Міністерство освіти і науки України

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра мережевих та інтернет технологій

ЗАТВЕРДЖУЮ
завідувач кафедри
мережевих та інтернет технологій
_____ Юрій КРАВЧЕНКО
«_____»_____2023 року

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

галузі знань 17 «Електроніка та телекомунікації»
за спеціальністю 172 «Телекомунікації та радіотехніка»
освітньо-професійна програма «Мережеві та інтернет технологій»

на тему:

ІНФОРМАЦІЙНА СИСТЕМА МОНІТОРИНГУ НАЯВНОСТІ ЕЛЕКТРОПОСТАЧАННЯ

Виконав: студент групи МІТ -41
Назар ЯРОЩУК _____
(ім'я та ПРІЗВИЩЕ) (підпис)

Керівник: доцент кафедри мережевих та інтернет технологій
(посада)
к.т.н., доцент Ольга ЛЕЩЕНКО _____
(науковий ступень, вчене звання, ім'я та ПРІЗВИЩЕ) (підпис)

Київ 2023

Міністерство освіти і науки України
«Київський Національний університет імені Тараса Шевченка»

Факультет інформаційних технологій

Кафедра мережевих та інтернет технологій

ЗАТВЕРДЖУЮ
завідувач кафедри
мережевих та інтернет технологій
_____ Юрій КРАВЧЕНКО
«_____»_____2023 року

ЗАВДАННЯ

НА ДИПЛОМНУ РОБОТУ

Здобувачу вищої освіти Ярощуку Назару Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи:
Інформаційна система моніторингу наявності електропостачання

затверджена на засіданні кафедри МІТ «07» грудня 2022 р. протокол №5
2. Термін здачі закінченої роботи «30» травня 2023р
3. Вихідні дані до проекту (роботи)
Мова програмування – TypeScript.
Технологія Raspberry Pi 3 model B, Linux, AWS, Node.js, React, MongoDB, Vercel
4. Зміст пояснювальної записки (перелік питань, що їх потрібно розробити, обсяг – 35-40 стор.)
Вступ

1. Аналіз предметної області розробки комерційного веб-додатку

1.1. Аналіз останніх досліджень
1.2. Актуальність роботи
1.3. Огляд аналогічних рішень
1.4. Постановка задачі та методи дослідження

2. Проектування інформаційної системи моніторингу електропостачання

2.1. Структурно-функціональне забезпечення системи
2.2. Стек технологій для розроблення інформаційної системи
2.3. Проектування Use-Case діаграму
2.4. Проектування бази даних
2.5. Методи оптимізації, масштабування та надійності

3. Реалізація інформаційної системи моніторингу електропостачання

3.1. Основні етапи розробки серверної частини системи
3.2. Основні етапи розробки користувацької частини системи
3.3 Дослідження роботи інформаційної системи. Результати тестування. Демонстрація роботи.

4. Дослідження роботи інформаційної системи

Висновки

5. Перелік графічного матеріалу 8-10 слайдів

Дата видачі завдання

Керівник роботи Ольга ЛЕЩЕНКО

(підпис) (посада, прізвище, ім'я, по батькові)

Завдання прийняв до виконання

Назар ЯРОЩУК

(підпис)

Дата видачі завдання

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ РОБОТИ

Номер	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Підготовчий	02.05.2023	
2	Розділ 1	10.05.2023	
3	Розділ 2	15.05.2023	
4	Розділ 3	20.05.2023	
5	Доповідь та слайди	25.05.2023	
6	Пояснювальна записка	30.05.2023	

Здобувач вищої освіти _____ Назар ЯРОЩУК

(підпис)

Керівник _____ Ольга ЛЕЩЕНКО

(підпис)

РЕФЕРАТ

Дипломна робота присвячена розробці інформаційної системи моніторингу електропостачання

Пояснювальна записка до дипломної роботи на тему «Інформаційна система моніторингу електропостачання». містить : 64 с., 31 рис., 8 додатків, 11 джерел.

Мета роботи: розробка «опенсорс» програмного забезпечення, яке фіксує наявність електроживлення в певному місці.

Об'єкт дослідження: моніторинг наявності електропостачання.

Предмет дослідження: інформаційна система в поєднанні з датчиками для відслідковування наявності електроенергії.

Методи дослідження: системний підхід, методи порівняння, структурний аналіз, моделювання, спостереження.

Наукова новизна: полягає у тому, що створено спеціалізоване програмне забезпечення, яке дозволяє перевірити наявність електропостачання в певному місці.

Актуальність: запропоновано розробити спеціалізоване програмне забезпечення яке використовується для перевірки наявності електроживлення.

Практичне значення роботи полягає у можливості перевірки та сповіщення користувача про відсутність електропостачання в певному місці.

Результати здійснених у дипломній роботі досліджень можуть бути використані на підприємствах, лікарнях, в побутовому використанні.

Ключові слова: ІНФОРМАЦІЙНА СИСТЕМА, ЕЛЕКТРОЖИВЛЕННЯ, ВЕБ-ДОДАТОК, МОНІТОРИНГ, ПРИСТРІЙ.

ЗМІСТ

ВСТУП

Електроенергетика в Україні завжди відігравала значну роль у розвитку країни та її економіки. Електроенергія необхідна для роботи підприємств, виробництва товарів і послуг, освіти, охорони здоров'я, транспорту та інфраструктури. Без електроенергії багато галузей господарства не змогли б функціонувати ефективно. Також розвиток електроенергетики стимулює інновації, впровадження нових технологій та розвиток сектора відновлювальної енергетики. Це сприяє залученню нових інвестицій, створенню робочих місць та підтримці розвитку малого та середнього бізнесу. Розвиток інноваційних рішень у сфері електроенергетики допомагає покращувати енергоефективність, знижувати витрати на енергію та сприяє сталому розвитку країни.

Тому галузь електроенергетики завжди була дуже важливою, а найбільш гостро кожен з нас відчув значимість цього аспекту у період масових відключень світла по усій країні. На даний момент країни втратила дуже великі енергетичні потужності, що збільшило проблематику даного питання.

Однією з проблем, що виникла у період «блекауту» була неможливість в деяких випадках встановлення де саме на даний момент відсутнє електроживлення. Через це багато приладів та підприємств не могли працювати нормально.

Тому темою дипломної роботи стала розробка інформаційної системи моніторингу наявності електропостачання. Вона являє собою «опенсорс» інформаційну систему, яка складається з пристрою, що приєднується до системи електропостачання, веб додатку на якому відображається інформація, що передається пристроєм та бота сповіщувача кінцевого користувача.

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ РОЗРОБКИ КОМЕРЦІЙНОГО ВЕБ-ДОДАТКУ

1.1 Аналіз останніх досліджень

Сьогодні існує дуже багато різноманітних способів того, як можна визначити чи присутнє електропостачання у певному місці. В умовах «блекауту» місцеві органи влади постійно інформували про планові відключення електропостачання, які легко можна було знайти на їх сайтах та офіційних телеграм каналах. Були створені спеціальні мобільні застосунки-сповіщувачі та сайти, які брали інформацію про планові відключення з офіційних джерел та нотифікували і відображали її. Проте дуже часто відбувались аварійні відключення, які ніяк не можна було передбачити і саме для відслідковування таких випадків можна використовувати деяку низку способів та технологій, які було досліджено та проаналізовано в ході даної роботи.

Звичайно можна було б розглянути такі варіанти, як використання домашніх гаджетів з вбудованим приєднанням до інтернету, як наприклад робот-пилосос або кондиціонер, також можна попросту запитати чи є світло у чаті свого ОСББ, або ж встановити у себе дома звуковий сигналізатор вимкнення мережі, який просто пролунає як сирена. Проте ці способи є просто тимчасовим виходом із ситуації, який до того ж підходить тільки для побутового використання.

Більш технологічним виходом стали телеграм-боти такі, як наприклад «Чи є світло?». Для їхньої роботи потрібно підключитись до WiFi мережі з телефону або ноутбуку у місці, яке потрібно відслідковувати та підв’язати до бота WiFi-роутер. Після цього програма починає відстежувати цю мережу та надсилати повідомлення, коли саме мережа зникає або з'являється. Це звичайно можна пов’язати з зміною стану електросистеми, проте можуть бути випадки, що попросту відімкнули інтернет або ж може бути таке, що світло вже з’явилось, а

мережа ні. Цей метод передбачає собою використання WiFi-роутеру, що попередньо повинен бути налаштований, тобто повинні бути відкриті порти та повинна бути встановлена статична ip-адреса, також він повинен відповідати на ping запити. Тому людина, яка у цьому не розбирається не зможе це все самотужки зробити. Також розроблено багато мобільних віджетів та застосунків, які повторюють даний функціонал з його недоліками. Їх приблизна робота та візуальний інтерфейс зображені на рис 1.1.

Рисунок 1.1 – Приклад одного із віджетів для смартфонів на системі IOS.

Ще одним способом відслідковування наявності електроенергії є встановлення розумних розеток. Суть їх роботи полягає у тому, що подача або припинення електроживлення до приладу, що підключений до розетки, відбувається за допомогою телефона або планшету. Для кожної одиниці існує спеціальний мобільний додаток у якому можна переглядати будь яку інформацію та статистику, також при відсутності електроживлення він відповідає за нотифікацію користувача. Зараз на ринку існує велике різноманіття даних приладів різних виробників та цінових категорій. Також існують спеціальні електронні лічильники електроенергії, які мають схожий функціонал та інтерфейс.

Роботу даного способу було перевірено на прикладі розумної розетки «Sanoff» у поєднані з мобільним застосунком «eWeLink». На рис 1.2 зображена панель керування пристроєм, на якій є можливість подачі живлення до водяного насосу, є можливість перегляду статистики та налаштування нотифікацій про стан мережі.

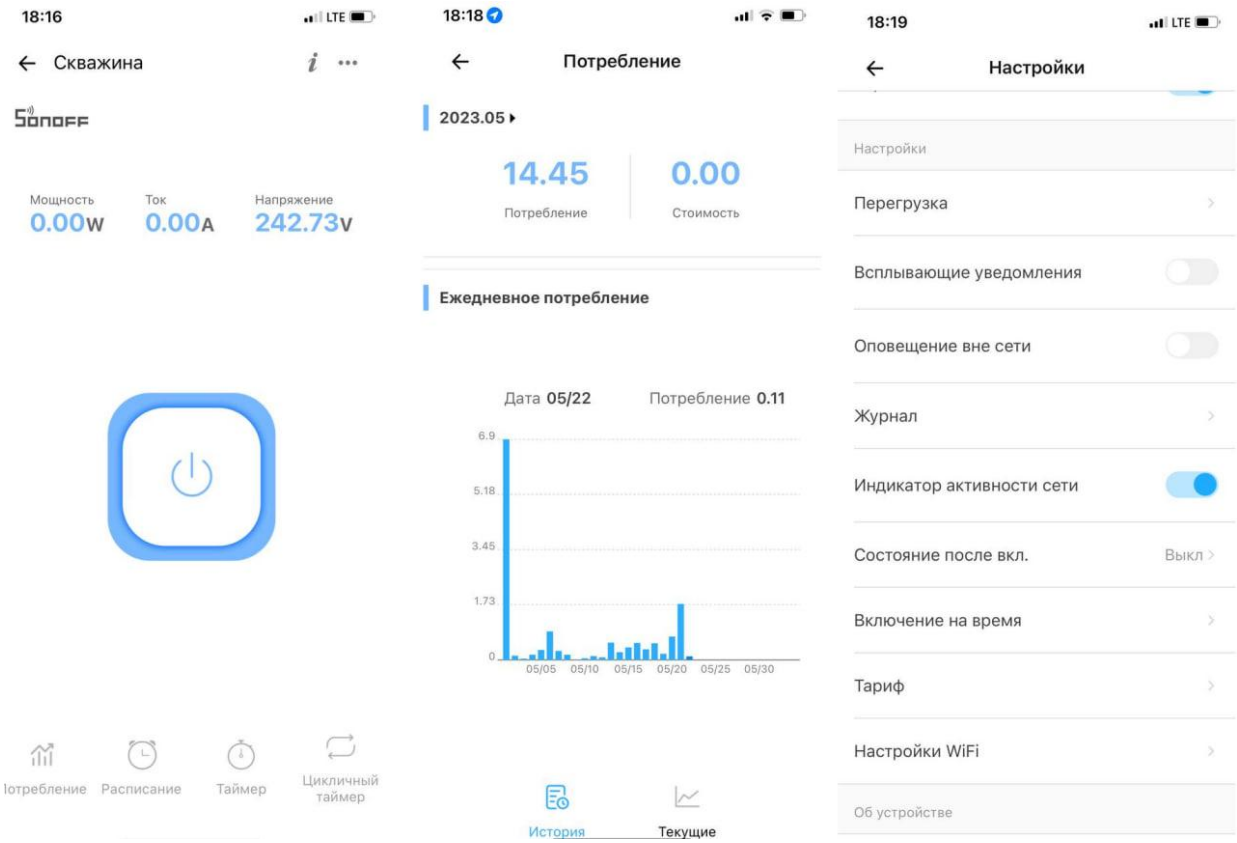

Рисунок 1.2 – Графічний інтерфейс розетки «Sanoff»

Проте постає така ж проблема, як і в ботах та додатках, якщо зникне мережа ми не зможемо здійснювати керування розеткою та приймати від неї нотифікації. Існують звичайно розетки з GSM модулем, що вирішує дану проблему, проте цей прилад є закритим до змін, що не дозволяє додати до нього якийсь інший функціонал. Також для багатьох моделей потрібен пристрій, що буде підключений до самої розетки, або ж якесь спеціальне встановлення та налаштування.

Ще одним способом фіксації є встановлення камер відеоспостереження, що мають подібний функціонал по нотифікації, як у розумних розеток, проте

використовувати цей метод для простого сповіщення користувача про стан мережі є дорогим та недоцільним.

Отже дослідивши та проаналізувавши дані методи фіксації наявності електропостачання було вирішено, що використовувати їх для даного завдання не є практично. На побутовому рівні, вони мають право на життя, проте в великих промислових масштабах або підприємствах їх використання є недоцільним, через їх недоліки та обмеженість функціоналу.

1.2 Актуальність роботи

Реалії нашого сьогодення дозволили зрозуміти, що здавалось би, буденні речі такі, як наприклад, присутність або відсутність електропостачання в, будь який, момент можуть кардинально змінити життя. Ще до недавнього часу такій термін, як «Блекаут» повсякденно використовувася тільки у людей, які спеціалізуються на енергетиці. Під час другої зими повномасштабної війни з Росією, кожен у нашій країні відчув на собі, що означає це слово.

Раптові відключення світла негативно впливали не тільки на загальний морально-психологічний стан та побутове життя, а й звичайно на будь які електроприлади. Якщо ж згорівша праска або холодильник, який протік не являють по своїй суті дуже глобальну проблему, то у масштабах великого бізнесу, підприємств, лікарень та інших установ належна та безперебійна робота електроніки є надважливою. Прикладів можна навести безліч від холодильних камер, які використовуються у харчовій промисловості, різноманітних охоронних систем та медичних пристроїв до різного роду серверів, що зберігають, оброблюють та відправляють інформацію.

Вирішенням даної проблеми є підключення техніки до генераторів електричного струму, що дозволяє їм працювати незалежно від головної системи електропостачання, проте потрібно швидко та вчасно проінформувати людей, які за це відповідальні про збій у роботі пристроїв.

