизації, ніж мобільні веб-сайти, наприклад покупки в програмі, підписки та реклама. Мобільні веб-сайти здебільшого покладаються на доходи від реклами.
- Виявленість: мобільні програми може бути важко знайти, і вони вимагають маркетингових зусиль, щоб охопити користувачів, тоді як мобільні веб-сайти можна знайти за допомогою пошукових систем і соціальних мереж.
- Офлайн-функція: мобільні програми можуть запропонувати офлайн-функцію, тоді як мобільні веб-сайти потребують стабільного підключення до Інтернету для належної роботи.
- Push-сповіщення: мобільні програми можуть надсилати push-сповіщення користувачам, навіть якщо програму закрито, що може бути корисним для сповіщень або нагадувань. Мобільні веб-сайти, з іншого боку, не можуть надсилати push-повідомлення.
- Інтеграція з функціями пристрою: Мобільні програми можуть інтегруватися з такими функціями пристрою, як камера, GPS і мікрофон, що може забезпечити більш багатий і персоналізований досвід користувача. Мобільні веб-сайти мають обмежений доступ до функцій пристрою.
- Життєвий цикл – мобільні веб-сайти не можна видалити: середній термін придатності програми досить короткий, згідно з деякими дослідженнями, менше 30 днів, тому, якщо ваша програма не є справді унікальною та/або корисною (в ідеалі, обидва), сумнівно, як довго він

протримається на пристрої користувача. З іншого боку, мобільні веб-сайти завжди доступні для користувачів, щоб на них повертатися.

- Мобільний веб-сайт може бути додатком: як і звичайний веб-сайт, мобільні веб-сайти можна розробляти як веб-програми, керовані базою даних, які діють дуже схоже на власні програми. Мобільний веб-додаток може бути практичною альтернативою нативній розробці додатків.
- Розповсюдження в магазині додатків: мобільні додатки можна розповсюджувати через магазини додатків, такі як Apple App Store і Google Play Store, що може забезпечити більшу потенційну аудиторію та кращу видимість. Мобільні веб-сайти покладаються на те, що користувачі знаходять сайт і створюють закладки.
- Взаємодія з користувачем: мобільні додатки можуть забезпечити більш спрощений і захоплюючий досвід користувача, з меншою кількістю відволікань і швидшим часом завантаження порівняно з мобільними веб-сайтами. Мобільні веб-сайти можуть завантажуватися повільніше та можуть мати проблеми з інтерфейсом користувача на менших екранах.
- Технічне обслуговування: Мобільні програми потребують постійного обслуговування та оновлень, що може зайняти багато часу та коштів, в свою чергу мобільні веб-сайти можна легше оновлювати та вони потребують менше обслуговування.

Вибираючи між мобільним веб-сайтом і мобільним додатком, важливо враховувати конкретні цілі проекту, цільову аудиторію та бюджет. Мобільні веб-сайти є чудовим варіантом для надання доступного вмісту широкій аудиторії, тоді як мобільні програми забезпечують більш захоплюючий і привабливий досвід із розширеними функціями. Зрештою, вибір між мобільним веб-сайтом і мобільним додатком залежатиме від конкретних потреб і цілей проекту.

1.4 Аргументація вибору мобільної версії сайту

У попередньому підрозділі було порівняно мобільну версію сайту та мобільний додаток за певними факторами. Як підсумок порівняльної характеристики, було обрано розробляти мобільну версію сайту (адаптувати сайт під різні розміри екранів – як планшети, так і мобільні телефони).

Ключовими критеріями для вибору саме мобільної версії сайту є:

- Безпосередність – мобільні веб-сайти доступні миттєво: мобільний веб-сайт миттєво доступний для користувачів через браузер на різних пристроях (iPhone, Android, BlackBerry тощо). З іншого боку, програми вимагають, щоб користувач спочатку завантажив і встановив програму з ринку програм, перш ніж вміст або програму можна буде переглянути – це суттєва перешкода між початковою взаємодією та дією/конверсією.
- Сумісність – мобільні веб-сайти сумісні на різних пристроях: один мобільний веб-сайт може охоплювати користувачів із багатьох різних типів мобільних пристроїв, тоді як рідні програми потребують розробки окремої версії для кожного типу пристроїв. Крім того, URL-адреси мобільних веб-сайтів легко інтегруються в інші мобільні технології, такі як SMS, QR-коди та зв’язок ближнього поля (NFC).
- Можливість оновлення – мобільні веб-сайти можна оновлювати миттєво: мобільний веб-сайт є набагато динамічнішим, ніж програма, з точки зору чистої гнучкості оновлення вмісту. Якщо ви хочете змінити дизайн або вміст веб-сайту для мобільних пристроїв, просто опублікуйте редагування один раз, і зміни одразу стануть видимими; оновлення програми, з іншого боку, потребує надсилання оновлень користувачам, які потім потрібно завантажити, щоб оновити програму на кожному типі пристрою.
- Можливість пошуку – мобільні веб-сайти можна легко знайти: користувачам набагато легше знайти мобільні веб-сайти, оскільки їхні сторінки можуть відображатися в результатах пошуку та перераховуватися в галузевих каталогах, завдяки чому кваліфіковані відвідувачі можуть легко вас знайти. Найважливіше те, що відвідувачі вашого звичайного веб-сайту можуть автоматично спрямовуватися на ваш мобільний сайт, коли вони користуються

портативним комп’ютером (за допомогою визначення пристрою). Навпаки, видимість програм значною мірою обмежена магазинами програм виробника.

- Можливість спільного використання – видавці та користувачі можуть легко ділитися веб-сайтами для мобільних пристроїв: URL-адресами веб-сайтів для мобільних пристроїв легко ділитися між користувачами за допомогою простого посилання (наприклад, у електронному чи текстовому повідомленні, публікації на Facebook чи Twitter). Видавці можуть легко спрямовувати користувачів на мобільний веб-сайт із блогу чи веб-сайту чи навіть у друкованому вигляді. Додатком просто неможливо поділитися таким чином.

- Охоплення – мобільні веб-сайти мають ширше охоплення: оскільки мобільний веб-сайт доступний на різних платформах і ним можна легко поділитися між користувачами та пошуковими системами, він має набагато більші можливості охоплення, ніж рідна програма.

- Життєвий цикл – веб-сайти для мобільних пристроїв не можна видалити: середній термін придатності програми досить короткий, згідно з деякими дослідженнями, менше 30 днів, тому, якщо ваша програма не є справді унікальною та/або корисною (в ідеалі, обидва), сумнівно, як довго він протримається на пристрої користувача. З іншого боку, мобільні веб-сайти завжди доступні для користувачів, щоб на них повертатися.

- Час і вартість – мобільні веб-сайти простіші та менш дорогі: останнє, але не менш важливе, розробка мобільних веб-сайтів є значно більшою за часом та рентабельною, ніж розробка рідної програми, особливо якщо вам потрібно мати присутність на різних платформах (вимагають розробки кількох програм).

- Підтримка та технічне обслуговування: інвестиційні міркування щодо програми чи веб-сайту не закінчуються з початковим запуском; Належна підтримка та обслуговування програми (оновлення, тестування, проблеми із сумісністю та постійна розробка) набагато дорожче та складніше, ніж підтримка веб-сайту з часом.

Звичайно, є також багато причин, чому хтось може вибрати розробку рідної мобільної програми замість мобільного веб-сайту. Це може включати такі речі, як доступ до специфічних функцій пристрою (таких як камера або GPS), можливість працювати в автономному режимі та потенціал для покращення продуктивності. Зрештою, вибір між мобільним веб-сайтом і власним мобільним додатком залежатиме від низки факторів, включаючи конкретні потреби компанії чи організації та вподобання її користувачів.

В нашому випадку, нам більш підходить розробка мобільної версії сайту, аніж мобільного додатку по вище зазначеним причинам. Так як цей проект є комерційною розробкою для певного замовника, вирішальне слово йде за ним.

1.5 Висновки за розділом

Було здійснено огляд існуючих платформ з управління IoT пристроями, розібрані їх недоліки та переваги.

Також було здійснено порівняльний аналіз мобільної версії сайту та мобільного додатку, після чого аргументовано вибір саме мобільної версії сайту для розробки.

РОЗДІЛ 2
ОГЛЯД ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ВЕБ-ЗАСТОСУНКІВ

Розробка веб-додатків передбачає використання широкого спектру технологій, від мов програмування та баз даних до фреймворків і бібліотек. Ці технології постійно розвиваються, і постійно впроваджуються нові.

Деякі з найпопулярніших технологій для розробки веб-додатків включають такі мови програмування, як JavaScript, Python, Ruby та PHP. Ці мови забезпечують основу для створення логіки та функціональності веб-додатків і часто використовуються разом з іншими технологіями, такими як HTML і CSS, для створення інтерфейсів користувача.

Фреймворки та бібліотеки також є важливими компонентами розробки веб-додатків. Ці технології надають розробникам готові інструменти та компоненти, які можуть допомогти пришвидшити розробку та покращити якість коду. До популярних фреймворків належать Angular, React і Vue.js для зовнішньої розробки, а також Django, Ruby on Rails і Laravel для внутрішньої розробки.

Бази даних є ще однією важливою технологією для розробки веб-додатків, оскільки вони дозволяють розробникам зберігати та керувати великими обсягами даних. Популярні бази даних, які використовуються у веб-розробці, включають MySQL, PostgreSQL і MongoDB.

Окрім цих основних технологій, розробка веб-додатків також включає в себе низку інших інструментів і технологій, таких як API, веб-сервери, системи контролю версій і тестування.

Загалом розробка веб-додатків включає складну екосистему технологій та інструментів, які вимагають високого рівня знань і досвіду. Слідкуючи за останніми технологіями та передовими методами, розробники можуть створювати високоякісні, масштабовані та безпечні веб-програми, які відповідають потребам користувачів.

В даний час веб-фреймворки є основними інструментами для створення динамічних та інтерактивних веб-додатків. Вони надають набір бібліотек, модулів та інструментів, які допомагають розробникам створювати та підтримувати складні веб-додатки.

Веб-фреймворки створені для спрощення процесу розробки, надаючи структуру для організації коду, керування залежностями та виконання типових завдань, таких як автентифікація користувачів, керування даними та маршрутизація. Ця структура допомагає розробникам зосередитися на логіці та функціональності своїх програм, а не на базовій технології.

Вибір правильного веб-фреймворку для проекту залежить від низки факторів, включаючи розмір і складність програми, набір навичок команди розробників і конкретні вимоги проекту. Незалежно від вибору веб-фреймворку, вони можуть значно спростити та прискорити розробку сучасних веб-додатків.

Статистика використання веб-фреймворків може надати цінну інформацію про поточні тенденції та вподобання веб-розробників. Ці статистичні дані можуть допомогти розробникам прийняти обґрунтовані рішення про те, які фреймворки використовувати для власних проектів, а також допомогти компаніям і організаціям прийняти стратегічні рішення щодо інвестицій у технології та найму.

Одна важлива річ, про яку слід пам’ятати, дивлячись на статистику використання веб-фреймворків, полягає в тому, що вона може відрізнятися залежно від джерела даних. Різні опитування, дослідження та опитування можуть використовувати різні методології та критерії для визначення того, які фреймворки включити та як вимірювати використання. Тому важливо розглядати кілька джерел даних і враховувати обмеження та потенційні упередження кожного з них.

Давайте розглянемо статистичні дані таких ресурсів як stackoverflow trends та npm trends задля виявлення найпоширеніших веб-фреймворків.

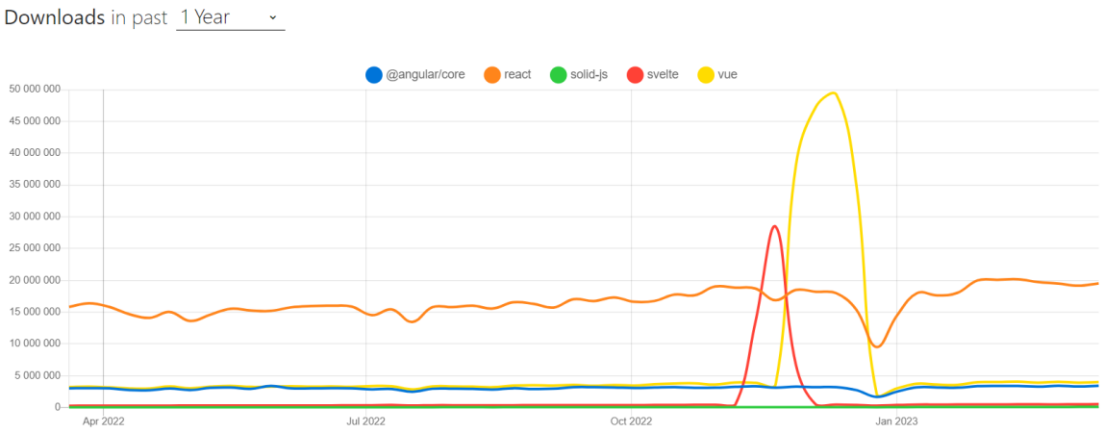

Рисунок 2.1 – Статистичні дані згідно npm trends [8]

Виходячи з статистики зібраної ресурсом npm trends, фаворитом серед скачувань веб-фреймворків є React, на другому місці знаходиться Vue, на третьому Angular.

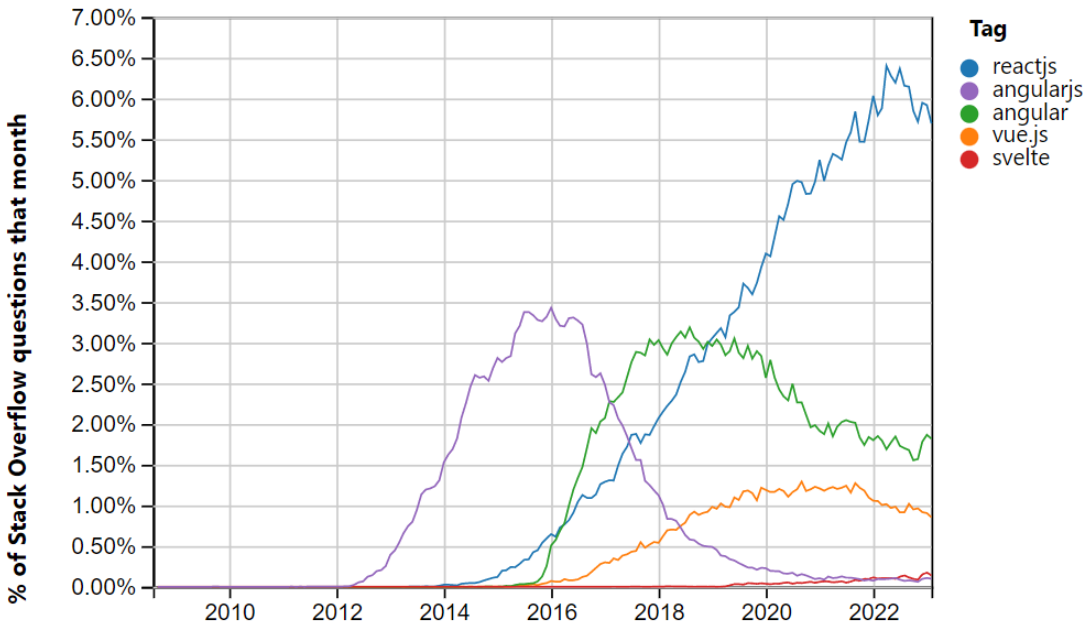

Рисунок 2.2 – Статистичні дані згідно stackoverflow trends [9]

Виходячи з статистки зібраної ресурсом stackoverflow trends, можна побачити, що останні декілька років лідируючі позиції займають React, Angular та Vue.

З розглянутих статистичних даних, можемо дійти висновку, що найпоширенішими веб-фреймворками за останні декілька років є React,

Angular та Vue, і якщо дивитися на інших ресурсах, таких як Github, Stateofjs та інші – дані не зміняться, бо вони є найпопулярнішими серед інших.

Для того, щоб обрати фреймворк який використовувати, потрібно провести більш детальний аналіз кожного з них. Аналізувати веб-фреймворки ми будемо за такими критеріями:

- продуктивність і швидкість;
- гнучкість і можливість налаштування;
- середовище проектування.

2.1 Огляд фреймворку React

React – це популярна платформа JavaScript з відкритим кодом для створення інтерфейсів користувача (UI) і веб-додатків. Він був розроблений Facebook і вперше випущений у 2013 році. З того часу він став одним із найпоширеніших веб-фреймворків і відомий своєю гнучкістю, продуктивністю та простотою використання. [10]

React використовує архітектуру на основі компонентів, що означає, що інтерфейси користувача розбиваються на менші компоненти, які можна багаторазово використовувати. Це полегшує керування складними інтерфейсами користувача, а також оновлення та змінення окремих частин інтерфейсу, не впливаючи на решту програми. React також використовує декларативну модель програмування, яка дозволяє розробникам описувати бажаний стан інтерфейсу користувача, а не писати чіткі інструкції щодо його оновлення.

React можна використовувати як для інтерфейсу, так і для рендерингу на стороні сервера, і його можна легко інтегрувати з іншими технологіями веб-розробки, такими як Redux для управління станом і React Native для створення мобільних додатків. React також має велику та активну спільноту розробників, що означає, що існує багато ресурсів та інструментів, доступних для вивчення та використання фреймворку.

Підсумовуючи, React – це гнучкий, високопродуктивний і широко використовуваний фреймворк JavaScript для створення веб-додатків та інтерфейсів користувача. Його компонентна архітектура та модель декларативного програмування дозволяють легко керувати складними інтерфейсами користувача та оновлювати окремі частини інтерфейсу користувача, не впливаючи на решту програми.

2.1.1 Продуктивність і швидкість

Продуктивність React зумовлена насамперед його компонентною архітектурою та екосистемою інструментів і бібліотек, які розвинулися навколо нього. Компонентна архітектура полегшує розбиття складних інтерфейсів користувача на менші багаторазово використовувані компоненти, якими з часом легше керувати та підтримувати їх. Цей модульний підхід також дозволяє розробникам швидко створювати нові компоненти інтерфейсу користувача або змінювати існуючі, не турбуючись про те, як зміни можуть вплинути на решту програми. Це може бути особливо корисним у великомасштабних проектах, де складність коду та обслуговування можуть бути серйозними проблемами.

