ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи на
ступінь вищої освіти бакалавр

на тему: **« РОЗРОБКА EMBEDDED ПРОДУКТУ ДЛЯ ТРЕКІНГУ СВІТЛА В БУДИНКУ З ВИКОРИСТАННЯМ ARDUINO, PYTHON, TELEGRAM API »**

Виконав: студент 5 курсу, групи ІПЗ–51 спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності/спеціалізації)

Коваль М. С.
(прізвище та ініціали)

Керівник Золотухіна О.А.
(прізвище та ініціали)

Рецензент

(прізвище та ініціали)

Київ – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти - «Бакалавр»
Спеціальність підготовки – 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____Негоденко О.В.
“____” _____2023 року

ЗАВДАННЯ НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТА

Коваль Максим Сергійович

(прізвище, ім’я, по батькові)

1. Тема роботи: « Розробка embedded продукту для трекінгу світла в будинку з використанням Arduino, Python, Telegram API »
Керівник роботи: Золотухіна Оксана Анатоліївна, к.т.н. доцент кафедри ІПЗ
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом вищого навчального закладу від «24» лютого 2023 року №26.
2. Строк подання студентом роботи«1» червня 2023 року
3. Вхідні дані до роботи:
3.1. Науково-технічна література з питань, пов’язаних з програмним забезпеченням щодо розробки embedded продукту.
3.2. Офіційна документація Python, Arduino, Android.
3.3. Офіційна документація Android Studio, Arduino IDE, PYcharm.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити).
4.1. Огляд та аналіз існуючих додатків та систем для трекінгу світла в будинку.
4.2. Огляд та аналіз існуючих мікроконтролерів для збору та передачі телеметрії.
4.3. Аналіз вимог до embedded продукту для трекінгу світла в будинку та формування технічного завдання до проекту.
4.4. Проектування та розробка пристрою для трекінга світла.

4.5. Проектування та розробка архітектури бази даних.
4.6. Проектування та розробка архітектури серверу обробки даних.
4.7. Проектування графічного інтерфейсу мобільного додатку.
4.8. Проектування та розробка архітектури мобільного додатку.
4.9. Тестування embedded продукту для трекінгу світла в будинку.
5. Перелік демонстраційного матеріалу (назва основних слайдів)
5.1. Мета, об’єкт, предмет, наукова новизна дослідження.
5.2. Аналіз аналогів.
5.3. Вимоги до додатку
5.4. Програмні засоби та інструменти реалізації.
5.5. Реалізація програми.
5.6. Апробація результатів дослідження.
5.7. Висновки.
6. Дата видачі завдання «25» лютого 2023р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	25.02.2023-27.02.2023	Виконано
2	Дослідження аналогів та актуальності додатку	12.03.2023-15.03.2023	Виконано
3	Аналіз та вибір інструментів для розробки додатку	15.04.2023	Виконано
4	Проектування та реалізація додатку	28.04.2023-05.05.2023	Виконано
5	Вступ, висновки, реферат	17.05.2023	Виконано
6	Розробка обов‘язкових демонстраційних матеріалів	18.05.2023	Виконано
7	Попередній захист роботи	19.05.2023	Виконано
8	Здача роботи	1.06.2023	Виконано

Студент _____ Коваль М.С.
(підпис) (прізвище та ініціали)
Керівник роботи _____ Золотухіна О.А.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи: 67 с., 3 табл., 15 рис., 13 джерел.

ІНТЕРНЕТ РЕЧЕЙ, EMBEDDED, ARDUINO, PYTHON, ANDROID, C++, TELEGRAM API.

Об’єкт дослідження – процес трекінгу світла в будинку.

Предмет дослідження – embedded продукти для трекінгу світла в будинку.

Мета роботи – спрощення процесу перевірки стану світла в будинку в режимі реального часу та підвищення рівня поінформованості користувача про зміну стану світла в будинку за рахунок використання embedded продукту на основі Arduino, Python, Telegram API.

У дипломному проекті проведений аналіз існуючих додатків та систем для трекінгу світла в будинку. Проведено аналіз існуючих мікроконтролерів для збору та передачі телеметрії. Продукт було створено за допомогою платформи Arduino мовою C++. Розроблено архітектуру бази даних, використовуючи Google Firebase. Розроблено архітектуру серверу обробки даних за допомогою Python та фреймворку FastApi, з використанням брокеру повідомлень RabbitMQ для обробки запитів асінхронно. Був розроблений дізайн графічного інтерфейсу мобільного додатку використовуючи Figma. Розроблено архітектуру мобільного додатку. Додаток базується на платформі Android за допомогою мови Kotlin та фреймворку Jetpack Compose.

