КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

До захисту допущено
В.о. завідувача кафедри ІСТ
_____ Олексій КОЛЕСНІКОВ
(підпис) (ім'я, ПРІЗВИЩЕ)
"___"_____2021р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

спеціальності 126 «Інформаційні системи та технології»

освітньої програми «Програмні технології інтернет речей»

на тему: "Комплексне IoT рішення для детектування та моніторингу лісових пожеж"

Виконав: студент 4 курсу, групи ПТІР-41

Сауляк Микола Русланович
(Ім'я, ПРІЗВИЩЕ) (підпис)

Керівник: д.т.н., проф. Михайло СТЕПАНОВ
(посада, науковий ступінь, вчене звання, Ім'я, ПРІЗВИЩЕ) (підпис)

Консультант нормо контроль к.т.н., доцент Ростислав ЛІСНЕВСЬКИЙ
(назва розділу) (посада, вчене звання, науковий ступінь, Ім'я, ПРІЗВИЩЕ) (підпис)

Рецензент:
(посада, науковий ступінь, вчене звання, науковий ступінь, Ім'я, ПРІЗВИЩЕ) (підпис)

Засвідчую, що у кваліфікаційній роботі немає запозичень з праць інших авторів без відповідних посилань.

Здобувач освіти _____
(підпис)

Київ – 2021 року

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра Інформаційні системи та технології
Освітній рівень Бакалавр
Спеціальність 126 Інформаційні системи та технології
Освітня програма Програмні технології інтернет речей

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри,
д.т.н., доцент
Олексій КОЛЕСНІКОВ

_____
«__» _____ 2021 року

ЗАВДАННЯ
НА ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

Здобувач освіти: Микола САУЛЯК

Група: ІР-41

1. **Тема кваліфікаційної роботи бакалавра**: «Комплексне IoT рішення для детектування та моніторингу лісових пожеж».

Затверджена протоколом засідання кафедри ІСТ №18/20 від 01.12.2021 року

2. **Строк подання студентом готової роботи −** «22» червня 2021 р.

3. **Вихідні дані до роботи**: дослідження в області детектування лісових пожеж. Програмно-апаратні рішення для детектування аномалій в показах температури. Дані від датчиків температури, вогню, вуглекислого газу, методи їх контролю та передачі даних з використанням систем IoT.

4. **Зміст роботи:** РОЗДІЛ 1 АНАЛІЗ СИСТЕМ ДЕТЕКТУВАННЯ ТА МОНІТОРИНГУ ЛІСОВИХ ПОЖЕЖ; РОЗДІЛ 2 ПРОЕКТУВАННЯ ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ ІОТ СИСТЕМИ; РОЗДІЛ 3 РЕАЛІЗАЦІЯ ІОТ СИСТЕМИ.

5. **Перелік графічного матеріалу**: Концептуальна модель архітектури системи IoT рішень для детектування лісових пожеж; фізична модель бази даних; алгоритм роботи системи, вхідні та вихідні дані; принцип роботи мобільного застосунку; екрани мобіл ного застосунку.

6. **Календарний план виконання роботи:**

Етапи виконання кваліфікаційної роботи бакалавра	**Термін виконання**	**Результат виконання**
1. Вибір тематики кваліфікаційної роботи бакалавра	до 01.12.2020	виконано
2. Наказ про затвердження тем кваліфікаційної роботи бакалавра та призначення керівників	01.12.2020	виконано
3. Розробка плану кваліфікаційної роботи бакалавра і його погодження з керівником	25.12.2020	виконано
4. Написання І розділу кваліфікаційної роботи	20.01.2021	виконано
5. Написання ІІ розділу кваліфікаційної роботи	19.02.2021	виконано
6. Написання ІІІ розділу кваліфікаційної роботи	05.03.2021	виконано
7. Підготовка висновків і пропозицій	05.04.2021	виконано
8. Попередній захист кваліфікаційної роботи	20.04.2021	виконано
9. Перевірка на плагіат	до 15.06.2021	виконано
10. Нормоконтроль	до 17.06.21	виконано
11. Рецензування кваліфікаційної роботи бакалавра і представлення роботи на кафедру в друкованому вигляді	до 21.06.2021	виконано
11. Захист кваліфікаційної роботи бакалавра	23.06.2021	

Дата видачі завдання «_01_» _грудня_ 2020 р.

Керівник роботи: д.т.н., проф. Михайло СТЕПАНОВ _____ (підпис)
Завдання прийняв до виконання:
Здобувач освіти на освітньому рівні «бакалавр» 4-го курсу групи ІР-41
Микола САУЛЯК _____
(Власне Ім'я, ПРІЗВИЩЕ) (підпис)

АНОТАЦІЯ

Бакалаврська робота на тему “Комплексне IoT рішення для детектування та моніторингу лісових пожеж” присвячена проектуванню, програмній реалізації мобільного застосунку та механізму відправки push сповіщень, складанням меш мережі із сенсорами.

Робота містить такі розділи: аналіз та опис сфери застосування даної системи, постановка задачі та список вимог, сценаріїв використання даної системи; опис засобів реалізації, вхідна, вихідна інформація, структурна схема зв’язків модулів програми, опис основного коду програми з посиланням на додаток, опис інтерфейсу програми.

У висновках підсумовано роботу, проведено аналіз виконаної праці.

Робота складається з 82 сторінок, містить 37 рисунків, 4 таблиці. При написанні роботи використано 25 інформаційних джерел.

Ключові слова: iot, javascript, сервер, дані, додаток, датчик, пожежі, ліси, температура.

SUMMARY

The bachelor's thesis on "Integrated IoT solution for detection and monitoring of forest fires" is devoted to the design, software implementation of mobile applications and the mechanism for sending push notifications, compiling a mesh network with sensors.

The work contains the following sections: analysis and description of the scope of forest fire prevention system, problem statement and list of requirements, system’s usecases; input, output information, block diagram of connections of program modules, description of the main code of the program with reference to the application, description of the program interface.

The conclusions summarize the work, analyze the work performed.

The work consists of 82 pages and contains 37 figures, 4 tables. 25 information sources were used in writing the work.

Keywords: iot, javascript, server, data, application, sensor, fires, forests, temperature.

ВСТУП

Актуальність теми. Кількість підключених до мережі Інтернет пристроїв росте з кожним роком. Вони дозволяють збирати та обробляти інформацію від нових джерел. Зокрема пристрої інтернету-речей можна використати, щоб краще розуміти навколишнє середовище та запобігати виникненню лісових пожеж, збитки від яких сягають мільярди доларів. Лісові пожежі стають частим та небезпечним явищем. За даними Європейської комісії [1] протягом 17 років на боротьбу з пожежами було витрачено понад 50 мільярдів доларів.

Збитки від масштабної пожежі на Луганщині у 2020, в ході якої вигоріло майже 8 тисяч гектарів лісу, можуть складати близько 4-5 мільярдів гривень.[2]

Метою бакалаврської роботи є:

- аналіз систем детектування лісових пожеж;
- проектування та реалізація комплексного IoT рішення для детектування лісових пожеж.

Завдання. Для досягнення мети необхідно виконати:

1) комплексне дослідження систем захисту лісів від пожеж;
2) зібрання вимог до системи та складання випадків використання;
3) розробка мобільного додатку згідно обумовленої теми за допомогою фреймворку react-native;
4) розробка веб сервера на базі фреймворку express;
5) фізичне під’єднання датчиків до плат та їх з’єднання зі сервером;
6) реалізація механізму відправки push сповіщень.

Об’єктом дослідження в бакалаврській роботі є складові системи захисту лісу від пожеж.

Предметом дослідження є проектування та програмна реалізація мобільного застосунку, веб сервера та механізму відправки push сповіщень.

За допомогою пошукових систем та аналізу публікацій в google scholar було зібрано та проаналізовано подібні системи, а також зібрано всю необхідну інформацію для проектування та реалізації системи.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ В НАПРЯМКУ МОНІТОРИНГУ ЛІСОВИХ ПОЖЕЖ

1.1 Аналіз систем детектування та моніторингу лісових пожеж.

Існує ряд систем виявлення та моніторингу лісових пожеж. Сюди входять моніторингові вежі, повітряний та супутниковий моніторинг, а також системи виявлення та моніторингу на основі цифрових камер та оптичних приладів, різних типів датчиків та їх поєднання. Способи виявлення лісових пожеж наведено на рисунку 1.1.

У наземних системах на основі оптичних приладів можуть використовуватися різні типи датчиків виявлення [3]:

- відеокамера, чутлива до видимого спектра диму;
- інфрачервоні (ІЧ), тепловізійні камери;
- системи виявлення та вимірювання світла - LIDAR, які вимірюють лазерні промені, відбиті від частинок диму.

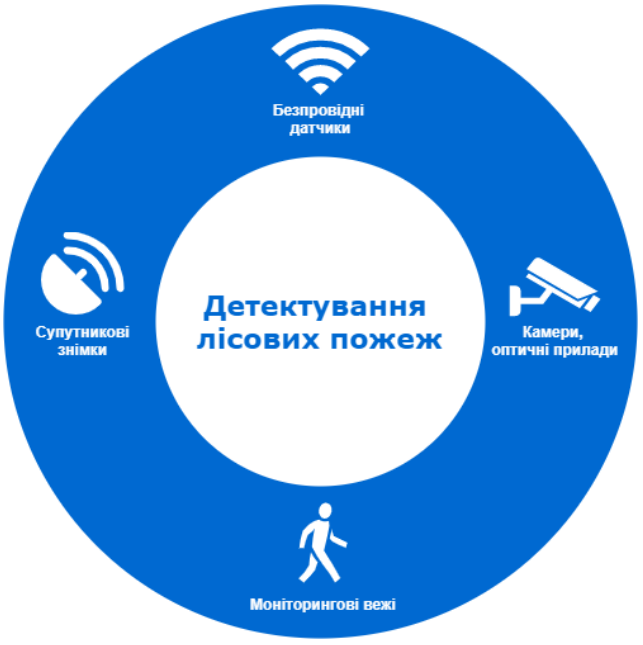


Рисунок 1.1. Способи детектування лісових пожеж

Таблиця 1.1 Порівняння спосіб детектування лісових пожеж

Назва способу	Переваги	Недоліки
1	2	3
Супутникові знімки	- можливість використання наявних апаратних засобів	- великий час відгуку - складність постійного покриття через зміну позицій супутників - вплив погодних умов на якість знімків
Цифрові камери та оптичні прилади	- велика площа покриття (15-80 км)	- висока вартість - необхідність побудови спеціальних приміщень - високий рівень помилок через погодні умови, зміну дня і ночі
Бездротові датчики	- швидкість детектування аномалій - можливість встановлення у важкодоступних місцях	- проблема локалізації датчиків - необхідність живлення кожному датчику
Моніторингові вежі	- більша ефективність у порівнянні з оптичними приладами	- людський фактор - складність покриття великих територій

Системи виявлення на основі оптичних сенсорів або цифрових камер є повільними і менш надійними, ніж навчений спостерігач. Однак може бути корисним використовувати ці системи як інструменти для підвищення продуктивності спостерігача вежі або у віддалених важкодоступних вежах.

Перевагою систем моніторингу лісових пожеж на основі безпровідних датчиків є здатність передавати потрібну інформацію в будь-який момент часу. Такі системи можна облаштовувати у важкодоступних місцях. Вон можуть працювати на сонячних батареях, використовувати штучний інтелект (ШІ) для аналізу даних, а також аналізу температури, вологості та тиску повітря.

Найчастіше вузли в такій мережі утворюють топологію меш (сітчаста топологія). Щонайменше два типи пристроїв наявні - сенсори та хаби, або шлюзи. Для утворення мережі такого типу можуть використовуватися найрізноманітніші технології - wifi, lorawan або zigbee.

Якщо система розташовується у віддаленому місці та важкодоступному, то можливе використання технології LoRa **(Lo**ng **Ra**nge), що дає змогу передавати дані на десятки кілометрів.

Основним недоліком подібних систем є вартість їх впровадження, адже потрібна величезна кількість різноманітних датчиків. Якщо вимірювати всі потрібні параметри для миттєвого виявлення пожежі, то ціна може сягати десятки доларів для одного девайсу із набором датчиків, що на даний момент є неприпустимим. За даними компанії amplia iiot ідеальним є розташування датчиків в подібній системі кожна 50 квадратних метрів [4], що дозволить детектувати пожежу менш, ніж за 5 хвилин після її початку.

В меш мережі (рис.1.2) вузли не потребують підключення до центрального вузла. Вузли відповідають за передачу один одному повідомлень. Це дозволяє кільком пристроям поширюватися на великій фізичній площі. Вузли можуть самоорганізовуватися та динамічно спілкуватися між собою, щоб переконатися, що пакет досягає кінцевого пункту призначення. Якщо будь-який вузол видалено з мережі, він може самоорганізуватися, щоб переконатися, що пакети досягли місця призначення [5].

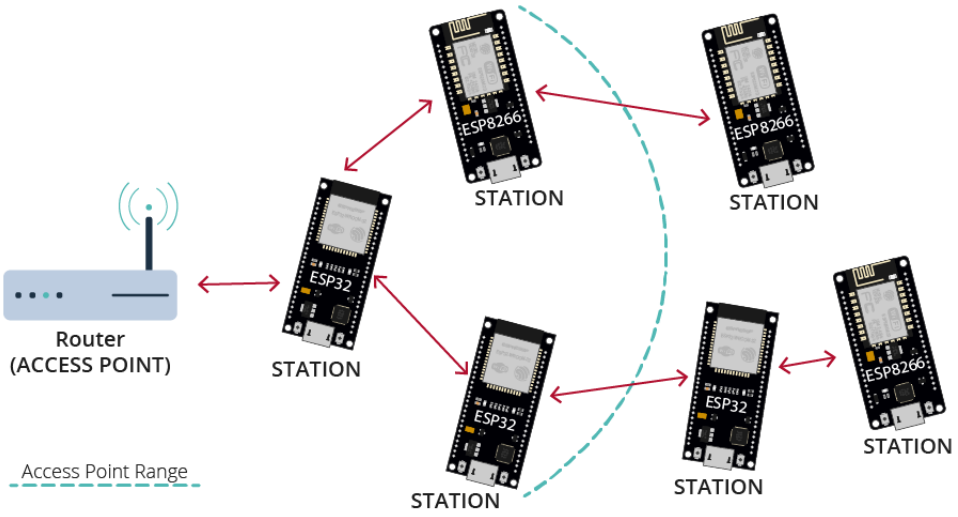


Рисунок 1.2. Приклад меш мережі на платах ESP32

Одним із способів донесення важливої інформації та інформування користувачів про певні події є push сповіщення на мобільних пристроях.

Напевно кожен хоча б раз бачив подібне сповіщення (рис.1.3) у себе на телефоні. Мобільна операційна система відображає таке сповіщення, або іншими словами просто повідомлення, щоб нагадати чи донести якусь інформацію від додатків, які бажають це зробити, у стислому вигляді до користувача. Сповіщення може містити додаткові кнопки для виконання певних дій безпосередньо у сповіщенні

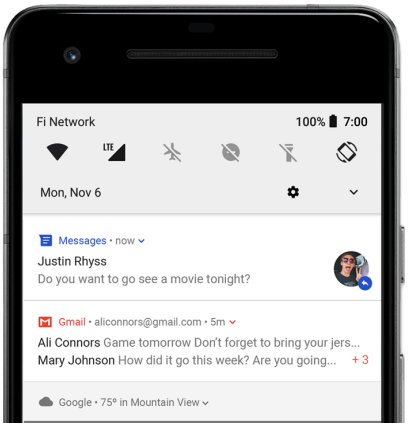


Рис 1.3 Приклад сповіщення на телефоні

Для нашої системи такі сповіщення є важливими оскільки вони допоможуть оперативно реагувати на аномалії в показах датчиках і приймати швидко потрібні рішення. Найпопулярніші мобільні операційні системи дають можливість додаткам на телефоні приймати різні типи сповіщень як за виглядом, так і за змістом інформації у сповіщеннях

Важливим компонентом для відправки сповіщень є push-токен. Push-токен (маркер пристрою) - це унікальний ключ для комбінації додатків і пристроїв, який видається шлюзами сповіщень Apple або Google. Це дозволяє шлюзам і провайдерам сповіщень маршрутизувати повідомлення та гарантує, що сповіщення доставляється лише до унікальних комбінації програм-пристроїв (тобто до певної програми на певному телефоні), для якої воно призначене.