На момент написання даної роботи, за допомогою спільних зусиль українського суспільства дана проблема не є настільки гострою в масштабах усієї країни, проте наразі вона залишається все ще актуальною, наприклад для прифронтових міст, бізнесу та простого повсякденного життя. Також не можна точно бути впевненими чи така ситуація не відбудеться знову.

1.3 Огляд існуючих рішень

1.3.1 AJAX

На сьогодні найдоцільнішим способом відслідковування наявності електроенергії є системи розумного будинку та системи охорони. Вони мають свої власні мобільні та веб додатки, у яких відбувається управління модулями, які підключені до центрального пристрою керування, наявна статистика по усім додатковим приладам, присутня можливість відправляти та приймати від них запити та нотифікації. У ході даної робити було практично досліджено та проаналізовано такий тип систем на основі української охоронної системи «AJAX».

«AJAX» - це комплекс охоронних приладів для захисту території та будівель таких, як житлові будинки, офіси, квартири та підприємства. Основним призначенням цієї системи є попередження та запобігання, подій які можуть нашкодити людям чи майну. Найчастіше це захист від проникнень, пожеж та потопів. Набір пристроїв системи може бути різним і залежить від технічного завдання. Комплектація буде відрізнятися залежно від типу будівлі території чи підприємства. У неї можуть входити датчики руху, відкривання та закривання вікон або дверей, пожежні датчики та датчики протікання. У нашому випадку головну роль відіграє сама центральна система керування.

Основою є пристрій під назвою «Hub». він слугує базовою станцією для усіх приладів системи, хаб миттєво сповіщає користувачів про тривоги на об'єкті,

що охороняється. Хаби Ajax оснащені вбудованим резервним акумулятором, що забезпечує роботу без живлення від електромережі. Також у них присутній GSM модуль, який дозволяє підтримувати роботу без підключення до вайфаю чи кабельного інтернету. Він відправляє та приймає запити від користувачів та керує усією інформацію, що до нього надходить. На рис 1.3 зображено приклад сповіщення користувача хабом про відсутність електропостачання та мережі у приватному будинку, що і є основою даної роботи, також зображений приклад стану самого приладу.

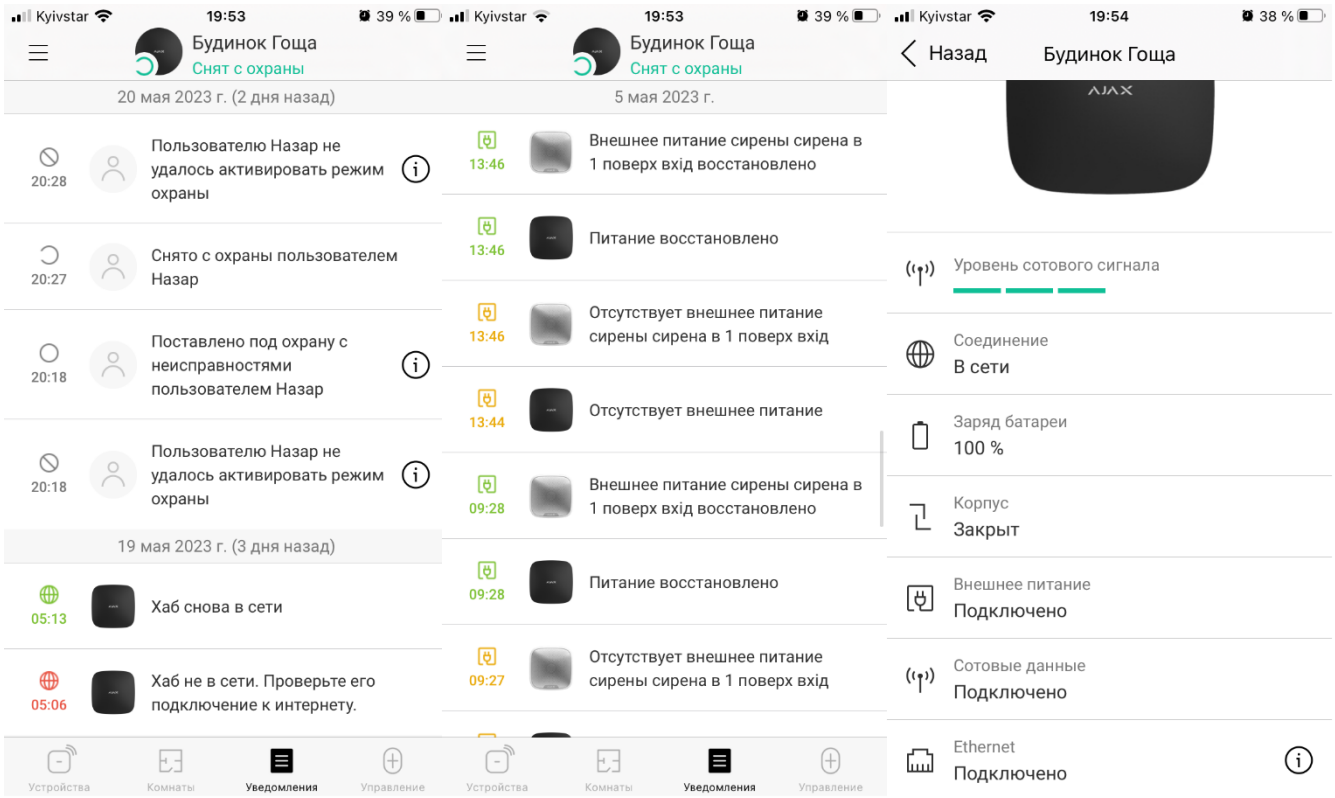

Рисунок 1.3 – Приклад нотифікацій, які надходять від хаба «AJAX»

Отже, перевагами даного пристрою є те, що він може працювати без електроживлення, наявний зручний користувацькій інтерфейс, простота використання та встановлення, для нього існує своя власна інфраструктура в, якій використовується багато новітніх технологій, присутня можливість налаштування різного роду нотифікацій та його модульність, тобто підключення до нього різних типів датчиків та приладів. Проте у нього наявні певні недоліки такі, як в першу чергу висока ціна, якщо нам потрібен пристрій, який буде просто відправляти та

аналізувати інформацію, що до нього надходить використання «Ajax» є недоцільним тому, що в першу чергу це охоронна система, яка базується на різному типу датчиків та приладів, що створюють собою певний комплекс приладів. Також система «Ajax» є закритою до змін, тобто до неї можливо підключити тільки пристрої або датчики, які випускаються цією компанією. Тому для даної роботи ця система не підходить, вона може являти собою лише альтернативу, якщо потрібно встановити охоронну систему, тоді її можна використовувати в якості датчика електроживлення, але це виступає «побічної» можливістю, а не основним призначанням даної системи.

1.3.2 smart-MAIC

Ще одним із способів відслідковування наявності електроживлення, є системи розумних лічильників та датчиків. Такі апаратно-програмні засоби використовуються для моніторингу та візуального аналізу різних фізичних процесів у режимі реального часу. Однією із таких систем є продукт української компанії smart-MAIC.

Дана розробка складається із різного типу лічильників та датчиків в поєднанні з веб-застосунком, що повністю візуалізує усю інформацію, що надходить. Функціонал даної системи дозволяє контролювати споживання електроенергії, води, газу, тепла, аналізувати температуру, вологість, тиск, CO2, TDS, pH та багато інших подій. Для аналізу та візуалізації даних використовується хмарний web-додаток smart-MAIC Dashboard доступ до якого можна отримати з телефону, планшету або настільного комп'ютера. Користувачеві доступне гнучке налаштування віджетів, індикаторів та графіків, велика кількість інформаційних панелей та підключених пристроїв до одного облікового запису. Перевагою також є те, що усі зібрані дані зберігаються у хмарі, при роботі в режимі реального часу показання з лічильників оновлюються з інтервалом 5 секунд. Існує можливість інтеграції веб-додатку з пристроями інших систем, для

чого використовується спеціальне API. Візуальна складова системи знаходиться на рис. 1.3.2.

В порівнянні з системою Ajax даний проект більш підходить для досягнення цілей, які були поставлені в ході даної роботи, проте у ній наявні такі ж недоліки, як і в охоронних системах, наприклад висока ціна та перевантаженість різною іншою інформацією, яка може бути і непотрібна користувачеві. Тому, було б не доцільно використовувати дану систему, лише для передачі поточного стану електромережі. Вона краще підходить для підприємств, де існує велика кількість різноманітних електричних пристроїв, за станом яких повинен бути постійний догляд. Тому було вирішено розроблювати подібну систему, проте з меншим функціоналом, що здешевлює її та підлаштовує лише до виконання невеликої кількості завдань.

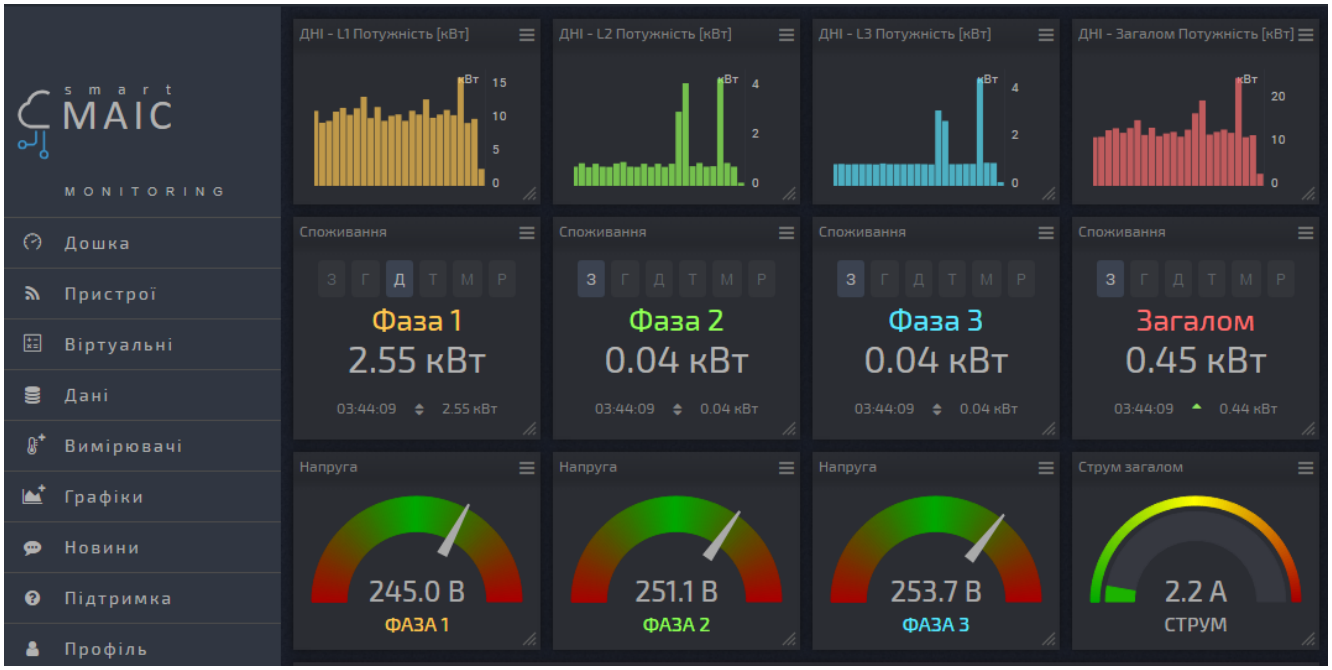

Рисунок 1.4 – візуальний інтерфейс smart-MAIC Dashboard

1.4 Постановка задачі та методи дослідження

Проаналізувавши усі підходи до даної проблеми було встановлено, що при розробці комерційного пристрою для відслідковування стану електромережі потрібно враховувати деякі чинники.

Перш за все - це простота у використанні. Кінцевий користувач не повинен довго розбиратися з налаштування пристрою для його роботи, тому було вирішено спростити етап активації пристрою нанесенням унікального QR-коду на корпус кожного приладу в наслідок сканування якого, з'явиться можливість, з під'єднанням пристрою до живлення та мережі, легко та швидко авторизуватися у веб-додатку та почати його використання. Також важливою є підтримка роботи декількох пристроїв одночасно на одному аккаунті та можливість підключити один пристрій до багатьох створених аккаунтів у веб-застосунку.

Другим чинником є зручний та зрозумілий веб-інтерфейс, що не повинен бути перевантажений лишніми функціями та налаштуваннями, проте відповідати та відображати увесь доступний функціонал системи. Також повинна бути наявна миттєва система нотифікацій, яка буде оформлена у вигляді зручного телеграм-боту. Третім чинником є відкритий код та модульність приладу, щоб користувач міг за бажанням доєднати до пристрою, будь які, свої датчики або модулі, інформацію з, яких пристрій буде оброблювати та передавати інформацію на головний сервер систем. У базовій комплектації пристрій буде налаштований лише на передачу стану електромережі та підключення його до інтернету. Пристрій повинен бути автономним від центрального джерела живлення, що означає наявність акумулятора та мати можливість підключення, як до провідного інтернету, так і наявність GSM-модуля.

У ході даної роботи пристрій в базовій комплектації прилад вирішено було налаштовувати на роботу лише з провідним інтернетом та без автономного джерела живлення, проте так, як це «опенсорс» проект, у кінцевого користувача буде змога додати це власноруч.

Мета дипломної роботи – розробити програмне забезпечення для моніторингу електропостачання.

Для досягнення визначеної мети, ставляться такі завдання:

– дослідити принципи роботи аналогічних рішень

– проаналізувати загальну архітектуру відповідних систем

– визначити основні вимоги до розроблюваної системи

– обрати інструмент для реалізації веб-додатка, БД, серверної частини, пристрою

– розробити програмні засоби для моніторингу енергоживлення

– протестувати розроблене програмне забезпечення.

РОЗДІЛ 2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ МОНІТОРИНГУ ЕЛЕКТРОПОСТАЧАННЯ

2.1 Структурно-функціональне забезпечення системи

Визначившись з основними вимогами та задачами, які повинна виконувати система було вирішено розділити її на декілька складових частин, які повинні бути зв’язані між собою та ефективно взаємодіяти. Такий підхід до розробки набагато її полегшує, а головне так, як це «опенсорс» програмне забезпечення це дозволить легко та зручно кінцевому користувачеві вносити у неї зміни та певним чином модифікувати. Отже, кожен модуль системи повинен виконувати свою роль та поставлені перед ним завданням та бути чітко сформульованим, як в теорії так і на практиці.

Головне це звичайно сам пристрій, було вирішено конструювати його на базі Raspberry Pi 3 model B, що буде поміщений у спеціальний пластиковий корпус з можливістю його відкрити, щоб можна було його налаштувати та вносити якісь зміни, наприклад підключити додатковий модуль

Основою та рушієм усієї системи є головний сервер. Він оброблює, приймає та передає усю інформацію, що до нього надходить від усіх інших модулів, а саме серверу приладу, веб застосунку та бази даних. Також однією із основних задач, що повинен виконувати головний сервер є нотифікація користувачів, яка була оформлена, як його складова частина. Складова частина головного серверу, що відповідає за сповіщення використовує інформацію про користувача, його прилади та головне телеграм «айді» на, який і відбувається нотифікація.

Сервер пристрою в свою ж чергу використовується для постійної передачі даних з самого пристрою, це може бути, як сам стан приладу, який було вирішено позначати, як «онлайн», «оффлайн» та «зміна статусу», так і при бажанні він передає інформацію з можливих додаткових датчиків. «Зміна статусу» це стан пристрою, що означає перехід приладу до іншого стану, також цей спеціальний статус був зроблений для оптимізації передачі даних.