Галузь використання додатку – системи розумного будинку, мережі IoT.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IoT - Інтернет речей (Internet-of-Things)

AR – доповнена реальність

Embedded - вбудовувана система;

API – прикладний програмний інтерфейс;

БД/СУБД – система управління базами данних;

IDE – інтегроване середовище розробки;

HTTP - протокол передачі даних;

SAAS – модель розгортання та реалізації програмного забезпечення;

ER-діаграма – діаграма сутність-зв'язок;

MVP - мінімальний життєздатний продукт;

GitHub – репозиторій;

ОС – операційна система.

CI/CD - безперервна інтеграція/безперервна доставка

UI – Інтерфейс користувача

ЗМІСТ

ВСТУП

В наші часи кожна сфера життя під впливом розвитку сучасних інформаційних систем та технологій, зокрема Інтернет речей (Internet-of-Things або IoT), які активно розвиваються та застосовуються, як в бізнесі, так і в домашньому господарстві. Наразі є актуальним питання використання технологій IoT в домашньому господарстві, а саме у відслідковуванні стану світла в оселі в режимі реального часу. Створений Embedded продукт вирішує вищезазначені питання та є незамінним аналогом на ринку, ґрунтуючись на перевагах і недоліках конкурентів.

Об’єкт дослідження – процес трекінгу світла в будинку.

Предмет дослідження – embedded продукти для трекінгу світла в будинку.

Мета роботи – спрощення процесу перевірки стану світла в будинку в режимі реального часу та підвищення рівня поінформованості користувача про зміну стану світла в будинку за рахунок використання embedded продукту на основі Arduino, Python, Telegram API.

Досягнення зазначеної мети роботи досягається вирішенням наступних задач:

– огляд та аналіз існуючих додатків та систем для трекінгу світла в будинку;

– огляд та аналіз існуючих мікроконтролерів для збору та передачі телеметрії;

– аналіз вимог до embedded продукту для трекінгу світла в будинку та формування технічного завдання до проекту;

– проектування та розробка пристрою для трекінгу світла;

– проектування та розробка архітектури бази даних;

– проектування та розробка архітектури серверу обробки даних;

– проектування графічного інтерфейсу мобільного додатку;

– проектування та розробка архітектури мобільного додатку;

– тестування embedded продукту для трекінгу світла в будинку.

В роботі проведений аналіз існуючих додатків та систем для трекінгу світла в будинку. Проведено аналіз існуючих мікроконтролерів для збору та передачі

телеметрії. Продукт було створено за допомогою платформи Arduino мовою C++. Розроблено архітектуру бази даних, використовуючи Google Firebase. Розроблено архітектуру серверу обробки даних за допомогою Python та фреймворку FastApi, з використанням брокеру повідомлень RabbitMQ для обробки запитів асінхронно. Був розроблений дізайн графічного інтерфейсу мобільного додатку використовуючи Figma. Розроблено архітектуру мобільного додатку. Додаток базується на платформі Android за допомогою мови Kotlin та фреймворку Jetpack Compose.

1 ОГЛЯД ТА АНАЛІЗ ІСНУЮЧИХ ДОДАТКІВ ТА СИСТЕМ ДЛЯ ТРЕКІНГУ СВІТЛА В БУДИНКУ

1.1. Огляд Embedded продуктів та додаткових реалістичних технологій в діджиталізованому світі

Embedded продукти лежать в основі багатьох різних продуктів, машин та інтелектуальних операцій, таких як машинне навчання та програми штучного інтелекту. Оскільки сьогодні вбудовані системи з'являються в кожній галузі та секторі, вбудовані пристрої та програмне забезпечення відіграють вирішальну роль у функціонуванні автомобілів, побутової техніки, медичних приладів, інтерактивних кіосків та іншого обладнання, яке ми використовуємо у повсякденному житті.

Хоча реальні вбудовані системи стали значною частиною нашого життя, вони спроектовані так, щоб працювати з мінімальним втручанням людини. Такі характеристики, як компактний розмір, проста конструкція та низька вартість, роблять їх корисною технологією в таких галузях, як аерокосмічна, автомобільна, медична та навіть "розумні" міста. Таким чином, вони є однією з рушійних сил сучасного цифрового, підключеного та автоматизованого світу.

Разом з IoT ми можемо спостерігати активний розвиток доповненої реальності (AR) - технологія з величезними перспективами, що стає ключем до розкриття повного потенціалу Інтернету речей. Їх поєднання дозволяє бізнесу вирішувати безліч питань та розширювати свої можливості на ринку.

Згідно з дослідженнями Gartner ринок Інтернету речей у 2025 році становитиме 58 мільярдів доларів, що на 34% більше, ніж у 2020 році. Найбільший сегмент доходів припадатиме на фазу створення, тоді як найбільше зростання відбудеться на фазі запуску [7].

Аналіз Gartner включають поглиблені власні дослідження, передовий досвід колег та галузі, аналіз тенденцій та кількісне моделювання, та інноваційні підходи,

За дослідженям і прогнозами аналітиків ми можемо бачити, що кількість пристроїв підклюсенних до IoT зростає і очікується, що в осяжному майбутньому він буде зростати з року в рік. Згідно з останніми доступними даними, налічується приблизно 19,18 мільярда підключених IoT-пристроїв [6].