Інший спосіб підвищення продуктивності React – це використання віртуальної DOM. Це дозволяє React ефективно оновлювати користувальницький інтерфейс, мінімізуючи кількість прямих маніпуляцій фактичним DOM, що призводить до швидшого та плавного оновлення навіть для великих і складних інтерфейсів користувача. Крім того, вбудовані функції React, такі як методи життєвого циклу та хуки, можуть допомогти розробникам заощадити час і оптимізувати робочий процес розробки.

Одна з причин, чому React настільки продуктивний, полягає в тому, що він має велику та активну спільноту розробників, які роблять внесок у фреймворк і створюють інструменти та бібліотеки навколо нього. Ця спільнота створила багато багаторазово використовуваних компонентів,

шаблонів і стартових наборів, які розробники можуть використовувати для швидкого запуску своїх проектів і економії часу.

Підсумовуючи, продуктивність React походять від його компонентної архітектури, віртуальної DOM, вбудованих функцій і сумісності з іншими бібліотеками та фреймворками. Ці якості роблять його потужним і універсальним інструментом для веб-розробки, що дозволяє розробникам швидко й ефективно створювати складні інтерфейси користувача та програми.

React створений як швидкий і ефективний, і є кілька причин, чому його вважають високошвидкісним фреймворком:

- Віртуальна DOM: React використовує віртуальну DOM (об’єктну модель документа), яка є представленням фактичної DOM у пам’яті. Віртуальний DOM дозволяє React ефективно оновлювати UI, мінімізуючи обсяг прямих маніпуляцій фактичним DOM, що призводить до швидшого та плавного оновлення навіть для великих і складних UI.
- Багаторазові компоненти: компонентна архітектура React дозволяє розробникам створювати багаторазові компоненти інтерфейсу користувача, які можна легко повторно використовувати в різних частинах програми. Це може зменшити кількість коду, який потрібно написати, а також може полегшити підтримку та оновлення коду з часом.
- Одностороннє прив’язування даних: одностороннє прив’язування даних React означає, що дані передаються в одному напрямку, від батьківського компонента до дочірніх компонентів. Це спрощує потік даних і зменшує ризик конфліктів або помилок.
- Відтворення на стороні сервера: React підтримує відтворення на стороні сервера, що означає, що сервер може попередньо відобразити HTML і надіслати його в браузер, зменшуючи обсяг роботи, яку потрібно виконати на стороні клієнта. Це може значно скоротити час початкового завантаження програми, особливо для користувачів із повільним або нестабільним підключенням до Інтернету.
- Оптимізація продуктивності: React включає кілька оптимізацій

продуктивності, таких як shouldComponentUpdate() і PureComponent, які можуть покращити продуктивність програми, зменшивши непотрібне повторне рендеринг компонентів. [11]

Підводячи підсумок, висока швидкість React зумовлена використанням віртуальної DOM, повторно використовуваних компонентів, одностороннього зв’язування даних, рендерингу на стороні сервера та оптимізації продуктивності. Ці функції роблять React ефективним і продуктивним фреймворком для створення сучасних веб-додатків.

2.1.2 Гнучкість і можливість налаштування

Гнучкість React пояснюється його здатністю використовуватися в різноманітних програмах і середовищах. React можна використовувати для створення чого завгодно: від простих статичних веб-сайтів до складних односторінкових програм (SPA). React також достатньо універсальний, щоб використовувати його для рендерингу на стороні сервера, що може покращити продуктивність і SEO веб-додатків. Крім того, сумісність React з іншими бібліотеками та фреймворками, такими як Redux для управління станом і React Router для маршрутизації на стороні клієнта, полегшує інтеграцію з іншими частинами стеку розробки.

Гнучкість React також поширюється на його простоту використання з різними інструментами розробки, такими як популярні редактори коду, такі як Visual Studio Code або Sublime Text, і засоби запуску завдань, такі як Gulp або Grunt. Це дозволяє розробникам налаштовувати середовище розробки та робочий процес відповідно до своїх потреб.

Інший варіант гнучкості React – це підтримка різних парадигм програмування. React можна використовувати як із функціональним, так і з об’єктно-орієнтованим стилями програмування, що дозволяє розробникам вибрати підхід, який найкраще відповідає потребам їх проекту. Ця гнучкість також поширюється на здатність React працювати з різними бібліотеками

керування станом, такими як Redux або MobX, що дозволяє розробникам вибирати підхід, який найкраще відповідає вимогам їхніх програм.

React надає багато варіантів для налаштування, зокрема:

- CSS Styling: React дозволяє розробникам використовувати свої бажані стилі, наприклад CSS, Sass або styled-components. Це дає розробникам можливість контролювати візуальний вигляд своєї програми, дозволяючи їм створювати власні дизайни та теми.
- Управління станом: React надає базові можливості керування станом, але також добре працює з іншими бібліотеками керування станом, такими як Redux, MobX або Apollo. Це дозволяє розробникам вибрати підхід, який найкраще відповідає вимогам їхніх програм.
- Методи життєвого циклу: React надає низку методів життєвого циклу, які дозволяють розробникам додавати спеціальну логіку до своїх компонентів. Це можна використовувати для обробки подій, оновлення стану або запуску інших дій у відповідь на зміни в програмі.
- Компоненти вищого порядку: компоненти вищого порядку (HOC) дозволяють розробникам додавати додаткові функції до своїх компонентів без необхідності переписувати вихідний код. HOC можна використовувати для автентифікації, кешування або інших поширених завдань.

Таким чином, гнучкість і можливості налаштування React роблять його потужним інструментом для створення програм, які відповідають певним вимогам. Його компонентна архітектура, сумісність з платформами та інтеграція з іншими бібліотеками та фреймворками надають розробникам гнучкість, необхідну для створення індивідуальних рішень, а його стилі CSS, керування станом, методи життєвого циклу та компоненти вищого порядку надають численні параметри налаштування для індивідуальних потреб. застосування для конкретних потреб.

2.1.3 Середовище проектування

React – це бібліотека JavaScript, що означає, що її можна кодувати за допомогою будь-якого текстового редактора або інтегрованого середовища розробки (IDE), що підтримує JavaScript. Однак є кілька IDE та редакторів, які особливо популярні серед розробників React завдяки своїм функціям та інтеграції.

Ось деякі популярні IDE та текстові редактори, які використовуються для розробки React:

- Visual Studio Code (VS Code) – популярний безкоштовний редактор коду з відкритим кодом, який широко використовується для розробки React. Він пропонує різноманітні функції та розширення, включаючи вбудований термінал, інструменти налагодження та підтримку інтеграції Git.
- WebStorm – це потужна IDE, спеціально розроблена для веб-розробки, включаючи React. Він пропонує такі функції, як доповнення коду, налагодження та підтримку різних веб-технологій, таких як TypeScript, CSS і HTML.
- Atom – ще один популярний текстовий редактор, який використовується для розробки React. Він пропонує широкий спектр функцій і плагінів, включаючи підтримку контролю версій, підсвічування коду та фрагментів коду.
- Sublime Text – це легкий текстовий редактор із можливістю налаштування, який підтримує розробку React. Він пропонує різноманітні функції, такі як підсвічування синтаксису, доповнення коду та система плагінів. [12]

З точки зору функцій, розробка React зазвичай вимагає кількох ключових функцій у IDE або текстовому редакторі, наприклад:

- Виділення та форматування коду: підсвічування та форматування синтаксису може зробити код більш читабельним і зрозумілим.
- Доповнення коду: доповнення коду може заощадити час розробників, надаючи пропозиції щодо коду під час введення.
- Інструменти налагодження: інструменти налагодження можуть допомогти розробникам усунути помилки та виявити проблеми в коді.

- Інтеграція з іншими інструментами: інтеграція з такими інструментами, як Git, менеджери пакунків і лінтери, може зробити розробку React більш ефективною та спрощеною.

Таким чином, React можна кодувати за допомогою будь-якого текстового редактора або IDE, що підтримує JavaScript. Популярні варіанти включають Visual Studio Code, WebStorm, Atom і Sublime Text. Важливі функції для розробки React у IDE або текстовому редакторі включають підсвічування та форматування коду, доповнення коду, інструменти налагодження та інтеграцію з іншими інструментами.

2.2 Огляд фреймворку Angular

Angular – це веб-фреймворк для створення клієнтських програм за допомогою HTML, CSS і TypeScript. Він був створений Google і зараз підтримується командою розробників і учасників спільноти. Angular – це повний фреймворк, який надає ряд функцій та інструментів для створення складних, масштабованих і високопродуктивних веб-додатків. Деякі з ключових особливостей Angular включають:

- Модульна архітектура: додатки Angular побудовані з використанням модульної архітектури, яка дозволяє розробникам розбивати програму на менші компоненти, які можна багаторазово використовувати.
- Двостороннє зв’язування даних: Angular забезпечує двостороннє зв’язування даних між поданням і моделлю, що означає, що будь-які зміни, внесені до моделі, автоматично відображаються у поданні, і навпаки.
- Маршрутизація: Angular надає потужну систему маршрутизації, яка дозволяє розробникам визначати маршрути для різних переглядів і керувати навігацією між ними.
- Шаблони: Angular використовує шаблони HTML для визначення рівня перегляду програми, що полегшує розробникам створення та керування складними інтерфейсами користувача. [13]

Підводячи підсумок, Angular – це повний фреймворк для створення складних і масштабованих веб-додатків. Він надає низку функцій, таких як модульна архітектура, двостороннє зв’язування даних, маршрутизація та шаблони.

2.2.1 Продуктивність і швидкість

Angular відомий своєю високою продуктивністю та швидкістю, що робить його популярним вибором для створення складних веб-додатків корпоративного рівня. Деякі фактори, які впливають на продуктивність і швидкість додатків Angular, включають:

- Попередня компіляція (AOT): Angular надає компілятор AOT, який компілює шаблони та генерує високооптимізований код під час збірки. Це призводить до швидшого часу завантаження та покращення продуктивності.
- Виявлення змін: Angular використовує виявлення змін для відстеження змін у стані програми та відповідного оновлення перегляду. Алгоритм виявлення змін Angular дуже оптимізований, що допомагає звести до мінімуму непотрібні оновлення DOM і підвищити продуктивність.
- Реактивне програмування: Angular використовує RxJS, бібліотеку для реактивного програмування, для керування асинхронними операціями та обробки подій. Це полегшує керування складними потоками даних і оптимізує продуктивність.
- Відкладене завантаження: Angular забезпечує механізм відкладеного завантаження, який дозволяє завантажувати модулі на вимогу, а не всі одночасно. Це допомагає скоротити час початкового завантаження та покращити продуктивність.
- Струшування дерева: Angular підтримує струшування дерева, процес, який усуває мертвий код і зменшує розмір остаточного комплекту. Це допомагає покращити час завантаження та продуктивність.
- Візуалізація на стороні сервера (SSR): Angular підтримує рендеринг на

стороні сервера, що дозволяє виконувати початковий рендеринг програми на сервері. Це допомагає підвищити продуктивність і скоротити час завантаження для користувачів із повільним підключенням до Інтернету.

Окрім цих функцій, Angular також надає низку інструментів і найкращих практик для оптимізації продуктивності, наприклад використання trackBy для оптимізації циклів ngFor, використання стратегії виявлення змін OnPush і зменшення кількості прив'язок і спостерігачів у додатку.

Загалом Angular – це високопродуктивний фреймворк, який надає ряд функцій та інструментів для оптимізації продуктивності та швидкості. Його компіляція AOT, алгоритм виявлення змін, реактивне програмування, відкладене завантаження, струшування дерева та можливості візуалізації на стороні сервера – усе це сприяє його продуктивності та швидкості.

2.2.2 Гнучкість і можливість налаштування

Angular – це фреймворк для веб-розробки з широкими можливостями налаштування, який надає розробникам багато варіантів адаптації своїх програм до своїх конкретних потреб. Ось кілька способів, якими Angular пропонує налаштування:

- Компоненти: Компонентна архітектура Angular дозволяє розробникам створювати власні компоненти, які інкапсулюють певну функціональність і можуть повторно використовуватися в різних частинах програми. Це забезпечує високий ступінь налаштування, оскільки розробники можуть створювати компоненти, які відповідають конкретним потребам їхньої програми.
- Директиви: функція директиви Angular дозволяє розробникам створювати власні HTML-теги та атрибути, які можна використовувати для створення багаторазових компонентів і додавання поведінки додатку. Це забезпечує ще один рівень налаштування, оскільки розробники можуть створювати власні директиви, які додають певну функціональність їхнім

програмам.

- Канали: функція каналів Angular дозволяє розробникам перетворювати дані в програмі, наприклад форматувати дати або фільтрувати списки. Це забезпечує високий ступінь настроювання, оскільки розробники можуть створювати спеціальні канали, які виконують певні перетворення відповідно до потреб їхньої програми.
- Сервіси: сервісна функція Angular дозволяє розробникам створювати багаторазовий код, який можна спільно використовувати між різними компонентами та частинами програми. Це забезпечує високий ступінь налаштування, оскільки розробники можуть створювати служби, які надають певні функції, які можна використовувати в усій програмі.
- Модулі: функція модуля Angular дозволяє розробникам організовувати свій код у модулі, які можна використовувати для інкапсуляції певних функцій і залежностей. Це забезпечує високий ступінь налаштування, оскільки розробники можуть створювати власні модулі, які забезпечують певні функції та залежності. [14]

Загалом, Angular надає розробникам багато можливостей для налаштування своїх програм і пристосування їх до своїх конкретних потреб.

Angular – це дуже гнучкий і універсальний фреймворк веб-розробки, який пропонує багато функцій і інструментів для створення складних і масштабованих програм. Ось кілька факторів, які доводять гнучкість Angular:

- Архітектура: Angular дотримується архітектури Model-View-Controller (MVC), яка забезпечує чіткий розподіл завдань між різними рівнями програми. Це дозволяє розробникам писати модульний і багаторазовий код, полегшуючи підтримку та масштабування програми з часом.
- Двостороннє зв’язування даних: функція двостороннього зв’язування даних Angular дозволяє розробникам автоматично синхронізувати дані між представленням і моделлю. Це означає, що будь-які зміни, внесені до подання, автоматично оновлять модель, а будь-які зміни, внесені до моделі, автоматично оновлять подання. Ця функція може допомогти спростити процес

розробки та зменшити кількість необхідного шаблонного коду.

- Тестування: Angular надає вбудовану підтримку для тестування, що дозволяє розробникам писати тести для своїх компонентів і служб. Це може допомогти переконатися, що програма працює належним чином, і зменшити ймовірність помилок і помилок.
- Кросплатформна розробка: Angular дозволяє розробникам створювати програми, які можуть працювати на кількох платформах, включаючи веб, мобільні пристрої та настільні комп'ютери. Це забезпечує високу гнучкість, оскільки розробники можуть створювати програми, до яких користувачі можуть отримати доступ із широкого спектру пристроїв і платформ. [14]

Загалом, архітектура Angular, зв'язування даних, впровадження залежностей, директиви, тестування та функції кросплатформної розробки надають розробникам велику гнучкість під час створення програм. Це робить Angular надзвичайно універсальним фреймворком, який можна використовувати для створення широкого спектру програм для різних варіантів використання та платформ.

2.2.3 Середовище проектування

Що стосується інструментів розробки, Angular можна кодувати за допомогою будь-якого текстового редактора або інтегрованого середовища розробки (IDE), що підтримує TypeScript. Однак є деякі популярні IDE та редактори, які спеціально розроблені для розробки за допомогою фрейворку Angular, до них можна віднести:

- Visual Studio Code – це популярний безкоштовний редактор коду з відкритим кодом, який широко використовується для розробки на Angular. Він пропонує ряд функцій і розширень, включаючи підтримку TypeScript, інструменти налагодження та інтеграцію з іншими інструментами, такими як Git.
- WebStorm – це потужна IDE, спеціально розроблена для веб-розробки,

включаючи Angular. Він пропонує такі функції, як доповнення коду, налагодження та підтримку різних веб-технологій, таких як CSS і HTML.

- Atom – популярний текстовий редактор, який використовується для розробки Angular. Він пропонує широкий спектр функцій і плагінів, включаючи підтримку контролю версій, підсвічування коду та фрагментів коду.
- Sublime Text – це легкий текстовий редактор із можливістю налаштування, який підтримує розробку Angular. Він пропонує різноманітні функції, такі як підсвічування синтаксису, доповнення коду та система плагінів. [12]

На додаток до цих IDE та редакторів коду, розробники Angular також часто використовують низку інших інструментів і бібліотек, щоб допомогти в розробці, таких як Angular CLI та різні інструменти налагодження та тестування. Загалом, Angular можна розробляти за допомогою різних інструментів і середовищ, залежно від уподобань і потреб розробника.

2.3 Огляд фреймфорку Vue

Vue.js (або просто Vue) – це прогресивна платформа JavaScript із відкритим вихідним кодом, яка використовується для створення інтерфейсів користувача та односторінкових програм. Він був створений Еваном Ю і вперше випущений у лютому 2014 року.

Vue часто порівнюють з іншими популярними фреймворками, такими як React і Angular, але він має певні відмінності та переваги. Ось деякі ключові функції Vue:

- Легкий: Vue є легким, займає лише близько 20 КБ після мінімізації та gzip-архіву. Це дозволяє швидко завантажити та легко інтегрувати в існуючі проекти.
- Простий і легкий у вивченні: Vue розроблений таким чином, щоб його було легко вивчати та використовувати, що робить його популярним вибором для

розробників, які тільки починають розробляти інтерфейс. Його синтаксис простий і легкий для читання, а його документація є вичерпною та зручною для користувача.