База даних також відіграє важливу роль, у ній зберігається повністю вся інформація, яка використовується системою, це може бути, як логін і пароль користувача, так і «айді» приладу або будь-що інше.

Найважливішою частиною системи для кінцевого користувача є саме веб застосунок, що включає у себе різноманітні можливості, як наприклад реєстрація акаунту, зв’язування з ним нових пристроїв, перегляд статистики та різноманітних графіків та інше.

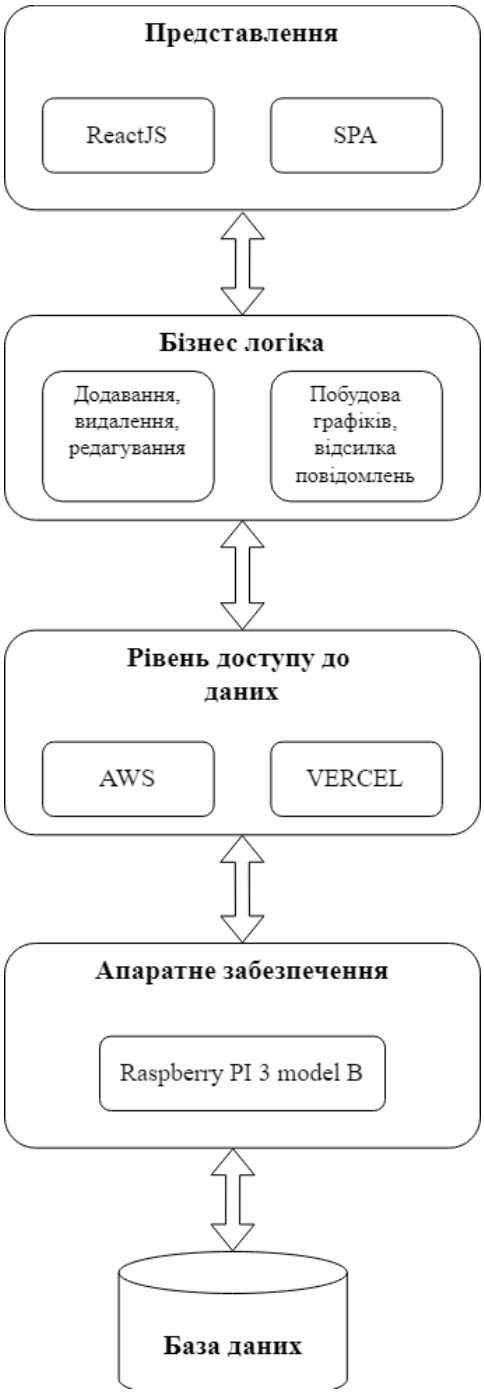

Рисунок 2.1 – Діаграма рівнів інформаційної системи

2.2. Стек технологій для розроблення інформаційної системи

UI (інтерфейс користувача):

Інтерфейс користувача розроблено за допомогою фреймворку для JavaScript що має назву React в поєднанні з використання багатьох допоміжних бібліотек, що розроблені спеціально для цього фреймворку. Ця технологія чудово підходить для розробки веб-сайтів, а так як задачею роботи було розробити саме веб-додаток, вирішено було використовувати саме її. Також в ході розробки інтерфейсу користувача використовувались мова розмітки HTML та формальна мови CSS.

React[1,4,6] - це відкрита JavaScript бібліотека, яка дозволяє розробникам будувати користувацький інтерфейс для веб-додатків. Основною особливістю даного фреймворку є віртуальний DOM він використовується для швидкого оновлення тільки тих частин сторінки, які змінилися, замість повного оновлення всього веб застосунку. Це дозволяє покращити продуктивність додатків та забезпечити ефективну взаємодію системи з користувачем. React використовує JSX (розширений синтаксис JavaScript), який дозволяє використовувати HTML-подібний синтаксис прямо в JavaScript коді. Це полегшує розробку компонентів та забезпечує зрозумілість структури інтерфейсу. Також перевагою React є те, що для нього розроблено багато додаткових бібліотек та інструментів, які дозволяють розширити його функціональність.

Для оформлення зовнішнього вигляду сторінок було використано CSS та допоміжні бібліотеки для нього. CSS - це формальна мова, яка слугую для опису та оформлення зовнішнього вигляду документа, створеного з використанням мови розмітки. Назва походить від англійського Cascading Style Sheets, що означає «каскадні таблиці стилів».

HTML - це мова гіпертекстової розмітки, за допомогою якого програмісти безпосередньо створюють структуру веб-сторінок і email-листів. Якщо пояснити суть даного поняття простими і наочними образами, то можна сказати, що HTML -

це тіло, а CSS (Cascading Style Sheets - каскадні таблиці стилів) - одяг. У той час як CSS визначає зовнішній вигляд web-сторінки, HTML формує її структуру (скелет) за допомогою заголовків, списків та інших подібних елементів, починаючи від початку сторінки - хедера, і до її кінця - футера.

Інтерфейс користувача вирішено було розміщувати на хмарному сервісі Vercel.

DATABASE (База даних)

Для зберігання усієї інформації, що приймає, віддає та оброблює система вирішено було використовувати нереляційну базу даних MongoDB.

MongoDB[5] - це документ-орієнтована NoSQL база даних, яка зберігає дані у вигляді документів у форматі JSON (JavaScript Object Notation). Перевагою цієї бази даних є те, що вона не потребує жорсткої схеми для зберігання даних. Уся інформація, що нею оброблюється може мати різну структуру, що дозволяє ефективно та швидко змінювати схему даних без необхідності оновлення всіх записів. Ще однією перевагою даної технології є її висока продуктивність. Вона забезпечує швидкий доступ до даних за рахунок використання внутрішнього кешування та ефективного алгоритму індексації, також існує підтримка паралельного виконання операцій, що дозволяє одночасно обробляти багато запитів.

BACKEND (серверна частина)

Для розробки серверної частини вирішено було використовувати TypeScript та Node.js з допоміжними для них бібліотеками.

Node.js[7] - це програмна платформа, яка робить мову JavaScript мовою загального призначення, її також називають середовищем виконання JS. Вона вміє зв'язуватися з зовнішніми бібліотеками, викликати команди з коду і виконувати

роль веб-сервера. Якщо пояснювати простіше, цей інструмент додає до повністю «фронтендової» мови «бекендову» частину, дозволяючи створювати з її допомогою не тільки веб-сайти, а й повноцінні програми, без задіяння браузера.

В свою ж чергу TypeScript[11] - це мова програмування, що розширює JavaScript, додаючи до нього статичну типізацію та нові функції. Перевагами даної мови є строга типізація даних, можливість наслідування класів та інтерфейсів, масштабованість та багато чого іншого.

Для відправки повідомлень користувачеві використовувалось Telegram API[12] та було створено спеціальний телеграм бот для цього завдання. Telegram API являє собою набір інструментів, які надають розробнику доступ до функціоналу та можливостей платформи Telegram. Він дозволяє розробникам створювати ботів, додатки та сервіси, які можуть взаємодіяти з користувачами через програмний інтерфейс месенджера.

Усю серверну частину вирішено було розміщувати на хмарному сервісі AWS.

DEVICE(пристрій)

Вирішено було використовувати Raspberry Pi 3 model B[10] тому, що він являє собою повноцінний комп’ютер з високою, для його невеликих розмірів, продуктивністю. В ньому наявна можливість підтримки багатьох операційних систем, присутність Wi-Fi та Bluetooth модулів, присутність різноманітних портів для підключення різних інших пристроїв. Також важливим фактором відіграє його ціна, для такого функціоналу вона є прийнятною та невисокою.

2.3 Проектування діаграми Use-case

Одним із важливих етапів розробки будь якої системи або додатку є проектування Use-case діаграми. Цей засіб дає великі можливості для розуміння та візуалізації зовнішнього та внутрішнього функціоналу програмного

забезпечення та, як користувач повинен з ним взаємодіяти. Проектування таким чином надає багато переваг таких, як розуміння вимог користувача, виявлення функціональних та нефункціональних вимог, що допомагає вирізнити, які частини системи будуть відповідати за функції, а, які за характеристики такі, як надійність, безпека, тощо, виявлення взаємодій між компонентами системи, представлення можливих сценаріїв роботи користувача з застосунком.

Одним з основних завдань даної діаграми є визначення границь системи та чітке розподілення програмного застосунок на модулі, що дозволяє нам встановити області відповідальності, тобто, для кого одні чи інші елементи системи будуть доступні та, які функції будуть при цьому використовуватись.

Визначившись з основними складовими частинами проекту вирішено було сконструйовати діаграму прецедентів для системи моніторингу електропостачання.

Усі користувачі системи є на одному рівні доступу до додатку тому, що це опенсорс проект з відкритим вихідним кодом. У них наявні можливості реєстрації та входу у систему, додавання та видалення пристроїв зі свого аккаунту, підключення пристроїв з іншого аккаунт, отримувати повідомлення про стан пристрою, перегляд стану, статистики та графіків кожного окремого пристрою, прив’язка до акаунту свого власного телеграму.

З’єднання у діаграмі поділяються на дві категорії include та extend. Include означає те, що попередня дія обов’язково повинна включатися у наступну. Extend означає те, що попередня дія не обов’язково повинна включатися у наступну, але може за необхідності розширювати її функціонал.

Робота системи розпочинається з реєстрації користувача у веб застосунку та прив’язки до нього свого власного телеграм аккаунту, після чого уся інформація про нього записується головним сервером у базу даних. Потім користувачеві потрібно зв’язати, прилади які він хоче додати, до свого аккаунту. В результаті цих дій з’являється можливість безпосередньо розпочати роботу з системою. Перш за все сервер пристрою постійно передає головному серверу запити про його

теперішній стан, в свою чергу основний сервер записує цю інформацію до логів та бази даних, після чого відбувається обробка усіх доступних даних, що включають в себе запити з приладів, що підв’язані до певного користувача. Потім у випадку зміни стану пристрою частина головного серверу, що відповідає за нотифікацію користувачів миттєво передає інформацію до телеграм-боту користувача, та веб застосунку. У можливості додатку входить перегляд поточного стану усіх доступних користувачеві приладів, побудова різноманітних графіків на основі зібраної інформації, що дозволяє переглядати статистику, також при необхідності можна додати нові пристрої до вже існуючого аккаунту. Весь цей функціонал зображений на діаграмі прецедентів на рис. 2.2.

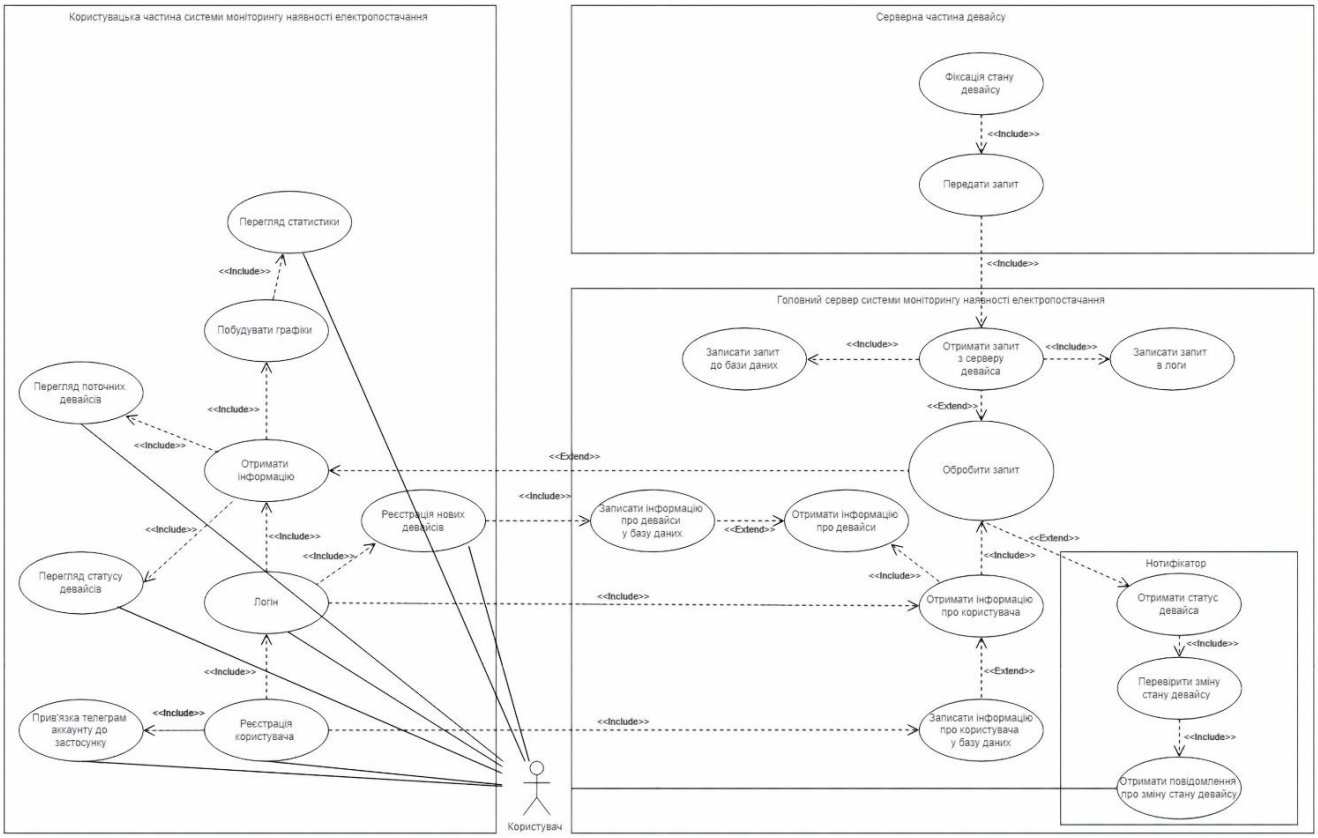

Рисунок 2.2 – Діаграма прецедентів для системи моніторингу електропостачання

2.4 Проектування бази даних

У наш час майже жоден програмний застосунок не може обійтися без бази даних, тому у реалізації системи вирішено було використовувати нереляційну базу даних MongoDb[5]. Перевагами даної бази даних є те, що непотрібно створювати

таблиці, MongoDB використовує колекції та гнучку модель документів, яка дозволяє зберігати інформацію в документах формату JSON-подібних об'єктів. Це дозволяє зберігати дані різної структури без потреби використовувати фіксовані таблиці зі стандартними реляційними базами даних. Увесь функціонал та дані знаходяться у хмарі, що є набагато зручніше, непотрібно окремо щось хостити або налаштовувати.

Для системи було створено три колекції для зберігання даних про користувачів, приладів та інформації, що відправляє пристрій.