Рисунок 1.1 демонструє стрімкий ріст використання IoT пристроїв та їх перевагу перед використанням пристроїв, що не належать до Інтернету речей.

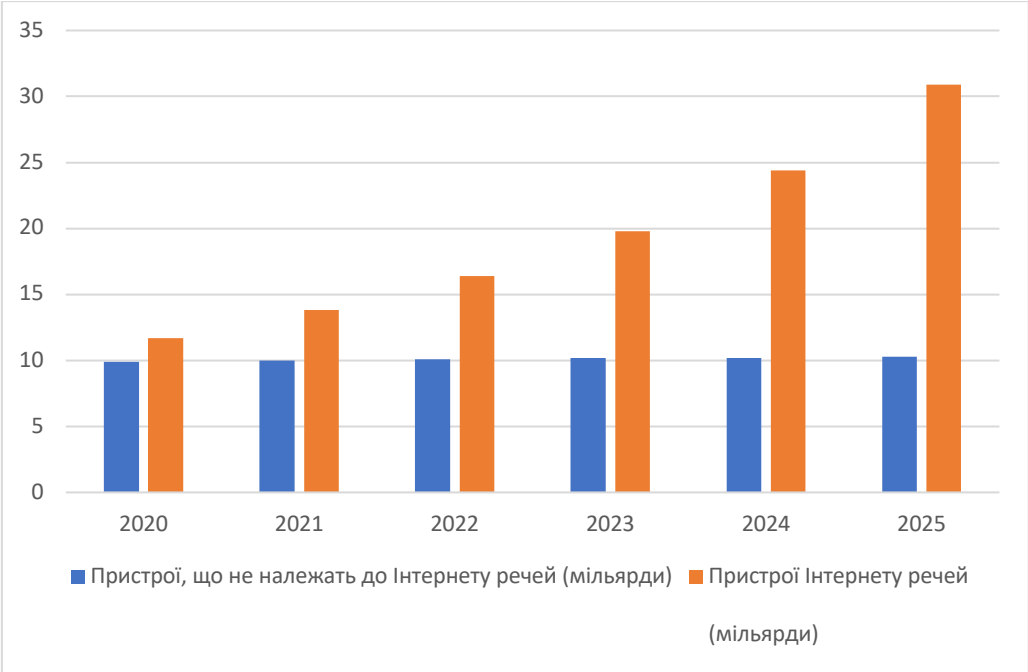


Рисунок1.1 - Порівняння росту використання не IoT пристроїв до IoT пристроїв

Ринок доповненої реальності також має прогнози на ріст. Згідно MarketsandMarkets ринок доповненої реальності може досягти $88,40 млрд до 2026 року із середньорічним темпом зростання 31,5% [3].

Таким чином, ці дослідження і звіти показують, що в найближчі роки ринок IoT та AR буде стрімко зростати, також за цими прогнозами можна бачити, що IoT і доповнена реальність мають великий потенціал для розвитку майже в усіх галузях нашого життя і може призвести до нових інновацій у бізнесі та промисловості.

Основні тенденції використання IoT та AR в різних сферах бізнесу:

- компанії очікують, що використання технологій IoT та AR стануть нормою для використання в їх галузі в найближчі 5 років [11];

- більшість компаній зараз використовують IoT, а також вивчають можливість використання AR, тоді як близько 76% тих, хто розробляє лише AR-рішення, вважають, що додавання IoT до їхніх додатків матиме цінність [4];

- індустрія нерухомості часто використовує доповнену реальність, щоб покращити досвід своїх споживачів, роблячи об'єкти нерухомості більш доступними. Крім того, IoT дозволяє використовувати розумні технології, які розвивають концепцію "розумних будинків";

- технології IoT та AR активно використовуються в дизайні продуктів. Наприклад, Microsoft HoloLens використовує змішану реальність для перетворення зображень у реальному світі, а Інтернет речей з'єднує пристрої та системи з мережею та хмарою.

- IoT і доповнена реальність в освіті. Simbionix Simulators - це 3D-системи, які є медичними тренажерами. Вони допомагають фахівцям і студентам отримати практичний досвід у проведенні малоінвазивних хірургічних операцій.

1.2. Особливості використання пристроїв IoT в різних сферах діяльності

Пристрої IoT служать різним цілям, включаючи підвищення ефективності роботи, покращення взаємодії з користувачем, уможливлення прогнозованого обслуговування та підтримку прийняття рішень на основі даних.

Поточне використання пристроїв IoT наведено нижче.

Розумний дім: пристрої IoT зазвичай використовуються в розумних будинках для автоматизації та керування різними аспектами, такими як освітлення, системи безпеки, термостати, прилади та розважальні системи.