Мобільні ОС надають змогу користувачам заборонити сповіщення для певного додатку, або видозмінити вигляд сповіщень (iOS), а також відключити звук та бейджики на іконках додатків, тому перед отриманням токену потрібно перевірити відповідні дозволи.

Подібні системи не використовуються поки досить широко і проектів за даною тематикою досить мало. Одним із таких проектів є Lali. Інформації про цю систему в мережі Інтернет мало, відомо що вона була створена під час хакатону командою із Еквадору.[6]

Команда Lali використала десятигодинний хакатон для створення прототипу пакету датчиків, який контролює температуру і відправляє дані на базову станцію кожні десять хвилин. Базова станція передає дані в хмарне програмне забезпечення для обробки. Команда згадує про загальну вартість менше 5 доларів, проте не вказуються основні компоненти системи.

Система FireBug - проект студентів та професорів університету берклі , складається з мережі бездротових термодатчиків з підтримкою GPS, рівня управління для обробки даних датчиків та командного центру для інтерактивної

комунікації з мережею датчиків. [7] На сайті проекту зазначається, що використовуються такі датчики (рис 1.4) як Sensirion SHT11 для зчитування показів температури, Intersema MS5534AP - для показів атмосферного тиску, акселерометр ADXL 202AE , датчик світла Taos TLS257. Проте ці компоненти не є популярними та їх загальна вартість може становити декілька десятків доларів. Система використовує GPS датчики. Доцільність використання цих компонентів на сайті проекту не описана. В якості інтерфейсу - програма написана на java.

Рисунок 1.4. Пристрій проекту Firebug

Ще одним прикладом системи, метою якої є захист лісів від пожеж є LADSensors. LADSensors - це нова платформа виявлення та управління пожежею, яка використовує технологію сенсорів IoT поряд з алгоритмами штучного інтелекту для прогнозування, виявлення та управління пожежами в лісах. Вимірюючи такі показники, як температура, рівень навколишнього CO2, вологість, напрямок вітру та швидкість, LADSensors не тільки виявляє наявність пожежі, але й передбачає, де вона поширюється.[8] Команда проекту говорить про 1300 років безперебійної роботи без заміни акумуляторів. Пристрої показано на рисунку 1.5.

Рисунок 1.5. Сенсор системи LAD Sensors

Всі системи використовують датчики температури, для більшої точності можна використати також датчики вологості, CO2. Дані відправляються на хмарну платформу, де і обробляються. Проте на нашу думку ці системи використовують надмірну кількість датчиків, які роблять практичне застосування даної системи менш раціональним. Наша системи міститиме мінімум потрібних датчиків для виявлення аномалій.

Аналіз систем наведено в таблиці 1.2 за такими показниками: наявність веб-інтерфейсу для перегляду даних, ціна датчиків та обслуговування, автономність системи, показники що вимірюють датчики.

Таблиця 1.2 Порівняння систем детектування лісових пожеж на основі безпровідних датчиків

Назва проекту	Web	Ціна датчику	Автономність	Показники
1	2	3	4	5
LAD Sensors	+	1x - 169€ ~ 5600 грн 10x 159€ 50x 139€ 100x 119€ 1000x - 99€ ~ 3200 грн Обслуговування: 1x - 84€, 10x - 56€ 50x - 44.80€, 100x - 36,40€, 1000x - 28€	наявність сонячних батарей, ~ 1000 років	1. температура 2. рівень CO2, 3. вологість, 4. напрямок вітру та швидкість.
FireBug	-	SHT11 (500+ грн), Taos TSL250RD ~ 60 грн, Intersema MS5534AP - не випускається, ~ 1000грн загалом: ~1600 грн	2 батарейки AA	1. Вологість 2. Температура 3. Інтенсивність світла 4. Прискорення 5. Географічне положення
Lali	+	~ менше 150 гривень	до 10 років	1. Температура 2. Напрям вітру (сторонній API)

При аналізу економічної ефективності потрібно враховувати збитки від лісових пожеж, а також кошти витрачені на боротьбу з пожежами, кількість датчиків та їх ціна. На рисунку 1.6 показано втрати від лісових пожеж в США протягом 2010-2019 років. [9]

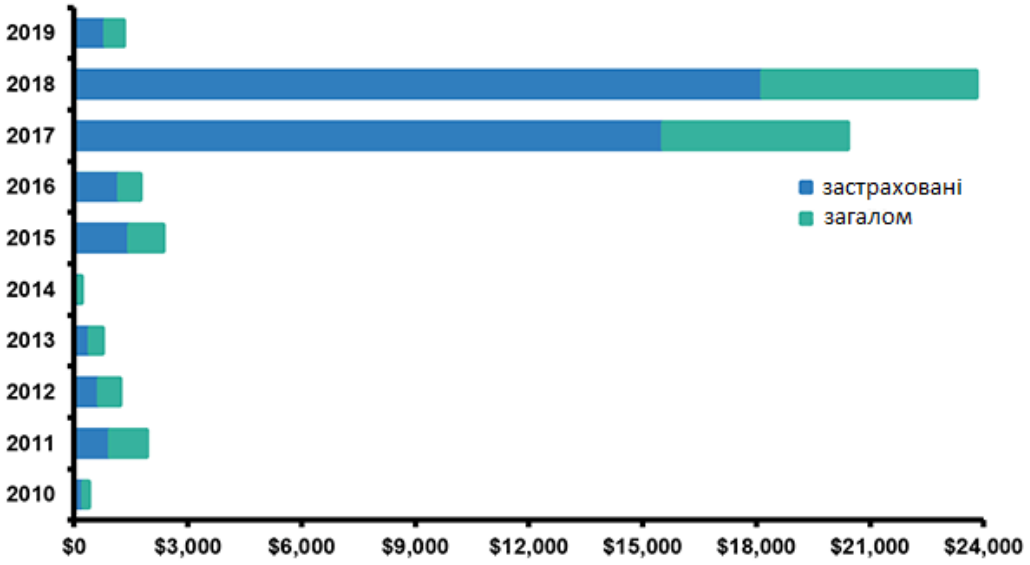


Рисунок 1.6. Втрати від лісових пожеж в США

На рисунку 1.7 показана ціна впровадження в різних країнах по всій території лісових насаджень за умови розміщення на одному гектарі 9 датчиків і ціни датчика 15 доларів без вартості підтримки такої мережі.

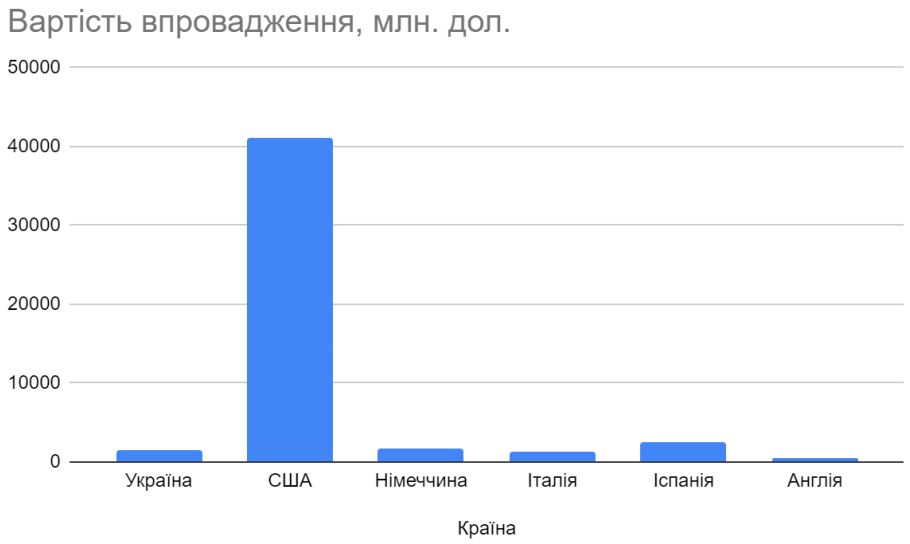


Рисунок 1.7 Вартість впровадження системи

1.2 Висновки до розділу

Проаналізувавши наявні типи систем моніторингу лісових пожеж можна дійти висновку, що найбільш популярними і практичними є системи на основі безпровідних датчиків, яку ми і будемо розробляти в даній роботі. Найголовнішим параметром для вимірювання є температура, яку вимірюють майже всі системи. Збитки від лісових пожеж, які були завдані протягом останніх 10 років співрозмірні з вартістю впровадження подібної системи.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ ІОТ СИСТЕМИ

2.1 Зібрання вимог до системи

Для будь якого проекту потрібний набір чітких вимог та задач, які буде виконувати дана система. Процес створення системи може продовжуватися досить довго та вимагати значних ресурсів, тому список задач, вимог не може бути безкінечним.

Вимогу найпростіше зрозуміти як конкретний опис потреб вашого клієнта, який може бути використаний для створення реального продукту.

Процес збирання вимог можна поділити на декілька етапів [10]:

1. Виявлення вимог.
2. Висловлення вимог.
3. Визначення пріоритетів вимог.
4. Аналіз вимог.
5. Управління вимогами.

Типові подання вимог включають, зокрема, випадки використання, історії користувачів чи розкадровки (use cases, user stories, або storyboards). Вони часто пристосовуються до проекту. В обов’язки менеджера програмного продукту та команди входить визначити та використовувати подання, які найкраще підходять для поточного проекту.

Для досягнення своєї мети процес розробки вимог до систем IoT має три підпроцеси: визначення обсягу проекту; визначення системи IoT та визначення системних вимог IoT. [11]

В результаті успішного виконання підпроцесу визначення обсягу проекту отримані такі результати:

1. Проблема або можливість виявляється, аналізується та деталізується;
2. Визначено зацікавлені сторони та потреби зацікавлених сторін

Проблемою в даному випадку є знищення домівок жителів поблизу лісових насаджень.

В нашому випадку зацікавленими сторонами є жителі, які живуть поблизу лісових насаджень та охоронні організації, які опікуються безпекою лісів. Як було сказано раніше однією з головних вимог є детектування пожежі в перші 5 хвилин та можливість отримання відповідного сповіщення.

В результаті успішного виконання підпроцесу визначення системи IoT отримують такі результати:

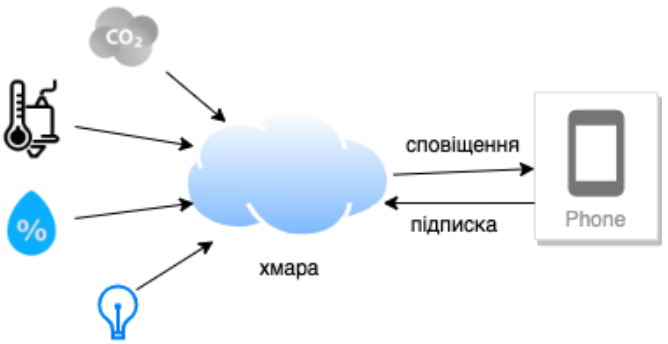
1. Визначено сценарії IoT;
2. Визначено компоненти та дії системи IoT;
3. Визначено склад альтернативних домовленостей;
4. Вибрано бажані механізми

Сценарій використання даної системи наведено в таблиці 2.1.

Таблиця 2.1. Сценарій використання IoT мережі

Назва проекту	1	Комплекс моніторингу лісових пожеж “Лісова куниця”
Відповідальний	2	Сауляк Микола
Цілі систем	3	Система має на меті забезпечити простий спосіб вимірювання, збору та зберігання даних (ідентифікатор, вага та зріст) про аномалії на території лісів. Дані будуть доступні в системі "Лісова куниця” (Інтернет та мобільний) для жителів та лісових працівників.
Домен системи	4	Охорона лісів
Актори (дійові особи)	5	Користувачі: жителі та працівники лісових охоронних організацій
		Речі: датчики, хаб (з esp8266)
		ПО: “Лісова куниця” (для зберігання та видобування даних)

Таблиця 2.1.Продовження

Типи даних	6	Температура, вологість, вуглекислий газ, світло з довжиною хвилі 760nm ~ 1100 Нм.	
Ідентифікатор сценарію	7	ЛК01	Назва: Автоматичне надсилання пуш сповіщення у випадку появи аномалії в даних користувачеві мобільного додатку.
Домовленості	8		
Кроки	9	Послідовність взаємодії Базовий порядок: 1. Датчики надсилають дані на сервер. 2. Хаб отримує дані від сенсорів. 3. Хаб формує корисне навантаження для севера та надсилає post запит на сервер. 4. Сервер отримує запит та отримує ідентифікатор сенсора, який надіслав ці дані. 5. Сервер перевіряє наявність та стан датчика в базі даних. 6. Сервер записує дані від датчика та аналізує. 7. При перевищені певного порогового значення сервер надсилає сповіщення.	

Таблиця 2.1.Продовження

Кроки	9	8. Сервер перевіряє чи було сповіщення про аномалію від датчиків на цій ділянці, або в певному радіуса раніше, якщо так, то сповіщення отримає вищий рівень небезпеки та виглядатиме по-іншому, а також програватиме інший звук при отриманні сповіщення. 9. Сповіщення відображається, якщо мобільний додаток згорнуто. 10. Сповіщення відображається, якщо мобільний додаток саме в цей час працює. Сповіщення матиме інший вигляд. 11. Користувач може натиснути на сповіщення та відкрити екран в мобільному додатку з показами датчиків в певному радіусі. Альтернативний порядок: [A1]: Користувач заборонив відправку сповіщень на телефоні 1. Додаток не дає змогу зареєструватися без дозволу на сповіщення. [A2]: Неадекватні покази датчиків 1. Працівники охоронних організацій (далі лісник) перевіряють покази датчиків. 2. Лісник змінює стан датчика в системі на “перевіряються покази”. 3. Лісник блокує відправку сповіщень на 15 хв. 4. Сервер надсилає сповіщення користувачам про перевірку датчиків в певній території. 5. Сервер перестає аналізувати сповіщення від даного датчика, але записує в базу даних.

Таблиця 2.1.Продовження

Кроки	9	Бізнес правила: [RN01]: Час відправки сповіщень повинен бути меншим або рівним 40 секунд після виявлення аномалії.
Середовище	10	Доступ до “Лісової куниці” здійснюється з будь якого місця, де є доступ до мережі WiFi. Датчики розташовуються в лісовій місцевості і взаємодіють між собою за допомогою технології Wifi, утворюючи меш мережу.
Підключення	11	Система потребує мобільного додатку та мережі Wi-Fi.

В результаті успішного виконання останнього підпроцесу зібрання вимог повинні бути:

1. Вказані та перевірені компоненти пристрою.
2. Вказані та перевірені програмні компоненти.
3. Описані та перевірені випадки використання.
4. Перевірено вимоги зацікавлених сторін до системи IoT та розроблена специфікація вимог.

Апаратні та програмні компоненти будуть наведені в наступному розділі.

2.2 Архітектура системи

Система повинна надсилати дані про аномалії та їх місце у корисному навантаженні сповіщень. Формат даних - json. На рисунку 2.1 зображено потік даних в мережі.