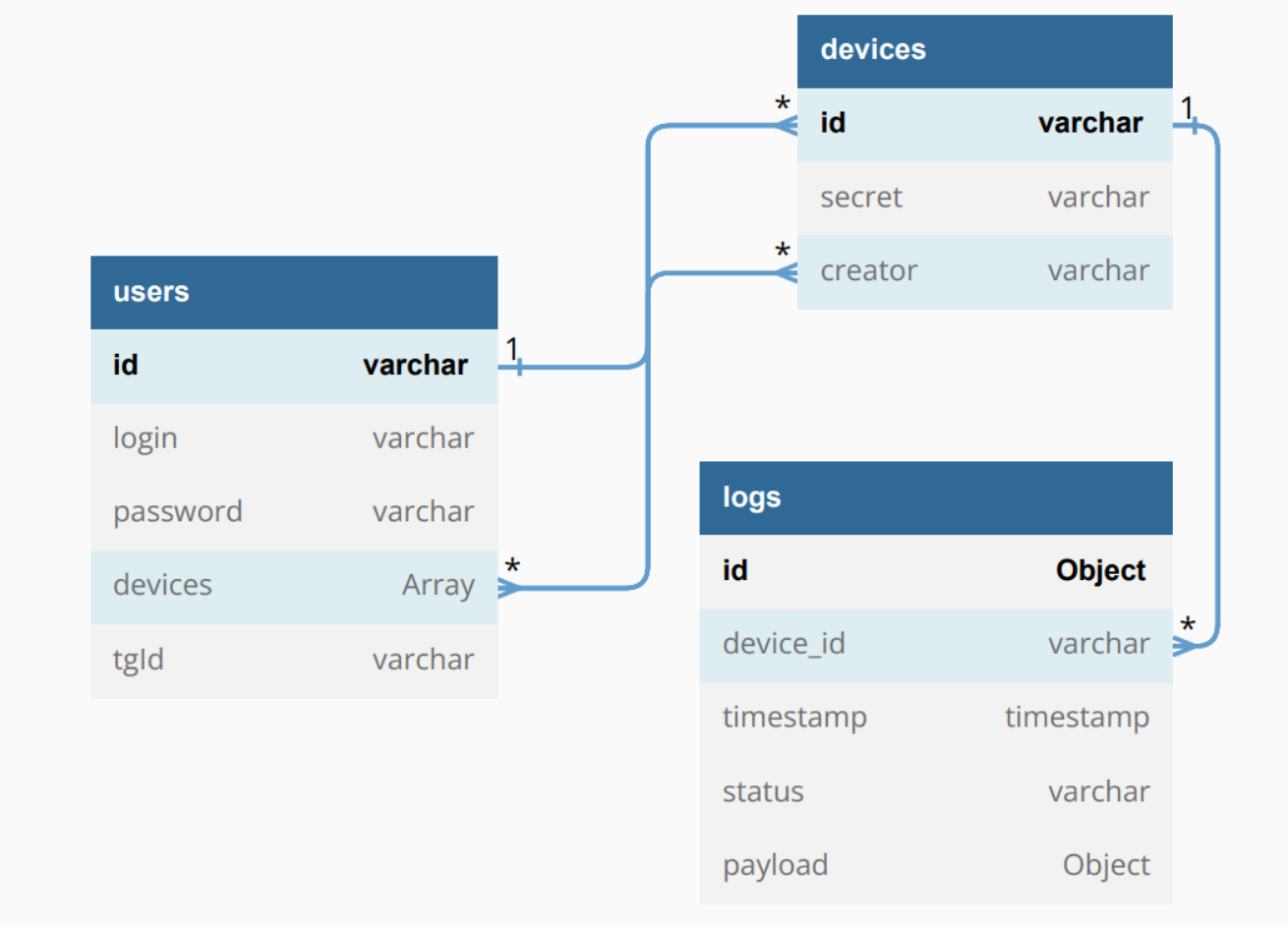

Рисунок 2.3 – Діаграма класів бази даних

У колекції «Users» зберігається уся наявна інформація, що використовується користувачем, а саме його id, логін, пароль, телеграм-id та масив рядків, що відповідає за наявні у користувача пристрої.

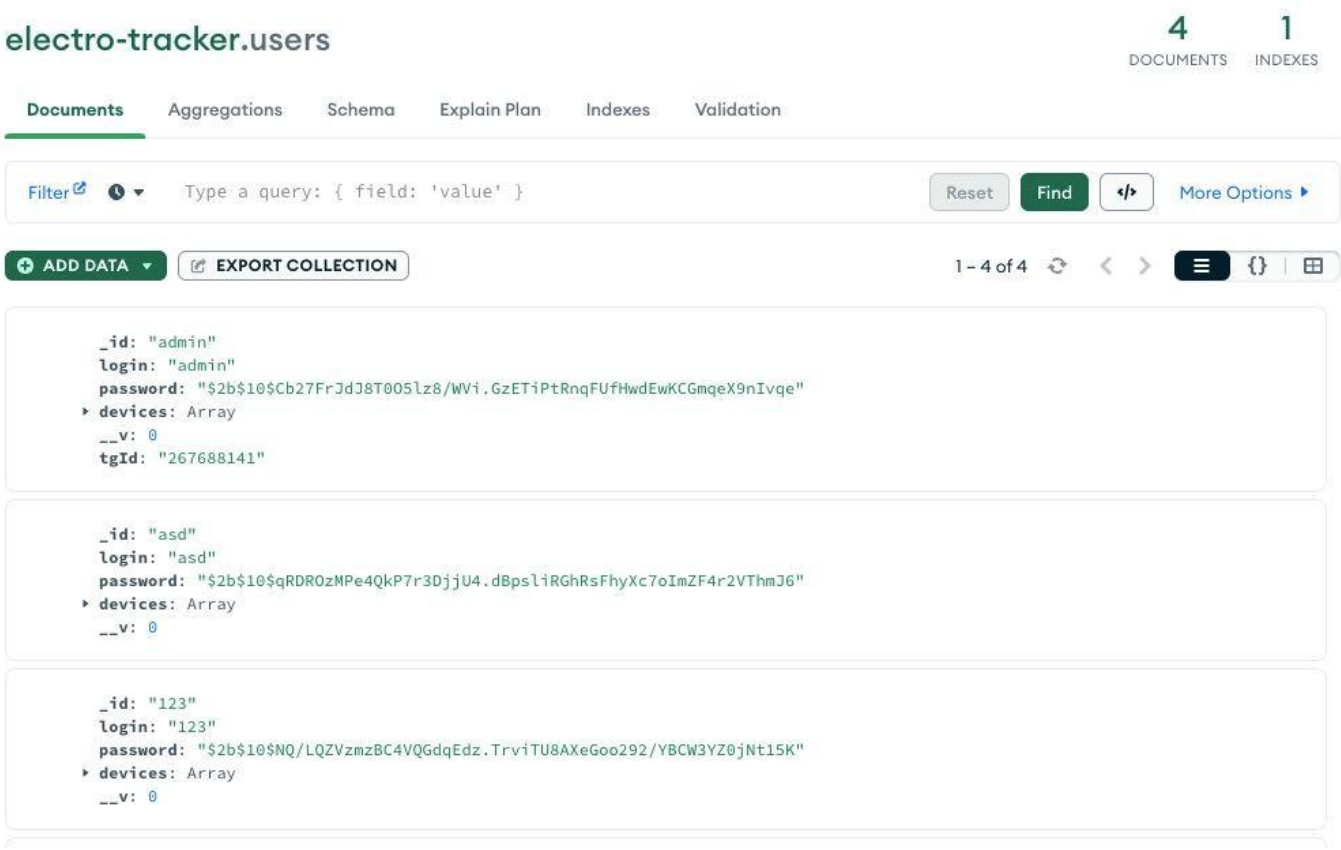

Рисунок 2.4 – Таблиця «users»

У колекції «devices» зберігається інформація про пристрій, що включає в себе його id та секретний ключ, що відповідає за ідентифікацію приладу,а також поле в якому зберігається кому саме належить цей пристрій.

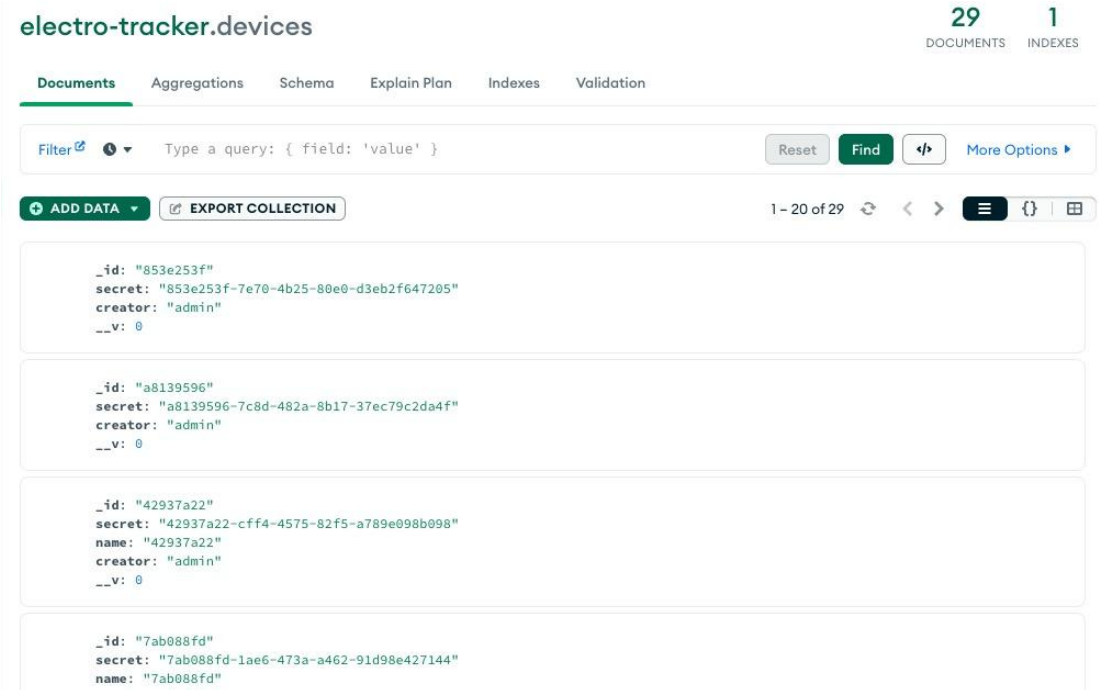

Рисунок 2.5 – Таблиця «devices»

У колекції «logs» знаходиться інформація, яку передає сервер приладу, вона включає в себе id запиту та пристрою, час коли надійшов запит, статус пристрою та об’єкт довільного змісту, що може включати в себе будь яку додаткову інформацію, що надсилає пристрій.

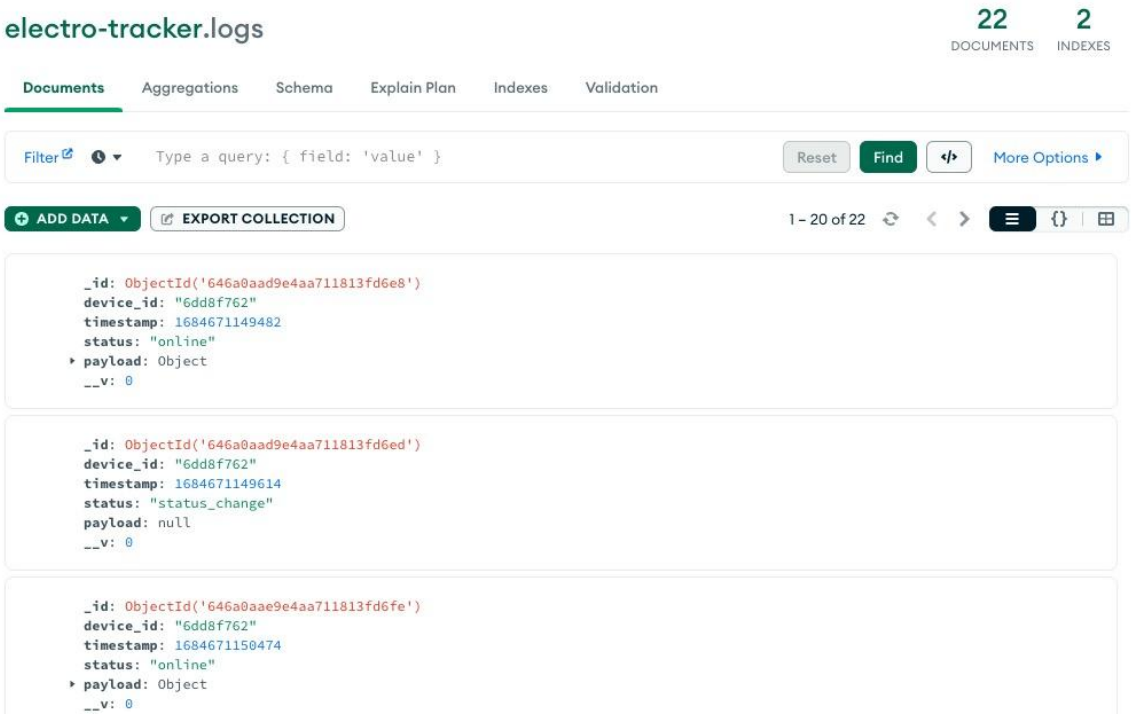

Рисунок 2.6 – таблиця «logs»

2.5 Методи масштабування оптимізації та надійності

Сервера усієї системи вирішено було хостити на Amazon Web Services. Сьогодні AWS[8] це один із провідних хмарних сервісів у якому наявна велика кількість можливостей та послуг для розробки, впровадження та керування веб-додатків та будь якого іншого програмного забезпечення. Перевагами даного сервісу є масштабованість, надійність, безпека, гнучкість, іноваційність та простота у використанні. Для масштабування та оптимізації розроблюваної системи можна використовувати засоби, які надає нам AWS. Присутня можливість збільшити обсяг оперативної пам’яті або збільшити обчислювальні частоти процесорів, при умові збільшення кількості користувачів або ж даних, які сервери

повинні будуть оброблювати. Також перевагою цього сервісу є те, що він повністю знаходиться у хмарі, що дозволяє легко та зручно зробити потрібні зміни з будь-якого місця та пристрою.

Одним із засобів масштабування та оптимізації системи також є використання MongoDb Cloud[5], ця технологія дозволяє збільшити або зменшити розмір кластеру бази даних, відповідно до нових вимог користувача, також існує можливість керування усією базою даних в автоматичному режимі, що набагато полегшує її використання та обслуговування. MongoDb Cloud також є хмарним сервісом і має подібний функціонал у налаштуванні, оптимізації та масштабування, як і сервіси AWS.

Усю «фронтенд» частину проекту вирішено було розміщувати на платформі Vercel[9], яка також являє собою набір хмарних технологій для розгортання веб-додатків та статичних сайтів. У цьому сервісі доступна можливість автоматичного масштабування що дозволяє динамічно адаптувати розмір додатку в залежності від потреб. Ця технологія може автоматично збільшувати розмір проекту під час пікових навантажень та в свою чергу зменшувати його, коли навантаження зменшується, забезпечуючи оптимальну продуктивність та ефективне використання ресурсів. Отже, майже вся інформація, що обробляється системою знаходиться на різноманітних хмарних сервісах, що відкриває великі можливості для масштабування, оптимізації та надійності додатку, ці технології надають величезний вибір різноманітних функцій, які можна впровадити миттєво, що і є перевагою їх використання.

Ще одним методом оптимізації проекту є використання бандлів для «фронтенду», який означає процес об'єднання та компіляції різних джерел інформації або ресурсів таких, як наприклад HTML, CSS, JavaScript в один або кілька файлів для їх ефективного навантаження та виконання користувачем.