Промислове застосування: пристрої IoT знаходять широке застосування в галузях промисловості для відстеження активів, управління ланцюжками поставок, прогнозованого обслуговування, віддаленого моніторингу та оптимізації виробничих процесів.

Охорона здоров’я. Пристрої IoT використовуються в охороні здоров’я для дистанційного моніторингу пацієнтів, переносних трекерів здоров’я, систем керування ліками та медичного обладнання з підтримкою IoT.

Сільське господарство: пристрої Інтернету речей використовуються в сільському господарстві для точного землеробства, моніторингу ґрунту, контролю зрошення, відстеження худоби та моніторингу здоров’я посівів.

Розумні міста: пристрої IoT сприяють ініціативам розумних міст, забезпечуючи інтелектуальні транспортні системи, розумне паркування, управління відходами, екологічний моніторинг та оптимізацію енергоспоживання.

Роздрібна торгівля: пристрої IoT використовуються в роздрібній торгівлі для управління запасами, відстеження клієнтів, персоналізованого маркетингу, розумних полиць і розумних платіжних систем.

Транспорт і логістика: пристрої IoT полегшують керування автопарком, відстеження активів, оптимізацію маршрутів, моніторинг стану транспортних засобів і видимість ланцюга поставок.

Управління енергією: пристрої IoT підтримують енергоефективність і управління за допомогою інтелектуальних лічильників, інтелектуальних мереж, систем реагування на попит і моніторингу енергії в реальному часі.

Переваги використання пристроїв IoT:

- зростаюче впровадження: використання пристроїв IoT швидко поширюється в різних секторах завдяки прогресу технологій, зниженню витрат і зростанню обізнаності про переваги, які вони пропонують;
- підключення та сумісність: пристрої IoT покладаються на протоколи підключення (наприклад, Wi-Fi, Bluetooth, Zigbee) і стандарти сумісності для забезпечення безперебійного зв’язку та інтеграції в екосистеми IoT;
- збір і аналіз даних - пристрої Інтернету речей генерують величезні обсяги даних, що вимагає ефективних методів збору, зберігання та аналізу даних, щоб отримати корисну інформацію;
- проблеми з безпекою та конфіденційністю - розповсюдження пристроїв Інтернету речей викликає занепокоєння щодо безпеки даних, порушень

конфіденційності та вразливості, що вимагає жорстких заходів безпеки та правил конфіденційності;

- масштабованість і керування - управління великомасштабним розгортанням пристроїв IoT створює проблеми, пов’язані з керуванням пристроями, оновленнями мікропрограм, мережевою інфраструктурою та масштабованістю платформ IoT;

- прийняття користувачами та зручність використання: успіх пристроїв IoT залежить від прийняття користувачами та простоти використання - зручні інтерфейси, інтуїтивно зрозуміла взаємодія та чіткі ціннісні пропозиції мають вирішальне значення для широкого впровадження.

1.3. Огляд існуючих мікроконтролерів для збору та передачі телеметрії

Мікроконтролери пропонують низку функцій, можливостей продуктивності та варіантів підключення, придатних для різноманітних програм IoT.

Для огляду та аналізу використано вибірку популярних мікроконтролерів, на ринку доступно багато інших варіантів, кожен зі своїми сильними сторонами та конкретними випадками використання. В таблиці 1.1 наведено зведені результати порівняння існуючих мікроконтролерів для збору та передачі телеметрії.

Таблиця 1.1 – Зведені результати порівняння існуючих мікроконтролерів для збору та передачі телеметрії

№ п/п	Назва мікроконтролеру	Основні характеристики	Особливості
1.	Arduino Uno	Процесор: ATmega328P Тактова частота: 16 МГц Флеш-пам'ять: 32 КБ SRAM: 2 Кб Підключення: USB, UART, I2C, SPI	Платформа з відкритим кодом, зручна для початківців, широка підтримка спільноти, багата екосистема щитів і бібліотек.

Продовження таблиці 1.1 – Зведені результати порівняння існуючих мікроконтролерів для збору та передачі телеметрії

№ п/п	Назва мікроконтролеру	Основні характеристики	Особливості
2.	Raspberry Pi Pico	Процесор: RP2040 Тактова частота: 133 МГц Флеш-пам'ять: 2 МБ SRAM: 264 КБ Підключення: USB, UART, I2C, SPI	Доступний, потужний, підтримує програмування MicroPython, C/C++, програмований ввід/вивід, універсальний для ряду додатків IoT.
3.	ESP32	Процесор: Xtensa Dual-Core 32-bit LX6 Тактова частота: до 240 МГц Флеш-пам'ять: 4 Мб SRAM: 520 Кб Підключення: Wi-Fi, Bluetooth, UART, I2C, SPI	Двоядерний, вбудований Wi-Fi і Bluetooth, великий обсяг пам'яті, широка підтримка периферійних пристроїв, широко використовується для проектів IoT, сумісний з Arduino IDE.
4.	Серія STM32 (наприклад, STM32F103C8)	Процесор: ARM Cortex-M3 Тактова частота: до 72 МГц Флеш-пам'ять: 64 КБ - 1 МБ SRAM: 20 КБ - 128 КБ Підключення: USB, UART, I2C, SPI	Висока продуктивність, багатий набір периферійних пристроїв, широкі інструменти розробки програмного забезпечення, широкий діапазон пам’яті та опцій пакетів, придатних для різноманітних додатків IoT.