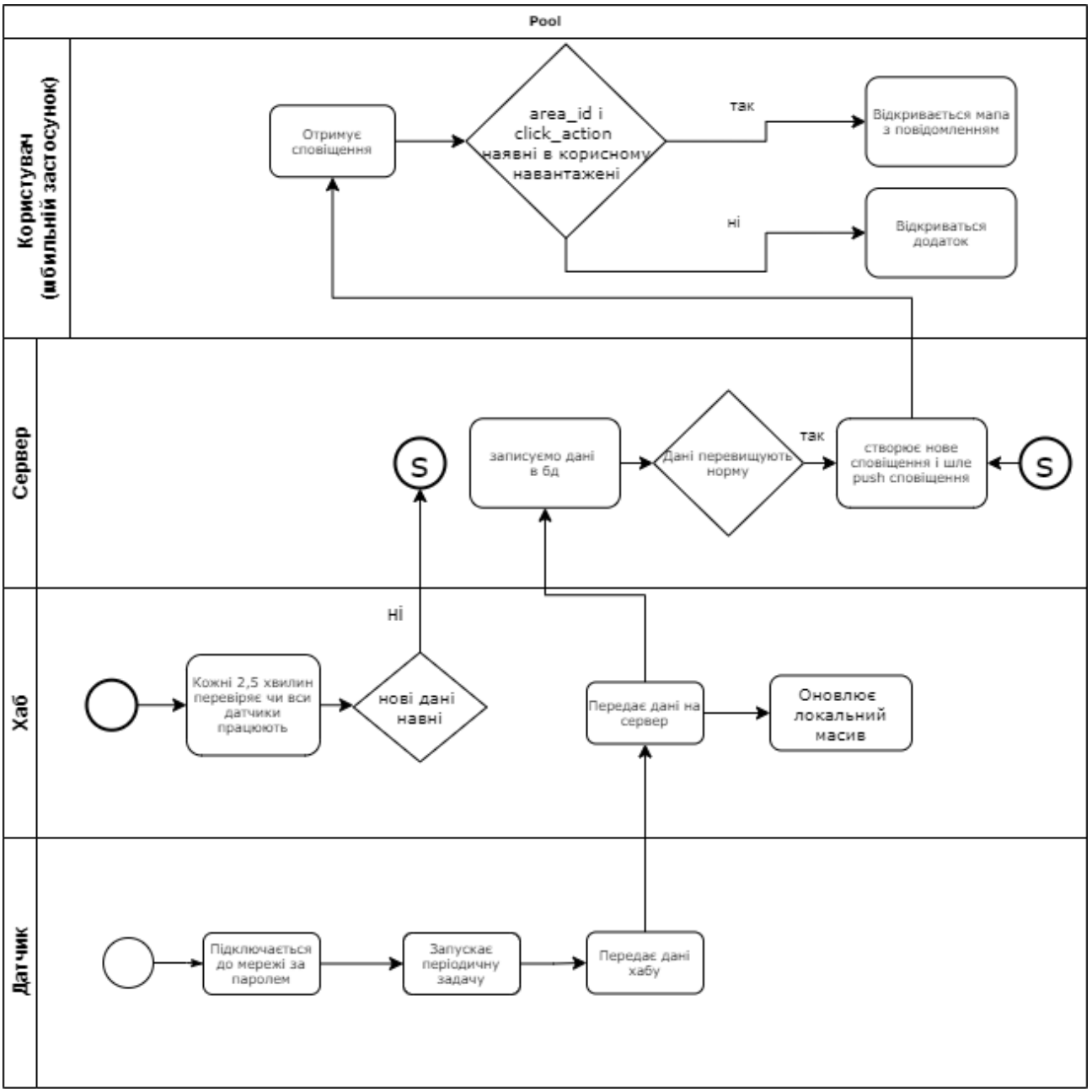


Рисунок 2.1. Потік даних в мережі

Для створення даної системи будуть використанні наступні програмні компоненти:

- технології react, react-native, expo для створення мобільного застосунку, що дозволяє застосувати js у процесі написання
- Nodejs, express, firebase-admin для імплементації сервера та відправки сповіщень
- Postgresql, pgAdmin4 - для реалізації бази даних та її підтримки

- painlessMesh.h - для утворення меш мережі

Архітектура системі показана на рисунку 2.2

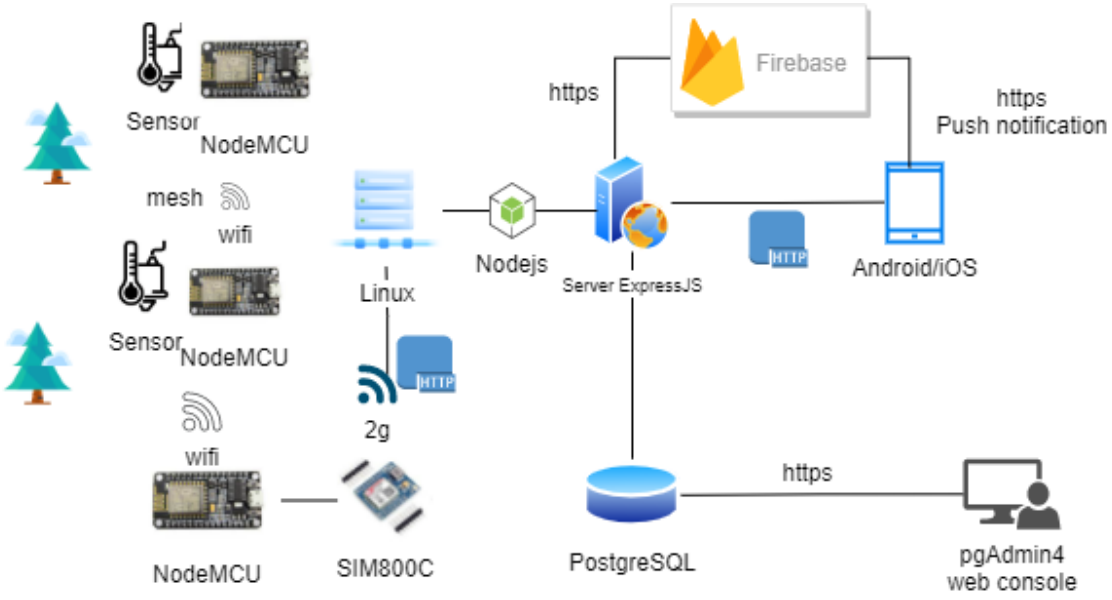


Рисунок 2.2. Архітектура системи

Вхідні та вихідні дані наведено на рисунку 2.3. Хаб може надсилати на сервер як і дані з датчиків, так і інформацію про відсутність таких даних. Мобільних застосунок може отримувати дані про аномалії в push сповіщення та у відповідь на запит до сервера. Для відображення мапи потрібні також дані від сервера.

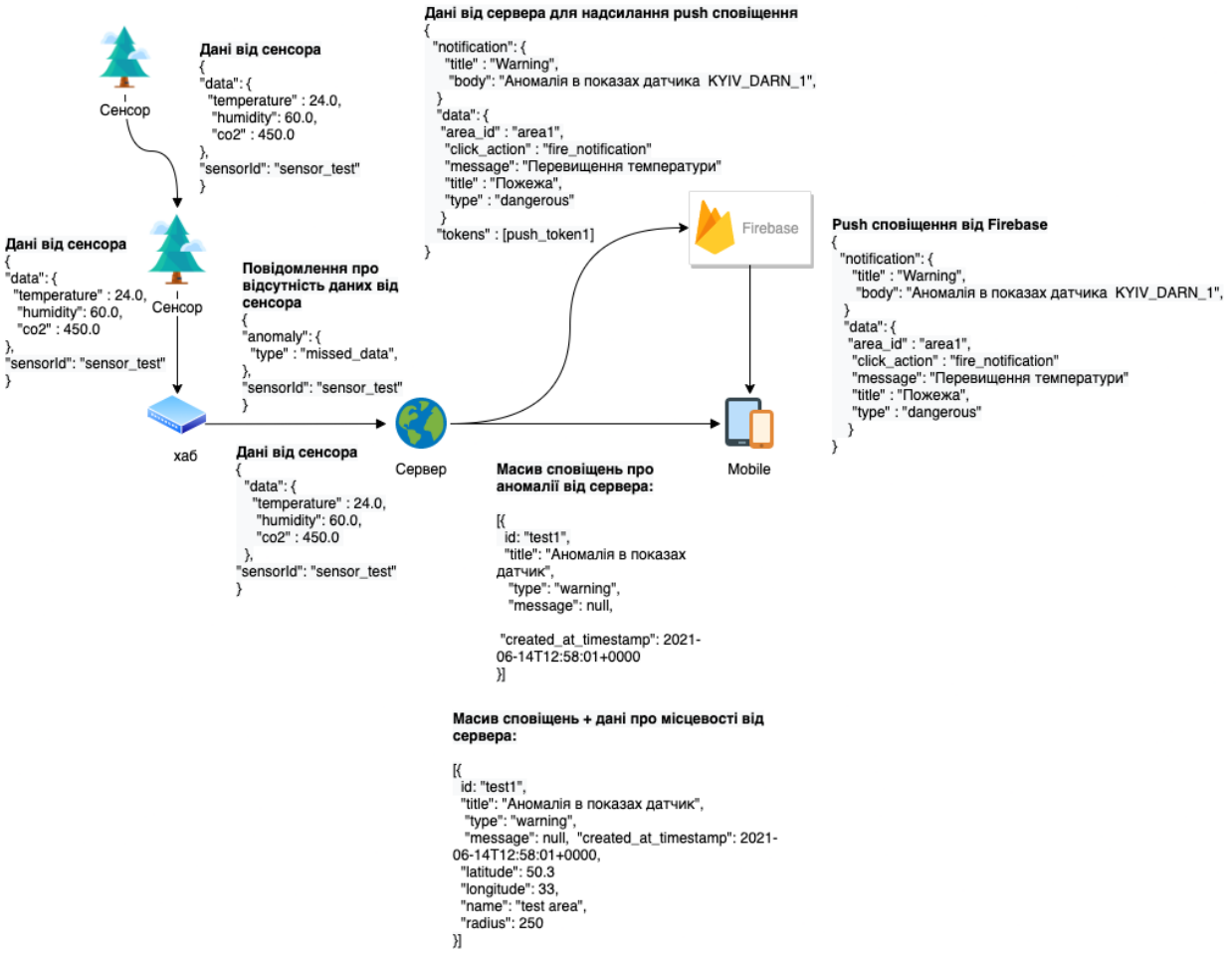


Рисунок 2.3. Вхідні та вихідні дані системи

Двома найпоширенішими мобільними операційними системами на даний момент є Android та iOS [12]. Оскільки автор даної роботи, тобто я, працюю з мовою програмування javascript та для пришвидшення розробки було обрано саме технологію React Native, що дозволить мати одну кодову базу для декількох платформ. А ще ця технологія підходить для створення веб-інтерфейсів [13].

Для збереження даних було обрано реляційну базу даних postgres через те, що у нас уже були sql коди для створення подібних реляційних таблиць із попередніх робіт, а postgres є безкоштовним проектом з відкритим програмним кодом. Для віддаленого доступу до бази даних можна також використати зручний інструмент pgAdmin4.

Апаратні засоби надані в таблиці 2.2.

Таблиця 2.2. Апаратні засоби для створення системи моніторингу лісових пожеж

Назва компоненту	Фото	Характеристики
1	2	3
SIM800C V2		Робота в мережах: 850/900/1800/1900 МГц Розміри чіпа: 15,7 x 17,6 x 2,3 мм Управління AT командами (3GPP TS 27.007,27.005 та пропрієтарні AT команди) Діапазон вхідного напруження: 3,4 ... 4,4 В Низьке енергоспоживання Робочий діапазон температур: -40 ... 85 градусів за Цельсієм Пакетна передача даних GPRS Вбудований протокол TCP / UDP FTP / HTTP
GSM антена SMA роз'єм		частота: 824 МГц ~ 960 МГц / 1710 МГц ~ 1990 МГц
Набір перемичок		

Таблиця 2. Продовження

1	2	3
Датчик температури LM35DZ аналоговий		Аналоговий датчик LM35DZ призначений для вимірювання температури в діапазоні від -55 до + 150 ° C. Точність перетворення 0.5 °C (при + 25 °C) Робочий діапазон вимірювань від -55 до +150 °C Робочий діапазон напруг від 4 до 30 Вольт Струм менше 60 мкА
Модуль датчиків якості повітря CCS811 + HDC1080 (CO2+VOCs)		Робоча температура: -20 ° C ~ 70 ° C Діапазон виявлення eTVOC: 0 ~ 32768ppb Діапазон виявлення eCO2: 400 ~ 32768ppm Напруга живлення: рекомендується 3,3 V Потужність: до 60 мВт Можливість налаштування і зчитування часу датчика після включення: не менше 20 хвилин Co2, температура, вологість
Макетна плата мікро безпаєчна SYB-170 170 точок		

Таблиця 2. Продовження

1	2	3
Wi-Fi модуль NodeMCU V3 ESP8266 (CH340)		WiFi 802.11 b / g / n підтримка STA / AP / STA + AP режимів вбудований стек протоколів TCP / IP з підтримкою множинних клієнтських підключень (до 5) D0 ~ D8, SD1 ~ SD3: можуть бути використані як GPIO, PWM, IIC, тощо. ток на виведення: 15 мА AD0: 1 виведення АЦП живлення: 4.5 - 9В (10В максимум), живлення від USB перепрошивка з хмари або через USB відстань між контактними пинами: 28 мм діапазон робочих температур: -40 ~ +125 ° C
Модуль автономного живлення 2А на 18650 з USB-виходом		Тип акумулятора: 18650 Li-Ion (без захисту) Напруга зарядного пристрою: від 5В до 8В Вихідні напруги: ● 3В - безпосередньо з акумулятора через захисний пристрій ● 5В - через підвищувальний перетворювач. Максимальний вихідний струм: ● Вихід 3В - 1А ● Вихід 5В - 2А

Таблиця 2. Продовження

1	2	3
Акумулятор LiitoKala NCR18650B 3400мАч без захисту		Модель: NCR18650B Тип: Li-ion 18650 без захисту Плата захисту: Немає Номінальна ємність 3350 mAh Макс.ток розряду 4.8 A Номінальний струм заряду: 0.5c * (1 625 мА) Номінальна напруга: 3.6 В Мінімальна напруга: 2.7 В Максимальна напруга: 4.2 В Температурні режими: заряд: 0-45 ° C / розряд: від -20 до 60 ° C Довжина: 68 мм Діаметр: 18.5 мм Вага: 46 г
Корпус пластиковий для електроніки		Модифікація: N8AA Розмір: 46x70x134mm Вага: 86 грам виробник: ТОВ «Амбокс», Україна
Хомут металевий E.Next PPA e.steel.tie.ppa.10.900		Тип хомута: Кабельний Матеріал: Нержавіюча сталь Довжина, мм: 900мм Ширина, мм: 10мм

На рисунку 2.4 показано фізична модель бази даних.

Оскільки ми використовується базу даних Postgresql, то використаємо наступні типи даних - text, numeric, timestamp.