- Гнучкий і настроюваний: Vue є дуже гнучким і настроюваним, що дозволяє розробникам вибирати особливості та функції, необхідні для своїх проектів. Це дозволяє легко інтегрувати Vue в існуючі проекти або створювати власні рішення з нуля.
- Реактивні компоненти: Vue використовує реактивну систему для обробки змін даних і введення користувача. Це означає, що зміни даних автоматично відображаються в інтерфейсі користувача без необхідності оновлювати їх вручну.
- Компонентна архітектура: Vue використовує компонентну архітектуру, подібну до React і Angular. Це полегшує створення багаторазових компонентів і керування складними інтерфейсами користувача.
- Чудова екосистема: Vue має велику та активну екосистему з низкою плагінів і бібліотек, доступних для стандартних випадків використання, таких як маршрутизація та керування станом. [15]

Загалом Vue – це гнучкий і потужний фреймворк, який добре підходить для створення сучасних веб-додатків. Його легкий синтаксис, який легко освоїти, і компонентна архітектура роблять його популярним вибором для розробників будь-якого рівня кваліфікації.

2.3.1 Продуктивність і швидкість

Vue.js відомий своєю чудовою продуктивністю та швидкістю, що робить його популярним вибором для створення високопродуктивних веб-додатків. Ось деякі фактори, які впливають на продуктивність Vue:

- Віртуальний DOM: як і React, Vue використовує віртуальний DOM для ефективного оновлення інтерфейсу користувача. Віртуальний DOM є спрощеним представленням фактичного DOM і дозволяє Vue робити

ефективні оновлення без необхідності повного перезавантаження сторінки.

- Реактивне зв’язування даних: Vue використовує реактивну систему для обробки змін даних, що означає, що зміни даних автоматично відображаються в інтерфейсі користувача. Це допомагає зменшити кількість коду, необхідного для керування станом і оновлення інтерфейсу користувача.
- Відкладене завантаження: Vue підтримує відкладене завантаження компонентів, що означає, що компоненти завантажуються лише тоді, коли вони потрібні. Це може допомогти скоротити початковий час завантаження програми та покращити продуктивність.
- Візуалізація на стороні сервера: Vue підтримує рендеринг на стороні сервера, що може допомогти скоротити початковий час завантаження програми та покращити пошукову оптимізацію.
- Мінімальний розмір: Vue легкий, з невеликим розміром файлу лише близько 20 Кб. Це дозволяє швидко завантажити та легко інтегрувати в існуючі проекти. [15]

Загалом Vue відомий своєю чудовою продуктивністю та швидкістю, що робить його чудовим вибором для створення високопродуктивних веб-додатків.

2.3.2 Гнучкість і можливість налаштування

Vue.js відомий своєю гнучкістю та можливостями налаштування, що робить його популярним вибором для розробників, які хочуть створювати власні рішення для своїх проектів. Ось деякі фактори, які сприяють гнучкості та можливостям налаштування Vue:

- Модульна архітектура: Vue використовує модульну архітектуру, що дозволяє легко вибирати функції та функції, необхідні для проекту. Розробники можуть використовувати лише ті частини Vue, які їм потрібні, і можуть легко інтегрувати Vue у існуючі проекти.
- Спеціальні директиви та фільтри: Vue підтримує спеціальні директиви

та фільтри, які дозволяють розробникам розширювати функціональні можливості Vue і створювати спеціальну поведінку для своїх програм.

- Спеціальні плагіни: Vue підтримує спеціальні плагіни, які дозволяють розробникам створювати власні функції та інтегрувати їх у Vue. Це може бути корисним для додавання таких функцій, як відстеження аналітики, автентифікація або керування даними.
- Розгалужена екосистема: Vue має велику й активну екосистему з рядом плагінів і бібліотек, доступних для типових випадків використання, таких як маршрутизація та керування станом. Це полегшує пошук та інтеграцію рішень, які відповідають потребам конкретного проекту.
- Однофайлові компоненти: Vue дозволяє розробникам писати компоненти в одному файловому форматі, що полегшує керування та налаштування компонентів. Цей формат містить шаблон, сценарій і стиль для компонента в одному файлі, що дає змогу легко побачити структуру компонента та внести зміни.
- CSS з областю: Vue підтримує CSS з областю, що означає, що стилі, визначені в компоненті, застосовуються лише до цього компонента та його дочірніх компонентів. Це полегшує керування стилями для компонента та уникає конфліктів з іншими компонентами на сторінці.
- Міксини та розширення: Vue підтримує міксини та розширення, які дозволяють розробникам повторно використовувати код у кількох компонентах. Міксини – це багаторазово використовувані фрагменти коду, які можна додавати до компонента, тоді як розширення дозволяють розробникам створювати нові компоненти, які успадковують існуючі компоненти.
- Спеціальні функції візуалізації: Vue підтримує спеціальні функції візуалізації, які дозволяють розробникам створювати спеціальну логіку візуалізації для своїх компонентів. Це може бути корисним для створення складних інтерфейсів користувача або для інтеграції з іншими бібліотеками та фреймворками. [15]

Загалом, підтримка Vue однофайлових компонентів, обмеженого CSS,

міксинів і розширень, спеціальних функцій візуалізації та реактивного зв’язування даних надає йому гнучкої структури, яку можна адаптувати до широкого діапазону проектів і випадків використання.

2.3.3 Середовище проектування

Vue можна кодувати в будь-якому текстовому редакторі чи інтегрованому середовищі розробки (IDE), яке підтримує розробку JavaScript. Однак є також кілька IDE, спеціально розроблених для розробки Vue, з функціями, які можуть підвищити продуктивність і покращити досвід розробки.

Ось деякі з популярних IDE для розробки Vue та їхні функції:

- Visual Studio Code – це популярна безкоштовна IDE із відкритим кодом, яка підтримує розробку Vue через пакет розширень Vue.js. Цей пакет розширень містить кілька корисних розширень, таких як підсвічування синтаксису та автозаповнення для шаблонів Vue, фрагменти коду та підтримку налагодження.
- WebStorm – це потужна комерційна IDE, яка пропонує комплексну підтримку розробки Vue, включаючи підсвічування синтаксису, автозавершення та налагодження. Він також включає такі функції, як рефакторинг коду, аналіз якості коду та підтримку інших популярних веб-технологій, таких як Node.js і TypeScript.
- Atom – це безкоштовний текстовий редактор із відкритим вихідним кодом, який підтримує розробку Vue за допомогою кількох пакетів, які підтримує спільнота. Ці пакети пропонують такі функції, як підсвічування синтаксису, автозавершення та фрагменти коду.
- Sublime Text – це популярний текстовий редактор, який підтримує розробку Vue за допомогою кількох плагінів. Ці плагіни надають такі функції, як підсвічування синтаксису, автозавершення та фрагменти коду. [12]

На додаток до цих IDE існує також кілька інструментів і бібліотек Vue,

які можна використовувати для покращення досвіду розробки, наприклад Vue DevTools, розширення для браузера, яке надає набір інструментів для налагодження та розробки програм Vue, і Vetur, розширення VS Code, яке пропонує розширені функції, такі як перевірка помилок, форматування коду та лінтування для програм Vue.

Загалом, Vue можна кодувати у будь-якому текстовому редакторі чи середовищі IDE, що підтримує розробку JavaScript, але використання спеціалізованої IDE або спеціальних інструментів Vue може підвищити продуктивність і покращити досвід розробки.

2.4 Аргументація вибору фреймворку для розробки веб-застосунку

Після розглядання найпопулярніших на теперішній час фреймворків для розробки веб-застосунків, було прийняте рішення зупинитись на фреймворці під назвою React. Одну з ключових ролей в виборі саме даного фреймворку зіграв замовник, але є і чіткі відмінності, за якими було прийняте рішення віддати перевагу реакту. Зумовлене дане рішення такими факторами:

- Віртуальний DOM: використання React віртуального DOM дозволяє ефективно відображати та оновлювати інтерфейс користувача, що призводить до кращої продуктивності та плавнішої взаємодії з користувачем.
- JSX: використання React JSX дозволяє більш інтуїтивно зрозумілий і стислий спосіб написання коду інтерфейсу користувача, що полегшує його читання та підтримку.
- Можливість повторного використання: компоненти React можна легко повторно використовувати на кількох сторінках і проектах, що може заощадити час розробки та покращити підтримку коду.
- Легкість: React – це легка бібліотека, що означає, що її можна легко інтегрувати в існуючі проекти, не додаючи значних витрат.
- Оптимізація продуктивності: React надає багато інструментів для оптимізації продуктивності, таких як shouldComponentUpdate і

PureComponent, які можуть зменшити кількість непотрібних рендерів і підвищити загальну продуктивність.

- Безпека типів: React можна використовувати з TypeScript, статично типізованою мовою, яка забезпечує безпеку типів і допомагає запобігти помилкам під час компіляції.
- Інструменти розробника: React має різноманітні доступні інструменти розробника, які спрощують налагодження та тестування, наприклад, розширення для браузера React Developer Tools і фреймворк тестування Jest.
- Підтримка спільноти: React має велику та активну спільноту розробників, що означає, що є багато ресурсів, доступних для навчання та усунення несправностей, а також велика кількість бібліотек і компонентів з відкритим кодом.
- Одностороннє зв’язування даних: React використовує одностороннє зв’язування даних, що спрощує потік даних і зменшує ризик побічних ефектів.
- Гнучкість: гнучкість React дозволяє легко інтегрувати його з іншими бібліотеками та фреймворками, що робить його популярним вибором для створення складних і динамічних веб-додатків.

Хоча Angular і Vue також мають свої сильні сторони та переваги, компонентна архітектура React, віртуальний DOM і JSX роблять його потужним і популярним вибором для веб-розробки. Крім того, популярність React і підтримка спільноти означають, що розробникам доступна велика кількість ресурсів, що полегшує швидкий запуск і усунення будь-яких проблем, які можуть виникнути.

2.5 Особливості отримання даних для відображення

У веб-розробці фронтенд відповідає за відтворення інтерфейсу користувача та взаємодію з користувачами, тоді як бекенд обробляє логіку на стороні сервера та спілкується з базами даних, API та іншими зовнішніми системами. Щоб створити динамічну веб-програму, часто необхідно

надсилати й отримувати дані між інтерфейсом і сервером.

Є кілька способів передачі даних із серверної частини до зовнішньої, зокрема:

- RESTful API: Representational State Transfer (REST) – це стиль архітектури програмного забезпечення, який визначає набір обмежень для створення веб-служб. RESTful API надають стандартний спосіб для інтерфейсу зв’язуватися з сервером за допомогою HTTP-запитів і відповідей. Сервер надає набір кінцевих точок, які інтерфейс може використовувати для запиту даних або виконання дій, таких як створення або оновлення записів у базі даних.

- WebSockets: WebSockets забезпечують двонаправлений канал зв’язку між інтерфейсом і сервером, що дозволяє надсилати й отримувати дані в реальному часі без запитів HTTP. WebSockets часто використовуються для програм чату, інформаційних панелей у режимі реального часу та інших програм, де потрібні оновлення в режимі реального часу.
- GraphQL: GraphQL – це мова запитів для API, яка забезпечує більш гнучкий і ефективний спосіб запиту даних із серверної частини. За допомогою GraphQL інтерфейс може точно вказати, які дані йому потрібні, а серверна частина відповідає лише запитаними даними, зменшуючи обсяг даних, які потрібно передати через мережу. [16]

Важливо розділити інтерфейс і серверну частину з кількох причин. По-перше, це дозволяє створювати більш модульний і зручний код. Розділивши інтерфейс і серверну частину, розробники можуть зосередитися на своїх відповідних сферах знань і уникнути створення монолітних кодових баз, якими важко керувати.

По-друге, розділення інтерфейсу та серверної частини забезпечує кращу масштабованість і продуктивність. Інтерфейс можна розгорнути в мережі доставки контенту (CDN) для швидшого завантаження, тоді як серверну частину можна масштабувати незалежно для обробки збільшеного трафіку та

робочого навантаження.

По-третє, це забезпечує кращу безпеку. Розділення інтерфейсу та серверної частини гарантує, що конфіденційні дані та бізнес-логіка зберігаються на стороні сервера, де їх легше захистити. Якщо логіка серверної частини не розкрита для зовнішньої частини, зловмисникам стає важче використовувати вразливості в коді.

Інша причина полягає в тому, що це дозволяє краще тестувати та налагоджувати. Відокремивши інтерфейс і сервер, кожен з них можна тестувати та налагоджувати окремо. Це може спростити процес тестування та полегшити виявлення та усунення проблем.

Розділення інтерфейсу та серверної частини також забезпечує більшу гнучкість щодо вибору технологій. Можна використовувати різні інтерфейсні та бекенд-фреймворки, бібліотеки та мови програмування, якщо вони можуть спілкуватися через API або інші протоколи зв’язку. Це може забезпечити більше інновацій та експериментів, а також можливість вибрати найкращу технологію для конкретного завдання.

Підводячи підсумок, можна сказати, що розділення інтерфейсу та серверної частини є важливим для модульного та підтримуваного коду, масштабованості та продуктивності, безпеки, тестування та налагодження, а також гнучкості технологій. Це дає змогу краще розділити проблеми, спростити співпрацю між розробниками та покращити загальну архітектуру програми.

2.6 Архітектура «Клієнт-Сервер»

Архітектура клієнт-сервер у веб-розробці – це модель, у якій функціональні можливості програми розподіляються між клієнтом і сервером. Клієнт, як правило, веб-браузер або мобільний додаток, відповідає за інтерфейс користувача та взаємодію. Сервер забезпечує зберігання, обробку та бізнес-логіку даних.

У цій архітектурі клієнт надсилає запити на сервер, вказуючи бажану дію (наприклад, отримання даних, надсилання форми). Сервер обробляє ці запити, виконує необхідні операції та надсилає відповідь із запитаними даними або підтвердженням. [16]

Клієнт і сервер спілкуються за допомогою стандартизованих протоколів, часто HTTP, що забезпечує сумісність і взаємодію. Це дозволяє безперебійно обмінюватися даними та діями між двома компонентами.

Архітектура клієнт-сервер пропонує кілька переваг. Це забезпечує модульну розробку, полегшуючи оновлення або заміну компонентів клієнта або сервера незалежно один від одного. Це також забезпечує масштабованість, оскільки кілька клієнтів можуть підключатися до сервера одночасно. Крім того, шляхом централізації зберігання та обробки даних на сервері це допомагає покращити безпеку та цілісність даних.

Загалом, клієнт-серверна архітектура є фундаментальною концепцією веб-розробки, яка забезпечує структурований та ефективний спосіб створення веб-додатків, розділяючи проблеми між клієнтським і серверним компонентами для кращої організації та продуктивності.

2.7 Архітектурні принципи REST

API, або інтерфейс програмування додатків, – це набір правил, які визначають, як програми або пристрої можуть підключатися та спілкуватися один з одним. REST API – це API, який відповідає принципам дизайну REST або архітектурному стилю передачі репрезентативного стану. З цієї причини REST API іноді називають RESTful API.

REST API можна розробляти за допомогою практично будь-якої мови програмування та підтримувати різні формати даних. Єдина вимога полягає в тому, щоб вони узгоджувалися з наступними п'ятьма принципами дизайну REST, також відомими як архітектурні обмеження [17]:

- Уніфікований інтерфейс. Усі запити API для одного ресурсу мають

виглядати однаково, незалежно від того, звідки надходить запит. Ресурси не повинні бути занадто великими, але повинні містити всю інформацію, яка може знадобитися клієнту.

- Розв'язка клієнт-сервер. У дизайні REST API клієнтські та серверні програми мають бути повністю незалежними одна від одної. Єдиною інформацією, яку має знати клієнтська програма, є URI запитаного ресурсу.
- Відсутність стану. API REST не мають стану, тобто кожен запит повинен містити всю інформацію, необхідну для його обробки. Іншими словами, API REST не потребують сеансів на стороні сервера.
- Кешування. Якщо це можливо, ресурси повинні кешуватися на стороні клієнта або сервера. Мета полягає в тому, щоб підвищити продуктивність на стороні клієнта, одночасно збільшуючи масштабованість на стороні сервера.
- Рівнева архітектура системи. В REST API виклики та відповіді проходять різні рівні. У комунікаційному контурі може бути кілька різних посередників. REST API потрібно розробити так, щоб ні клієнт, ні сервер не могли визначити, чи він спілкується з кінцевою програмою чи посередником.

REST API обмінюються даними через запити HTTP для виконання стандартних функцій бази даних, таких як створення, читання, оновлення та видалення записів (також відомих як CRUD) у межах ресурсу. Наприклад, REST API використовуватиме запит GET для отримання запису, запит POST для його створення, запит PUT для оновлення запису та запит DELETE для його видалення.

2.8 Інструментарій для розробки

2.8.1 Мова програмування TypeScript

TypeScript – це строго типізована мова програмування, розроблена корпорацією Майкрософт, яка вдосконалює JavaScript, додаючи статичний тип і розширені функції. Він служить надмножиною JavaScript, тобто весь

дійсний код JavaScript є дійсним кодом TypeScript. Використовуючи статичну типізацію, TypeScript дозволяє розробникам виявляти помилки, пов’язані з типом, на етапі розробки, що призводить до покращення якості та надійності коду. [18]

Однією з помітних переваг TypeScript є його підтримка принципів об’єктно-орієнтованого програмування (ООП). Він представляє такі функції, як класи, інтерфейси, успадкування та модифікатори доступу, сприяючи створенню більш структурованого та багаторазового коду. Ці можливості ООП сприяють кращій організації коду та зручності обслуговування, особливо у великомасштабних проектах.

Крім того, TypeScript легко інтегрується з існуючими кодовими базами JavaScript, що дозволяє розробникам поступово адаптувати його. Код TypeScript компілюється в звичайний JavaScript, що дає змогу запускати його в будь-якому браузері або середовищі виконання JavaScript. Ця сумісність гарантує, що проекти, створені за допомогою TypeScript, залишаються доступними та сумісними з ширшою екосистемою JavaScript.

2.8.2 Redux Toolkit для контролю стану

Redux Toolkit – це бібліотека, яка вийшла на сцену кілька років тому з метою спрощення розробки Redux. Він обертається навколо ядра Redux, надаючи простіший інтерфейс для роботи з ним та надає набір утиліт і угод, які зменшують шаблонний код і підвищують продуктивність розробника.