Продовження таблиці 1.1 – Зведені результати порівняння існуючих мікроконтролерів для збору та передачі телеметрії

№ п/п	Назва мікроконтролеру	Основні характеристики	Особливості
5.	Particle Photon	Процесор: STM32F205RGY6 Тактова частота: 120 МГц Флеш-пам'ять: 1 Мб SRAM: 128 Кб Підключення: Wi-Fi, UART, I2C, SPI	Підключення до хмари, вбудований Wi-Fi, оновлення прошивки через OTA, проста інтеграція з хмарною платформою Particle, підходить для прототипів Інтернету речей і масштабованих розгортань.
6.	Arduino MKR GSM 1400	Процесор: SAMD21 Cortex-M0+ Тактова частота: 32,768 кГц (RTC), 48 МГц (ЦП) Флеш-пам'ять: 256 КБ SRAM: 32 Кб Підключення: GSM, UART, SPI, I2C	Підключення GSM, вбудований слот для SIM-карти, низьке енергоспоживання, підходить для додатків IoT, які потребують стільникового зв’язку, сумісний з екосистемою Arduino

Arduino MKR GSM 1400 — це мікроконтролер, орієнтований на IoT, який пропонує вбудоване підключення GSM, що забезпечує бездоганну інтеграцію зі стільниковими мережами для віддаленого зв’язку та передачі даних. Завдяки низькому енергоспоживанню та сумісності з бібліотеками та інструментами Arduino, він забезпечує зручне рішення для IoT-проектів, які потребують бездротового зв’язку через мережі GSM.

1.4. Аналоги існуючих додатків та систем для трекінгу світла в будинку

На ринку України є великій вибір пристроїв розумного будинку та приладів побуту які мають зв’язок з мережею інтернет та мобільний застосунок, використовуючи який можна побачити стан приладу або не отримати його стан і таким чином так або інакше дізнатись стан світла. Також є спеціалізовані прилади за слідкуванням стану мережі її навантаженням, витратами та станом.

Нижче представлений огляд на популярні додатки, які використовуються для трекінгу світла в Україні.

Додаток Світло - доступний на ios/android платформах, має гарний дизайн, є можливість додавати різні місця для перевірка стану світла та перевіряти графік відключень згідно з ДТЕК. Приклади екранних форм додатку Світло наведено на рис.2.1.