Щодо надійності, система використовує протокол HTTPS, основною перевагою якого є захист конфіденційності даних, що забезпечує шифрування інформації, які передаються між сервером та користувачем. Це означає, що ніхто

крім сервера та клієнта не зможе доступитися до цих даних. Цей протокол використовується для підтвердження автентичності сервера, в ньому наявні цифрові сертифікати, що дозволяють упевнитись, що користувач взаємодіє саме з оригінальними та перевіреними серверами, що унеможливлює атаку з перехопленням даних.

Ще одним методом забезпечення безпеки системи є використання бібліотеки «bcrypt» для Node.js[7] вона використовується для хешування паролів користувачів, шляхом генерації «солі», тобто випадкового набору текстових значень, що формує собою рядок. В цілому принцип дії даного методу полягає у тому, що при реєстрації нового користувача, його пароль ніде не зберігається, тому ідентифікація користувача відбувається шляхом порівняння збереженої солі та хешу, що унеможливлює будь які атаки зловмисників з цілю викрадення секретних паролів.

```
// Verifying password
if (originUser && !.compareSync(password, originUser.password)) {
    logger.debug(`Login request from ${login} (${req.ip}) | Failed: invalid password`)
    reply.code(400).send({ error: 'Invalid password' })
    return
}
```

Рисунок 2.7 – Приклад використання у коді бібліотеки «bcrypt»

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ МОНІТОРИНГУ ЕЛЕКТРОПОСТАЧАННЯ

3.1 Основні етапи розробки серверної частини системи

Першим та одним із найголовніших етапів розробки інформаційної системи моніторингу електропостачання було написання «бекенд» частини проекту, а саме серверу пристрою та головного серверу.

Спочатку потрібно було розробити сервер пристрою, в можливості якого входить підключення до головного серверу системи та передача даних з самого пристрою на нього, де ці всі дані в подальшому повинні оброблюватися. В якості місця де розташовується цей сервер слугує сам пристрій. Щоб головний сервер міг приймати ці дані з пристрою у коді серверу приладу потрібно вказати секретний ключ пристрою та посилання на головний сервер.

```
1   DEVICE_SECRET=29570ca7-13cc-4dc2-b620-b86d7ee4c417
2   MAIN_SERVER_URL=https://electro-tracker.ddns.net/heartbeat
```

Рисунок 3.1 – Приклад даних для підключення до головного серверу

В базовому варіанті, він налаштований лише на передачу інформації про його теперішній стан, тобто чи знаходиться пристрій в мережі на поточний момент. Проте система позиціонується, як відкрита до будь яких змін, тому код було написано так, що при бажанні користувач може легко змінити спеціальну частину скрипту для додаткової інформації, що повинен відправляти пристрій, наприклад додати у програму функціонал відслідковування показників датчику температури.

```
setInterval(() => {
    fetch(MAIN_SERVER_URL, {
        method: 'POST',
        headers: { auth: DEVICE_SECRET },
        body: JSON.stringify({ temperature: (Math.random() * 100).toFixed(2), humidity: (Math.random() * 100).toFixed(2)}),
    })
        .then(res => res.json())
        .then(json => console.log(json))
        .catch(err => console.error('Error:', err))
}, 1000)
```

Рисунок 3.2 – Частина коду, що відповідає за додатковий функціонал

На рис. 3.2. зображена частина коду, що відправляє прості випадкові значення на головний сервер, так, як можливості встановлення додаткових датчиків на даний момент немає. Проте цей скрипт працює так, що він надсилає результати своєї роботи у вигляді об’єкту, тому головний сервер та база даних налаштовані таким чином, що вони зможуть оброблювати та зберігати майже будь який тип інформації. В тому вигляді в якому вся система і задумувалася сам пристрій був сконструйований на базі Raspberry Pi 3 model B[10], але ще однією перевагою такого підходу є те, що даний сервер можна запустити на будь-якому пристрої, що підтримує виконання цього коду, чим надає велику варіативність свого використання.

Рисунок 3.3 – Фото пристрою сконструйованого на базі Raspberry Pi 3 model B

На рис 3.3. зображено базова модель пристрою, що сконструйована на базі Raspberry Pi 3 model B, який поміщений у спеціальний пластиковий корпус та налаштовано під роботу з скриптом для передачі з нього даних на головний сервер.

1. Далі потрібно було перейти до написання головного серверу усієї системи. Він відіграє у проекті ключову роль, тобто відповідає за обробку усієї наявної інформації, взаємодію з базою даних, реєстрацію та логування користувачів, взаємодію з телеграм ботом для сповіщення, створення та додавання до аккаунту нових або вже існуючих пристроїв.

З самого початку розробки головного серверу, потрібно було підключити до нього спроектовану в попередньому розділі роботи базу даних на основі MongoDb, що надає можливість зберігати та оброблювати усі дані про користувачів, пристрої та інформацію, що з них надходить.

```
// User schema
export const UserModel = connection.model('user', new Schema({
    _id: String,
    login: String,
    password: String,
    devices: [String],
    tgId: String,
}))

// Device schema
export const DeviceModel = connection.model('device', new Schema({
    _id: String,
    secret: String,
    name: String,
    creator: String,
}))

// Log schema with indexes
export const LogModel = connection.model('log', new Schema({
    device_id: String,
    timestamp: Number,
    status: String,
    payload: Object
}).index({ timestamp: -1 }))
```

Рисунок 3.4 – Створення схем колекції баз даних

2. Після цього потрібно було оформити реєстрацію та логування користувача. Воно зроблено таким чином, що при введені даних у поле для

логування відбувається також і реєстрація нового користувача, якщо при спробі зайти до аккаунту такого не існувало.

```
if (originUser) {
    logger.debug(`Login request from ${login} (${req.ip}) | User found. Password correct`)
}

if (!originUser) {
    logger.debug(`Login request from ${login} (${req.ip}) | User not found. Creating new user`)
    const hashedPassword = bcrypt.hashSync(password, 10)
    const user = new UserModel({
        _id: login,
        login,
        password: hashedPassword,
    })

    originUser = await user.save()
}
```

Рисунок 3.5 – Фрагмент коду, що відповідає за логування та реєстрацію

3. Далі потрібно було реалізувати підключення створеного аккаунту до телеграм нотифікатора, щоб це зробити потрібно перейти по спеціальному посиланню на сайті проекту по якому і відбувається прив’язка. Даний функціонал було реалізовано через застосування Telegram Api та створення боту, що при будь яких змінах у стані пристроїв миттєво надсилає користувачеві про це повідомлення.

```
tgBot.sendMessage(tgId, `Device ${device.name} #${device._id} is now ${status}. \nIt was ${lastStatus} for ${timeDifferenceHHMMSS}.`)
    .then(() => logger.debug(`Message sent to ${user._id} about device ${device.name} #${device._id} status change (${status})`))
    .catch((err) => {
        logger.error(`Error sending message to ${user._id} about device ${device.name} #${device._id} status change (${status})`)
        logger.error(err)
    })
```

Рисунок 3.6 – Приклад формування повідомлення, що надсилає телеграм бот

4. Після цього потрібно було приступити до написання коду, що відповідає за логіку взаємодії з пристроями. Було реалізовано два варіанта додавання до свого аккаунту нових приладів. В першому випадку користувач може підв’язати до свого аккаунту вже існуючий пристрій, щоб був створений іншим користувачем. Для цього потрібно або ж перейти по спеціальному посилання, що формується для кожного пристрою окремо, або ввести його унікальний ідентифікаційний код, яким при бажанні може поділитися власник приладу.

Другим же варіантом є власноруч створити новий пристрій, проте для цього потрібно буде самому налаштовувати та запускати для нього окремий сервер. При створенні нового приладу йому потрібно дати ім’я, id ж та ідентифікаційний код створюються та додаються до бази даних автоматично. Також звичайно, було реалізована можливість видалити вже існуючий пристрій, що прибирає його прив’язку до аккаунту, якщо цей аккаунт не є власником пристрою, або ж повністю його видалити, якщо ця людина створила цей пристрій.

```
        // Add device if not exists
        await UserModel.findOneAndUpdate({ login }, { $addToSet: { devices: deviceId } }, { new: true })
            .then((doc) => {
                if (!doc) {
                    logger.debug(`Link device request from ${login} (${req.ip}) | Failed: user not found`)
                    reply.code(400).send({ error: 'User not found' })
                    return
                }

                logger.debug(`Link device request from ${login} (${req.ip}) | Success`)
                reply.send({ success: true })
            })
            .catch((err) => {
                logger.error(`Link device request from ${login} (${req.ip}) | Failed: db error`)
                logger.error(err)
                reply.code(500).send({ error: 'Internal server error' })
            })
    },
})
```

Рисунок 3.7 – Код, що показує створення нового пристрою

Потім залишилось лише прописати логіку для оброки інформації, що надсилає сервер пристроїв. Вона включає в себе їхній статус, якому користувачеві належить даний пристрій та інформацію про час коли він в останній раз відповідав на запити. Також там міститься його id, ідентифікаційний код та ім’я. В кінці головний сервер було розміщено на сервісах AWS.

```
    // Detect device gone offline
    if (lastLog.status === DeviceStatus.ONLINE && diff > THRESHOLD_MS) {
        await pushLog(device, DeviceStatus.OFFLINE, null)
    }
```

Рисунок 3.8. – Частина коду, що відповідає за фіксацію стану пристрою «офлайн»

3.2 Основні етапи розробки інтерфейсу користувача

Другим етапом розробки інформаційної системи моніторингу електропостачання було написання «фронтенд» частини проекту. Її було реалізовано за допомогою бібліотеки для мови програмування JavaScript, яка називається React з використанням багатьох допоміжних бібліотек. Цей фреймворк відмінно підходить для створення величезних веб-додатків, де дані можуть змінюватися на регулярній основі. Перевагами ReactJS є простота його синтаксису, високий рівень гнучкості та віртуальна DOM (document object model).

Спочатку потрібно було створити головну сторінку проекту, що було найлегшим завданням у цьому етапі тому, що вона представляє собою просто статичний текст.

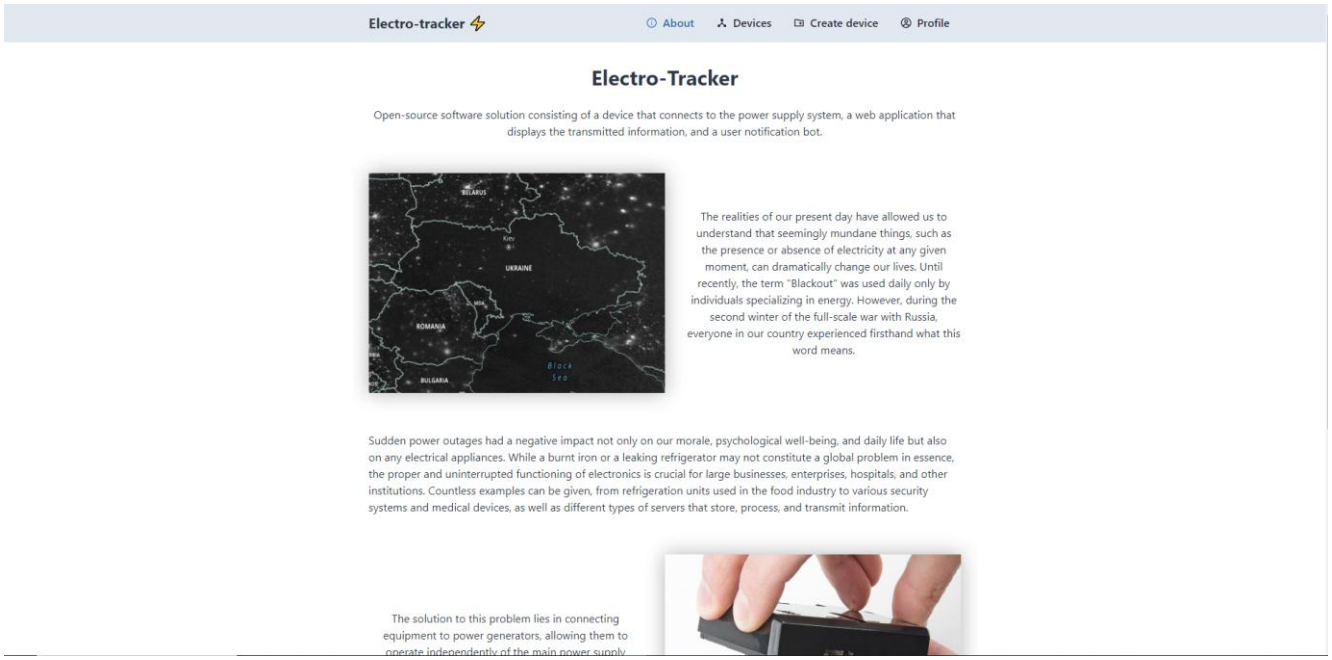

Рисунок 3.9 – Головна сторінка системи моніторингу електропостачання

Далі потрібно було створити сторінку де відбувається реєстрація та логування користувачів. Вона включає в себе два поля у, яких користувач вводить свої персональні дані.

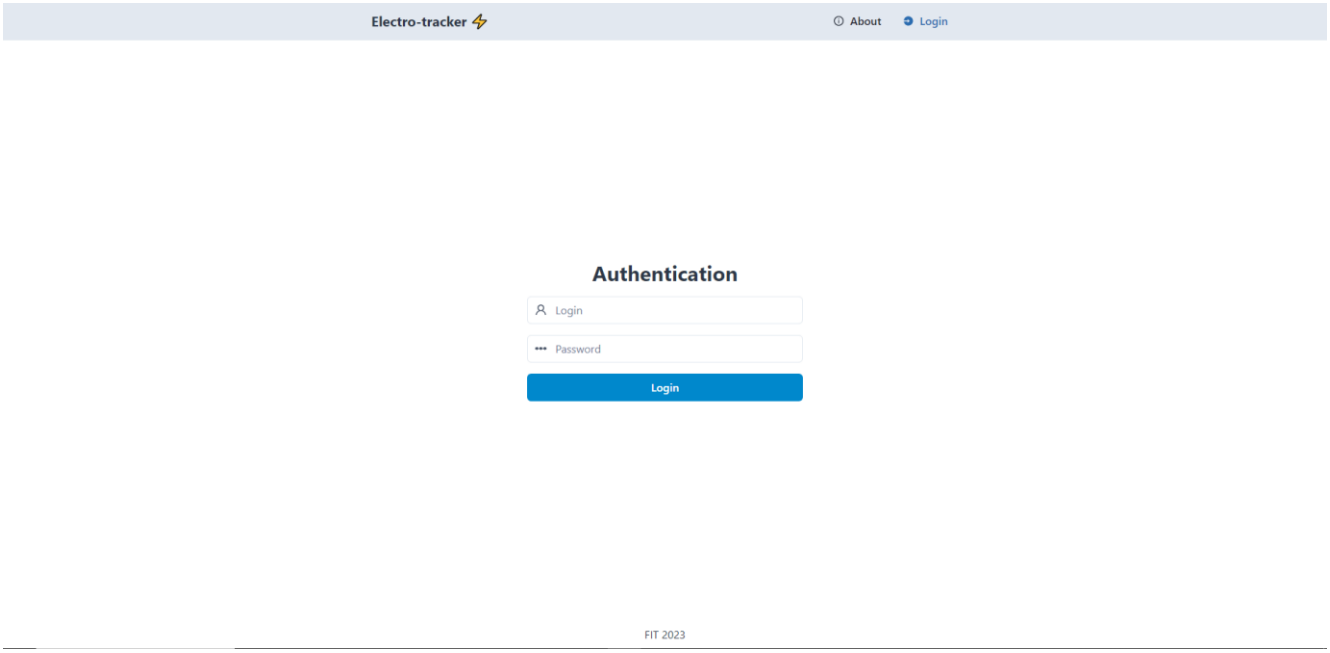

Рисунок 3.10 – Сторінка аутентифікації системи моніторингу електропостачання

Після цього потрібно було створити сторінку де знаходяться усі доступні користувачеві прилади з невеликою кількістю інформації про них, а саме їх статус, ім’я, останній раз коли вони відповідали на запити та кому вони належать. Також присутнє поле, при заповненні якого спеціальним ідентифікаційним ключем можна додати новий прилад до списку.