Тож, якщо кажучи більш детально - це бібліотека керування передбачуваним станом для програм JavaScript, яка зазвичай використовується з React фреймворком. Вона забезпечує централізований і передбачуваний спосіб керування станом програми, полегшуючи розробку складних програм із узгодженою та надійною поведінкою. А ключовими перевагами з використання Redux Toolkit є [19]:

- Передбачуваність: Redux забезпечує передбачуваний шаблон

управління станом, що полегшує міркування про те, як відбуваються зміни стану в програмі.

- Централізований стан: наявність єдиного джерела правдивих даних спрощує потік даних і полегшує відстеження та налагодження змін стану.
- Масштабованість: Redux добре працює з великомасштабними програмами, забезпечуючи чітку структуру для керування станом і полегшуючи організацію коду.
- Можливість тестування: Redux сприяє тестуванню, відокремлюючи бізнес-логіку від рівня інтерфейсу користувача, що полегшує написання модульних тестів для редукторів і дій.
- Інструменти розробника: Redux надає багату екосистему інструментів розробника, які допомагають у налагодженні, налагодженні подорожей у часі та перевірці змін стану програми.

2.8.3 Середовище розробки WebStorm

Під час професійної розробки, для підвищення зручності та швидкості роботи з програмним кодом, проектуванням та рефакторингом, застосовуються програмні засоби, які називаються IDE (Integrated Development Environment).

Для мови програмування TypeScript існує чимало IDE з різними особливостями, але найбільш популярними з них є Visual Studio Code та WebStorm. Для розробки даного веб-застосунку була використана IDE під назвою WebStorm, через такі переваги над своїм головним супротивником:

- Глибокі функції IDE: WebStorm – це повноцінне інтегроване середовище розробки (IDE), спеціально розроблене для веб-розробки. Він пропонує такі розширені функції, як інтелектуальне завершення коду, навігація по коду, інструменти рефакторингу та можливості налагодження. У WebStorm ці функції часто є більш потужними та налаштованими порівняно з легшим VSCode.

- Спеціалізовані робочі процеси веб-розробки: WebStorm розроблено для оптимізації робочих процесів веб-розробки. Він підтримує різні веб-фреймворки та бібліотеки, такі як React, Angular, Vue.js і Node.js із спеціальною інтеграцією. Він розуміє їхні специфічні структури коду, забезпечує перевірку коду та пропонує вбудовані інструменти для запуску, налагодження та тестування веб-додатків.
- Розумний аналіз коду та виявлення помилок: WebStorm містить інтелектуальні функції аналізу коду, які можуть виявляти потенційні помилки, пропонувати виправлення та покращувати якість коду. Він також надає підсвічування помилок у реальному часі та пропозиції швидкого виправлення. Ці можливості можуть допомогти виявити помилки на ранніх стадіях процесу розробки та підвищити надійність коду.
- Безперебійне налагодження: WebStorm пропонує потужний вбудований налагоджувач для JavaScript і TypeScript. Він забезпечує контрольні точки, покрокове налагодження та перевірку змінних, що дозволяє розробникам налагоджувати свій код у IDE. Налагодження в WebStorm оптимізовано для веб-розробки та забезпечує інтуїтивно зрозумілий інтерфейс, що полегшує пошук і вирішення проблем.

Підсумовуючи, WebStorm – це потужне інтегроване середовище розробки, спеціально розроблене для веб-розробки. Він пропонує широкий спектр функцій та інструментів, які задовольняють потреби веб-розробників, забезпечуючи покращений досвід кодування та підвищення продуктивності.

2.9 Висновки за розділом

Було здійснено огляд популярних фреймворків для розробки веб-застосунків: React, Angular, Vue, та аргументовано вибір одного з них - React. Були розглянуті особливості отримання даних для відображення, клієнт-серверна архітектура та принципи RESTful API. Аргументовано вибір інструментарію для розробки веб-застосунку.

Для розробки використовується мова програмування TypeScript з використанням фреймворку React. В допомогу до реакту, задля контролю стану веб-застосунку використовується бібліотека Redux Toolkit.

Здійснено огляд середовища для розробки WebStorm, яке було обране для розробки програмного продукту.

РОЗДІЛ 3
РОЗРОБКА ТА ІНТЕГРАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

3.1 Проектування системи

Замовником було поставлене завдання створити frontend частину додатку на базі single-page application (SPA). Веб-додаток повинен бути розроблений за допомогою мови програмування Typescript, задля більш зручної підтримки в майбутньому. Як основну технологію розробки, після узгодження, було обрано – React фреймворк. Більш детальний аналіз та аргументація вибору саме його – наведені у другому розділі даної роботи.

Веб-додаток має виконувати функції з дистанційного управління пристроями, надання доступу до телеметрії пристрою, менеджменту пристроїв та користувачів. Повинно бути 4 ролі користувачів (Admin, retailer, manager, customer) та можливе листування між ними.

Веб-застосунок складається з двох частин:

- Backend – ця частина розроблена за допомогою такого технологічного стеку: PHP, Laravel, MySQL – та в нашому випадку вже є реалізованою.
- Frontend – SPA, React, Typescript.

На рисунку 3.1 зображено діаграму послідовності, яка відображає взаємодію частин веб-застосунку між собою.

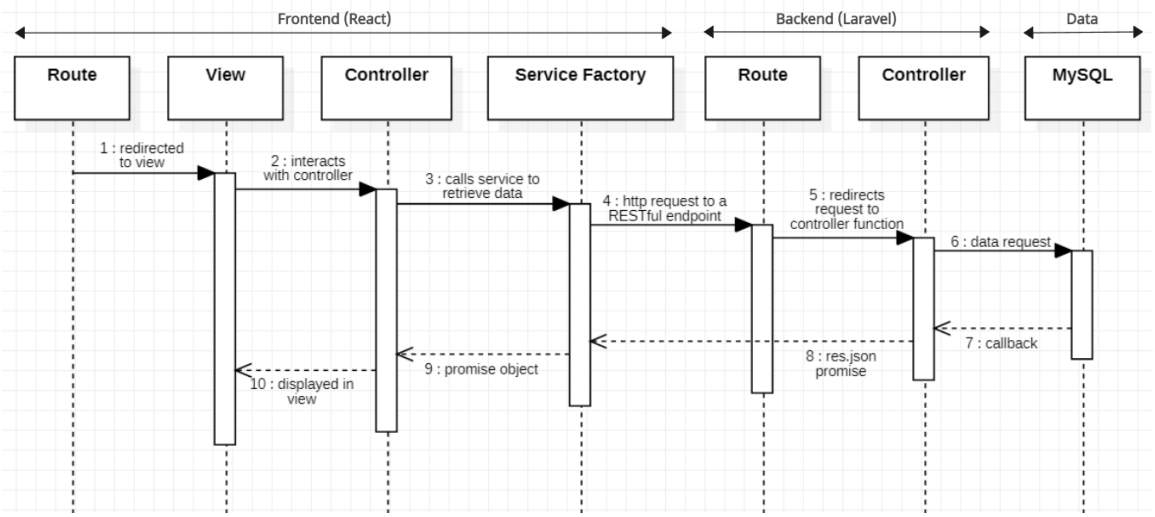

Рисунок 3.1 – Діаграма послідовності

З рисунку можна зазначити, що дана діаграма складається з трьох частин, а саме:

- Frontend – включає в себе такі об’єкти: Route, View, Controller, Service Factory.
- Backend – містить такі об’єкти: Route, Controller.
- Data – відображає базу даних MySQL.

Дана діаграма визначає типову послідовність подій у взаємодії частин веб-застосунку. Об’єкт Route відповідає за переадресацію користувача між об’єктами View, який в свою чергу взаємодіє з контролером. Після цього контролер визиває об’єкт сервісу, який відповідає за запити до серверної частини застосунку.

Сервіс звертається до спеціальної кінцевої точки (endpoint) серверної частини об’єкту Route. До кожної кінцевої точки приєднаний свій контролер, який відповідає за обробку HTTP-запитів, що може включати автентифікацію, маршрутизацію та виклик відповідної бізнес-логіки. Контролер взаємодіє з базою даних: виймає, вставляє, корегує та видаляє певні дані, надсилаючи необхідні SQL або інші специфічні для бази даних команди. Після виконання поставленої операції, контролер генерує відповідь на основі отриманих даних та іншої обробки. Після чого надсилає відповідь назад до сервісу фронтенду, який в свою чергу передає його контролеру. Він отримує відповідь і може обробити його, щоб отримати необхідні дані. Після обробки дані передаються до об’єкту View, який відтворює дані, отримані з серверної частини, за потреби оновлюючи інтерфейс користувача.

3.1.1 Функціональні можливості користувачів

За завданням від замовника, у веб-додатку повинно бути реалізовано розмежування можливостей користувачів відповідно до їх ролі. Усього веб-додаток буде налічувати 4 ролі, а саме: Admin, manager, retailer та customer. Саме у такому порядку змінюються функціональні можливості користувачів.

Задля більш детального розуміння функціональних можливостей користувачів з певними ролями, було розроблено діаграми прецедентів під кожну з них.

Діаграма прецедентів можливостей користувача з роллю «Admin», зображена на рисунку 3.2.

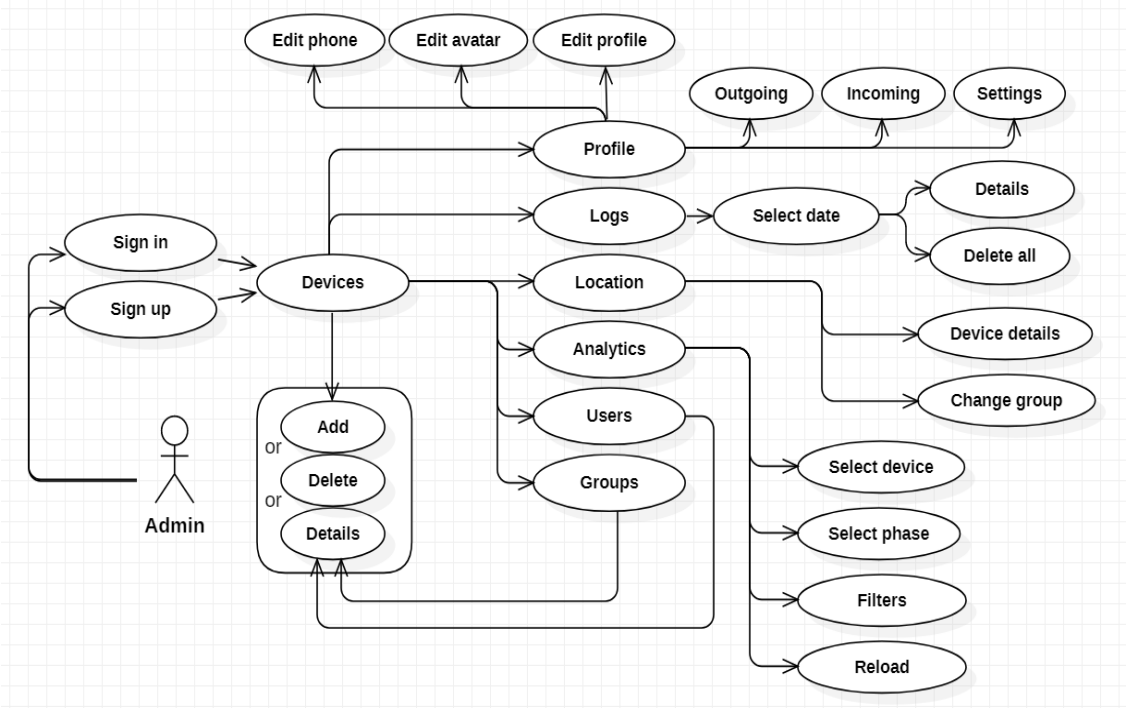

Рисунок 3.2 – Діаграма прецедентів ролі «Admin»

Роль адміністратора знаходиться на найвищому рівні можливостей взаємодії у веб-додатку. Вона наділена усіма функціональними можливостями по контролю користувачів та пристроїв, включаючи видалення, редагування та додавання. Також користувачам з роллю адміністратора доступна сторінка логів, в яку записується уся активність користувачів у вигляді екшенів.

Адміністратор може переглядати та редагувати майже усю інформацію про користувачів та їх пристрої, окрім платіжних даних та правил, призначених для певних пристроїв. Також користувач з даною роллю не може видаляти собі подібних – інших адміністраторів, в той час коли будь-яких інших користувачів – може.

Слідуючою за старшинством роллю йде Manager, діаграма прецедентів можливостей користувача з даною роллю зображена на рисунку 3.3.

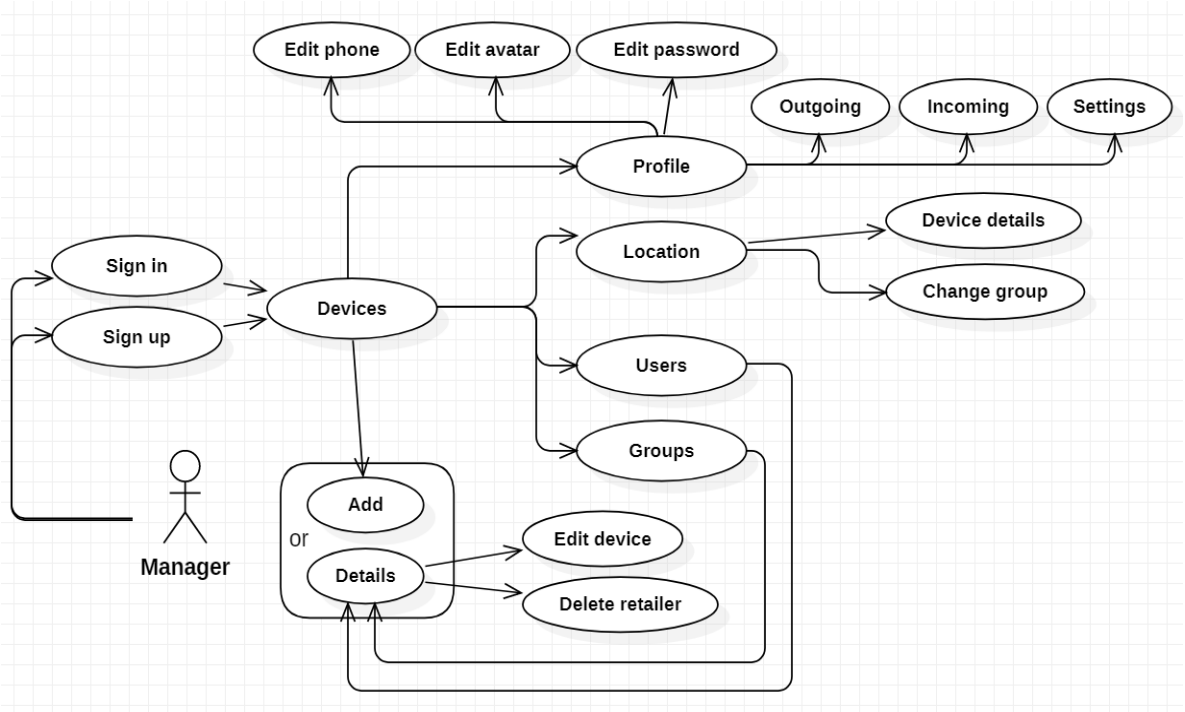

Рисунок 3.3 – Діаграма прецедентів ролі «Manager»

Як можна побачити з діаграми, можливості користувача з роллю Manager значно зменшились в порівнянні з адміністратором. Йому недоступні вкладки логів та аналітики пристроїв.

Йому доступні можливості додавання та редагування користувачів та пристроїв, в той час як видалення – вже не є доступним. Також якщо подивитись на вкладку профілю, йому недоступна можливість цілком редагувати свій профіль, а лише – змінювати пароль, номер телефону та аватар. Як і у ролі адміністратора, в нього є можливість редагування пристроїв, тож менеджер може видаляти та прикріпляти нових ретейлерів до певних пристроїв.

Усі інші можливості залишились незмінними як у адміністратора, до них входять вкладки: місцезнаходження – переглядати де саме знаходяться пристрої на мапі та переглядати коротку інформацію про них, профіль – вхідні та вихідні повідомлення та налаштування, групування пристроїв.

Іншою за старшинством йде роль Retailer, діаграма прецедентів можливостей користувача з даною роллю зображена на рисунку 3.4.

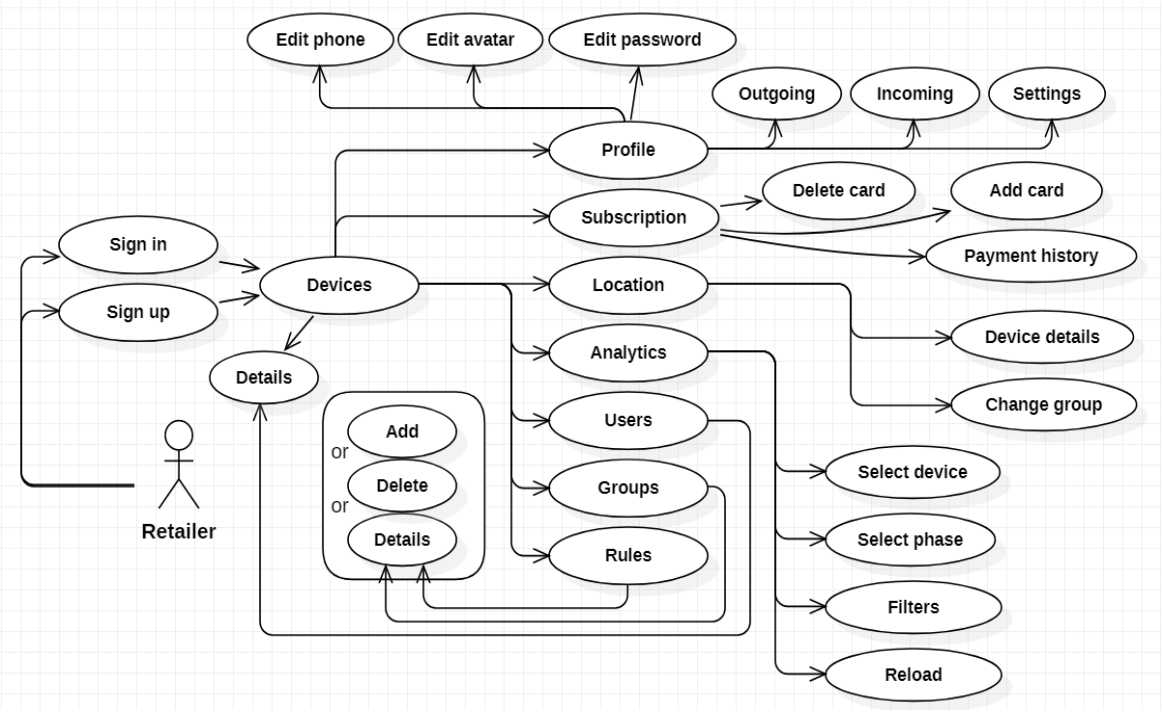

Рисунок 3.4 – Діаграма прецедентів ролі «Retailer»

Як можна почити з даної діаграми, у ролі Retailer з’являються нові вкладки, а саме: Rules та Subscription.