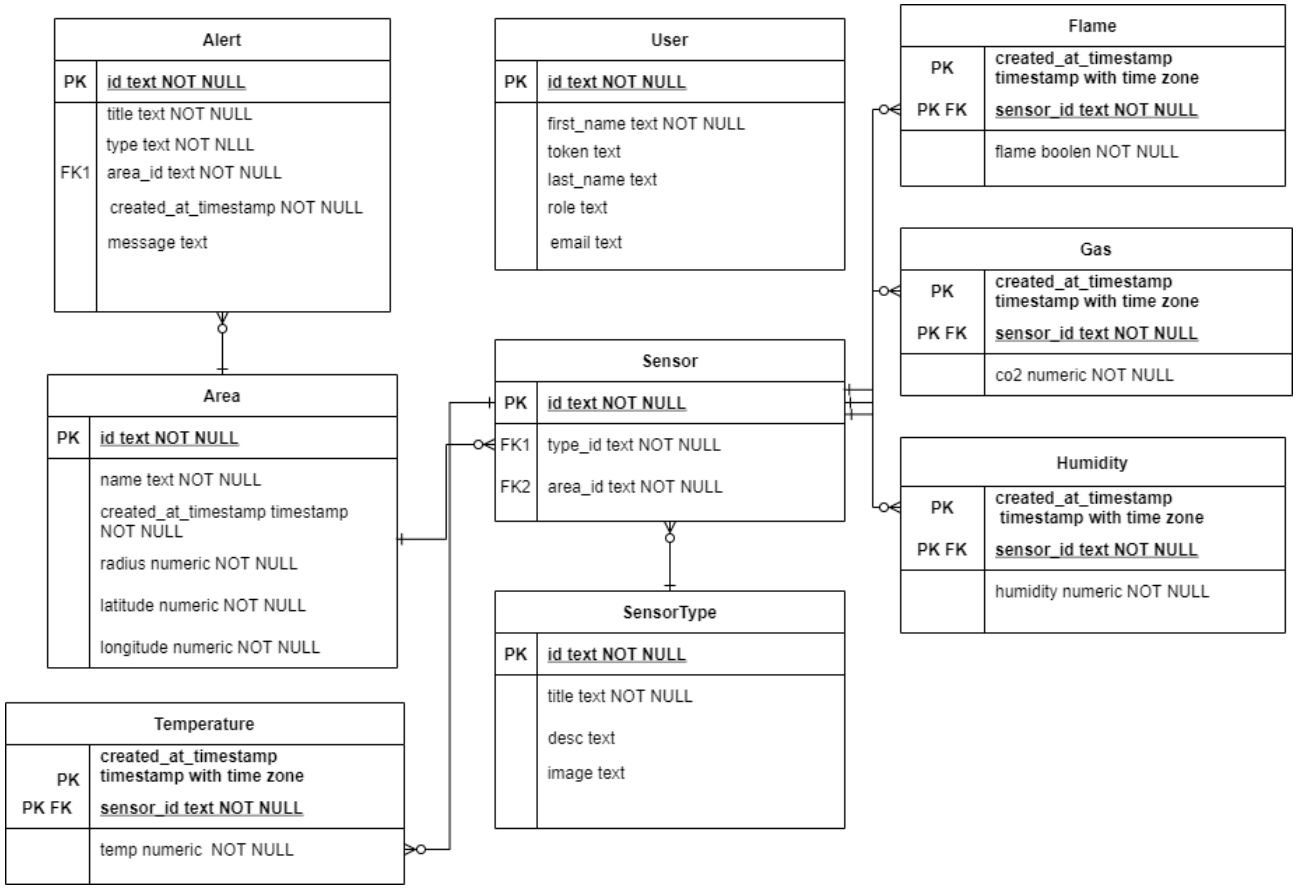


Рисунок 2.4. Фізична модель бази даних системи детектування лісових пожеж.

База даних містить 9 таблиць:

Alert (сповіщення)

Area (місцевість)

User (користувачі)

Temperature (покази температури)

Sensor (датчики)

SensorType (типи датчиків)

Flame (Для датчиків вогню)

Humidity (Вологість)

Gas (покази вуглекислого газу)

Зв’язки між таблицями :

SensorType - Sensor - один до багатьох.

Sensor - Area - багато до одного.

Alert - Area - багато до одного.

Temperature – Sensor - багато до одного

Flame – Sensor - багато до одного

Humidity – Sensor - багато до одного

Gas – Sensor - багато до одного

Опис таблиці здійснюється у вигляді: назва таблиці, назва поля, тип даних поля та відмітка ключа.

Таблиця Alert має поля:

id text NOT NULL PRIMARY KEY,

title text NOT NULL,

type text NOT NULL

area_id text NOT NULL FOREIGN KEY

message text

created_at timestamp NOT NULL

Таблиця Area має поля:

id text NOT NULL PRIMARY KEY,

name text NOT NULL

created_at_timestamp timestamp NOT NULL

radius numeric NOT NULL

latitude numeric NOT NULL

longitude numeric NOT NULL

Таблиця User має поля:
id text NOT NULL PRIMARY KEY,
first_name text NOT NULL,
last_name text,
email text
role text,
token text;

Таблиця Sensor має поля:
id text NOT NULL PRIMARY KEY,
type_id text NOT NULL FOREIGN KEY,
area_id text NOT NULL FOREIGN KEY

Таблиця SensorType має поля:
id text NOT NULL PRIMARY KEY,
title text NOT NULL,
desc text,
image text;

Таблиця TEMPERATURE має поля:
created_at_timestamp timestamp NOT NULL PRIMARY KEY,
sensor_id text NOT NULL PRIMARY KEY FOREIGN KEY,
temp numeric NOT NULL,

Таблиця Flame має поля:
created_at_timestamp timestamp NOT NULL PRIMARY KEY,
sensor_id text NOT NULL PRIMARY KEY FOREIGN KEY,
flame boolean NOT NULL,

Таблиця Gas має поля:

created_at_timestamp timestamp NOT NULL PRIMARY KEY,
sensor_id text NOT NULL PRIMARY KEY FOREIGN KEY,
co2 numeric NOT NULL,

Таблиця Humidity має поля:
created_at_timestamp timestamp NOT NULL PRIMARY KEY,
sensor_id text NOT NULL PRIMARY KEY FOREIGN KEY,
humidity numeric NOT NULL

В якості веб-сховища не було обрано хмарні сервіси, тому що amazon [14], firebase [15] враховують кількість звернень до сервера, в нашому випадку кількість датчиків може бути велика, тому було вирішено створити власний веб-сервер на орендованій машині на операційній системі Linux. Express було обрано адже це найвідоміший фреймворк для nodejs [16]. Для відправки сповіщень було обрано хмарний сервіс від Firebase, тому що на даний момент він безкоштовний та уже має нативні бібліотеки для Android та iOS, які приймають дані від шлюзу сповіщень, до якого ми з власного серверу уже будемо через REST API відправляти потрібні дані. Нам потрібно лише зберігати унікальні push токени мобільних телефонів у власній базі даних. Для react-native існують декілька бібліотек-обгорток для нативних бібліотек, наприклад, react-native-push-notifications, expo-notification, react-native-firebase та інші [17].

Для зв’язку з сервером було обрано модуль SIM800C вміє працювати з 5В чи 3,3В, а у SIM800L наприклад напруга логічних рівнів сягає 2.8В, що потребує додаткових перетворювачів до 3.3В або 5В.

Даний модуль на максимумі може споживати струм 2А, тому потрібне відповідне джерело енергії. Модуль автономного живлення 2А на 18650 з USB-виходом здатний видавати такий струм, він працює з акумулятором типу Li-ion 18650 без захисту.

LM35DZ було обрано через дешевизну та хороші відгуки

NodeMCU дозволяє використовувати технологію Wi-FI і в той же час ця плата містить мікроконтролер та потрібну кількість пінів для підключення усіх потрібних компонентів, ціна становить у межах 100 гривень на час написання даної роботи.

CCS811 має менший час розігріву, ніж MQ-7, а також більш мініатюрний і коштує менше, ніж інші датчики якості повітря такі як датчик CO2 MH-Z19B.

Для реалізації push сповіщень потрібний сервер, який буде відправляти дані на мобільний телефон та сервіс, який буде працювати у фоні, приймати дані із сервера та викликати методи операційної системи для появи сповіщень. Існують хмарні сервіси, які надають послугу відправки push сповіщень. Одним із таких сервісів є Firebase Cloud Messaging. Цей хмарний сервіс працює у співпраці з нативними бібліотеками для iOS та Android. Для роботи в React Native для цих бібліотек існує обгортка у вигляді ще однієї бібліотеки - react-native-firebase, за допомогою якої можна згенерувати унікальний для даного девайсу push токен, підписатися та отримувати віддалені сповіщенння. Бібліотека розрізняє сповіщення, які прийшли в той час, коли мобільний застосунок працював та був відкритий або закритий чи згорнутий.

React - це бібліотека для створення графічних інтерфейсів за допомогою мови програмування javascript. Під час написання інтерфейсу може застосовуватися спеціальний синтаксис написання програми - jsx, що є поєднанням HTML та javascript, що дозволяє пришвидшити процес написання інтерфейсу:

```
function renderData(user) {
  if (user) {
    return <h5>Hello, {formatName(user)}!</h>;
  }
  return <h5>Hello, Stranger.</h5>;
}
```

Будь яка сторінка в інтернеті містить певний набір HTML тегів, які можуть бути вкладеними один в одного. Для спрощення доступу до цих тегів,

якщо така потреба є, використовується стандарт DOM (об’єктна модель документа). Таким чином за допомогою концепту DOM можна створювати динамічні інтерфейси і змінювати теги та їх вміст уже під час користування інтерфейсом. Щоб було зрозуміліше наведемо приклад методу, який визначений у стандарті DOM - getElementsById, даний метод повертає список всіх елементів, у яких ідентифікатор має певне значення.

ReactJS має свою віртуальну об’єктну модель документа, яка зберігається в оперативній пам’яті та дозволяє ефективніше відображати зміни на сторінці. Одним із принципів, за допомогою яких досягається оптимізація продуктивності роботи бібліотеки, є наявність так званих ключів для елементів, які відображають змінився елемент чи ні. [18]

ReactJS використовується для написання веб-застосунків, проте він може застосовуватися у зв'язку з іншими бібліотеками та фреймворками для написання наприклад мобільних або десктопних застосунків. [19]

Javascript постійно розвивається і додаються нові можливості, проте не всі браузери поспівають за цими змінами, тому розробники інколи використовують утиліти для перетворення новішого коду в такий, що розуміється більшістю браузерів. Однією із таких утиліт є Babel. Крім вищезгаданих функції Babel уміє також перетворювати синтаксис jsx у звичайний javascript код. Babel компілює JSX до викликів React.createElement().

У 2015 році, після появи ReactJS, з’явився фреймворк для створення мульти-платформних застосунків - React Native, який використовується принципи ReactJS та дозволяє одночасно створювати програми і на iOS, і на Android. Оскільки в мобільних додатках не існує такого поняття як DOM, натомість кожна операційна система для створення графічних інтерфейсів має власний набір “тегів” (елементів), з якими ядро операційної системи вміє працювати. Після запуску програми на react-native ті елементи, якими ми описували інтерфейс (View, Text,Image) перетворюються на відповідні елементи на кожній із платформ (TextView на андроїді

або UITextView на айос). Але цього не достатньо, адже наша програма може змінювати інтерфейс в процесі роботи, тому потрібно якось реагувати на події користувача. React-Native використовує javascript і задача перетворення коду написаного на одній мові програмування на іншу мову програмування є складною і вимагає додаткових зусиль і зацікавлених осіб, які могли б підтримувати такі засоби перетворення, адже з часом додаються все новіші функції і можливості, які потрібно постійно було б вносити та додавати. Сучасні мобільні телефони можуть виконувати код не тільки написаний спеціалізованими мовами програмування, а й і джаваскрипт.

Всі сучасні телефони надають можливості переглядати веб-документи (веб-сайти), які можуть містити код javascript, тому компаніям, що випускають дані телефони довелося знайти спосіб виконувати javascript код. На айос наприклад це WebKit, який використовує JavaScriptCore для виконання коду та інші механізми та утиліти для зберігання історії відвідувань або для імплементації механізму гіперпосилань.[20]

JavaScriptCore - це оптимізована віртуальна машина.

Для зв'язку та обміном даними між віртуальною машиною JavaScriptCore та нативним, таким що написаний рідною мовою мобільної операційної системи, перетвореним у React Native кодом існує спеціальний міст зі своїм форматом повідомлен [21].

Javascript не є ідеальною мовою програмування та має певні особливості, які роблять її легкою у вивченні.

TypeScript намагається заповнити прогалини javascript. Для роботи з typescript достатньо компілятора, який перетворюватиме написане в javascript код.

JavaScript - це інтерпретована мова. Отже, код потрібно запустити, щоб перевірити, чи є він справним. Це означає, що інколи можна витратити доволі багато час на те, щоб віднайти помилку в коді. Транслятор TypeScript надає функцію перевірки помилок. TypeScript відразу повідомить, якщо виявить певні синтаксичні помилки, ще під час процесу написання програми.

JavaScript не є суворо типізованою мовою. TypeScript постачається з необов’язковою системою статичного введення та виведення тексту через TLS (мовна служба TypeScript). Тип змінної визначається на основі її значення, якщо спочатку не вказаний. А це означає що можна навіть не завжди вказувати напряму тип змінної, а дана служба самостійно визначить тип, що пришвидшує процес написання програми

В TypeScript можна визначати типи для існуючих бібліотек JavaScript у спеціальному файлі із розширенням .d.ts. Отже, код TypeScript може містити ці бібліотеки.

TypeScript підтримує концепції об’єктно-орієнтованого програмування, такі як класи, інтерфейси, успадкування тощо. [22]

В основі TypeScript є такі три компоненти -

- Мова - вона складається із синтаксису, ключових слів та анотацій типу.
- Компілятор TypeScript - перетворює інструкції, написані на TypeScript, в еквівалент JavaScript.
- Мовна служба TypeScript - підтримує загальний набір типових операцій редактора, таких як завершення операторів, форматування та окреслення коду, забарвлення тощо.

Redux - це популярна бібліотека JavaScript для управління станом програми.

Стан програми це глобальне сховище, яке зберігає інформацію, яку ви використовуєте для різних цілей пізніше в додатку (наприклад, прийняття рішень щодо того, які компоненти і коли відображати, візуалізація збережених даних тощо).

Прикладом, з яким ми часто стикаємось, є відображення індикатора завантаження під час завантаження сторінки. У цьому випадку, якби ми використовували сховище лише для цієї мети, об’єкт стану зберігав би логічне поле, чи сторінка завантажена, і ми б використовували це поле для перемикання відображення індикатора завантаження.

Великі програми мають великі стани додатків, і управління ними стає все більш незручним у міру зростання додатка. Крім того, ми можете мати компоненти, які використовують ті самі дані, але розміщуються випадково в дереві DOM, саме тому і потрібна дана бібліотека.

Є дві найважливіші моделі, яких дотримується Redux.

Один шаблон, якого дотримується Redux, називається “Single Source Of Truth” [23], що означає, що у нас є лише одне місце (зване Store), де ми зберігаємо єдиний стан для всієї програми.

Іншими словами, один додаток - одне сховище.

Однак компоненти в React або інших фреймворках також можуть містити свій власний внутрішній стан.

Ще одна закономірність, якої дотримується Redux, називається «незмінність».[24] Незмінність означає, що ми не змінюємо об'єкт стану та його властивості безпосередньо. Натомість ми робимо новий об’єкт, перераховуємо новий стан програми та оновлюємо його за допомогою нашого новоствореного об’єкта. Ми хочемо залишити наш старий об’єкт цілим та неспотвореним. Це потрібно щоб уникнути невловимих багів.

Postgresql - об'єктно-реляційна система керування базами даних. База даних може бути розташовано на власному сервері. Однією з операційних систем для серверів є Ubuntu, для якої існують такі утиліти як netdata, що стежить за використанням ресурсів сервера; ufw - програма для керування фаєрволом, забороняє чи дозволяє трафік через певні порти для певної групи айпі адрес, а також забороняє багаторазові повторні підключення до сервера за певний проміжок часу.

PainlessMesh створює мережу, що самоорганізується та відновлюється, де всі вузли з'єднані. Усі вузли в сітці рівні. Мережа використовує зіркову топологію, уникаючи кругових шляхів. Повідомлення між різними вузлами надсилаються у форматі JSON, що полегшує їх розуміння та створення. Інформацію про схему JSON, яка використовується для різних повідомлень, можна знайти тут. [25]

2.3 Висновки до розділу

Зібравши вимоги до даної системи та розробивши приклад використання системи було підібрано компоненти системи, розроблено архітектуру системи; проаналізовано та підібрано програмні засоби для створення мобільного застосунку, підібрано апаратні засоби для створення системи датчиків для збирання потрібних даних

РОЗДІЛ 3. РЕАЛІЗАЦІЯ IOT СИСТЕМИ

3.1 Реалізація mesh мережі

В якості основи для кінцевих девайсів була обрана плата nodemcu через простоту та швидкість розробки. В системі будуть наявні два типа девайсів - хаби та сенсори.