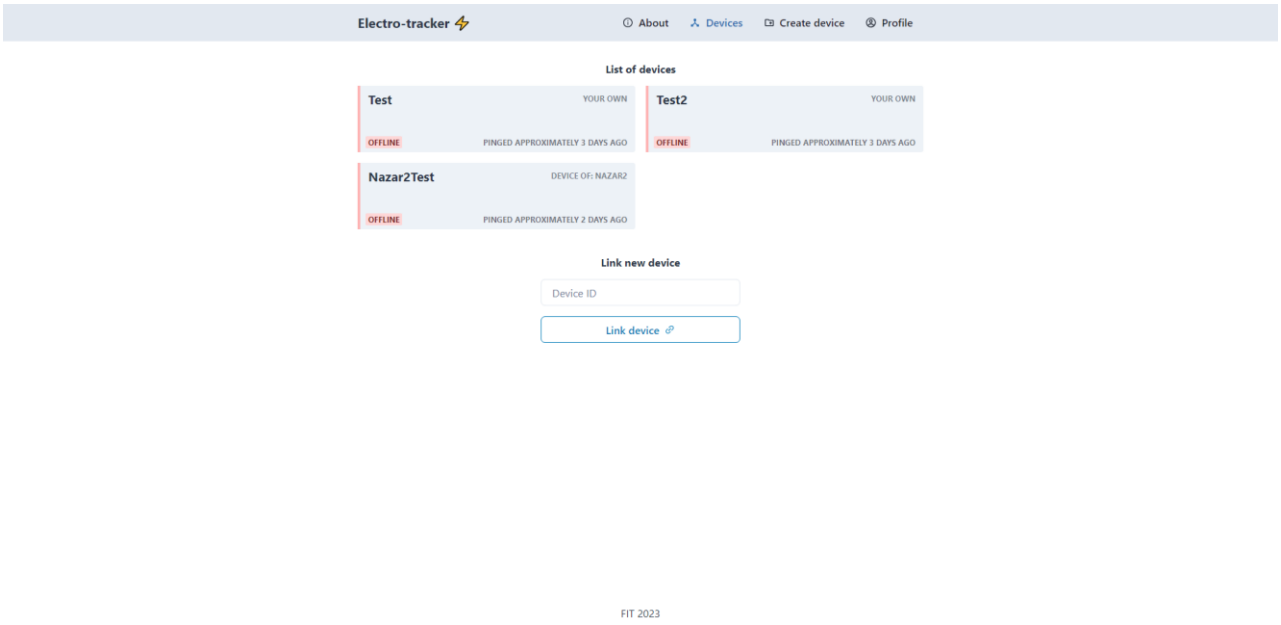

Рисунок 3.11 – Сторінка доступних пристроїв

Логічним продовженням розробки було б створення детальної інформації про кожен прилад окремо, тому далі було написано сторінку на якій знаходиться статистика відключень пристрою до мережі, його інформація та три кнопки. Кнопка «Show secret» відповідає за відображення ідентифікаційного ключа пристрою, якщо користувач є його власником. Кнопка «Share device» створює посилання для під’єднання вже існуючого пристрою до якогось іншого аккаунту. Кнопка «Unlink device» видаляє пристрій з аккаунту.

Рисунок 3.12 – Сторінка детальної інформації про пристрій

Також було розроблено сторінку для створення своїх власних пристроїв. Для цього потрібно просто написати ім’я приладу, все інше створиться автоматично. Після цього новий пристрій буде відображатись на сторінці з усіма іншими приладами, проте в нього не буде статусу поки для пристрою не буде налаштований свій власний сервер та скрипт.

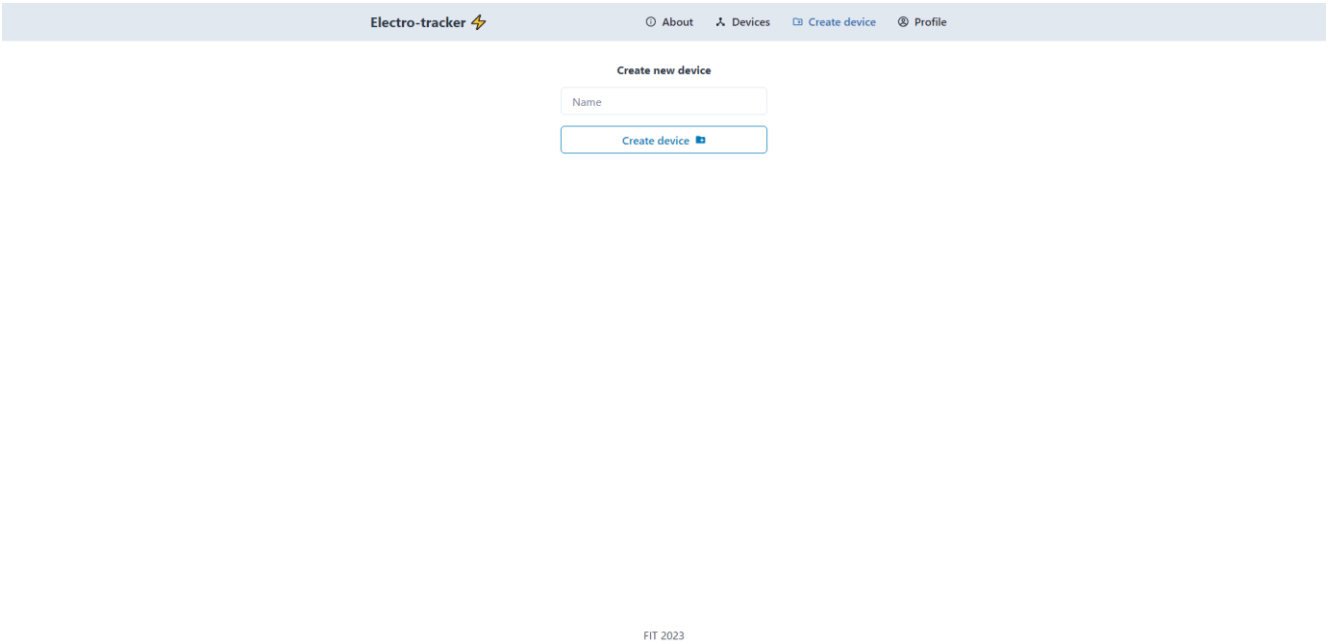

Рисунок 3.13 – Сторінка створення нових пристроїв

В кінці була створена сторінка користувача, на якій є можливість прив’язати телеграм акаунт та вийти з акаунту.

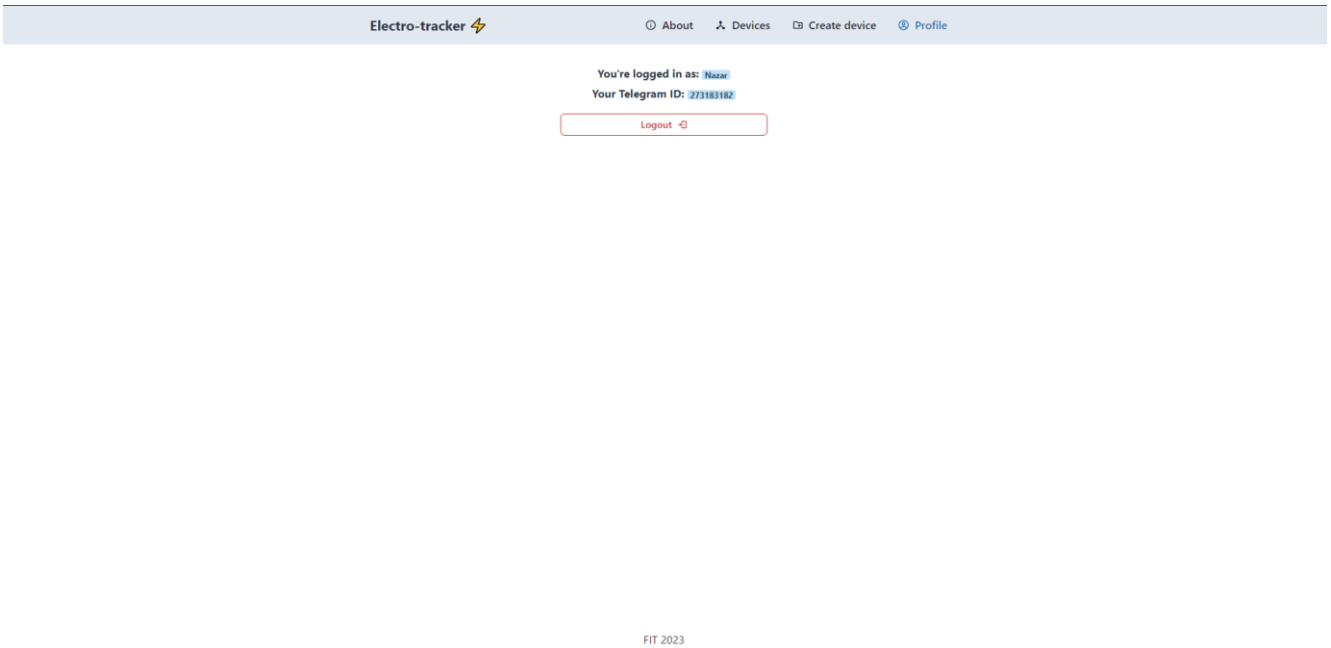

Рисунок 3.14 – Сторінка користувача

Звичайно, при розробці користувацької частини не обійшлося без використання різноманітних бібліотек таких, як наприклад ReactIcons[1] за

допомогою якої беруться іконки для розділів, мови розмітки HTML та CSS в поєднанні з JavaScript за допомогою якого було прописано логіку для усіх функціональних частин. Візуальна частина проекту розміщується на хмарному сервісі Vercel[9].

РОЗДІЛ 4 ДОСЛІДЖЕННЯ РОБОТИ ІНФОМРМАЦІЙНОЇ СИСТЕМИ

В результаті розробки була отримана інформаційна система моніторингу електропостачання, яка працює таким чином, сервер пристрою передає запити, головному серверу, який їх оброблює та разом з візуальною частиною додатку та ботом-сповіщувачем доносить всю цю інформацію користувачеві.

На початку роботи із застосунком користувач повинен пройти реєстрацію у застосунку, після цього прив’язати до нього свій власний телеграм аккаунт для можливості отримання сповіщень від системи. Для цього потрібно натиснути на кнопку «Link Telegram account» на сторінці інформації про користувача. Ця дія перенаправить користувача на спеціально сторінку прив’язки.

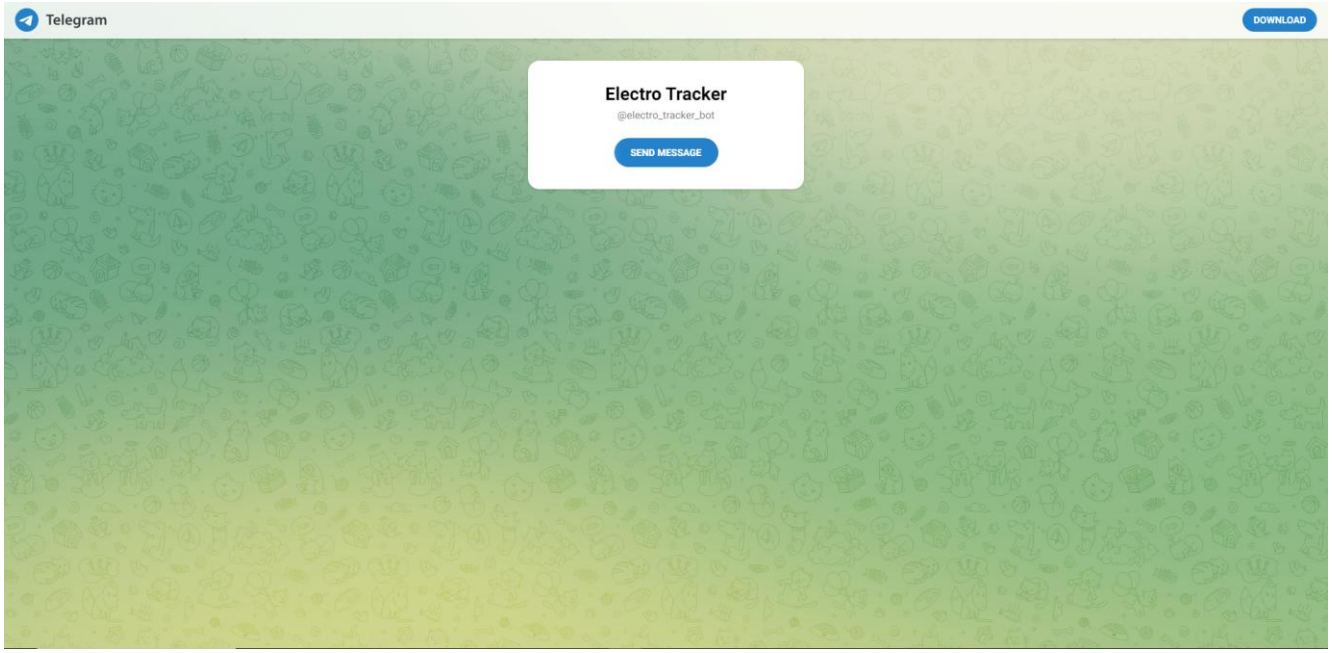

Рисунок 4.1 – Сторінка під’єднання телеграм акаунту

Після цього відкривається месенджер телеграм та сторінка з ботом сповіщувачем, автоматично надсилається команда /start та відбувається з’єднання з акаунтом застосунка.

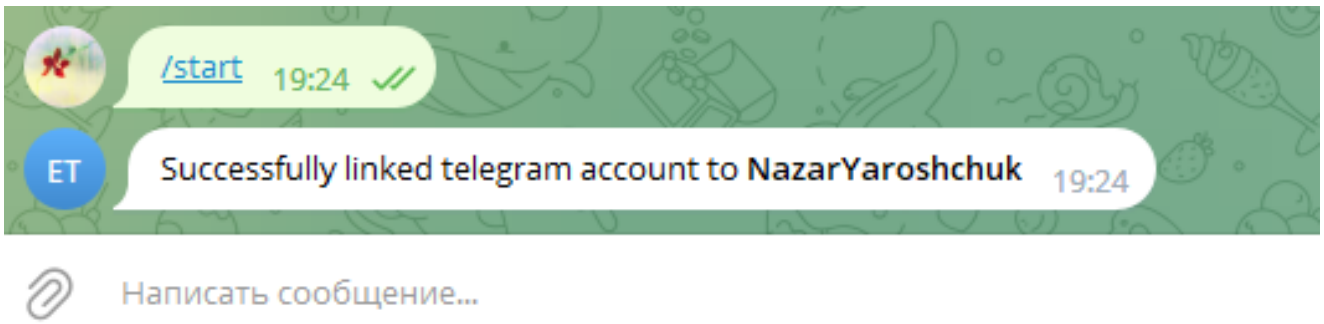

Рисунок 4.2 – Приклад повідомлення про успішне під’єднання акаунту

У додатку існує два способи підключення нових пристроїв до акаунту. Перший це підв’язати до нього вже існуючий пристрій. Для цього потрібно перейти по спеціальному посиланню, яке повинен згенерувати власник пристрою, або ж ввести у спеціальне поле його секретний ключ.