Вкладка правил (rules) відповідає за виставлення, редагування та видалення правил роботи пристроїв. Привила можна додавати як до окремого так і до груп пристроїв, тож можна специфікувати роботу певних пристроїв у певний час.

Вкладка під назвою підписка (subscription) відповідає за відображення тарифного плану користувача з даною роллю. В ній знаходиться інформація про поточний тарифний план, усі плани доступні у веб-додатку, платіжна історія та функціональність взаємодії з картками.

Дана роль обмежена за можливостями по взаємодії з користувачами та пристроями, не можна редагувати та видаляти. Інші вкладки залишилися без змін, а саме: профіль, місцезнаходження та аналітика пристоїв.

І останньою за старшинством роллю є Customer, діаграма прецедентів можливостей користувача з даною роллю зображена на рисунку 3.5.

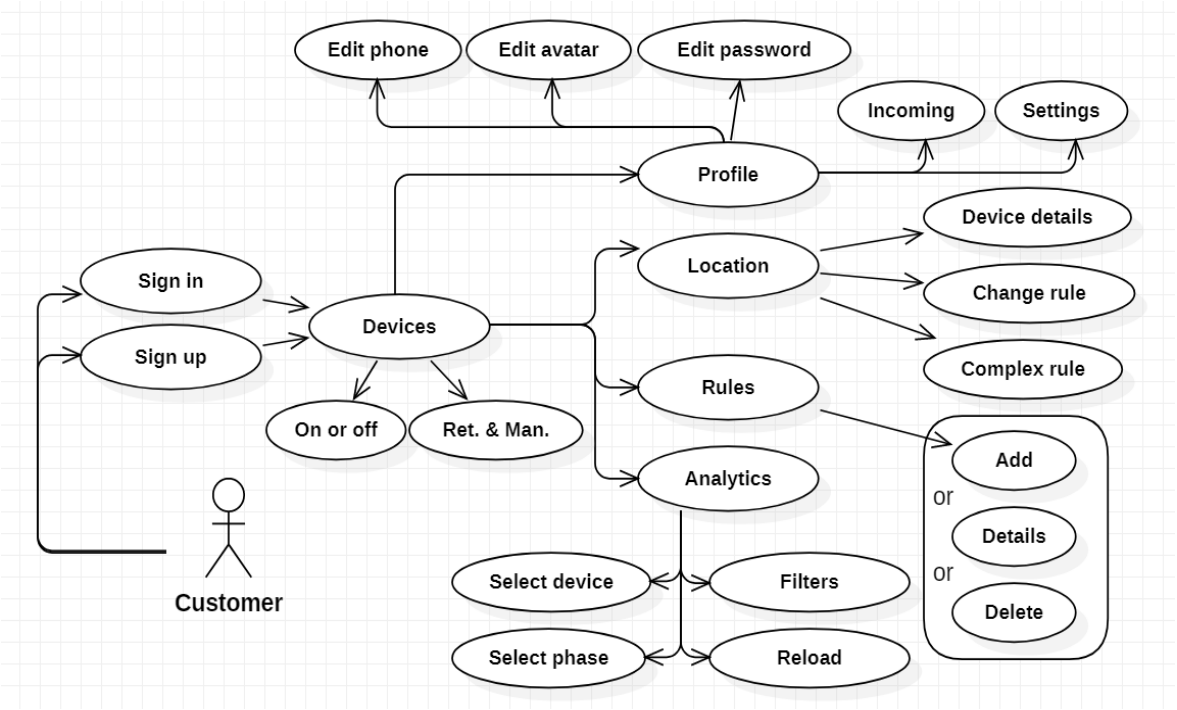

Рисунок 3.5 – Діаграма прецедентів ролі «Customer»

Як можна побачити з діаграми, якщо порівнювати дану роль з ретейлером, то тут відсутні вкладки групування, підписки та користувачів. Користувач з даною роллю не може переглядати інших користувачів та їх пристрої, та відповідно провидити будь-які інші дії з ними.

Вкладки аналітики та правил залишилися без змін. В той час як з вкладки профілю зникла можливість відправки вихідних повідомлень. Інші вкладки також потерпіли таких змін:

- місцезнаходження – додана функціональність перегляду комплексних правил (свої + ретейлерів);
- пристрої – перегляд тільки своїх пристроїв з більш деталізованою інформацією з можливістю включення та виключення певних частин, а також перегляд інформації про ретейлерів та менеджерів по кожному девайсу.

3.1.2 Алгоритм входу до веб-додатку

Першим, що побачить користувач буде сторінка логіна, тож потрібно

більш поглиблено розуміти алгоритм роботи даної сторінки. Даний алгоритм зображений у вигляді діаграми послідовності на рисунку 3.6.

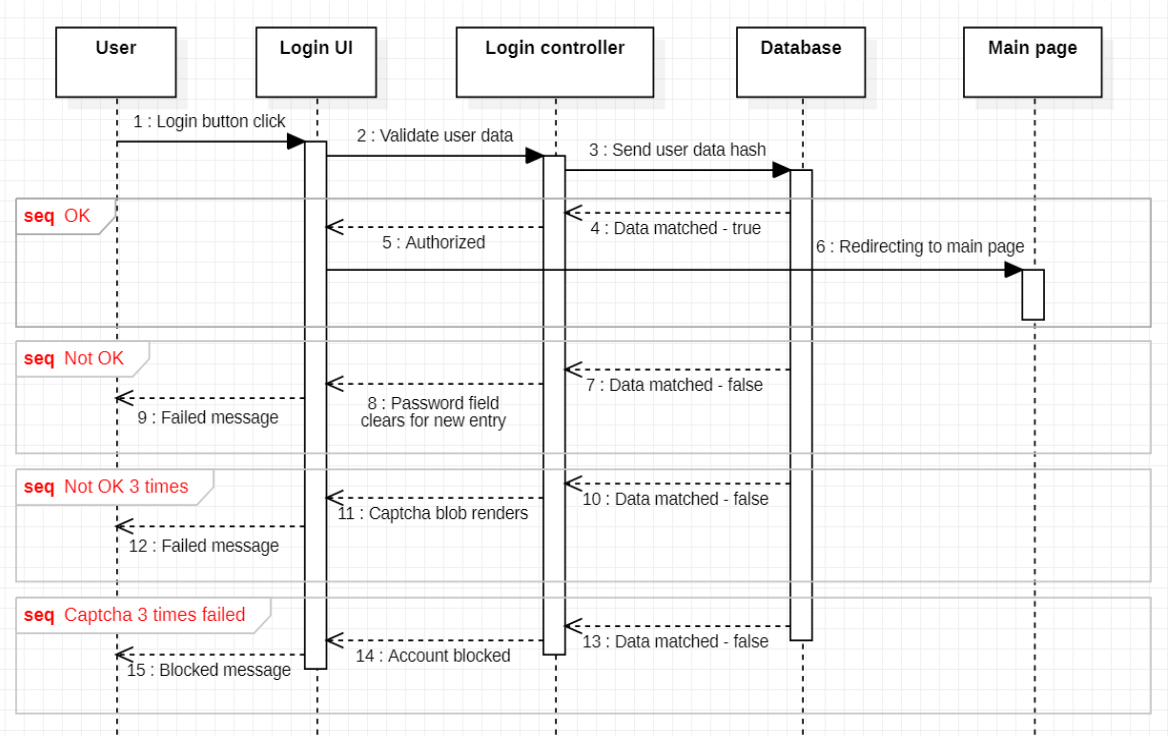

Рисунок 3.6 – Діаграма послідовності алгоритму логіну

Дана діаграма відображає алгоритм роботи аутентифікації та авторизації до веб-застосунку.

Спочатку користувач вводить персональні дані, після чого натискає на кнопку входу. Після чого дані перевіряються за певними параметрами та передаються до контролеру, який взаємодіє з базою даних. Якщо дані співпадають – користувачу присвоюється статус авторизованого та він перенаправляється до головної сторінки.

Якщо дані не співпадають з тими що є в базі даних – користувачу присвоюється статус неавторизованого, та виводиться відповідне повідомлення. Після чого користувач пробує змінити дані, та механізм повторюється.

Якщо користувач вводив невірні дані 3 рази – на екрані з’являється капча, без вводу якої послідуючі спроби входи не є можливими. Механізм

повторюється, але у цей раз окрім персональних даних, користувач вводить ще й код з капчі.

У разі якщо користувач вводить невірні дані з капчею ще 3 рази – йому присвоюється статус заблокованого, та висвічується відповідне попередження. При блокуванні аккаунту, користувач не може увійти навіть з правильними даними до того моменту, поки блокування не завершиться.

3.1.3 RESTful API

Як вже було зазначено, веб-додаток складається з двох частин: фронтенду та бекенду. Проектування бекенду проводилося за принципами архітектурного стилю REST.

Обмін даними проводиться за допомогою ресурсів, як правило у форматі JSON (нотація JS). Доступ до ресурсів отримується шляхом запиту до певного, обов’язково унікального URL, які ще називаються кінцевими точками. Як правило, при описі кінцевих точок базовий URL ігнорується, бо є спільним для усіх точок.

Усі кінцеві точки проекту розкривати я не маю можливості, через підписання угоди про нерозголошення, але можу навести для прикладу декілька з них:

- /login – для здійснення аутентифікації та отримання токену доступу.
- /refresh – для здійснення поновлення токену доступу.
- /subscription – задля отриманні інформацію про підписку.
- /devices – задля отримання пристроїв.
- /customers – задля отримання користувачів.
- /messages – задля отримання повідомлень.
- /user/captcha – для отримання капчі.

Всі запити додатку, окрім аутентифікації, повинні мати заголовок з токеном доступу, саме з його поміччю бекенд розуміє чи авторизований

користувач. Цей заголовок має такий вигляд: `Authorization: <acess_token>`, та у разі його відсутності, користувачу замість ресурсу з певною інформацією, буде повертатись помилка зі статусом 401 Unauthorized.

3.1.4 База даних

Задля зберігання даних в веб-додатку використовується реляційна модель. На рисунку 3.7 зображена структура таблиць бази даних та їх відносини.

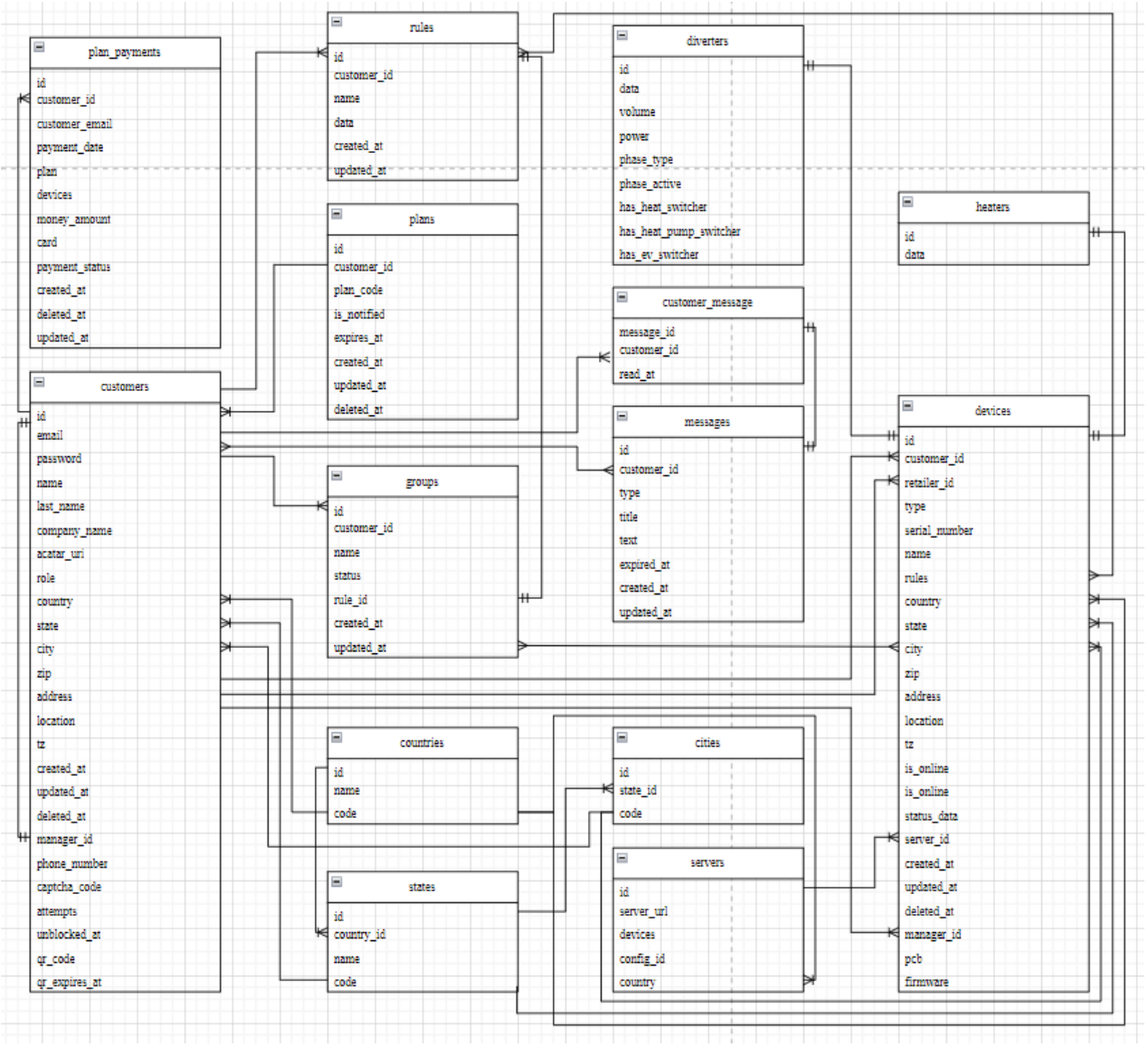

Рисунок 3.7 – Структура таблиць БД

Таблиця «customers» відображає користувача додатку. В ній зберігаються персональні дані користувача. Користувач може мати багато пристроїв, груп, правил, повідомлень та платежів, тож з цими таблицями користувач має зв’язок «один до багатьох». До кожного користувача з роллю customer підключений свій менеджер, який може бути тільки один, тож тут зв’язок між ними «один до одного». Якщо ж у користувача роль retailer – в нього є свій тарифний план (підписка), та відповідно платежі за цією підпискою.

Таблиця «devices» відображає пристрої користувачів додатку. В ній зберігається інформація про усі пристрої. Кожен пристрій належить користувачу, в нього є свій менеджер та ретейлер («один до багатьох»). В залежності від типу пристрою, телеметрична інформація пристроїв зберігається у двох таблицях – heaters та diverters та має зв'язок з ними «один до одного».

Таблиця «rules» зберігає в собі інформацію про правила користувачів до певних пристроїв, або груп пристроїв. До кожної групи може прикріплятися одне певне правило («один до одного»), в той час як до конкретного пристрою – декілька («багато до багатьох»).

Таблиця «plans» зберігає інформацію про доступні тарифні плани користувачів.

Таблиця «plan_payments» – інформація про платежі за відповідним тарифним планом.

Таблиця «groups» зберігає інформацію про пристрої додані до певної групи та її правило.

Таблиця «messages» – інформація про повідомлення конкретних користувачів. Користувач може мати багато повідомлень, як і одне повідомлення може мати багато користувачів, тому зв’язок між ними «багато до багатьох». Дана таблиця зв’язана з таблицею «customer_message», яка зберігає інформацію про статус повідомлення (було воно прочитане, чи ні), зв’язок між ними – «один до одного».

Таблиці «diverters» та «heaters» – зберігають інформацію про телеметрію певного пристрою за його типом відповідно.

Таблиця «countries» – зберігає інформацію про країни, в одній країні може бути безліч користувачів та пристроїв.

Таблиця «states» – зберігає інформацію про штати країн, також як і країна, може налічувати безліч пристроїв та користувачів.

Таблиця «cities» – зберігає інформацію про міста штатів, може також мати безліч пристроїв та користувачів.

Таблиця «servers» – зберігає інформацію про сервера, до яких підключені пристрої, до одного серверу може бути підключено безліч пристроїв, а серверів може багато в певній країні.

3.2 Програмування системи

Для забезпечення безпеки та конфіденційності проекту між залученими сторонами було укладено угоду про нерозголошення. Ця угода підкреслює важливість захисту конфіденційної інформації, комерційної таємниці та будь-яких інших конфіденційних даних, пов’язаних із проектом. Як особа, яка підписала NDA, я зобов’язаний дотримуватися її умов щодо конфіденційності, які вона передбачає. Але вищим керівництвом було дозволено висвітлювати моменти, які не є власним рішенням.

3.2.1 Сторінка логіну

Робота веб-застосунку починається зі сторінки логіну, тож першим кроком потрібно було реалізовувати її. Алгоритм роботи даної сторінки був наведений у пункті 3.1.2. На рисунку 3.8 зображений зовнішній вигляд сторінки логіну.