Було обрано такі датчики для використання в системі:

- Модуль датчиків якості повітря CCS811 + HDC1080 (CO2+VOCs).
- Датчик температури LM35DZ аналоговий.
- Модуль датчика вогню.

Додаткові дані від датчика вогню та датчиків якості повітря допоможуть прийняти рішення у випадку аномалії в показах датчика температури.

Схема підключення датчиків для першого девайсу представлена на малюнку 3.1. Фізичне підключення показано на рисунку 3.2.

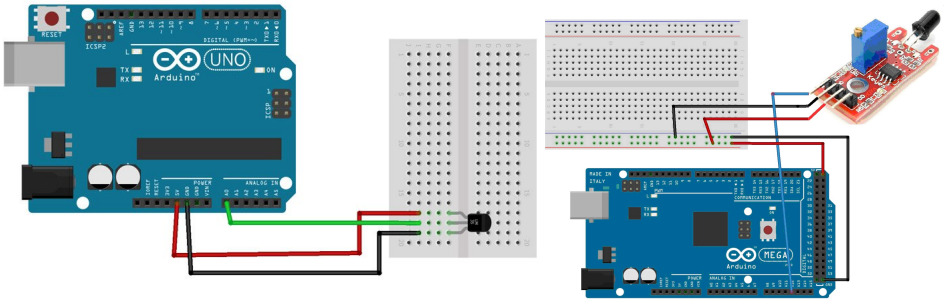


Рисунок 3.1. Схематичне підключення модуля датчика вогню та датчика температури

Схема підключення модуль датчиків якості повітря CCS811 + HDC1080 до плати nodemcu (рис 3.3) наведено в таблиці 3.1

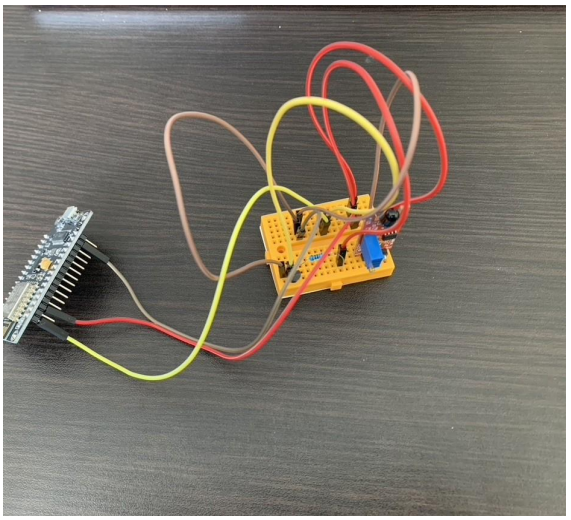


Рисунок 3.2. Фізичне під'єднання модуля датчика вогню та датчика температури до плати NodeMCU

Таблиця 3.1 Співставлення портів CCS811 і плати nodemcu

Піни CCS811 + HDC1080	**Піни nodemcu**
1	**2**
VCC	3V
GRN	GRN
SCL	D1
SDA	D2
WAK	D3

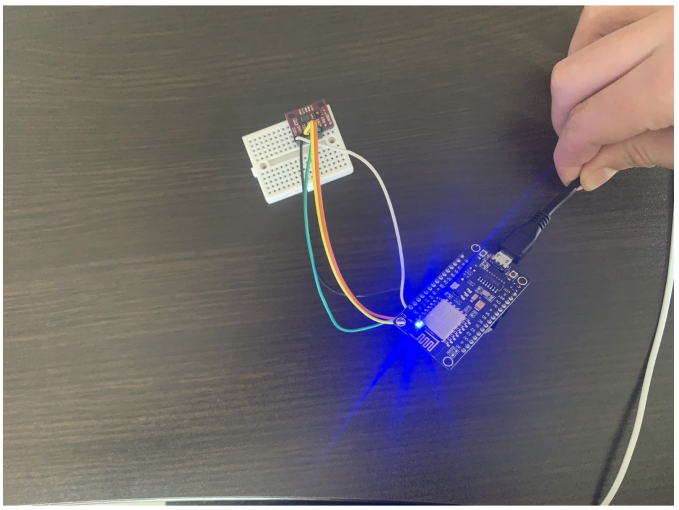


Рисунок 3.3. Фізичне під'єднання модуль датчиків якості повітря CCS811 + HDC1080 до плати nodemcu

Для під'єднання до мережі було використано модуль GSM сотового зв'язку SIM800C, схема підключення якого до плати esp32 наведена на рисунку 3.4, що має ті ж потрібні порти що й nodemcu

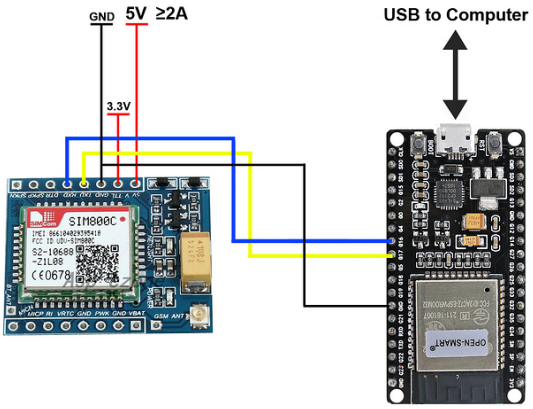


Рисунок 3.4. Схема підключення SIM800C до ESP32

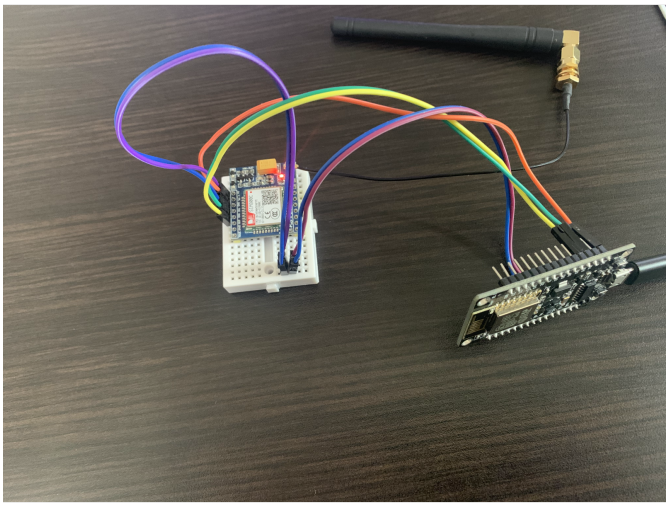


Рисунок 3.5. Фізичне підключення модуля сотового зв'язку SIM800C до плати nodemcu

Для утворення mesh-мережі була використана бібліотека painlessMesh (рис 3.6). Кожен вузок повинен знати префікс мережі, пароль для мережі та порт. Використаємо наступні значення для нашої мережі:

```
#define   MESH_PREFIX        "FOREST_FIRES_NETWORK"
#define   MESH_PASSWORD   "forestlive"
#define   MESH_PORT    5555
```

Для зручності роботи з даними інсталюємо бібліотеку Arduino_JSON

Для запуска вузла в даній мережі потрібно використати метод

```
mesh.init( MESH_PREFIX, MESH_PASSWORD, &userScheduler, MESH_PORT );
```

Вузол який буде виступати в ролі хаба має також функцію яка спрацьовує після отримання даних надісланих будь яким вузлом мережі, підписатися на такі повідомлення можна наступним чином:

```
mesh.onReceive(&receivedCallback);
void receivedCallback( uint32_t from, String &msg ) {
  Serial.printf("startHere: Received from %u msg=%s\n", from, msg.c_str());
}
```

Для кожного вузла, окрім хаба, створимо періодичну задачу для збору даних і відправки їх в мережу. Для цього скористаємося класом userScheduler із бібліотеки painlessMesh:

Task taskSendMessage(TASK_SECOND * 60 (може 60 секунд?), TASK_FOREVER, &sendMessage);

Для створення екземпляру класу потрібно задати час, через який буде запускатися задача, кількість разів (в нашому випадку постійно) та функція яка буде запускатися - sendMessage, в якій відбуватиметься відправка даних в мережу: mesh.sendBroadcast(msg);

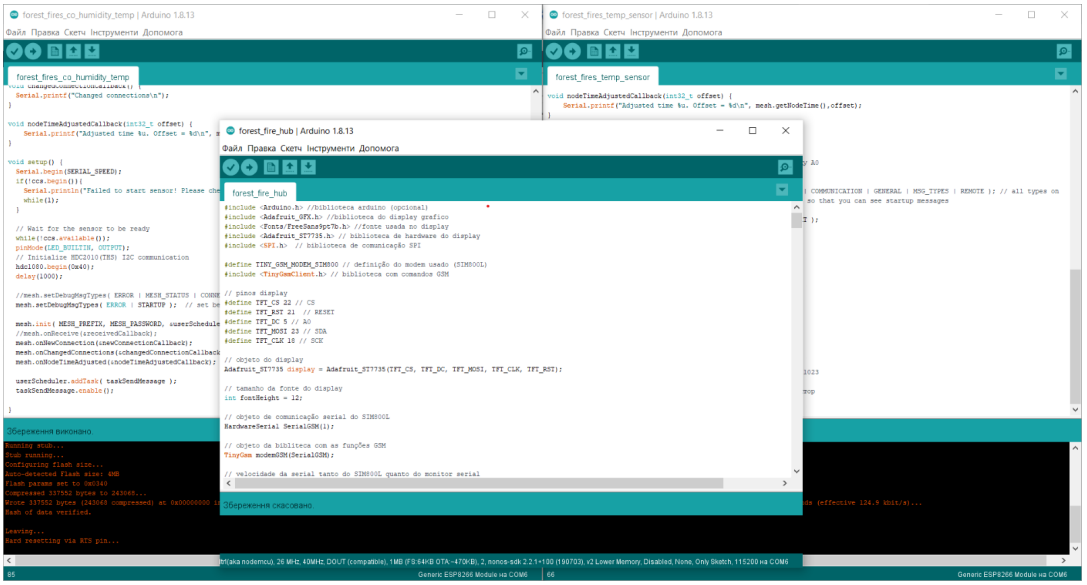


Рисунок 3.6. Робота з Arduino IDE та використання бібліотеки painlessMesh

Сервер було реалізовано за допомогою технології nodejs. Для реалізації серверу було використано фреймворк express, для підключення до бази даних - бібліотека pg. Запуск сервера відбувається за допомогою команди nodemon server.js. Пакет nodemon автоматично перезапустить сервер, якщо в ході роботи виникне помилка та сервер закінчить свою роботу передчасно.

В папці routes створимо файли для кожної таблиці (рисунок 3.18) та в кожному файлі опишемо тип запиту та відповідні дії, sql запити та повернення потрібних даних у відповідь на запит.

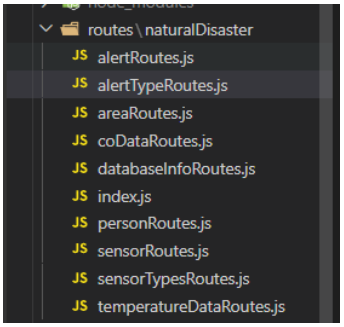


Рисунок 3.7. Список “маршрутів” нашого сервера

Створимо посилання за допомогою яких будемо взаємодіяти з сервером та базою даних. Приклад створення:

```
const temperatureDataRoutes = express.Router({mergeParams: true});
temperatureDataRoutes
  .route('/')
  .get(async function(req, res) {
```

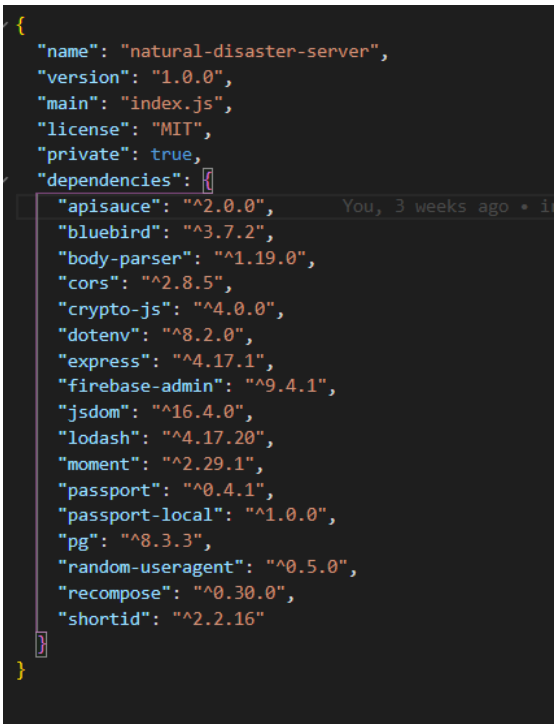


Рисунок 3.8. Список залежностей, які були використані для створення власного сервера

Створимо новий файл server.js, в якому підключимо усі описані шляхи для запитів та створимо новий екземпляр сервера на порту 4040 (Рисунок 3.9). Запустимо сервер на власному віддаленому віртуальному приватному сервері командою nodemon server.js

```
                const user = dbResponse.rows[0]
                req.user_role = user.role;
                client.release()

            }
        }catch(err){
            console.log('auth failed',err)
        }

        next();
    };

    app.use(firebaseAuth);

    app.use(cors());
    app.use(express.json());
    app.use('/natural-disaster', naturalDisasterRoute);
    app.route('/test', (req, res) => {
        res.sendStatus(200)
    });


    console.log('database connected...')
    app.use((req, res, next) => {
        next();
    });

    const server = http.createServer(app)
    server.listen(PORT, "0.0.0.0", (err) => {
        if (err) {
            return console.log('something bad happened', err)
        }
        console.log(`server is listening on ${PORT}`)
    })
```

Рисунок 3.9. Файл для запуска сервера на заданому порті

Для тестування використаємо таку програму як postman (рис 3.9)

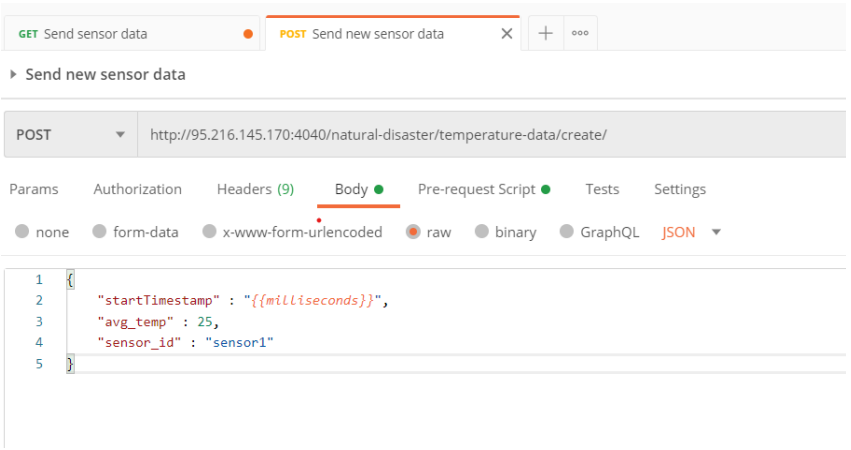


Рисунок 3.10. Тестування посилання для взаємодії із сервером

Відкриємо панель керування базою даних та перевіримо наявність даних (рис 3.11)