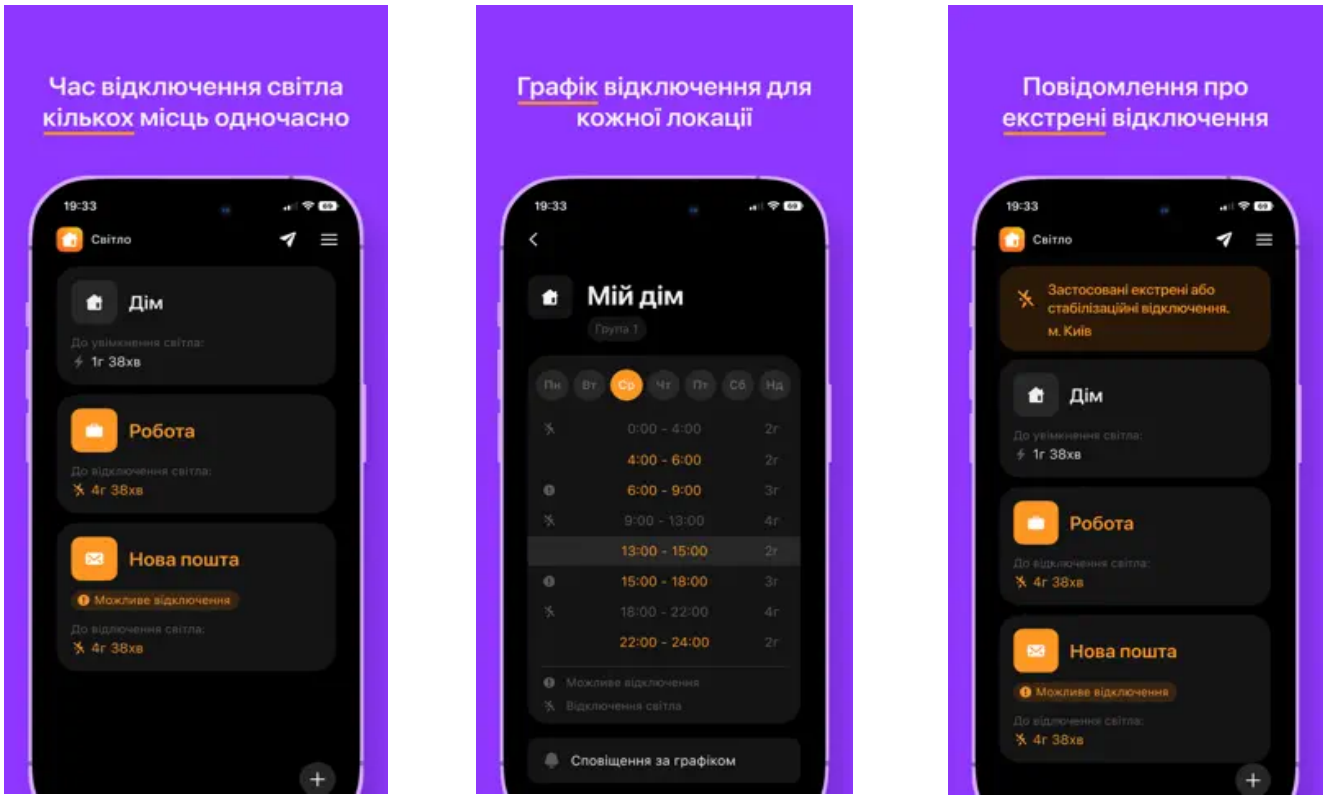


Рисунок1.2 – Інтерфейс додатку Світло.

Недоліками виступає відсутність даних в режимі реального часу, додаток працює згідно графіку відключень від ДТЕК та має низку обмежень по областям де ці графікі доступні на разі поки що доступні регіони:

- м. Київ;
- Київська область;
- Львівська область;
- Дніпропетровська область;
- Одеська область;
- Харківська область;
- Миколаївська область;
- Чернігівська область;
- Тернопільська область.

Телеграм-бот SvetaTyDeBot має наступні функції:

- перевірка наявності світла відбувається через ping (ICMP) чи HTTP;
- це SAAS, що підтримує багато локацій та багато ботів;
- основна ідея – легкий запуск нових локацій/ботів з мінімумом технічних складнощів;
- підтримуються функції поточного стану, оповіщення про зміну стану та перегляд статистики [11].

Приклади екранних форм телеграм-боту SvetaTyDeBot наведено на рис.1.3.

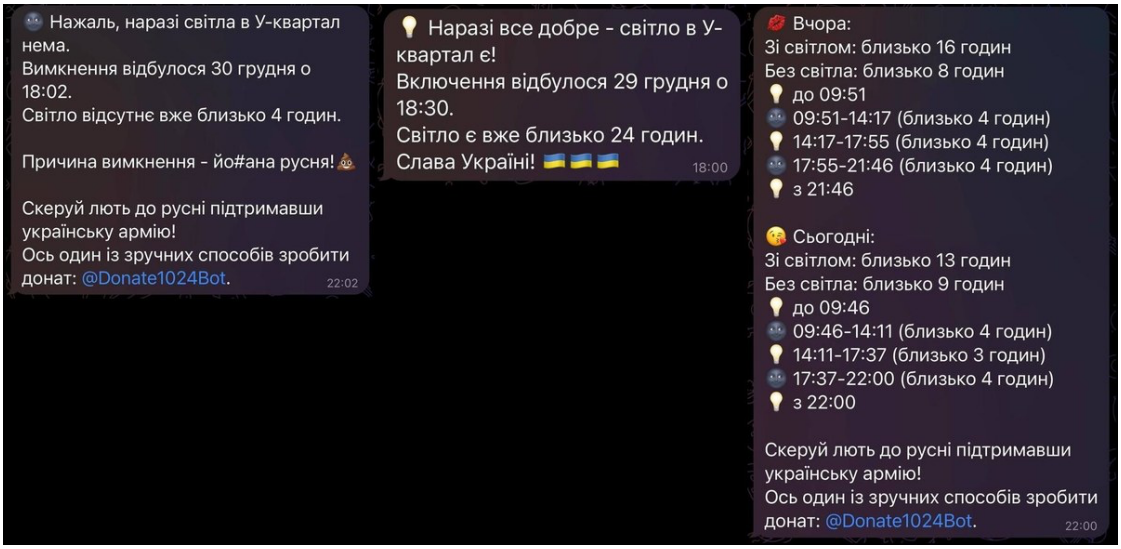


Рисунок 1.3 – Інтерфейс телеграм-боту SvetaTyDeBot

Недоліком телеграм-боту SvetaTyDeBot є то, що прилад який буде виступати для пінгу має працювати від мережі що унеможливює варіант підключення резервного живлення.

Платформа Schneider electric PowerTag sensor + Acti9 PowerTag Link включає програмне та апаратне забезпечення.