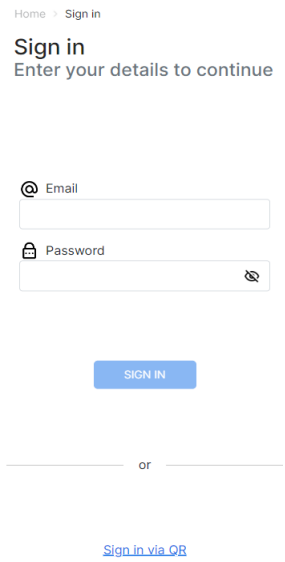

Рисунок 3.8 – Сторінка логіну

Після введення персональних даних користувачем та натискання на кнопку «Sign in», посилається запит до бекенду з введеними даними, у разі успіху користувач переходить до сторінки пристроїв.

Якщо ж користувач вводить не вірні дані – користувачу виводиться сповіщення з помилкою, яке зображене на рисунку 3.9.

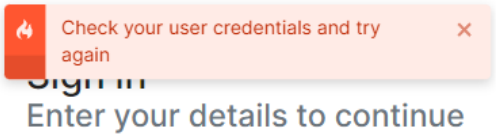

Рисунок 3.9 – Сповіщення про невдалий логін до веб-додатку

Дане сповіщення було реалізовано за допомогою сторонньої бібліотеки під назвою «react-toast-notifications». Реалізація виводу сповіщення наведена у лістингу 3.1.

Лістинг 3.1 – Реалізація сповіщень

```
import {useToasts} from "react-toast-notifications";
const {addToast} = useToasts();
```

```
addToast(res.error?.error_description || "Check your user credentials and try again ",
{appearance: "error"});
```

Якщо користувач 3 рази вводить неправильні дані, на екрані з’являється капча, без вводу якої користувач не зможе увійти до веб-додатку.

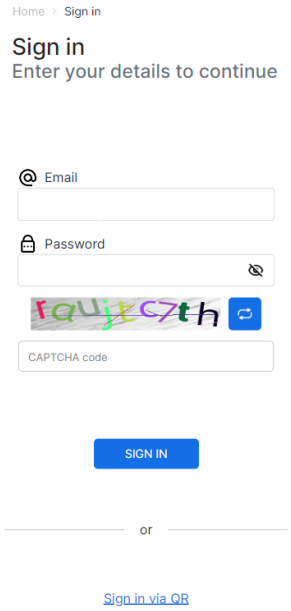

Рисунок 3.10 – Сторінка авторизації користувача з тестом підтвердження

Як можна побачити з рисунку, біля капчі є кнопка для її оновлення, у разі якщо вона є незрозумілою. По натисканню на неї, відправляється новий запит до серверу на капча перемальовується. Також був реалізований механізм по запам’ятовуванню імейлу користувача до котрого прийшла капча, тож якщо користувач буде вводити новий – капча пропаде. Реалізація виводу капчі зображена у лістингу 3.2.

Лістинг 3.2 – Реалізація виводу капчі

```
    addToast(res.error?.error_description || "Check your user credentials and try
again ", {appearance: "error"})
    setCaptchaImage(res.error.captcha_code)
    setUserEmail(emailContext.text)
    setCaptchaShow(true)
}
```

```
. . .
{
    captchaShow &&
        <div className={"d-flex flex-column justify-content-center"}>
            <div className={"d-flex ms-3 me-3 mt-2 mb-2 justify-content-end sign-in-
padding"}>
                <img className={"sing-in-captcha"} src={captchaImage}/>
                <LibraButton icon={ReloadIcon2} iconSize={20} cornerRadius={5}
height={40} width={40} iconColor={"color-ffffff"} backgroundHover={"#005CD3"}
background={"#146FE7"} onClick={reloadCaptcha}/>
            </div>
            <TextInput error={captchaInput.error} errorLabel={captchaInput.message}
height={38} watermark={"CAPTCHA code"} maxLength={10} onInput={onCaptchaInput} />
        </div>
}
```

Якщо користувач продовжує вводити невірні дані ще 3 рази, в кінцевому випадку його аккаунт блокується і виводиться відповідне повідомлення.

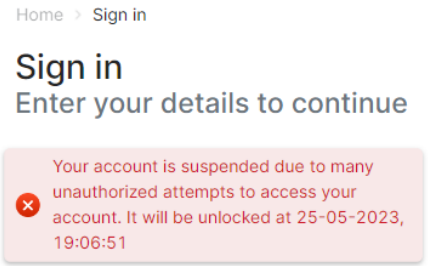

Рисунок 3.11 – Повідомлення про блокування аккаунту

Після блокування аккаунту, користувач не зможе зайти до веб-додатку, поки аккаунт не буде розблокований, навіть якщо буде вводити вірні дані.

Якщо ж користувач вводив правильні дані та в кінцевому випадку був авторизований, він опиняються на сторінці з пристроями.

3.2.2 Сторінка пристроїв

В залежності від ролі користувача, сторінка пристроїв може приймати

різний вигляд. Якщо користувач має роль адміністратора, йому доступні усі пристрої. Зовнішній вигляд сторінки зображений на рисунку 3.12.

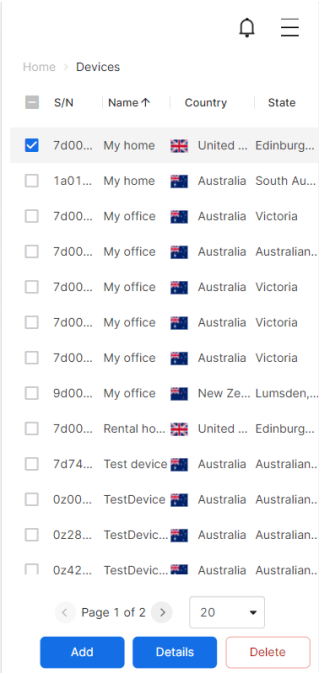

Рисунок 3.12 – Сторінка пристроїв з ролі адміністратора

Як можна зазначити з рисунку, пристрої представлені у вигляді таблиці. В нижній половині екрану знаходяться кнопки для взаємодії з певним пристроєм. Кнопка «Add» відповідає за додавання нового пристрою, «Details» - відкриває діалогове вікно з детальною інформацією про певний пристрій, а «Delete» - видаляю певний, або декілька пристроїв. Окрім кнопок в нижній частині, можна зазначити ще й можливості пагінації, можна перемикатися між сторінками, якщо усі пристрої не влізли на одній, а також змінювати кількість пристроїв, які будуть відображатися на одній сторінці.

Дана таблиця була реалізована з використанням сторонньої бібліотеки, під назвою «ag-grid-react». Задля повторного використання, було розроблено окремий компонент, в який за допомогою пропсів в подальшому будуть передаватися дані для заповнення даної таблиці.

Лістинг 3.3 – Головний компонент таблиці

```
<AgGridReact  defaultColDef={defaultColDef}
              ref={ref}
              overlayNoRowsTemplate={
                  '<span>No rows to show</span>'
              }
              onModelUpdated={(e)=>{
                  if (prop.selectionRows &&
ref?.current?.api?.getSelectedRows().length === 0 && prop.selectionRows.length > 0 &&
canModelUpdated)
                      selectRows(prop.selectionRows);
                  canModelUpdated = true;
              }}
              isRowSelectable={isRowSelectable}
              getRowStyle={getRowStyle}
              pagination={prop.isPagination}
              paginationPageSize={prop.pageSize ?? 20}
              paginationNumberFormatter={paginationNumberFormatter}
              rowMultiSelectWithClick={false}
              enableBrowserTooltips={true}
              suppressPaginationPanel={true}
              rowSelection={prop.multipliesSelection ? 'multiple' : 'single'}
              onGridReady={onGridReady}
              getRowId={getRowId}
              className={`ag-theme-alpine`}
              rowClassRules={rowClassRules}
              onRowDataChanged={prop.rowDataChanged}
              onSelectionChanged={onSelectionChanged}
              onPaginationChanged={onPaginationChanged}
              rowData={prop.rows}
>
    {columns}
</AgGridReact>
```

Як можна зазначити з лістингу, в даному компоненті проходить налаштування функцій таблиці, передавання даних та її стилізування. В середину компоненту передається константа `columns`, яка має такий вигляд. (Лістинг 3.4)

Лістинг 3.4 – Колонки таблиці

```
const columns = prop.columns.map((c, index) => (

    <AgGridColumn key={index}
                  filter={true}
                  maxWidth={c.maxWidth}
                  minWidth={c.minWidth}
                  width={prop.gridRef ? prop.gridRef[index]?.num ??  c.width :
c.width}
                  initialSort={sortInit(c)}
                  tooltipValueGetter={c.disableTooltip ? undefined :
(c.tooltipValueGetter ? c.tooltipValueGetter : (params:any)  =>
(params.data[c.name]))}
                  cellRenderer={c.cellRender ? (params:any) => c.cellRender!(params)
: undefined}
                  valueGetter={c.valueGetter}
                  headerClass={prop?.multipliesSelection === true ? index === 0 ?
"ag-header-first-cell-custom-class" : "ag-header-cell-custom-class" : "ag-header-
cell-custom-class"}
                  headerName={c.title}
                  field={c.name}/>
))
```

За допомогою методу ітерації масиву «map», ми проходимо по кожній колонці та обертаємо її в компонент колонки із бібліотеки з певними налаштуваннями.

Після розробки даного компоненту, ми можемо використовувати його для відображення великих об’ємів даних всього лише передавши в нього певні параметри, такі як: колонки та рядки таблиці, параметри зі стилізування даних та інші налаштування на прикладі кількості рядків на одній сторінці таблиці.

У разі якщо користувач має роль менеджера, сторінка буде мати той самий вигляд і функціонал. В той час як у ролі ретейлера – залишиться можливість лише переглядати деталі певного пристрою.

Якщо користувач має роль звичайного користувача, сторінка пристроїв буде приймати такий вигляд. (Рис. 3.13)

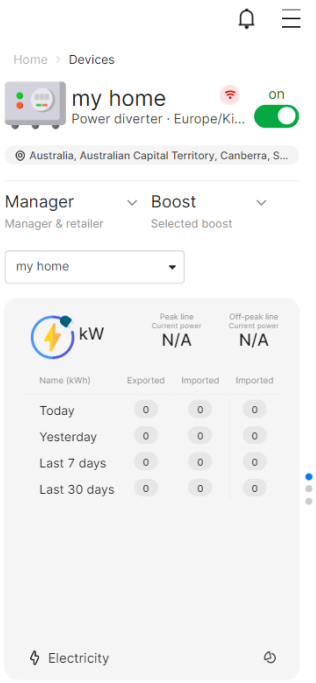

Рисунок 3.13 – Сторінка пристроїв ролі користувача

Зовнішній вигляд сторінки повністю змінюється, становиться більш інформативним. Користувач може переглядати лише пристрої, які до нього прикріплені.

Дану сторінку можна розділити на 2 частини, верхня частина включає в собі інформацію про тип пристрою, його повну адресу, є можливості з перегляду інформації про менеджера та ретейлера пристрою, включення та виключення, та налаштування так званого буста пристрою. Також можна переключатися між пристроями за допомогою випадаючого списку, який зображений на рисунку 3.14.

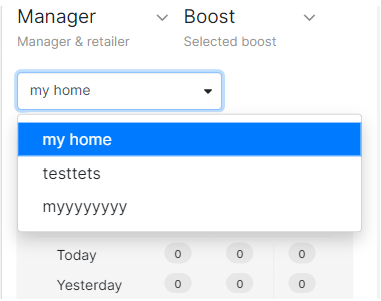

Рисунок 3.14 – Список доступних пристроїв

Випадаючий список був реалізований з використанням бібліотеки «react-widgets», використання якої наведене у лістингу 3.5.

Лістинг 3.5 – Використання випадаючого списку

```
      <DropdownList filter={null}  style={{height:40, paddingRight:140,
marginLeft:5}} value={selected.name} data={globalDevices.devices ?
globalDevices.devices.map(device=>device.name) : undefined} onSelect={(dataItem,
metadata)=>{
    let index = globalDevices.devices ? globalDevices.devices.findIndex(item =>
item.name===dataItem) : -1;
    if (index !== -1)  setSelectedDeviceIndex(index);
}}/>
```

В нижній частині екрану можна побачити карточки з детальною інформацією про статистику роботи пристрою. Переключатися між ними можна за допомогою свайпів доверху та донизу. Функціональність з перемикання карточок була реалізована з використанням бібліотеки React під назвою «swiper/react», і представлена у лістингу 3.6.

Лістинг 3.6 – Реалізація механізму перемикання між карточками

```
<Swiper speed={300} slidesPerView={1} direction={"vertical"} navigation={{ prevEl:
'.swiper-prev', nextEl: '.swiper-next' }} pagination={{ clickable: true }} style={{
width:"100%",minHeight:"100%",  height:"100%", paddingRight:10,}}>
    <SwiperSlide  className={"swiper-slide"}  >
        {swiperSlideElectricityElement}
    </SwiperSlide>
    <SwiperSlide  className={"swiper-slide"} >
        {swiperSlideHotWaterElement}
    </SwiperSlide>
    <SwiperSlide  className={"swiper-slide"}  >
        {swiperSlideEVElement}
    </SwiperSlide>
</Swiper>
```

Як можна зазначити з лістингу, кожна карточка обертається в спеціальнім компонент `SwiperSlide`, а усі карточки в компонент `Swiper`.

3.2.3 Сторінка користувачів

Сторінка користувачів має такий самий зовнішній вигляд та функціонал, як і сторінка пристроїв, зображена на рисунку 3.15.

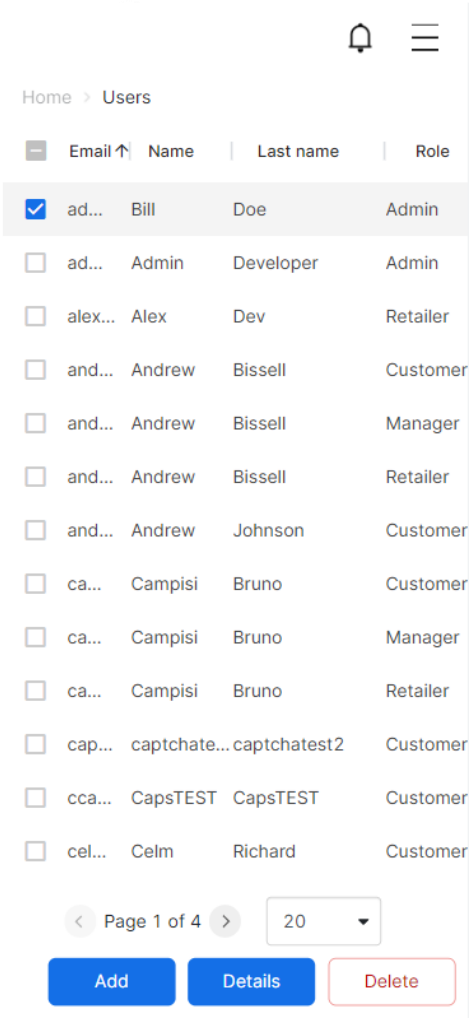

Рисунок 3.15 – Зовнішній вигляд сторінки користувачів

Вона була реалізована так само як і сторінка пристроїв, з тією самою логікою під кожну с ролей, окрім звичайного користувача, бо він не має доступу до цієї сторінки (вона попросту не відображається).

При натисканні на кнопку «Add» з’являється модальне вікно для заповнення даних нового користувача, яке зображене на рисунку 3.16.

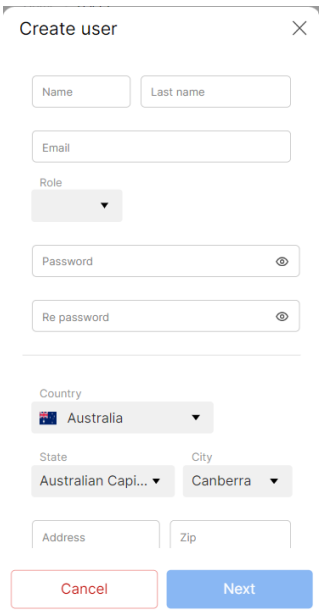

Рисунок 3.16 – Модальне вікно додавання нового користувача

Кожне поле даної форми перевіряється за певним фактором, і у разі якщо щось введено невірно – користувач побачить таку помилку. (Рис. 3.17)

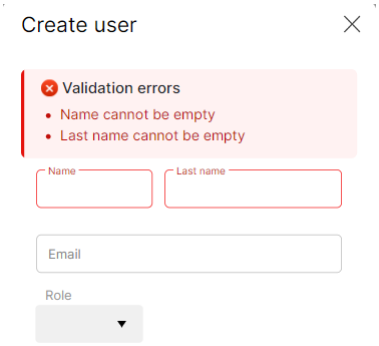

Рисунок 3.17 – Помилки при заповненні форми

Валідація полів для вводу була реалізована за допомогою популярної бібліотеки з валідації під назвою «react-hook-form» та винесено окремим компонентом.

Лістинг 3.7 – Реалізація механізму валідації

```
const formValidation = useForm<Inputs>({
    defaultValues: {"input": prop.defaultText},
    mode: prop.manualValidation ? undefined : "onChange",
```

```
    reValidateMode: "onChange",
})
<input type={type}
       autoComplete={"nope"}
       {...prop?.formValidation?.register("input", {
           required: prop.isRequired,
           maxLength: prop.maxLength,
           minLength: prop.minLength,
           pattern: prop.pattern,
           onChange: (event: any) => prop.onTextChange(event.target?.value)
       })}
>
```

При натисканні на кнопку «Details», з’являється модальне вікно з інформацією про обраного користувача, яке зображене на рисунку 3.18.