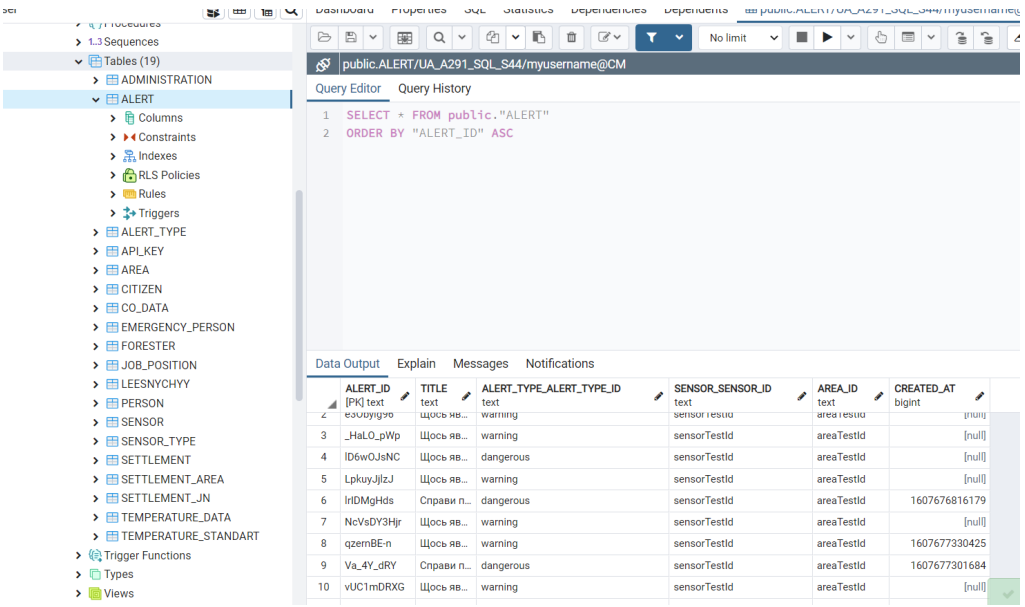


Рисунок 3.11. Вигляд адмінки для адміністрування бази даних Postgresql

3.2 Реалізація мобільного застосунку

Загальний принцип роботи мобільного застосунку наведено на рисунку 3.12. Користувач повинен ввести логін та пароль для авторизації, які будуть оброблені на хмарному сервісі Firebase. Далі запрошуємо дозвіл на сповіщення,

на андроїд цей дозвіл за замовчуванням надано. Якщо дозвіл надано надсилаємо пуш токен на сервер і записуємо в базу даних postgres. Завантажуємо список усіх отриманих раніше сповіщень. При отриманні сповіщення, користувача буде на екран з мапою з детальною інформацією про аномалію після кліку на сповіщення. Сповіщення надсилаються за допомогою хмарного сервісу Firebase.

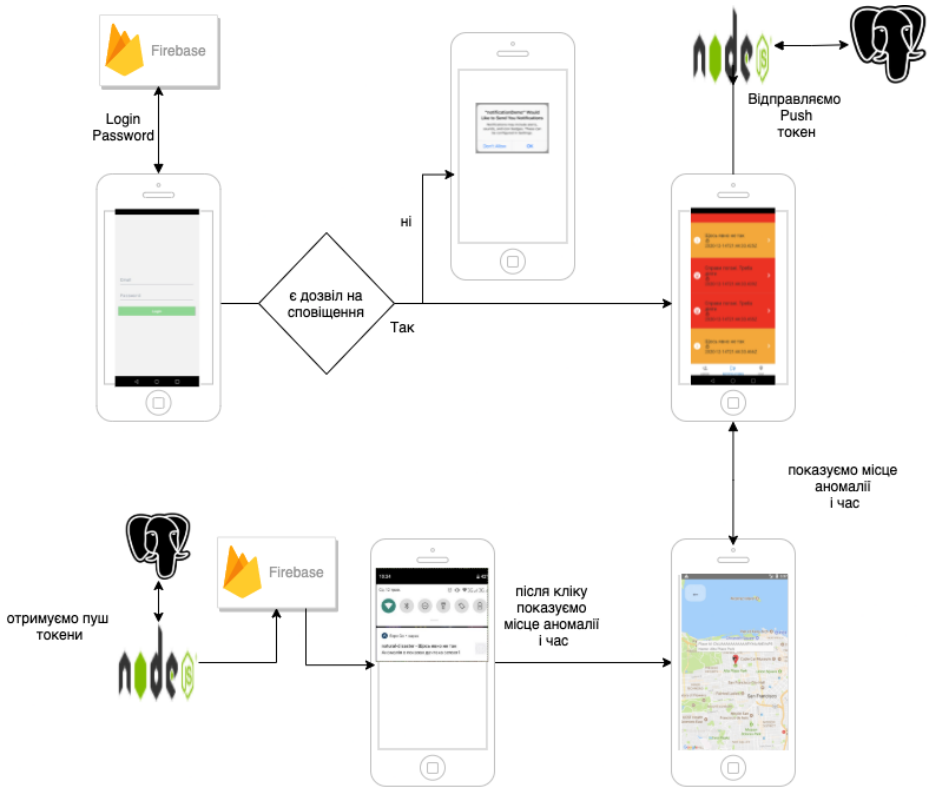


Рисунок 3.12. Принцип роботи мобільного застосунку

Для налаштування авторизації спочатку потрібно створити проект у Firebase. Переходимо за посиланням - https://console.firebase.google.com/u/1/ та створюємо новий проект (рисунок 3.13).

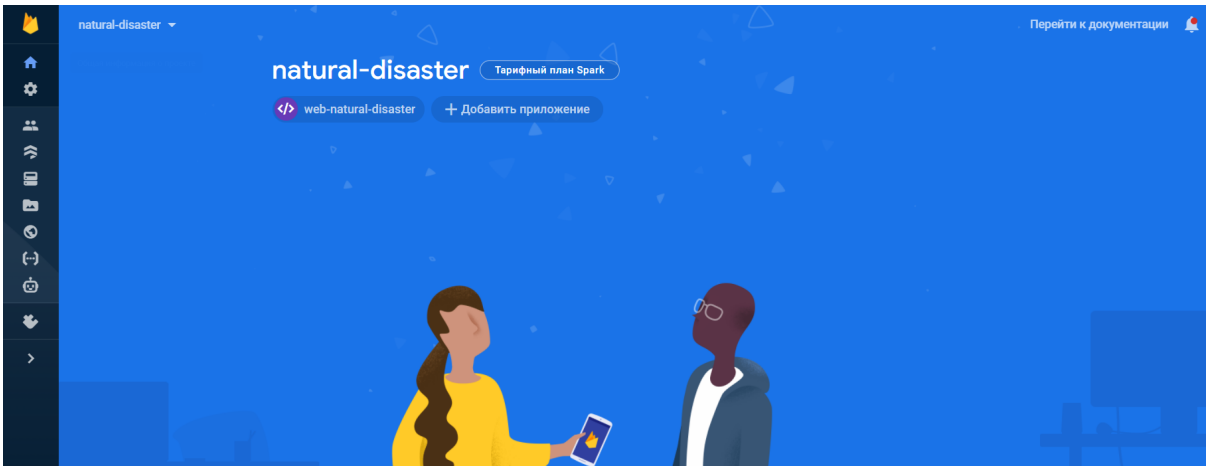


Рисунок 3.14. Стартова сторінка нового проекту firebase

Потрібні дані ми занесемо в додаток для зв’язку консолі та додатку

На вкладці Authentication створимо нових користувачів (рисунок 3.14) , дані яких будемо використовувати в додатку. Потрібно не забути внести потрібні дані та ролі користувачів також в нашу базу даних.

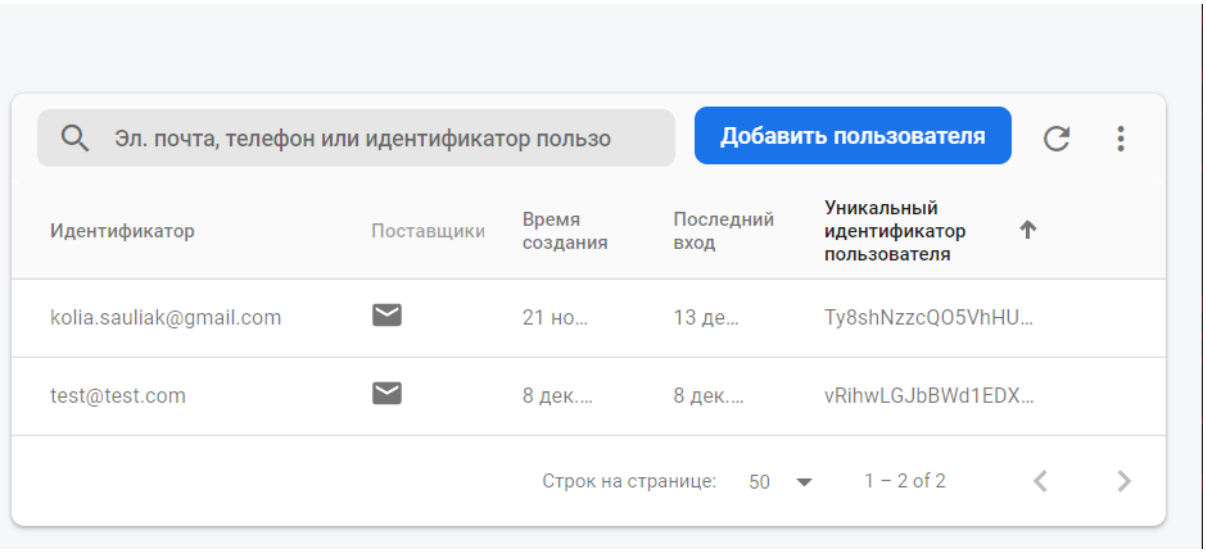


Рисунок 3.15. Створення нових користувачів в консолі firebase

Для навігації використаємо бібліотеку react-navigation. В папці navigation створимо файл index.tsx із якого імпортуватимемо кореневий навігатор (компонент, який відповідає за навігацію) нашого додатку (рис.3.16) Додатково

створимо файл AdminNavigator.tsx, в який помістимо навігатор, який буде видно лише користувачу з правами адміна

```
const Stack = createStackNavigator();

export default function AdminNavigator() {        You, 3 week
    return (
      <Stack.Navigator
        headerMode="none"
        initialRouteName={screens.TableList}
        screenOptions={{
            headerTitleStyle: {fontFamily: 'Jfwildwood'}
          }}
        >
        <Stack.Screen
            name={screens.TableList}
            component={TableList}
            />
        <Stack.Screen
            name={screens.AlertTypeEditScreen}
            component={AlertTypeEditScreen}
            />
        <Stack.Screen
            name={screens.AlertEditScreen}
            component={AlertEditScreen}
            />
```

Рисунок 3.16. Навігатор для адміністрування бази даних

Оскільки кожна таблиця має свої відмінні поля було вирішено зробити для кожної таблиці свій екран редагування (рисунок 3.17), з якого користувач зможе або створювати новий екземпляр, або редагувати уже наявні дані.

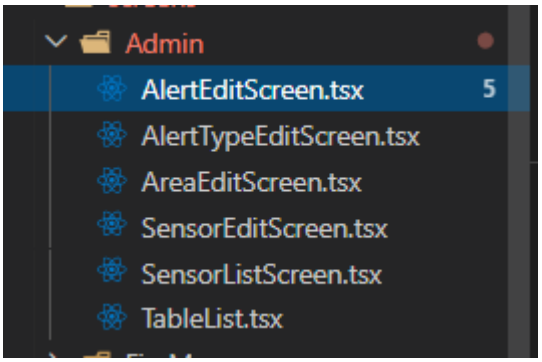


Рисунок 3.17. Файли екранів для редагування даних бази даних

Для зв’язку зі сервером створимо в папці services файл ApiService.tsx, в якому створимо окремий клас, з методами для доступу до кожного endpoint нашого API.

Для управління авторизацією створимо окремий файл AuthService.tsx, в якому помістимо методи для виходу з акаунту, а також обробники подій, таких як зміна користувача, або зміна токену (в такому випадку користувачу потрібно заново ввести дані).

Як уже було сказано раніше для зберігання даних була використана бібліотека redux, умовно розділили усі дані на так звані домени, або фічі, і для кожного домену зробили свою папку, в кожну з яких помістили наступні файли (рисунок 3.18).

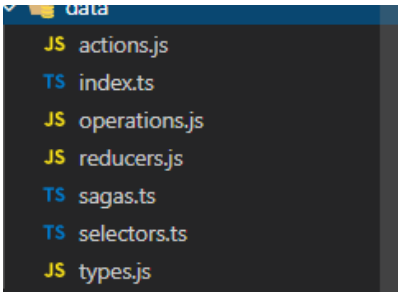


Рисунок 3.18. Список файлів для кожного домену даних

Для створення нового екземпляру та налаштування сховища на основі описаних схем створимо в папці store файл configureStore.tsx (рис 3.19)

Рисунок 3.19. Налаштування та експорт нового сховища із файлу

3.3 Реалізації сповіщень

Механізм спрацювання відправки сповіщень запускається на сервері при отримані даних які перевищують певні порогові значення. Для відправки сповіщень з боку сервера використаємо бібліотеку firebase-admin. Спочатку створимо проект у firebase консолі, завантажимо файл доступу до проекту у форматі json та зробимо ініціалізацію бібліотеки:

Спочатку потрібно завантажити файл з консолі firebase файл доступу на вкладці “Налаштування проекту”, “Сервісний акаунт”

serviceAccount - це і є наш файл доступу, який використовуємо для відправки сповіщень:

```
admin.initializeApp({
  credential: admin.credential.cert(serviceAccount),
  ...
});
```

Для безпосередньої відправки сповіщень використаємо метод:

admin.messaging().sendMulticast(message).

В якості вхідних даних (об'єкт message) передаємо поле notification із текстом, який і буде показано у сповіщенні на телефоні, та масив токенів із бази даних

З боку клієнта (мобільного застосунку) використаємо бібліотеку expo-notifications та метод getDevicePushTokenAsync, який власне і поверне нам потрібний токен, якщо користувач надав всі потрібні дозволи. Для отримання дозволу використаємо бібліотеку expo-permissions та метод: Permissions.askAsync(Permissions.NOTIFICATIONS).

Дозвіл може бути частковим та абсолютний (на всі можливі комбінації сповіщень, звуку та бейджиків)

Запуск всього проекту для мобільного застосунку здійснюється за допомогою команди:

expo start

Для публікації змін використовується команда:

expo publish

Проект мобільного застосунку знаходить у приватному репозиторії на github

3.4 Інструкція користувача

Користувач отримує сповіщення на телефон. Вигляд сповіщення показано на рисунку 3.20.

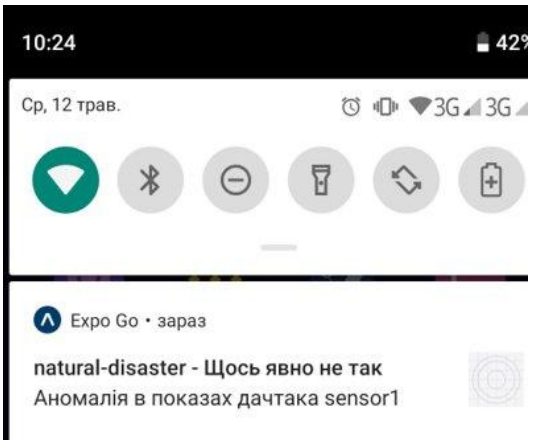


Рисунок. 3.20. Приклад сповіщення

Користувача може відкрити додаток натиснувши на сповіщення або безпосередньо через іконку додатку. Після першого запуску мобільного додатку, користувач попадає на сторінку авторизації. (Рис. 3.21)

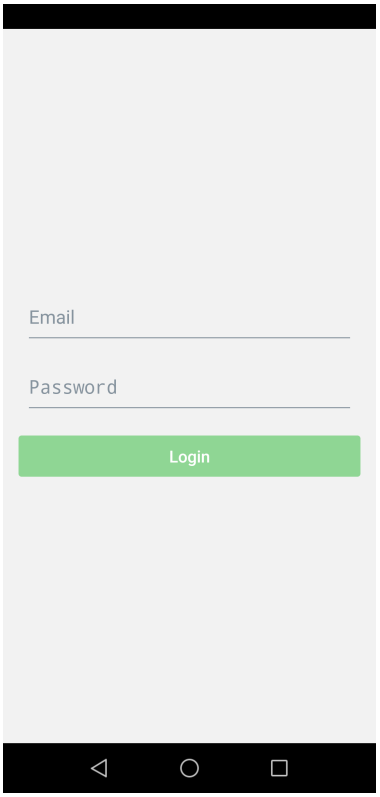


Рисунок 3.21. Сторінка авторизації

Для проходження авторизації потрібно використовувати вже існуючі дані користувача. Ми будемо використовувати користувача “test@test.com” та пароль для нього “123123”.