PowerTag - це бездротовий датчик IoT, який можна встановити на автоматичних вимикачах або електричних панелях для моніторингу електричних параметрів. Він надає дані в реальному часі про споживання енергії, якість електроенергії та збої в електропостачанні. Завдяки інтеграції з хмарними платформами PowerTag забезпечує віддалений моніторинг і сповіщає комунальні компанії або власників будинків про відключення електроенергії. Інтерфейс хмарної платформи PowerTag представлено на рис. 1.4.

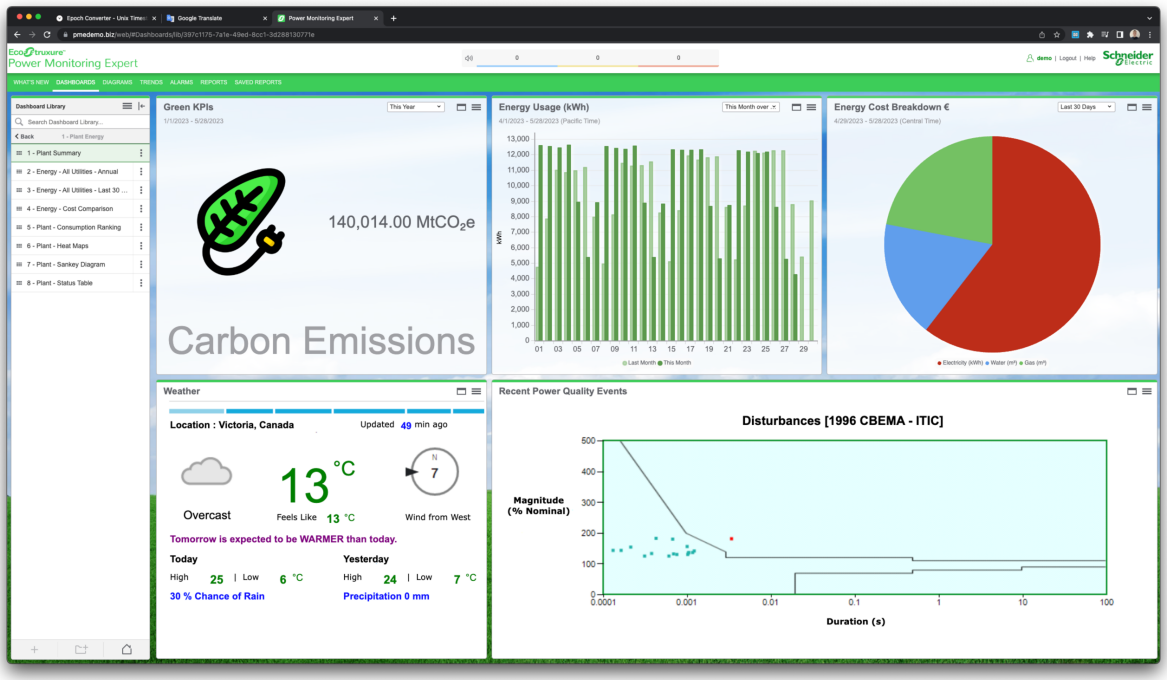


Рисунок1.4 – Інтерфейс хмарної платформи PowerTag

З недоліків платформи можливо зазначити те, що їх встановлення можливо тільки в вузлах комутації електроживлення, також висока вартість обладнання, яка ключає сенсор та коммутатор.

1.5. Обґрунтування вибору інструментів розробки системи

Продукт для трекінгу світла складається з 3 компонентів а саме:

- мікроконтроллер або ж IoT пристрій для збору та передачі даних по стану світла та самого пристрою;
- сервер для прийому та обробки даних з пристрою а також API для комунікації з мобільним додаток;
- мобільний додаток.

В якості пристрою обрано Arduino MKR GSM 1400, оскільки він має вбудований модуль GSM та має можливість працювати від акумулятора. Програмний код пристрою складається мовою C++.

Сервер обробки даних буде побудовано з використанням технології ізольованих контейнерів за допомогою Docker та Docker Compose.

В якості мови розробки вибрано Python. Він є однією з найпопулярніших мов програмування в світі, завдяки своїй простоті, елегантності та широкій спільноті розробників. Python можна використовувати для створення різноманітних веб-застосунків, включаючи веб-сайти, веб-сервери, веб-додатки та інші. Задля пришвидшення розробки сервера було обрано мікро фреймворк FastAPI. FastAPI завоював популярність серед розробників завдяки своїй продуктивності, сучасним функціям і простоті використання. Він поєднує найкращі аспекти мови Python, асинхронного програмування та веб-розробки, щоб забезпечити надійну структуру для створення високопродуктивних API.