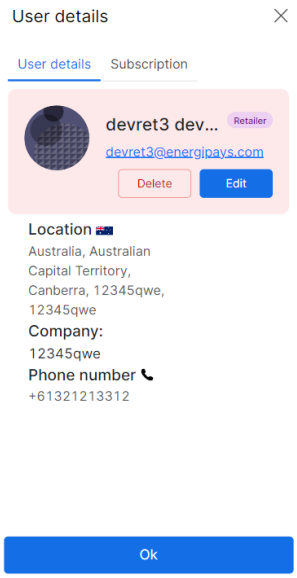

Рисунок 3.18 – Модальне вікно з інформацією про обраного користувача

Дане вікно наповнене інформацією про обраного користувача, так як зараз ми знаходимось на аккаунті з роллю адміністратора, нам доступні функції з видалення та редагування. Так як ми оглядаємо користувача з роллю ретейлера, можна зазначити, що діалогове вікно включає в собі 2 вкладки, при перемиканні на другу – ми побачимо інформацію про підписку даного користувача. (Рис. 3.19)

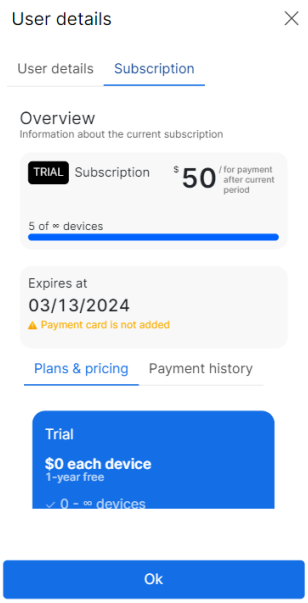

Рисунок 3.19 – Сторінка підписки користувача в модальному вікні

Тут представлена інформація про поточну підписку користувача, але більш детально ця сторінка буде розглянута далі.

При натисканні на кнопку «Delete» - користувачу буде виведено модальне вікно з повідомленням чи точно він хоче видалити обраного користувача. (Рис. 3.20)

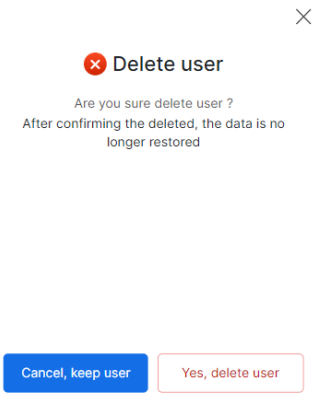

Рисунок 3.20 – Модальне вікно з видалення користувача

3.2.4 Сторінка підписок

Дана сторінка відображає тарифний план користувача, вона доступна лише користувачам з роллю ретейлера, як і тарифний план є тільки в них.

Зовнішній вигляд сторінки підписок зображений на рисунку 3.21.

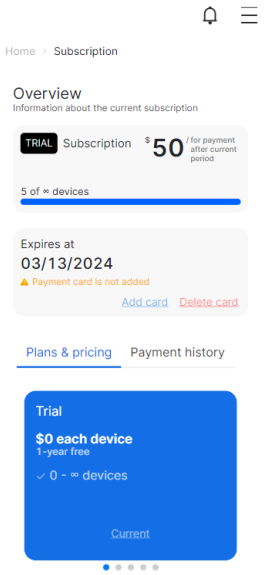

Рисунок 3.21 – Зовнішній вигляд сторінки підписок

Дану сторінку можна розділити на 2 частини, у верхній частині виводиться інформація про поточний тарифний план користувача, а саме: назва тарифного плану, кількість пристроїв підключених до користувача, сума до оплати за усі пристрої, дата завершення поточного плану та кнопки по роботі з картками. Нижня частина складається з двох вкладок, а саме:

- Plans & pricing – відображає поточний тарифний план та список усіх планів доступних у веб-додатку у вигляді свайперу.
- Payment history – відображає історію платежів за всіма планами, що були раніше.

Перша вкладка реалізована за допомогою бібліотеки «swiper/react». В цей раз кількість карточок може змінюватися, по мірі введення нових тарифних планів, тому вивід був перероблений. (Лістинг 3.8)

Лістинг 3.8 – Реалізація свайперу тарифних планів

```
<Swiper speed={300} navigation={{ prevEl: '.swiper-prev', nextEl: '.swiper-next' }}
pagination={{ clickable: true }}spaceBetween={planList.length}
slidesPerView={currentSlidesPerView}
>
```

```
 {
    planList.map((plan, index)=>{
let isCurrent = plan.plan_code.toLowerCase() === currentSubscription?.plan_code;
return (
        <SwiperSlide className={"subscription-swiper-slide"} key={plan.plan_id}>
                    <div>INFO</div>
        </SwiperSlide>
       );
    })
 }
</Swiper>
```

Друга вкладка реалізована з використанням тієї ж самої таблиці, яка була використана на сторінці пристроїв та користувачів та зображена на рисунку 3.22.

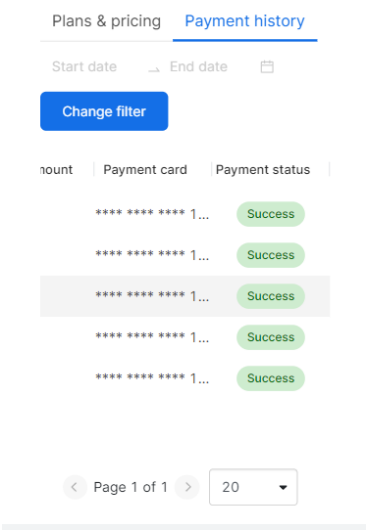

Рисунок 3.22 – Зовнішній вигляд вкладки з історією платежів

Як можна зазначити з рисунку, знизу немає ніяких кнопок, бо історія платежів доступна лише для перегляду, не можна видаляти або редагувати її. Але у верхній частині є кнопка «Change filter», яка відповідає за фільтрацію платежів за певною датою, а над нею виведене поле для відображення даного фільтру. При натисканні користувачу буле виведене модальне вікно, в якому можна обрати певний часовий період. (Рис. 3.23)

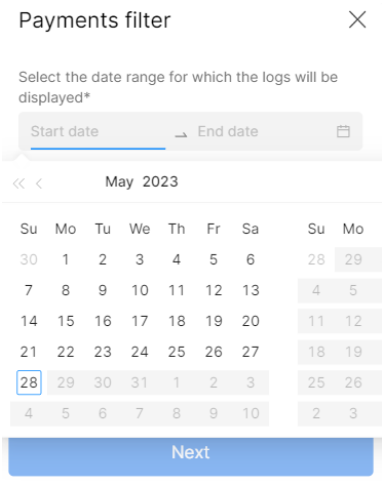

Рисунок 3.23 – Модальне вікно з вибором дати

Панель з вибором дати була реалізована за допомогою бібліотеки під назвою «ant-design», реалізація зображена у лістингу 3.9.

Лістинг 3.9 – Реалізація поля для вибору дати

```
<RangePicker bordered={false}  defaultValue={prop.intervalDate ?
[dateToAnyMoment(prop.intervalDate[0]), dateToAnyMoment(prop.intervalDate[1])] :
undefined} use12Hours={true} dropdownClassName={"antd-time-picker-range-picker-
dropdown"} onChange={onRangePickerChange} disabledDate={d => !d ||
d.isAfter(formatDate(dateToNextDay()))}/>
```

Після того як користувач вибере дату для фільтрації та натисне на кнопку «Next», модальне вікно закриється, а список платежів буде відфільтровано, також обрана дата буде відображатися над кнопкою по зміні фільтру.

3.2.5 Сторінка місцезнаходження

Дана сторінка представляє з себе карту, на якій можна буде бачити пристрої та їх точне місцезнаходження. Зовнішній вигляд даної сторінки зображений на рисунку 3.24.

Рисунок 3.24 – Зовнішній вигляд сторінки місцезнаходження

Як можна побачити з рисунку, на карті можна побачити точку з пристроєм. Також можна побачити кнопку «Add to group», яка відповідає за додавання пристроїв до груп прямо з мапи. При натисканні на пристрій – відкриється модальне вікно з короткою інформацією про даний пристрій, яке зображене на рисунку 3.25.

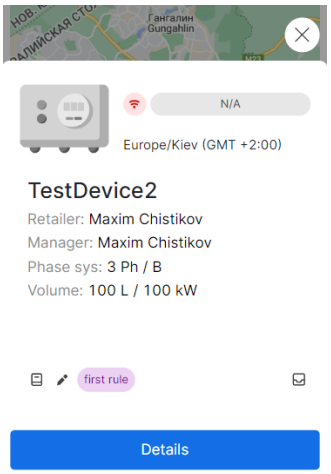

Рисунок 3.25 – Модальне вікно обраного пристрою

Тут можна побачити коротку інформацію про пристрій, а в нижній частині знаходяться такі кнопки:

- Перша кнопка (перша з лівого краю) – відповідає за відображення даних про усі правила даного пристрою.
- Друга (друга з лівого краю) – відповідає за редагування правила даного пристрою.
- Третя – (з правого краю) – відповідає за редагування групи даного девайсу.
- «Details» – перенаправляє користувача на сторінку пристроїв та відкриває модальне вікно з деталями користувача.

Вивід мапи був реалізований за допомогою бібліотеки «google-map-react», реалізація виводу карти наведена у лістингу 3.10.

Лістинг 3.10 – Реалізація виводу мапи

```
<GoogleMapReact defaultZoom={prop.zoom}
                onClick={prop.onClickMap}
                onDrag={prop.onDrag}
                defaultCenter={prop.center}
                style={prop.style}
                draggable={draggable}
                resetBoundsOnResize
                onGoogleApiLoaded={prop.onGoogleApiLoaded ? ({ map, maps }: any) =>
prop.onGoogleApiLoaded!(map, maps) : undefined}
                yesIWantToUseGoogleMapApiInternals
>
    { prop.children }
</GoogleMapReact>
```

Вона обернена в окремий компонент, тож потім за допомогою `prop.children` виставляються точки з пристроями на даній мапі.

3.2.6 Сторінка профілю

Сторінка профілю включає в себе інформацію про аккаунт користувача, його вхідні та вихідні повідомлення та налаштування. Зовнішній вигляд сторінки профілю наведений на рисунку 3.26.

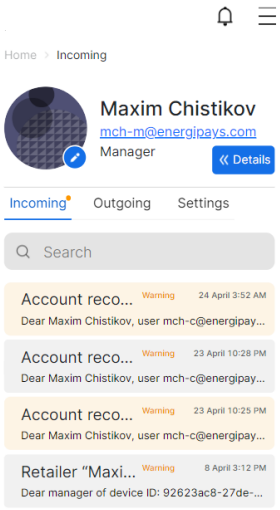

Рисунок 3.26 – Зовнішній вигляд сторінки профілю

Як можна побачити з рисунку, сторінка складається з двох частин, у верхній знаходиться інформація про користувача, а в нижній 3 вкладки:

- Incoming – відображає вхідні повідомлення користувача.
- Outgoing – вихідні повідомлення користувача.
- Settings – налаштування аккаунту користувача.

В частині з інформацією користувача є можливості зі зміни аватару користувача та відкриття деталей даного профілю. При натисканні на кнопку «Details» - з’являється модальне вікно з іншою інформацією користувача. (Рис. 3.27)

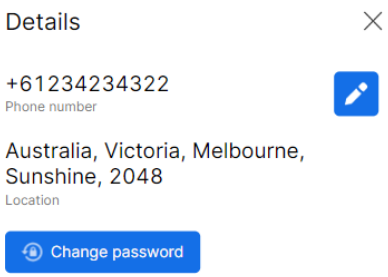

Рисунок 3.27 – Модальне вікно з інформацією користувача

В ньому доступні функції зі зміни номеру телефону та паролю, при натисканні на дані кнопки – поверх цього модального вікна відкриється відповідне з них. (Рис. 3.28 – 3.29)

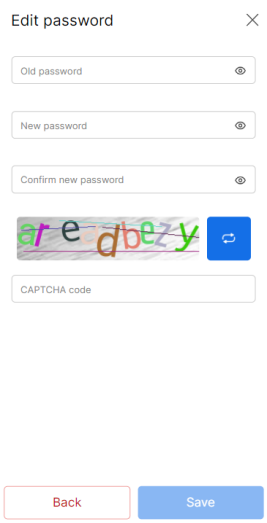

Рисунок. 3.28 – Модальне вікно зміни паролю

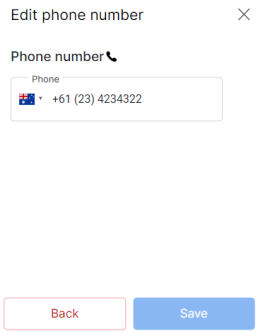

Рисунок 3.29 – Модальне вікно зміну номеру

При натисканні кнопок «Back» або «Save», користувач повернеться назад до вікна з детальною інформацією про профіль. Як можна зазначити з рисунку зміни паролю, там присутня капча, яка є обов’язковою для вводу. Вивід капчі реалізований так само як і при логіні до веб-додатку (отримується з бекенду та виводиться як картинка).

Усі поля вікна зміни паролю та редагування номеру перевіряються за певними правилами з допомогою методу із пункту 3.2.3 цього розділу.

Вкладка Incoming включає в себе усі вхідні повідомлення користувача та фільтр повідомлень при натисканні на який – він розкривається. (Рис. 3.30)

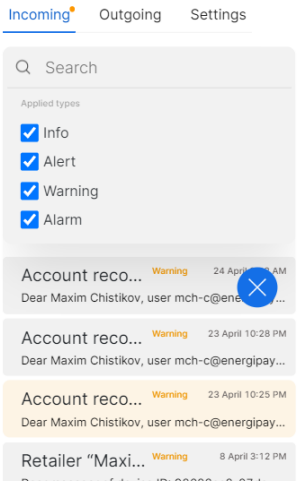

Рисунок 3.30 – Фільтр вхідних повідомлень

Як можна зазначити з рисунку, користувач може фільтрувати повідомлення двома способами: ввести власноруч інформацію присутню в повідомленні, або скористатися фільтрацією за типом повідомлення, реалізація якої наведена у лістингу 3.11.

Лістинг 3.11 – Реалізація фільтрації повідомлень

```
      const onInfoFilterClick = (e: FormEvent<HTMLInputElement>) => {
    prop.setFilterSearch({...prop.filterSearch, info:e.currentTarget.checked})
}

const onAlertFilterClick = (e: FormEvent<HTMLInputElement>) => {
    prop.setFilterSearch({...prop.filterSearch, alert:e.currentTarget.checked})
}

const onWarningFilterClick = (e: FormEvent<HTMLInputElement>) => {
    prop.setFilterSearch({...prop.filterSearch, warning:e.currentTarget.checked})
}

const onAlarmFilterClick = (e: FormEvent<HTMLInputElement>) => {
    prop.setFilterSearch({...prop.filterSearch, alarm:e.currentTarget.checked})
}
```

Далі йде вкладка Outgoing – включає в себе вихідні повідомлення користувача, їх можна направляти як одному користувачеві, так і до декількох. Зовнішній вигляд вихідних повідомлень зображений на рисунку 3.31.

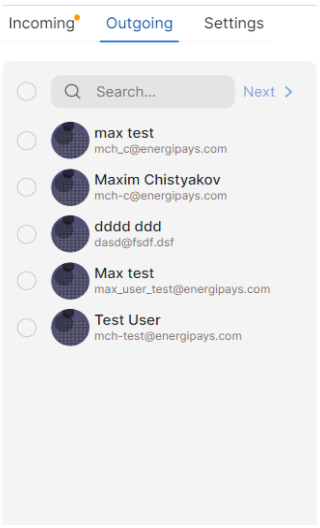

Рисунок 3.31 – Вихідні повідомлення - користувачі

Спочатку користувач обирає до кого буде йти написане їм повідомлення, потім натискає на кнопку «Next» та може вводити текст повідомлення.

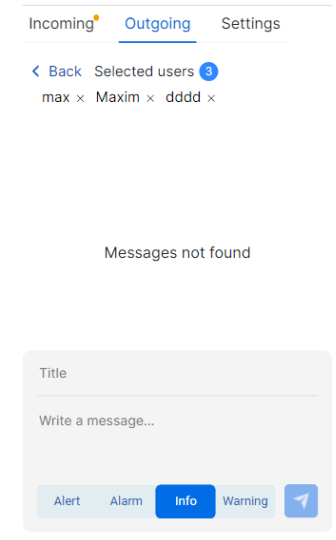

Рисунок 3.32 – Вихідні повідомлення – текст

Користувач може обирати тип, видаляти та додавати учасників повідомлення.

Далі йде вкладка Settings яка включає в себе усі налаштування доступні в додатку. На даний момент додаток налічує лише 2 налаштування, які зображені на рисунку 3.33.

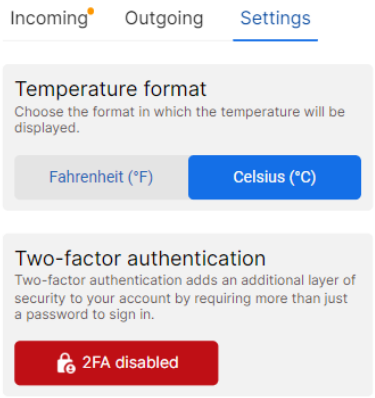

Рисунок 3.33 – Вкладка з налаштуваннями

Першим налаштуванням виступає регулювання відображення температури у додатку – можна обрати Цельсії та Фаренгейти. Другим налаштуванням виступає підключення двохфакторної авторизації до додатку, але вона несе в собі тільки інформативний характер, так як регулюється із мобільного додатку.

3.2.7 Сторінка груп

Сторінка груп відповідає за роботу з групами пристроїв, їх можна додавати, видаляти та редагувати. Додавання пристроїв до груп можливе як із вкладки з пристроями, так і на мапі. Зовнішній вигляд наведений на рисунку 3.34.

Як можна зазначити з рисунку, ця сторінка включає в себе велику кнопку для додавання груп та перелік груп, які вже були додані раніше. У кожної групи є своя назва, кількість пристроїв під’єднаних до неї, статус та контекстне меню: яке включає можливість видалення, редагування назви та очищення усіх пристроїв, які в ній знаходяться.

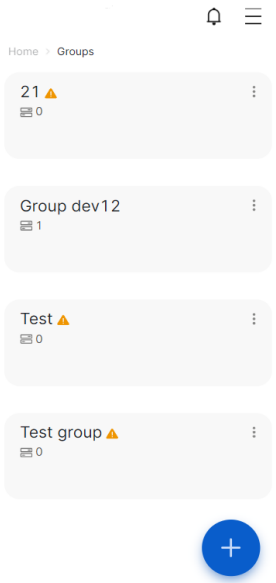

Рисунок 3.34 – Зовнішній вигляд сторінки груп

При натисканні на групу, з’являється модальне вікно з інформацією про неї, яке зображене на рисунку 3.35.