Після авторизації, ми попадаємо на головну сторінку додатку, де ми можемо переглянути сповіщення створені на основі датчиків (рис. 3.22).

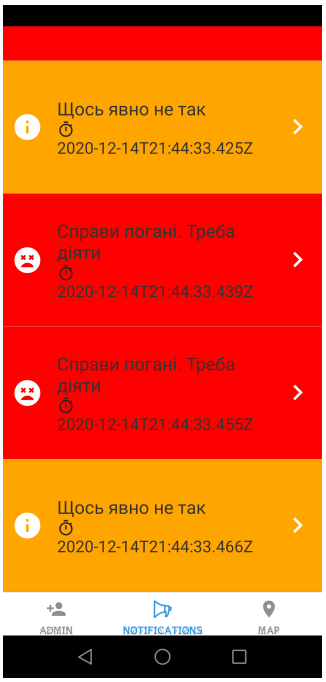


Рисунок 3.22. Екран сповіщень

Знизу можна помітити вкладку “ADMIN”, вона з'являється лише коли користувач має права адміна. На даному екрані (вкладці) знаходиться список деяких доступних таблиць в базі даних (рис. 3.23)

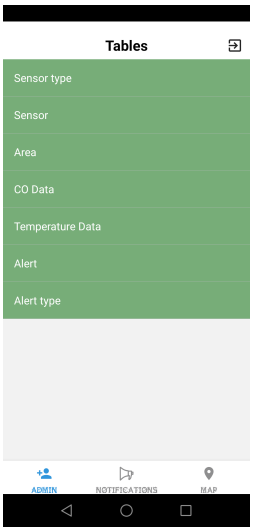


Рисунок 3.23. Екран з переліком таблиць бази даних

Клікнувши, наприклад, на компонент Sensor Type, ми перейдемо на екран, на якому будуть показані всі записи в таблиці SENSOR_TYPE (рис. 3.24)

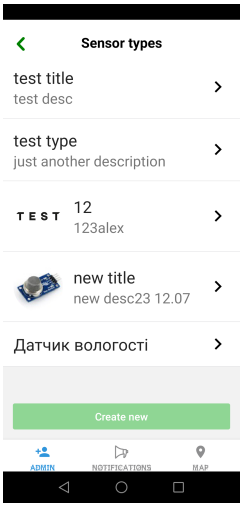


Рисунок 3.24. Екран з показом даних таблиці SENSOR_TYPE

Клікнувши на обраний елемент, ми перейдемо на екран редагування (рис.3.25), також ми можемо створити новий запис в таблиці, перейшовши на

екран створення (рис.3.26) після натиску на кнопку “create new”. В деяких випадках поле id є необов'язковим, та буде заповнене автоматично сервером, у випадку якщо ми нічого не надішлемо.

Рисунок 3.25. Екран редагування запису

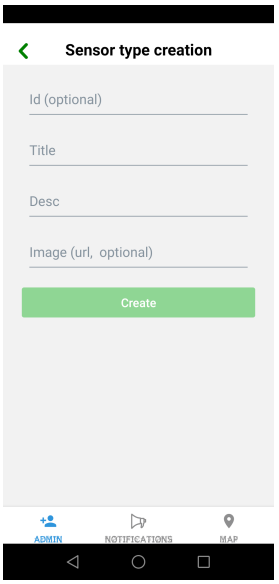


Рисунок 3.26. Екран створення нового запису

Якщо натиснути на системне сповіщення або на екрані сповіщень, то відкривається екран з мапою та короткими відомостями про аномалію (рис 3.27).

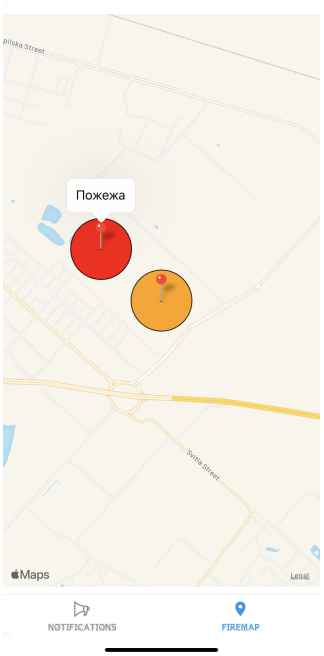


Рисунок 3.27. Приклад відображення мапи з сповіщенням про аномалію

3.5 Висновки до розділу

На основі вибраних раніше технологій було створено мобільний застосунок для платформ iOS та Android, реалізовано меш мережу на основі плати NodeMCU та створено сервер для обробки запитів мобільного додатку та хабів з меш мережі.

ВИСНОВКИ

Під час виконання даної бакалаврської роботи було спроектовано та реалізовано систему детектування та моніторингу лісових пожеж.

Було зібрано базові вимоги до даної мережі.

В якості бази даних використано postgresql, сервер створено та запущено за допомогою таких технологій: nodejs, express, nodemon

Створено меш мережу за допомогою таких компонентів: плата nodemcu, датчик LM35DZ ,модуль зв’язку sim800c, датчик CCS811.

Відправка сповіщень здійснювалася за допомогою пакету firebase-admin, з боку клієнта для отримання токену була використана бібліотека expo-notifications. Для отримання відповідних дозволів використана бібліотека expo-permissions

Наведена логічна та фізична модель бази даних з повним описом складових даних моделей; створено мобільний додаток для роботи з базою даних; створено сервер для роботи з базою даних; наведено порядок створення додатку: налаштування авторизації, виводу таблиць бази даних, операцій з таблицями; також була написання інструкція для нових користувачів.

Підводячи підсумок, можна сказати що тема захисту лісів не є дуже популярною в наш час. Проте є проекти, які хочуть використати сучасні технології для захисту природи, а для цього потрібно вчасно реагувати на аномальні явища, техногенні катастрофи. Розроблений мобільний додаток направлений на мобільне управління базою даних системи захисту лісів та своєчасного отримання сповіщень про виявлену небезпеку, на основі отриманих даних з датчиків температури.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Why action is needed on forest fires [Електронний ресурс] – Режим доступу до ресурсу: https://ec.europa.eu/info/research-and-innovation/research-area/environment/climate-action/forest-fire-research-and-innovation_en.
2. У Держлісагентстві оцінили збитки від пожежі на Луганщині у 4-5 млрд грн [Електронний ресурс] - Режим доступу до ресурсу: https://hromadske.ua/posts/u-derzhlisagentstvi-ocinili-zbitki-vid-pozhezhi-na-luganshini-u-4-5-mlrd-grn
3. A Review on Forest Fire Detection Techniques [Електронний ресурс] – Режим доступу до ресурсу: https://journals.sagepub.com/doi/full/10.1155/2014/597368.
4. IoT for detecting wildfires [Електронний ресурс] – Режим доступу до ресурсу: https://www.amplia-iiot.com/iot-detecting-wildfires/.
5. ESP-MESH with ESP32 and ESP8266: Getting Started (painlessMesh library) [Електронний ресурс] – Режим доступу до ресурсу: https://randomnerdtutorials.com/esp-mesh-esp32-esp8266-painlessmesh/.
6. Stopping a Wildfire with a Low-Cost Sensor Network [Електронний ресурс] / – Режим доступу до ресурсу: https://spectrum.ieee.org/view-from-the-valley/at-work/start-ups/stopping-a-wildfire-with-a-lowcost-sensor-network
7. Design and Construction of a Wildfire Instrumentation System Using Networked Sensors [Електронний ресурс] – Режим доступу до ресурсу: http://firebug.sourceforge.net/index.php.
8. ladsensors [Електронний ресурс] – Режим доступу до ресурсу: https://www.ladsensors.com/.

9. Wildfire Lossses In The United State [Електронний ресурс] – Режим доступу до ресурсу: https://www.iii.org/graph-archive/208963.
10. Client Needs and Software Requirements [Електронний ресурс] – Режим доступу до ресурсу: https://www.coursera.org/learn/client-needs-and-software-requirements.
11. A Requirements Engineering Process for IoT Systems [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/publication/337780133_A_Requirements_Engineering_Process_for_IoT_Systems.
12. Mobile Operating System Market Share Worldwide [Електронний ресурс] – Режим доступу до ресурсу: https://gs.statcounter.com/os-market-share/mobile/worldwide.
13. React Native Components and APIs on the Web [Електронний ресурс] – Режим доступу до ресурсу: https://necolas.github.io/react-native-web/.
14. AWS Lambda Pricing [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/lambda/pricing/.
15. Firebase pricing plans [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/pricing.
16. Frameworks for Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialsteacher.com/nodejs/open-source-frameworks-for-nodejs.
17. Best React Native Notification Libraries [Електронний ресурс] – Режим доступу до ресурсу: https://openbase.com/categories/js/best-react-native-notification-libraries.
18. React [Електронний ресурс] – Режим доступу до ресурсу: https://uk.reactjs.org/.
19. React Native for Windows + macOS [Електронний ресурс] – Режим доступу до ресурсу: https://microsoft.github.io/react-native-windows/.
20. A fast, open source web browser engine. [Електронний ресурс] – Режим доступу до ресурсу: https://webkit.org/.

21. Understanding the React Native bridge concept [Електронний ресурс] – Режим доступу до ресурсу: https://hackernoon.com/understanding-react-native-bridge-concept-e9526066ddb8.
22. TypeScript Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.typescriptlang.org/docs/.
23. Single source of truth [Електронний ресурс] / – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Single_source_of_truth
24. Immutability in JavaScript using Redux [Електронний ресурс] / – Режим доступу до ресурсу: https://www.toptal.com/javascript/immutability-in-javascript-using-redux
25. PainlessMesh Technical Documentation [Електронний ресурс]/ - Режим доступу до ресурсу:https://gitlab.com/painlessMesh/painlessMesh/-/wikis/home

ДОДАТКИ

Додаток А. Фрагменти коду програми

Компонент для відображення сповіщень на екрані сповіщень:

```
import React from 'react';
import { View, StyleSheet} from 'react-native'
import { Avatar, ListItem } from 'react-native-elements';
import Text from '../Text/Text'


const AlertComponent = ({
    alert_id = "",
    area_id = "",
    title = "",
    created_at = "",
    sensor_id = "",
    alert_type_id = "",
    ...otherProps
}) => {
    return (
        <ListItem
            bottomDivider
            {...otherProps}
        >
            <ListItem.Content>
```

```
            <Text>{`alert_id: ${alert_id}`}</Text>

            <Text>{`area_id: ${area_id}`}</Text>

            {created_at.length > 0 && <ListItem.Subtitle
style={styles.subtitle}>{`created_at: ${created_at}`}</ListItem.Subtitle>}

            {title.length > 0 && <ListItem.Subtitle style={styles.subtitle}>{`title:
${title}`}</ListItem.Subtitle>}

            {sensor_id.length > 0 && <ListItem.Subtitle
style={styles.subtitle}>{`sensor_id: ${sensor_id}`}</ListItem.Subtitle>}

            {alert_type_id.length > 0 && <ListItem.Subtitle
style={styles.subtitle}>{`alert_type_id: ${alert_type_id}`}</ListItem.Subtitle>}

          </ListItem.Content>

          <ListItem.Chevron size={35} color="black"/>

        </ListItem>

    );

};


export default AlertComponent;


const styles = StyleSheet.create({

    title: {fontSize: 25},

    subtitle: {fontSize: 22},

})
```

Приклад екрану редагування (створення) даних таблиці:

```
import React from 'react';
import { View, Image, Alert } from 'react-native'
import { Input } from 'react-native-elements'
import Button from '../../components/Button/Button';
import { Formik } from 'formik';
import ApiService from '../../services/ApiService';
import { useNavigation } from '@react-navigation/native';
import BackHeader from '../../components/BackHeader/BackHeaderCenter';
import { screens } from '..';
import shortid from 'shortid';



const SensorTypeEditScreen = ({
    route: {params}
}) => {


    const {replace} = useNavigation()
    const {
        sensor_type_id,
        title,
        desc,
        image
    } = params || {}
```

```
    return (
        <View style={{flex:1, justifyContent:'center'}}>
            <BackHeader title={sensor_type_id ? 'Sensor type editing': "Sensor type
creation"} />
            <Formik
                initialValues={{
                    sensor_type_id,
                    title,
                    desc,
                    image,
                }}
                onSubmit={async (values, { setSubmitting }) => {
                    // update
                    if(sensor_type_id){
                        const response = await ApiService.updateSensorType(sensor_type_id,
{
                            TITLE: title,
                            desc: values.desc,
                            image : values.image
                        })
                        if(response.ok){
                            replace(screens.SensorType)
```

```
                }else{
                    Alert.alert('Error',response.originalError.message)
                }
            }
            // create new
            else{
                const response = await ApiService.createSensorType({
                    sensor_type_id: values.sensor_type_id || shortid.generate(),
                    title: values.title,
                    desc: values.desc,
                    image : values.image
                })
                console.log('response',response)
                if(response.ok){
                    replace(screens.SensorType)
                }else{
                    Alert.alert(response.originalError.message,
response.config?.data.detail)
                }
            }
        }}
    >
    {({
```

```
    values,
    errors,
    touched,
    handleChange,
    handleBlur,
    handleSubmit,
    isSubmitting,
}) => (
<View style={{flex:1, padding: 25}}>
    <Input
        onPress={() => console.log('click')}
        placeholder="Id (optional)"
        value={sensor_type_id}
        />
    <Input
        placeholder="Title"
        value={values.title}
        onChangeText={handleChange('title')}
        />
    <Input
        placeholder="Desc"
        value={values.desc}
        onChangeText={handleChange('desc')}
```

```
            />
          <Input
            placeholder="Image (url,  optional)"
            value={values.image}
            onChangeText={handleChange('image')}
            />
          {!!values.image && (
            <Image
              source={{uri: values.image}}
              style={{width: 100, height: 100}}
              />
          )}
          <Button
            title={sensor_type_id ? 'Update' : "Create"}
            onPress={handleSubmit}
          />
          </View>
        )}
        </Formik>
      </View>
    );
};
```

```
export default SensorTypeEditScreen;
```