Приклад посилання: https://electro-tracker-coral.vercel.app/devices/01657ac3.

Приклад секретного ключа: 01657ac3-5de3-43fd-941f-99c5a1b39695.

Після того, як користувач переходить наприклад по посиланню, пристрій з’являється у його таблиці доступних пристроїв.

Другий спосіб підключення пристрою це його безпосереднє створення у веб додатку. Для цього потрібно перейти у вікно створення нового пристрою та просто заповнити поле з його назвою. Цей спосіб обов’язково включає в себе налаштування алгоритму приладу та налаштування серверу для нього.

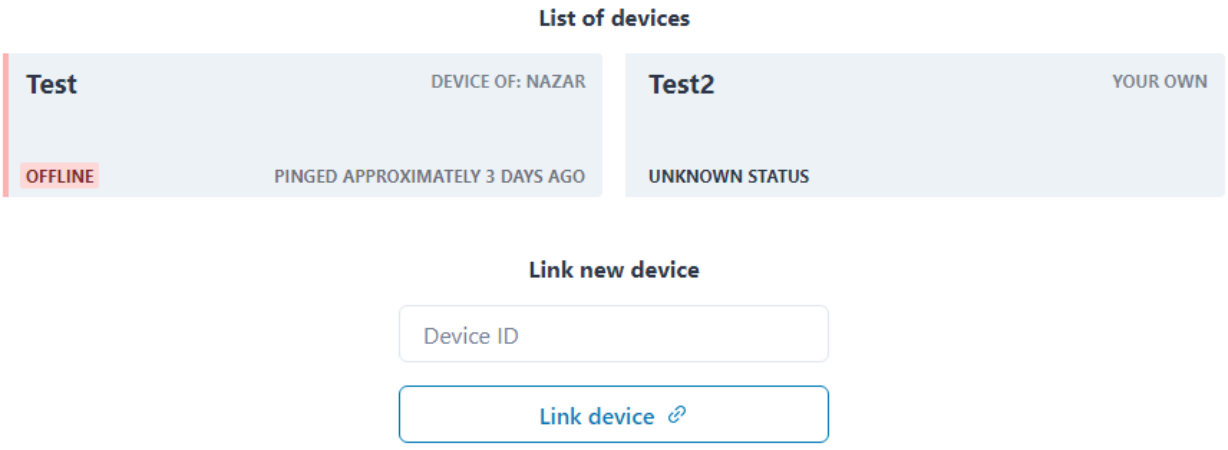

Рисунок 4.3 – Під’єднані пристрої двома різними способами

На рис 4.3. зображено два пристрої, один вже існуючий, тобто яким володіє інший користувач та тільки, що створений, яким володіє користувач цього аккаунту.

Після того як пристрій змінить свій стан ця інформація буде відображатися у веб-застосунку та буде надіслано повідомлення до телеграму користувача. Це зображено на рис. 4.4.

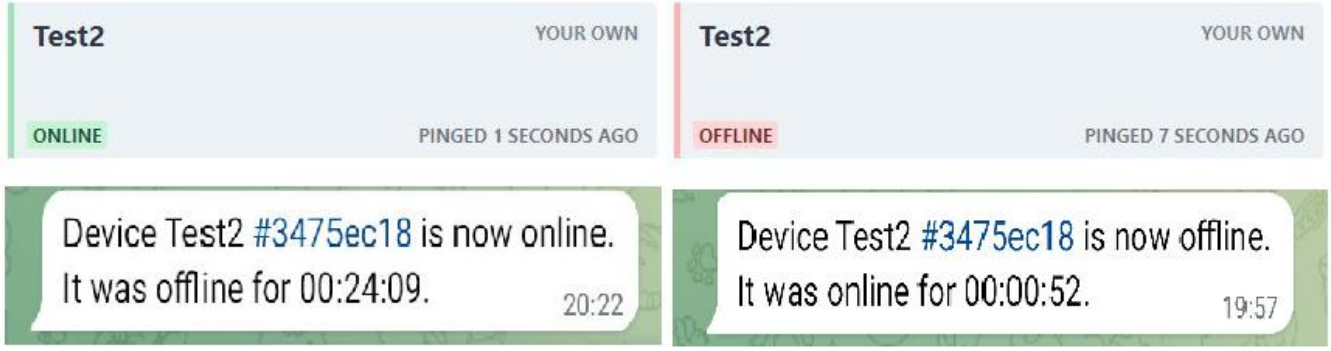

Рисунок 4.4 – Різні стани пристрою та повідомлення про них

Про кожен із підключених пристроїв можна переглядати детальну інформацію, відображення якої змінюється в реальному стані.

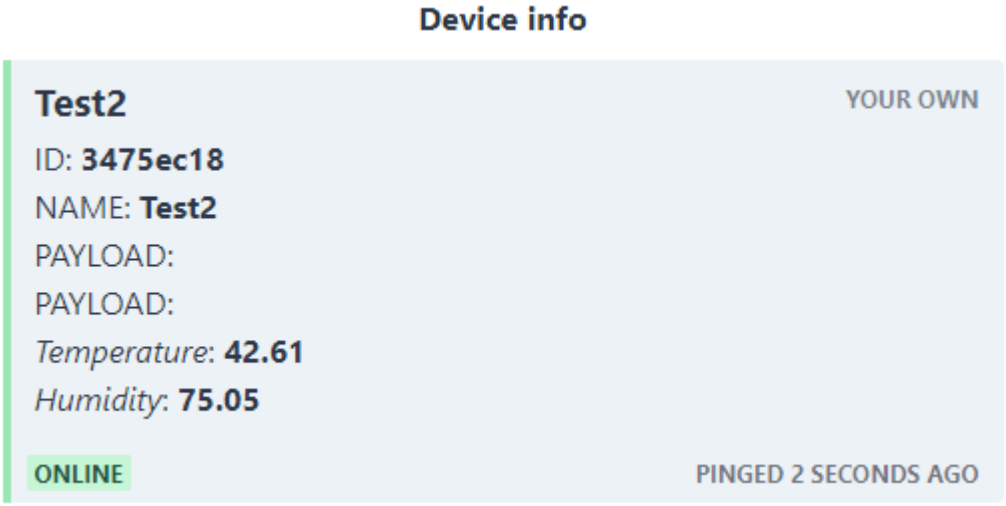

Рисунок 4.5 – Детальна інформація про стан пристрою.

Також веб-застосунок відображає статистику роботи пристрою, вона відображає проміжки часу коли пристрій був або не був у мережі рис. 4.5.

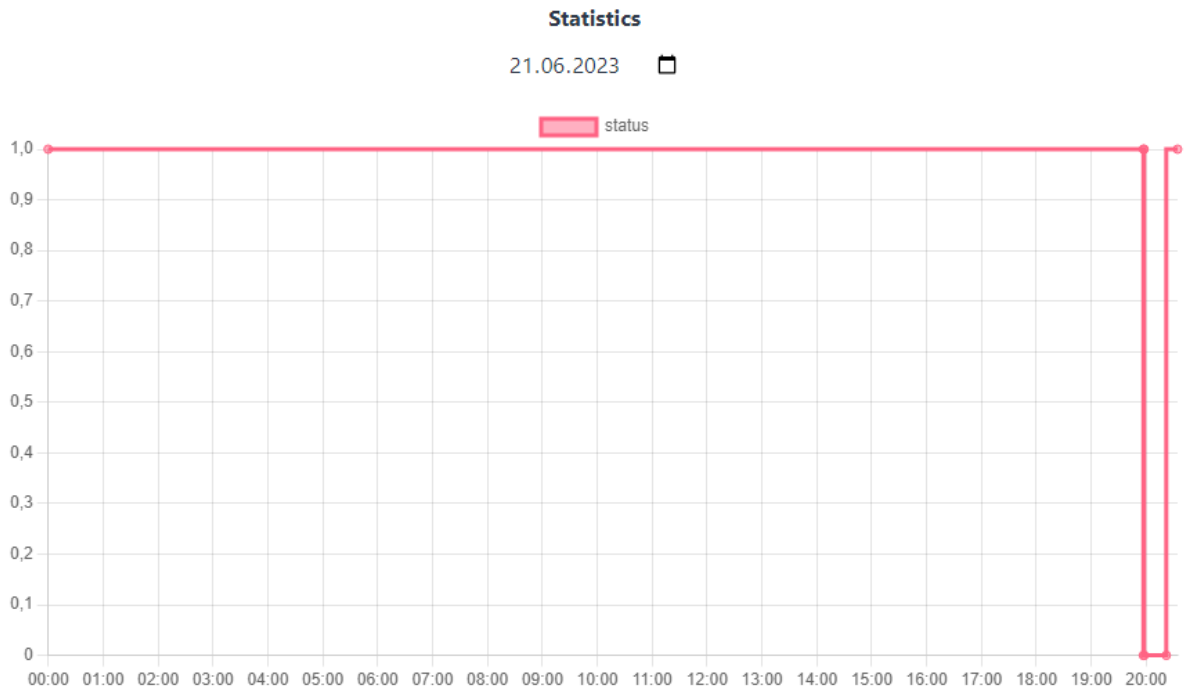

Рисунок 4.6 – Детальна статистика стану пристрою.

Отже під час тестування програмного забезпечення, було виявлено, що всі функції та завдання, які були поставлені в ході роботи було повністю виконано.

ВИСНОВКИ

В ході кваліфікаційної роботи бакалавра на тему Інформаційна система моніторингу електропостачання було:

1. Проаналізовано останні дослідження та методи, які використовуються для моніторингу електропостачання та визначено, що ці методи не цілком підходять для даної цілі через обмеженість свого функціоналу та певні недоліки;
2. Розглянуто та досліджено подібні та аналогічні рішення разом з їх архітектурою, було визначено, що системи розумних лічильників та охоронні системи також не цілком підходять для цієї задачі через їх велику вартість та дуже різноманітний функціонал, слідкувати за станом електромережі є їхньою додатковою функцією, а не основним завданням;
3. Визначено структурно-функціональне забезпечення системи та побудовано діаграму рівнів системи, вирішено було розділити систему на користувацьку та серверну частину в поєднанні з базою даних та пристроєм, це проста та водночас функціональна структура система, яка виконує усі поставлені перед нею задачі.
4. Досліджено та обрано стек технологій для розробки інформаційної системи, вибрані технології цілком розкривають весь розроблений функціонал та чудово з ними взаємодіють ;
5. Визначено методи масштабування, оптимізації та надійності, головним засобом для цього є використання різноманітних хмарних технологій, що дозволяють зручно вносити зміни у систему та забезпечувати її належну роботи;

В результаті тестування розробленої системи було визначено те, що обрані технології та засоби чудово розкривають весь задуманий функціонал, який було реалізовано в програмному застосунку. Проте в майбутньому планується додати

до пристрою автономне джерело живлення та GSM-модуль, що дозволить використовувати її у майже будь-яких умовах.

Отже, було виконано усі завдання та задачі, що були поставлені у ході роботи результатом було розроблено інформаційну систему моніторингу електропостачання, що візуалізує та повідомляє інформацію про стан електромережі в певному місці.

ЛІТЕРАТУРА

1. ReactJS. URL: https://uk.reactjs.org/docs/ (електронний ресурс)
2. About JavaScript. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript (електронний ресурс)
3. How Javascript Event Loop Works? URL: https://www.techboxweb.com/javasript-event-loop/ (електронний ресурс)
4. React Virtual DOM. URL: https://programmingwithmosh.com/react/react-virtual-dom-explained/ (електронний ресурс)
5. MongoDB. Getting started. URL: https://www.mongodb.com/blog/post/getting-started-with-python-andmongodb (електронний ресурс)
6. Quick Start with React router. URL: https://v5.reactrouter.com/web/guides/quick-start (електронний ресурс)
7. NodeJS. URL: https://nodejs.org/uk/docs/ (електронний ресурс)
8. AWS Quick Start Guide: Back Up Your Files to Amazon Simple Storage Service. URL: https://docs.aws.amazon.com/quickstarts/latest/s3backup/welcome.html (електронний ресурс)
9. Serverless Functions Quickstart. URL: https://vercel.com/docs/concepts/functions/serverless-functions/quickstart (електронний ресурс)
10. Raspberry Pi Documentation. URL: https://www.raspberrypi.com/documentation/computers/getting-started.html (електронний ресурс)
11. TypeScript Documentation. URL: https://www.typescriptlang.org/docs/ (електронний ресурс)
12. From BotFather to ‘Hello world’. URL: https://core.telegram.org/bots/tutorial (електронний ресурс)

Додаток А

Лістинг файлу index.ts для серверної частини пристрою

```
import * as dotenv from 'dotenv'
import fetch from 'cross-fetch'
dotenv.config()
const DEVICE_SECRET = process.env.DEVICE_SECRET as string
const MAIN_SERVER_URL = process.env.MAIN_SERVER_URL as string
if (!DEVICE_SECRET) {
  throw new Error('DEVICE_SECRET is not specified')
}
if (!MAIN_SERVER_URL) {
  throw new Error('MAIN_SERVER_URL is not specified')
}
setInterval(() => {
  fetch(MAIN_SERVER_URL, {
    method: 'POST',
    headers: { auth: DEVICE_SECRET },
    body: JSON.stringify({ temperature: (Math.random() * 100).toFixed(2), humidity: (Math.random() * 100).toFixed(2)}),
  })
    .then(res => res.json())
    .then(json => console.log(json))
    .catch(err => console.error('Error:', err))
}, 1000)
```

Додаток Б

Лістинг файлу requests.ts для користувацької частини пристрою

```
import { useAxios } from './axios'
interface LinkDeviceData  {
  deviceId: string;
}
export const useRequests = () => {
  const { postRequest, getRequest, loading, error } = useAxios()
  const postLogin = async (data: any) => {
    return postRequest('/login', data)
  }
  const getDevices = async (noLoading: boolean = false) => {
    return getRequest('/devices', noLoading)
  }
  const linkDevice = async (data: LinkDeviceData) => {
    return postRequest('/link-device', data);
  }
  const unlinkDevice = async(data: LinkDeviceData) => {
    return postRequest('/unlink-device', data);
  }
  const createDevice = async (data: any) => {
      return postRequest('/create-device', data)
  }
  const getUser = async () => {
   return getRequest('/user')
```

```
  }
  const getLogs = async (deviceId: string) => {
    return postRequest('/logs', { deviceId })
  }
  return { loading, error, postLogin, getDevices, linkDevice, unlinkDevice,
createDevice, getUser, getLogs }
}
```