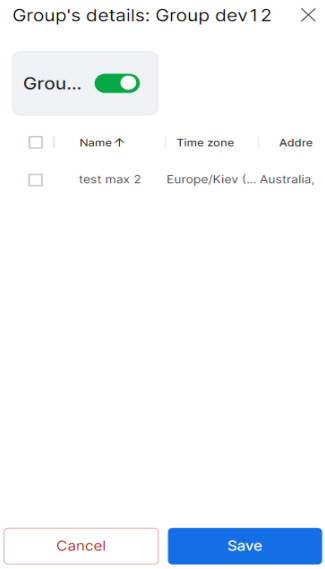

Рисунок 3.35 – Модальне вікно інформації про обрану групу

В кожної групи є перемикач, який активує та дезактивує усі пристрої які в ній знаходяться. Пристрої в групі відображаються за допомогою таблиці, яка була реалізована раніше, при обиранні пристрою – знизу з’являється кнопка видалення.

3.2.8 Сторінка правил

Сторінка правил має точно такий же дизайн як і сторінка груп, але функціонал в них різний. Правила задаються до пристроїв і виконують функції з керування пристроями за певними параметрами.

При натисканні на певне правило, з’являється модальне вікно, яке зображене на рисунку 3.36.

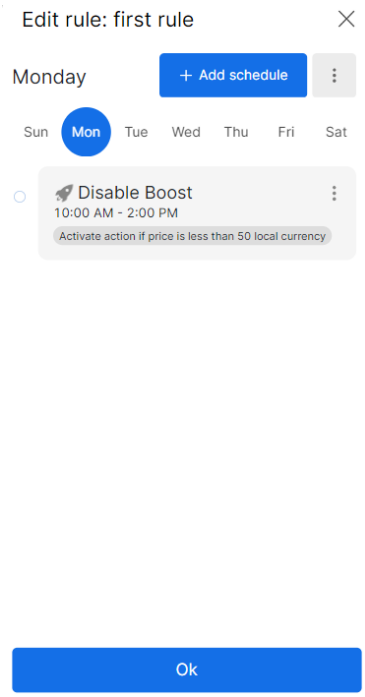

Рисунок 3.36 – Модальне вікно інформації про обране правило

Як можна зазначити з рисунку, можна додавати правила на кожен день неділі окремо, а при натисканні на певне правило, з’являється контекстне меню з можливістю редагування та видалення. При натисканні на кнопку «Add schedule», з’являється нове модальне вікно, зовнішній вигляд якого зображений на рисунку 3.37.

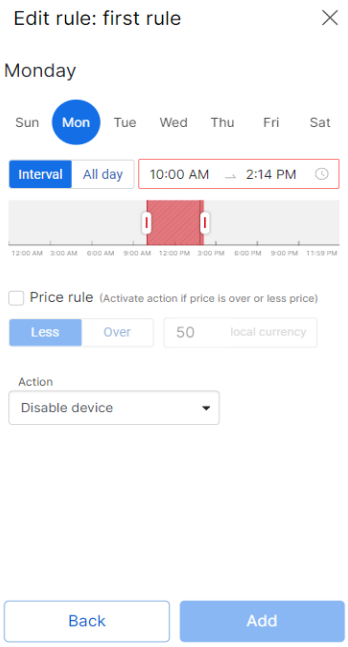

Рисунок 3.37 – Модальне вікно з налаштуванням правила

Як можна зазначити з рисунку, можна обрати день та часовий інтервал виконання правила, додавати активацію правила за певної ціни та обирати дію, яка буде виконуватися: виключення пристрою, або виключення бусту пристрою.

3.2.9 Сторінка аналітики

Сторінка аналітики надає аналітичні дані з роботи пристрою, вона є доступною для усіх ролей веб-додатку. Зовнішній вигляд сторінки аналітики зображений на рисунку 3.38.

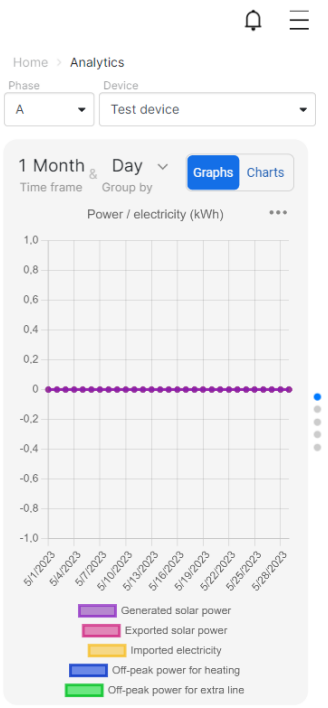

Рисунок 3.38 – Зовнішній вигляд сторінки аналітики

Як можна зазначити з рисунку, в верхній частині знаходиться фільтрація пристрою за його фазністю та вибір конкретного пристрою. А вже нижче знаходяться картки з самими аналітичними даними. Перша картка є графічним представленням вказаних в ній даних та має фільтрацію за часом.

Представлення даних у вигляді графіку було виконане за допомогою бібліотеки під назвою «react-chartjs», використання якої зображене у лістингу 3.12.

Лістинг 3.12 – Реалізація графічного представлення аналітичних даних

```
let options = { maintainAspectRatio: false, animation: { duration: 0 },
responsiveAnimationDuration: 0, responsive: true, plugins: { legend: { position:
'bottom' as const, }, title: { display: true, text: 'Power / electricity (kWh)',
fullSize:true, padding:20, color:"#666", font: { size: 14, family: "Helvetica",
weight:"normal", } }, },

{(chartsPointsEvents[1] === ChartsPointsEventsEnum.none) ? <Line options={options}
data={dataChart}/> :  <Bar options={options} data={dataChart}/>  }
```

Перемикання між карточками виконане за допомогою бібліотеки

«swiper», використання якої вже було описане у попередніх підрозділах.

3.2.10 Сторінка логів

Сторінка з логами доступна лише користувачам з роллю адміністратора та відображає майже всю активність інших користувачів, наприклад видалення чи додавання пристроїв або користувачів. При переході на дану сторінку першим чином користувач бачить модальне вікно, яке зображене на рисунку 3.39.

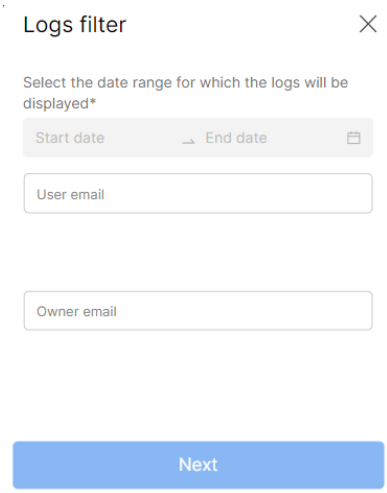

Рисунок 3.39 – Модальне вікно з фільтрацією логів

Користувач не може перейти на сторінку не обравши хоча б один з фільтрів: за датою, або за користувачем. Якщо користувач натисне на кнопку з хрестиком, він буде перенаправлений на головну сторінку додатку – сторінку пристроїв.

Після вибору методу фільтрації на натискання кнопки «Next», користувач опиняється на сторінці з логами, яка представлена на рисунку 3.40.

Як можна зазначити з рисунку, на сторінці присутня таблиця з логами за обраним фільтром, який відображається у верхній частині сторінки. В нижній частині знаходяться кнопки для взаємодії з логами: «Delete all» – видаляє усі записи, «Details» – відкриває деталі обраного логу, «Filter» – відкриває

модальне вікно для редагування фільтру. Таблиця записів була реалізована за допомогою раніше зробленого компоненту таблиці в яку передаються дані про логи.

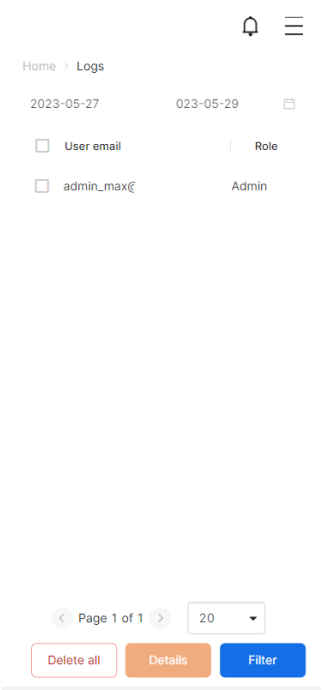

Рисунок 3.40 – Зовнішній вигляд сторінки логів

Після натискання кнопки «Details» – відкриється модальне вікно з повною інформацією про обраний запис. (Рис 3.41)

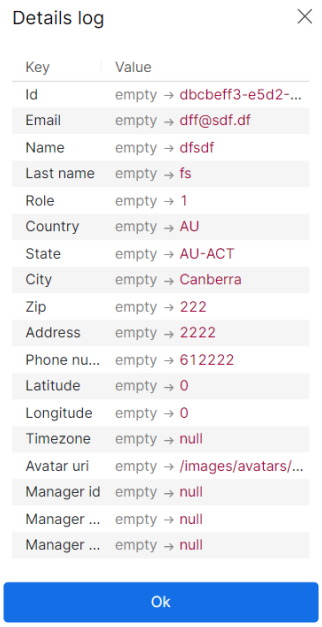

Рисунок 3.41 – Модальне вікно з інформацією про обраний запис

Інформація про обраний запис представлена у вигляді таблиці, зліва якої знаходиться колонка під назвою «Key», яка відображає тип даних, а з правої сторони – попереднє та поточне значення даних. Реалізація виводу даних у вигляді таблиці зображена у лістингу 3.13.

Лістинг 3.13 – Реалізація виводу інформації про обраний запис

```
{
    logData.map((value, index)=>{
        return (
            <div style={{display:"grid", padding:"3px 0 3px 0"}}>
                <div style={{whiteSpace:"nowrap", title={value.key}>
                    {value.key}
                </div>

                <div style={{title={`${value.data} ${ value.newData ? "->" +
value.newData : ""}`}>
                    <text style={{color:value.newData ? "#878787" : "black"}}>
                        {value.data}
                    </text>
                    {
                        value.newData &&
                        <>
                            <Icon style={{margin:"0 5px 0 5px"}} size={15}
position={"relative"} src={ArrowPen} classColor={"color-878787"}/>
                            <text style={{color:"#AA0039"}}>
                                {value.newData}
                            </text>
                        </>
                    }
                </div>
            </div>
        )})
}
```

3.2.11 Навігаційна панель

Навігаційна панель присутня на кожній сторінці веб-додатку, бо без неї користувач не буде мати можливості перемикатися між сторінками.

Наповнення даної панелі змінюються виходячи з ролі користувача, так у ролі адміністратора не може бути сторінки підписок, так як дана функціональність не призначена для цієї ролі. Зовнішній вигляд навігаційної панелі зображений на рисунку 3.42.

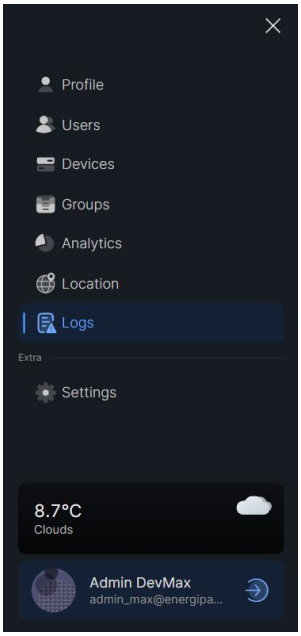

Рисунок 3.42 – Зовнішній вигляд навігаційної панелі

В навігаційній панелі, окрім назв сторінок, ще присутні кнопка виходу з аккаунту з інформацією про поточний аккаунт та поле з виведенням поточної температури за адресою, вказаною при реєструванні аккаунту.

3.3 Висновки за розділом

Відповідно до висунутих вимог, було спроектовано функціональні можливості користувачів з різними ролями. Спроектовано структуру таблиць бази даних для зберігання даних веб-застосунку. Спроектовано модель взаємодії фронтенду та бекенду.

Було розроблено дизайн веб-застосунку, який відповідає побажанням замовника. Було розроблене комплексне рішення для дистанційного керування пристроями, менеджменту пристроїв та користувачів.

ВИСНОВКИ

В процесі кваліфікаційної бакалаврської роботи було виконано всі поставлені задачі щодо розробки мобільного інтерфейсу з управління IoT пристроями.

Було здійснено огляд існуючих платформ з управління IoT пристроями, розібрані їх недоліки та переваги.

Також було здійснено порівняльний аналіз мобільної версії сайту та мобільного додатку, після чого аргументовано вибір саме мобільної версії сайту для розробки.

Було здійснено огляд популярних фреймворків для розробки веб-застосунків: React, Angular, Vue, та аргументовано вибір одного з них – React. Були розглянуті особливості отримання даних для відображення, клієнт-серверна архітектура та принципи RESTful API.

Для розробки використовувалася мова програмування TypeScript з використанням фреймворку React. В допомогу до реакту, задля контролю стану веб-застосунку використовувалася бібліотека Redux Toolkit. Вибір усього інструментарії для розробки веб-застосунку був оговорений за узгоджений із замовником.

Відповідно до висунутих вимог, було спроектовано функціональні можливості користувачів з різними ролями. Спроектовано структуру таблиць бази даних для зберігання даних веб-застосунку. Спроектовано модель взаємодії фронтенду та бекенду.

У підсумку було розроблене комплексне рішення для дистанційного керування пристроями, менеджменту пристроїв та користувачів. Завдяки зручній та інтуїтивно зрозумілій платформі користувачі можуть легко керувати своїми пристроями IoT і контролювати їх, що сприяє підвищенню ефективності, продуктивності та якості життя кінцевих користувачів. Створений веб-застосунок у повному обсязі задовільнив вимоги технічного завдання замовника.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Best IoT Cloud Platforms [Електронний ресурс]. – Режим доступу: www. URL: https://euristiq.com/best-iot-cloud-platforms/ (дата звернення: 13.02.23)
2. Build IoT web applications using AWS IoT [Електронний ресурс]. – Режим доступу: www. URL: https://aws.amazon.com/ru/blogs/iot/build-iot-applications-using-aws-iot-application-kit/ (дата звернення: 14.02.23)
3. Customizing of Azure IoT dashboard [Електронний ресурс]. – Режим доступу: www. URL: https://learn.microsoft.com/en-us/azure/iot-central/retail/ tutorial-in-store-analytics-customize-dashboard (дата звернення: 14.02.23)
4. IBM Watson IoT Platform [Електронний ресурс]. – Режим доступу: www. URL: https://www.g2.com/products/ibm-watson-iot-platform/reviews (дата звернення: 21.02.23)
5. Amazon Web Service (AWS) [Електронний ресурс]. – Режим доступу: www. URL: https://www.techtarget.com/searchaws/definition/Amazon-Web-Services (дата звернення: 07.03.23)
6. What is Azure Internet of things (IoT) [Електронний ресурс]. – Режим доступу: www. URL: https://learn.microsoft.com/en-us/azure/iot/iot-introduction (дата звернення: 07.03.23)
7. Mobile app vs Mobile website [Електронний ресурс]. – Режим доступу: www. URL: https://www.cleveroad.com/blog/mobile-app-vs-mobile-website/ (дата звернення: 20.03.23)
8. NPM Trends [Електронний ресурс]. – Режим доступу: www. URL: https://npmtrends.com/@angular/core-vs-react-vs-solid-js-vs-svelte-vs-vue (дата звернення: 25.03.23)
9. Stack Overflow Trends [Електронний ресурс]. – Режим доступу: www. URL: https://insights.stackoverflow.com/trends?tags=reactjs%2C-vue.js%2Cangular%2Csvelte%2Cangularjs%2Cvuejs3 (дата звернення: 25.03.23)
10. Everything you need to know about React [Електронний ресурс]. – Режим доступу: www. URL: https://medium.com/the-research-nest/everything-you-

need-to-know-about-react-ab24da4275ea (дата звернення: 26.03.23)

11. Why you should use React for Web Development [Електронний ресурс]. – Режим доступу: www. URL: https://www.techmagic.co/blog-/why-we-use-react-js-in-the-development/ (дата звернення: 26.03.23)
12. Best JavaScript IDE’s [Електронний ресурс]. – Режим доступу: www. URL: https://jelvix.com/blog/best-javascript-ides (дата звернення: 06.04.23)
13. What is Angular? [Електронний ресурс]. – Режим доступу: www. URL: https://www.simplilearn.com/tutorials/angular-tutorial/what-is-ang-ular (дата звернення: 06.04.23)
14. Angular Features [Електронний ресурс]. – Режим доступу: www. URL: https://angular.io/features (дата звернення: 06.04.23)
15. Benefits of Vue.js [Електронний ресурс]. – Режим доступу: www. URL: https://www.tatvasoft.com/outsourcing/2021/10/what-is-vue-js-and-its-benefits.html (дата звернення: 07.04.23)
16. Modern client server communications [Електронний ресурс]. – Режим доступу: www. URL: https://tech-lead.medium.com/grpc-vs-restful-api-vs-graphql-web-socket-tcp-sockets-and-udp-beyond-client-server-43338eb02e37 (дата звернення: 12.04.23)
17. Client Server Architecture [Електронний ресурс]. – Режим доступу: www. URL: https://www.enjoyalgorithms.com/blog/client-server-archit-ecture (дата звернення: 12.04.23)
18. What is TypeScript [Електронний ресурс]. – Режим доступу: www. URL: https://thenewstack.io/what-is-typescript/ (дата звернення: 14.04.23)
19. Why RTK is how to use Redux today [Електронний ресурс]. – Режим доступу: www. URL: https://redux.js.org/introduction/why-rtk-is-redux-today (дата звернення: 14.04.23)
20. WebStorm vs Visual Studio Code [Електронний ресурс]. – Режим доступу: www. URL: https://www.techrepublic.com/article/webstorm-vs-vscode/ (дата звернення: 19.04.23)