Хук (функція) для видобування даних таблиці:

```
import { useState, useEffect,  useCallback} from 'react'
import { AppState } from 'react-native';
import ApiService from '../services/ApiService'

function capitalizeFirstLetter(string) {
   return string.charAt(0).toUpperCase() + string.slice(1);
 }

export default function useFetchData(name:string, transformItems?:any, getSchema = false){

   const [loading, setLoading] = useState(false)
   const [data, setData] = useState<any[]>([])
   const [schema, setSchema] = useState<any[]>({})

   const _fetchSensorTypes = async () => {
      try{
         setLoading(true)
```

```
        const response = await ApiService[`get${capitalizeFirstLetter(name)}`]()
        // console.log('response',response.data)
        if(response.ok && response.data){
            let _data = response.data || []
            if(transformItems){
                _data = _data.map(transformItems)
            }
            setData(_data as any[])
        }
        if(getSchema === false){

        }else{
            const tableInfoResponse = await ApiService.getTableInfo(name)
            if(tableInfoResponse.ok){
                const _shema = tableInfoResponse.data.reduce((ac, curr) => {
                    ac[curr.column_name] = curr.data_type == 'text' ? "" : curr.data_type
=== 'ARRAY' ? [] : 0
                    return ac
                }, {})
                setSchema(_shema)
            }
        }
    }catch(err){
```

```
        console.log('ERRRO DURING FETCH SENSOR TYPES',err)
    }finally{
        setLoading(false)
    }
}

useEffect(() => {
    _fetchSensorTypes()
}, [])

const handleAppState = useCallback(
    (state) => {
        if(state == 'active'){
            _fetchSensorTypes()
        }
    },
    [setLoading, setData],
)

useEffect(() => {
    AppState.addEventListener('change', handleAppState)
    return () => {
        AppState.removeEventListener('change', handleAppState)
```

```
        }
    },[])

    return {
        loading,
        data,
        schema
    }
}
```

Компонент-навігатор для панелі адміна (вкладка “Admin”) :

```
import * as React from 'react';
import { createStackNavigator } from '@react-navigation/stack';
import { screens } from '../screens';
import SensorTypeScreen from '../screens/Sensors/SensorTypeScreen';
import TableList from '../screens/Admin/TableList';
import SensorTypeEditScreen from '../screens/Sensors/SensorTypeEditScreen';
import SensorEditScreen from '../screens/Admin/SensorEditScreen';
import AreaEditScreen from '../screens/Admin/AreaEditScreen';
import AlertTypeEditScreen from '../screens/Admin/AlertTypeEditScreen';
import AlertEditScreen from '../screens/Admin/AlertEditScreen';

const Stack = createStackNavigator();
```

```
export default function AdminNavigator() {
    return (
      <Stack.Navigator
        headerMode="none"
        initialRouteName={screens.TableList}
        screenOptions={{
            headerTitleStyle: {fontFamily: 'Jfwildwood'}
          }}
        >
        <Stack.Screen
            name={screens.TableList}
            component={TableList}
            />
        <Stack.Screen
            name={screens.AlertTypeEditScreen}
            component={AlertTypeEditScreen}
            />
        <Stack.Screen
            name={screens.AlertEditScreen}
            component={AlertEditScreen}
            />
        <Stack.Screen
```

```
        name={screens.SensorEditScreen}
        component={SensorEditScreen}
        />
      <Stack.Screen
        name={screens.SensorType}
        component={SensorTypeScreen}
        />
      <Stack.Screen
        name={screens.AreaEditScreen}
        component={AreaEditScreen}
        />
      <Stack.Screen
        options={{
          title: "Sensor Type Edit "
        }}
        name={screens.SensorTypeEditScreen}
        component={SensorTypeEditScreen}
        />
    </Stack.Navigator>
  );
}
```

Додаток Б. Презентація захисту роботи

Комплексне IoT рішення для детектування та моніторингу лісових пожеж

Виконав: студент 4 курсу, групи ПТІР-41
Сауляк Микола Русланович

Керівник: доктор технічних наук
Степанов Михайло Миколайович

@sauliak, 2021

Рисунок А 1 - Слайд 1

Актуальність теми

Лісові пожежі стають частим та небезпечним явищем. За даними Європейської комісії протягом 17 років на боротьбу з пожежами в Європі було витрачено понад 50 мільярдів доларів.

Збитки від масштабної пожежі на Луганщині у 2020, в ході якої вигоріло майже 8 тисяч гектарів лісу, можуть складати близько 4-5 мільярдів гривень.

Рисунок А 2 - Слайд 2

Метою бакалаврської роботи є:

- аналіз систем детектування лісових пожеж
- проектування та реалізація комплексного IoT рішення для детектування лісових пожеж

Завдання. Для досягнення мети необхідно виконати:

- комплексне дослідження систем захисту лісів від пожеж
- зібрання вимог до системи та складання випадків використання
- розробка мобільного додатку згідно обумовленої теми за допомогою фреймворку react-native.
- розробка веб-сервера на базі фреймворку express
- фізичне під'єднання датчиків до плат та їх з'єднання зі сервером
- реалізація механізму відправки push сповіщень

Рисунок А 3 - Слайд 3

Назва способу	Переваги	Недоліки
Супутникові знімки	- можливість використання наявних апаратних засобів	- великий час відгуку - складність постійного покриття через зміну позицій супутників - вплив погодних умов на якість знімків
Цифрові камери та оптичні прилади	- велика площа покриття (15-80 км)	- висока вартість - необхідність побудови спеціальних приміщень - високий рівень помилок через погодні умови, зміну дня і ночі
Бездротові датчики	- швидкість детектування аномалій - можливість встановлення у важкодоступних місцях	- проблема локалізації датчиків - необхідність живлення кожному датчику
Навчені спостерігачі	- більша ефективність у порівнянні з оптичними приладами	- людський фактор - складність покриття великих територій

Рисунок А 4 - Слайд 4

Вимоги до системи

- В нашому випадку зацікавленими сторонами є жителі, які живуть поблизу лісових насаджень та охоронні організації, які опікуються безпекою лісів. Як було сказано раніше однією з головних вимог є детектування пожежі в перші 5 хвилин та можливість отримання відповідного сповіщення.

Рисунок А 5 - Слайд 5

Потік даних в системі

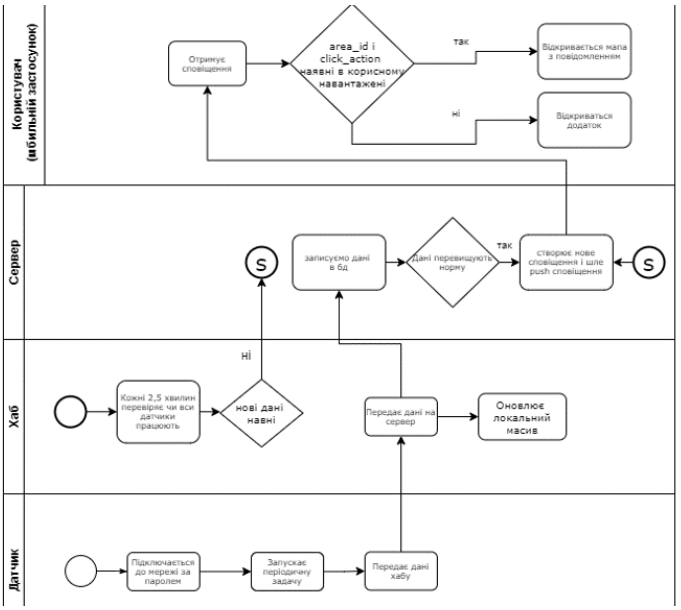


Рисунок А 6 - Слайд 6

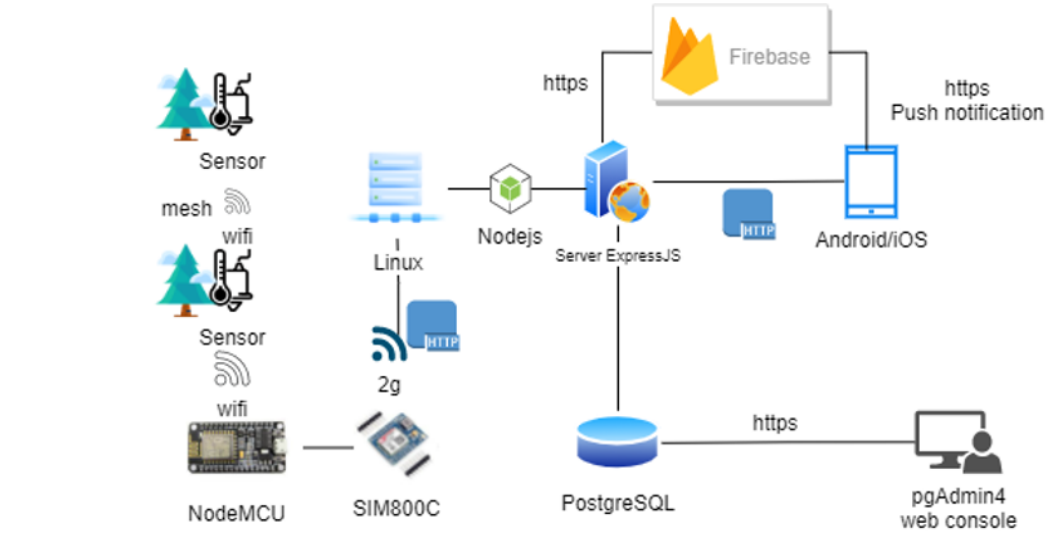


Рисунок А 7 - Слайд 7

Фізична модель бази даних PostgreSQL

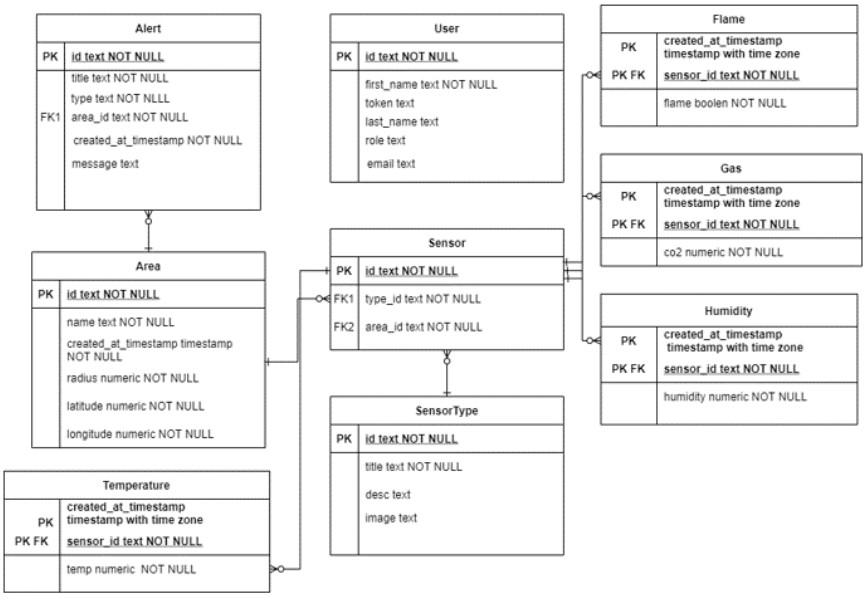


Рисунок А 8 - Слайд 8

Апаратні засоби

Рисунок А 9 - Слайд 9

Вхідні та вихідні дані

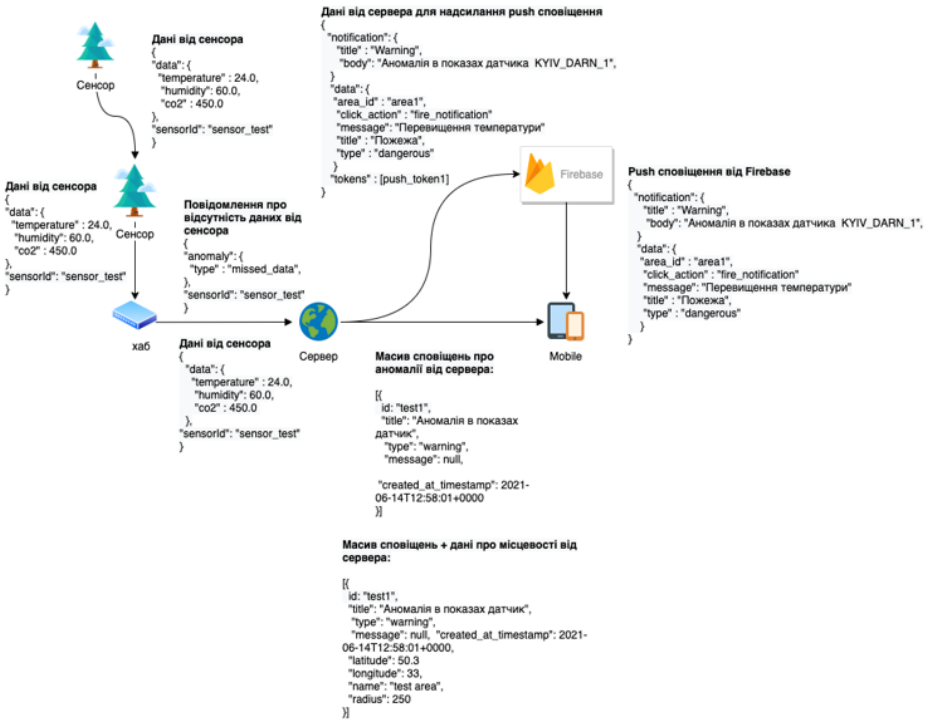


Рисунок А 10 - Слайд 10

Порядок дій мобільного застосунку

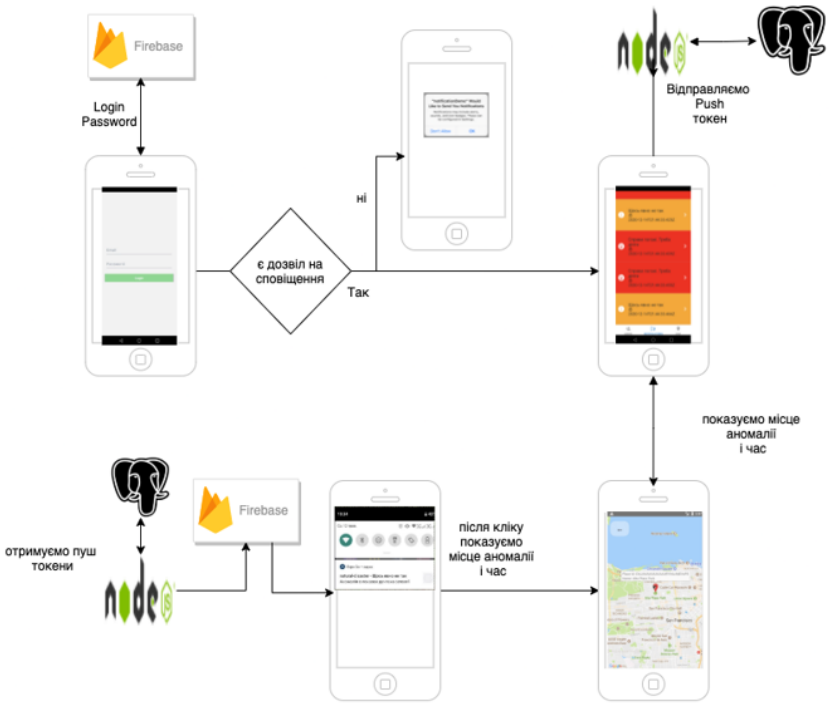


Рисунок А 11 - Слайд 11

Висновки

- Під час виконання даної бакалаврської роботи було спроектовано та реалізовано систему детектування та моніторингу лісових пожеж.
- Було зібрано базові вимоги до даної мережі. Проаналізовано подібні системи
- В якості бази даних використано postgresql, сервер створено та запущено за допомогою таких технологій: nodejs, express, nodemon
- Створено меш мережу за допомогою таких компонентів: плата nodemcu, датчик LM35DZ ,модуль зв'язку sim800c, датчик CCS811.
- Відправка сповіщень здійснювалася за допомогою пакету firebase-admin, з боку клієнта для отримання токену була використана бібліотека expo-notifications. Для отримання відповідних дозволів використана бібліотека expo-permissions
- Створено мобільний додаток для роботи з базою даних; створено сервер для роботи з базою даних; наведено порядок створення додатку: налаштування авторизації, виводу таблиць бази даних, операцій з таблицями;

Для своєчасного детектування лісових пожеж потрібно вчасно реагувати на аномальні явища, в чому допоможе зібрана системи Розроблений мобільний додаток направлений на мобільне управління базою даних системи захисту лісів та своєчасного отримання сповіщень про виявлену небезпеку, на основі отриманих даних з датчиків температури.

Рисунок А 12 - Слайд 12