Додаток В

Лістинг файлу router.tsx для користувацької частини пристрою

```
import { createBrowserRouter, createRoutesFromElements, Route } from 'react-router-
dom'
import Overlay from '../../components/templates/overlay/Overlay'
import Root from '../../components/pages/root/Root'
import Login from '../../components/pages/login/Login'
import Error from '../../components/pages/error/Error'
import {
  CREATE_DEVICE_ROUTE,
  DEVICE_ROUTE,
  DEVICES_ROUTE,
  ERROR_403_ROUTE,
  ERROR_500_ROUTE,
  HOME_ROUTE,
  LOGIN_ROUTE,
  PROFILE_ROUTE,
} from './routes'
import Profile from '../../components/pages/profile/Profile'
```

```
import Devices from '../../components/pages/devices/Devices'
import Device from '../../components/pages/device/Device'
import CreateDevice from '../../components/pages/create-device/CreateDevice'
export const router = createBrowserRouter(
  createRoutesFromElements(
    <Route>
      <Route element={<Overlay />}>
        <Route path={HOME_ROUTE} element={<Root />} />
        <Route path={PROFILE_ROUTE} element={<Profile />} />
        <Route path={LOGIN_ROUTE} element={<Login />} />
        <Route path={DEVICES_ROUTE} element={<Devices />} />
        <Route path={DEVICE_ROUTE} element={<Device />} />
        <Route path={CREATE_DEVICE_ROUTE} element={<CreateDevice />} />
      </Route>
      <Route path={ERROR_403_ROUTE} element={<Error error={{ status: 403, message: 'Forbidden' }} />} />
      <Route path={ERROR_500_ROUTE} element={<Error error={{ status: 500, message: 'Internal Server Error' }} />} />
      <Route path='*' element={<Error error={{ status: 404, message: 'Page not found' }} />} />
    </Route>,
  ),
)
```

Додаток Г

Лістинг файлу overlay.tsx для користувацької частини пристрою

```
import React, { useContext } from 'react'
```

```
import { Outlet } from 'react-router-dom'
import Header from '../../organisms/header/Header'
import Footer from '../../organisms/footer/Footer'
import ContentWrapper from '../../organisms/content-wrapper/ContentWrapper'
import { useAuth } from '../../../hooks/useAuth'
import { AuthContext } from '../../../context/AuthContext'
import Login from '../../pages/login/Login'

const Overlay = () => {
  const { isAuthenticated } = useContext(AuthContext)
  if (!isAuthenticated){
    return (<>
      <Header />
      <ContentWrapper>
        <Login />
      </ContentWrapper>
      <Footer />
    </>);
  }
  return (
    <>
      <Header />
      <ContentWrapper>
        <Outlet />
      </ContentWrapper>
      <Footer />
    </>
  )
```

```
}
export default Overlay
```

Додаток Ґ

Лістинг файлу schemas.ts для серверної частини пристрою

```
import mongoose from 'mongoose'
import { MONGO_URI } from '../util/env'

const Schema = mongoose.Schema
const connection = mongoose.createConnection(MONGO_URI)
// User schema
export const UserModel = connection.model('user', new Schema({
  _id: String,
  login: String,
  password: String,
  devices: [String],
  tgId: String,
}))
// Device schema
export const DeviceModel = connection.model('device', new Schema({
  _id: String,
  secret: String,
  name: String,
  creator: String,
}))
// Log schema with indexes
```

```
export const LogModel = connection.model('log', new Schema({
  device_id: String,
  timestamp: Number,
  status: String,
  payload: Object
}).index({ timestamp: -1 }))
```

Додаток Д

Лістинг файлу auth.ts для серверної частини пристрою

```
import fastifyJwt from '@fastify/jwt'
import { JWT_SECRET } from '../util/env'
import { FastifyInstance, FastifyReply, FastifyRequest } from 'fastify'
import { UserModel } from '../schema/schemas'
import { UserDB } from '../route/routes-type'
import { UserJwtPayload } from '../type/type'
declare module '@fastify/jwt' {
  interface FastifyJWT {
    payload: UserJwtPayload
    user: UserDB
  }
}
export default function(fastify: FastifyInstance) {
  fastify.register(fastifyJwt, {
    secret: JWT_SECRET,
  })
```

```
  fastify.decorate('authenticate', async function(request: FastifyRequest, reply: FastifyReply) {
    try {
      await request.jwtVerify()
      const login = request?.user?.login
      console.log(login);
      request.user = await UserModel.findOne({ login }) as UserDB
      if (!request.user) {
        reply.code(401).send({
          error: 'User not found',
        })
      }
    } catch (err) {
      reply.send(err)
    }
  })
}
```

Додаток Е

Лістинг файлу deviceEventHandler.ts для серверної частини пристрою

```
import { UserModel } from '../schema/schemas'
import { DeviceStatus } from '../type/type'
import { tgBot } from './telegramBot'
import { logger } from './logger'
export const onDeviceStatusChange = async (device: any, status: DeviceStatus, timeDifference: number) => {
```

```
  const timeDifferenceHHMMSS = new Date(timeDifference).toISOString().substr(11, 8)
  const lastStatus = status === DeviceStatus.ONLINE ? 'offline' : 'online'
  logger.info(`Device ${device.name} #${device._id} is now ${status}. It was ${lastStatus} for ${timeDifferenceHHMMSS}.`)
  const deviceUsers = await UserModel.find({ devices: device._id })
  for (const user of deviceUsers) {
    logger.debug(`User ${user._id} has device ${device.name} #${device._id}`)
    const tgId = user.tgId
    if (!tgId) {
      logger.warn(`User ${user._id} has no tgId (device ${device.name} #${device._id})`)
      continue
    }

    tgBot.sendMessage(tgId, `Device ${device.name} #${device._id} is now ${status}. \nIt was ${lastStatus} for ${timeDifferenceHHMMSS}.`)
      .then(() => logger.debug(`Message sent to ${user._id} about device ${device.name} #${device._id} status change (${status})`))
      .catch((err) => {
        logger.error(`Error sending message to ${user._id} about device ${device.name} #${device._id} status change (${status})`)
        logger.error(err)
      })
  }
}
```

Додаток Є

Лістинг файлу routes.ts для серверної частини пристрою

```
import type { FastifyRequest } from 'fastify'
import { FastifyInstance, FastifyReply } from 'fastify'
import { CreateDeviceRequestBody, LoginRequestBody, PostDeviceBody } from
'./routes-type'
import { verifyLength } from '../util/verify'
import { jwtOpts, loginOpts, passwordOpts } from '../util/const'
import { DeviceModel, UserModel, LogModel } from '../schema/schemas'
import { generateDeviceSecret } from '../util/uid'
import { DeviceStatus } from '../type/type'
import { logger } from '../module/logger'
import { getDeviceLastStatus, pushLog } from '../module/logAnalyzer'
const bcrypt = require('bcrypt')
export const registerRoutes = (fastify: FastifyInstance) => {
  fastify.route({
    method: 'POST',
    url: '/login',
    handler: async (req: FastifyRequest<{ Body: LoginRequestBody }>, reply:
FastifyReply) => {
      const { login, password } = req?.body
      logger.debug(`Login request from ${login} (${req.ip})`)
      // Verifying username
      if (!verifyLength(login, loginOpts.minLength, loginOpts.maxLength)) {
        logger.debug(`Login request from ${login} (${req.ip}) | Failed: invalid login
length`)
        reply.code(400).send({ error: `Login must be between
${loginOpts.minLength} and ${loginOpts.maxLength} characters` })
        return
      }
      // Verifying password length
```

```
        if (!verifyLength(password, passwordOpts.minLength,
passwordOpts.maxLength)) {
            logger.debug(`Login request from ${login} (${req.ip}) | Failed: invalid
password length`)
            reply.code(400).send({ error: `Password must be between
${passwordOpts.minLength} and ${passwordOpts.maxLength} characters` })
            return
        }
        // Create user if not exists
        let originUser = await UserModel.findOne({ login })
        logger.debug(`User found: ${originUser}`)
        // Verifying password
        if (originUser && !.compareSync(password, originUser.password)) {
            logger.debug(`Login request from ${login} (${req.ip}) | Failed: invalid
password`)
            reply.code(400).send({ error: 'Invalid password' })
            return
        }
        if (originUser) {
            logger.debug(`Login request from ${login} (${req.ip}) | User found.
Password correct`)
        }
        if (!originUser) {
            logger.debug(`Login request from ${login} (${req.ip}) | User not found.
Creating new user`)
            const hashedPassword = bcrypt.hashSync(password, 10)
            const user = new UserModel({
                _id: login,
                login,
                password: hashedPassword,
```

```
            })
            originUser = await user.save()
        }
        // Sending token to user
        logger.debug(`Login request from ${login} (${req.ip}) | Success. Sending token`)
        reply.send({
            token: fastify.jwt.sign({ login }, jwtOpts),
            user: {
                login: originUser.login,
            },
        })
    },
})


// Register device route (Add device to user)
fastify.route({
    method: 'POST',
    url: '/link-device',
    onRequest: [fastify.authenticate],
    handler: async (req: FastifyRequest<{ Body: PostDeviceBody }>, reply: FastifyReply) => {
        const { deviceId } = req.body
        const { login } = req.user
        logger.debug(`Link device request from ${login} (${req.ip})`)
        // Check if device exists
        if (!await DeviceModel.findOne({ _id: deviceId })) {
            logger.debug(`Link device request from ${login} (${req.ip}) | Failed: device with id ${deviceId} not found`)
```

```
            reply.code(400).send({ error: 'Device not found' })
            return
        }
        logger.debug(`Link device request from ${login} (${req.ip}) | Device found`)
        // Add device if not exists
        await UserModel.findOneAndUpdate({ login }, { $addToSet: { devices: deviceId } }, { new: true })
            .then((doc) => {
                if (!doc) {
                    logger.debug(`Link device request from ${login} (${req.ip}) | Failed: user not found`)
                    reply.code(400).send({ error: 'User not found' })
                    return
                }

                logger.debug(`Link device request from ${login} (${req.ip}) | Success`)
                reply.send({ success: true })
            })
            .catch((err) => {
                logger.error(`Link device request from ${login} (${req.ip}) | Failed: db error`)
                logger.error(err)
                reply.code(500).send({ error: 'Internal server error' })
            })
    },
  })
  // Unlink device
  fastify.route({
    method: 'POST',
```

```
    url: '/unlink-device',
    onRequest: [fastify.authenticate],
    handler: async (req: FastifyRequest<{ Body: PostDeviceBody }>, reply:
FastifyReply) => {
        const { deviceId } = req.body
        const { login } = req.user
        logger.debug(`Unlink device request from ${login} (${req.ip})`)
        // Check if device exists
        if (!await DeviceModel.findOne({ _id: deviceId })) {
            logger.debug(`Unlink device request from ${login} (${req.ip}) | Failed:
device with id ${deviceId} not found`)
            reply.code(400).send({ error: 'Device not found' })
            return
        }

        logger.debug(`Unlink device request from ${login} (${req.ip}) | Device found`)
        // Remove device from user
        await UserModel.findOneAndUpdate({ login }, { $pull: { devices: deviceId } },
{ new: true })
            .then((doc) => {
                if (!doc) {
                    logger.debug(`Unlink device request from ${login} (${req.ip}) | Failed:
user not found`)
                    reply.code(400).send({ error: 'User not found' })
                    return
                }
                logger.debug(`Unlink device request from ${login} (${req.ip}) | Success`)
                reply.send({ success: true })
            })
```

```
            .catch((err) => {
                logger.error(`Unlink device request from ${login} (${req.ip}) | Failed: db error`)
                logger.error(err)
                reply.code(500).send({ error: 'Internal server error' })
            })
        },
    })

    // Get user devices
    fastify.route({
        method: 'GET',
        url: '/devices',
        onRequest: [fastify.authenticate],
        handler: async (req: FastifyRequest, reply: FastifyReply) => {
            const devices = req.user.devices
            const userLogin = req.user.login
            // Add device names and statuses to response
            const devicesWithInfo = await Promise.all(devices.map(async (deviceId) => {
                const device = await DeviceModel.findOne({ _id: deviceId })
                if (!device) return null
                const deviceLastStatus = await getDeviceLastStatus(deviceId)
                return {
                    ...device.toObject(),
                    secret: device.creator === userLogin ? device.secret : undefined,
                    status: deviceLastStatus,
                }
            }))
```

```
        logger.debug(`Devices request from ${req.user.login} (${req.ip}) | Success`)
        reply.send({ devices: devicesWithInfo })
      },
    })
    fastify.route({
      method: 'POST',
      url: '/create-device',
      onRequest: [fastify.authenticate],
      handler: async (req: FastifyRequest<{ Body: CreateDeviceRequestBody }>, reply: FastifyReply) => {
        const { name } = req.body
        const { login } = req.user
        logger.debug(`Create device request from ${login} (${req.ip}) | Creating device ${name}`)


        // Create device
        logger.debug(`Create device request from ${login} (${req.ip}) | Generating device secret`)
        const secret = await generateDeviceSecret()
        const _id = secret.slice(0, 8)
        logger.debug(`Create device request from ${login} (${req.ip}) | Saving device to database`)
        const device = new DeviceModel({
          _id,
          name: name || _id,
          secret,
          creator: login,
        })
        await device.save()
```

```
        logger.info(`Create device request from ${login} (${req.ip}) | Device ${_id} created`)
        // Add device to user devices
        await UserModel.findOneAndUpdate({ login }, { $addToSet: { devices: _id } }, { new: true })
            .then(doc => {
                logger.debug(`Create device request from ${login} (${req.ip}) | Device ${_id} added to user ${login}`)
                reply.code(200).send({
                    device,
                })
            })
            .catch(err => {
                logger.error(`Create device request from ${login} (${req.ip}) | Failed to add device ${_id} to user ${login}`)
                reply.code(500).send({ error: 'Internal server error', err })
            })
    },
  })
  fastify.route({
    method: 'POST',
    url: '/heartbeat',
    handler: async (req: FastifyRequest, reply: FastifyReply) => {
        const deviceSecret = req.headers['auth'] as string
        const payload = JSON.parse(req.body as string)
        logger.debug(`Heartbeat request from ${deviceSecret} (${req.ip}) | Payload: ${JSON.stringify(payload)}`)
        const device = await DeviceModel.findOne({ secret: deviceSecret })
        // Check if device exists
```

```
        if (!device) {
            logger.debug(`Heartbeat request from ${deviceSecret} (${req.ip}) | Failed: device not found`)
            reply.code(400).send({ error: 'Device not found' })
            return
        }
        logger.debug(`Heartbeat request from ${deviceSecret} (${req.ip}) | Device found. Saving log`)
        // Create log
        await pushLog(device, DeviceStatus.ONLINE, payload)
        logger.debug(`Heartbeat request from ${deviceSecret} (${req.ip}) | Log saved. Sending response`)
        reply.code(200).send({ status: 'ok' })
    },
  })

  fastify.route({
    method: 'GET',
    url: '/user',
    onRequest: [fastify.authenticate],
    handler: async (req: FastifyRequest, reply: FastifyReply) => {
        const { login, tgId, devices } = req.user
        logger.debug(`User request from ${login} (${req.ip}) | Sending user info`)
        reply.send({
            login,
            tgId,
            devices,
        })
    },
```

```
    })
    // Get device logs
    fastify.route({
        method: 'POST',
        url: '/logs',
        onRequest: [fastify.authenticate],
        handler: async (req: FastifyRequest<{ Body: PostDeviceBody }>, reply:
FastifyReply) => {
            const { deviceId } = req.body
            // Check if device exists
            const device = await DeviceModel.findOne({ _id: deviceId })
            if (!device) {
                logger.debug(`Logs request from ${req.user.login} (${req.ip}) | Failed: device
not found`)
                reply.code(400).send({ error: 'Device not found' })
                return
            }
            // Get logs
            const logs = await LogModel.find({device_id: deviceId}).sort({timestamp: -1})
            reply.send({
                logs
            })
        }
    })
}
```