Головною метою є забезпечення обробки великого обсягу даних, для досягнення цієї мети буде використано асинхроний підхід обробки даних та брокер повідомлень, буде створено окремий мікросервіс/контейнер, який буде обробляти повідомлення які надходять з черги. В якості брокера повідомлень обрано RabbitMQ та бібліотека Dramatiq, яка спрощує роботу з чергами в програмному коді. Також буде використано кешування даних для зменшення запитів до БД та прискоренню обробці. Для кешування обрано Redis.

В якості СУБД системи було обрано Firebase. Вона надає можливість стрімінгу даних для мобільного застосунку та швидкої розробки, оскільки дозволяє працювати з нереляційною БД.

В якості платформи для розробки мобільного застосунку обрано Android. Для розробки коду обрано мову Kotlin та функціональний підхід розробки інтерфейсів використовуючи бібліотеку Jetpack Compose.

Для розробки обрано сучасне програмне забезпечення, як для написання коду, так і для реалізації інтерфейсу додатку – PyCharm (для написання бекенду), Arduino IDE (для написання коду для IoT), Android Studio (для написання мобільного додатку).

PyCharm - одне з найпопулярніших середовищ розробки на Python від компанії JetBrains. Підтримка веб-розробки за допомогою Django є ще однією вагомою причиною вибору цього середовища.

Arduino IDE - популярний інструмент розробки для програмування мікроконтролерів. Arduino IDE - це платформа з відкритим вихідним кодом, яка надає простий і зручний інтерфейс для написання та завантаження коду на різні плати Arduino та сумісні пристрої [2].

Android Studio - це IDE (інтегроване середовище розробки), створене компанією Google для розробки додатків для Android. Але на відміну від інших IDE, Android Studio розроблена спеціально для Android і містить набір функцій та інструментів, призначених для цієї мобільної ОС. Загалом, будучи офіційним IDE для розробки додатків для Android, Android Studio покликана полегшити роботу розробників[1].

Продукт створено за допомогою платформи Arduino мовою C++. Для розробки архітектури бази даних обрано Google Firebase. Для розробки архітектури серверу обробки даних обрано Python та фреймворк FastApi, з використанням брокеру повідомлень RabbitMQ для обробки запитів асінхронно.

В якості засобів розробки дизайну графічного інтерфейсу мобільного додатку обрано фреймворк Figma. Додаток базується на платформі Android за допомогою мови Kotlin та фреймворку Jetpack Compose[4].

2 ПРОЕКТУВАННЯ ТА РОЗРОБКА EMBEDDED ДОДАТКУ ДЛЯ ТРЕКІНГУ СВІТЛА

2.1 Аналіз вимог до embedded продукту для трекінгу світла в будинку та формування технічного завдання до проекту

Розробка програмного забезпечення - це процес, який використовують програмісти для створення комп'ютерних програм. Цей процес, також відомий як життєвий цикл розробки програмного забезпечення (SDLC), включає кілька етапів, які забезпечують метод створення продуктів, що відповідають технічним специфікаціям і вимогам користувачів.

SDLC є міжнародним стандартом, який компанії-розробники програмного забезпечення можуть використовувати для створення та вдосконалення своїх комп'ютерних програм. Він пропонує визначену структуру для команд розробників, якої вони повинні дотримуватися при проектуванні, створенні та підтримці високоякісного програмного забезпечення. Метою процесу розробки програмного забезпечення є створення ефективних продуктів в рамках визначеного бюджету та строків[8]. Основні етапи процесу розробки програмного забезпечення включають наступне:

- визначення потреб;
- аналіз вимог;
- проектування;
- розробка та впровадження;
- тестування;
- розгортання та підтримка продукту.

В попередньому розділі був проведений аналіз ринку embedded продуктів для трекінгу світла. В цьому розділі будуть визначені вимоги до продукту, виконано його проектування та розробка.

Згідно проведеного аналізу, можна визначити наступні вимоги до додатку:

- реєстрація нового користувача;

- можливість змінювати налаштування оповіщень;
- додавання пристрою трекінгу світла в свій аккаунт;
- видалення пристрою трекінгу світла з аккаунта;
- перегляд інформації наданої пристроєм трекінга світла;
- додаток повинен мати сторінку з списком усіх доданих пристроїв аккаунта;
- сервер обробки данних пристрою повинен мати велику продуктивність та відмовостійкість.

2.2 Моделювання об’єкту проектування

З точки зору розробки продукту, дизайн та моделювання програмного забезпечення є важливим, оскільки дизайн є основою для всіх інших видів діяльності з побудови програмного забезпечення. Дизайн програмного забезпечення дозволяють інженерам-програмістам створювати моделі, які представляють структуру та поведінку програмної системи. За допомогою цих моделей визначаються основні компоненти та їхній взаємозв'язок для вирішення проблеми. Характеристики якісного коду, такі як модульність, зв'язність і сполучуваність, зароджуються на етапі проектування. Для складних завдань при розробці програмного забезпечення використовуються абстракції та інкапсуляція як засоби для забезпечення систематичного системного підходу до вирішення проблем. Крім того, програмні проекти можна використовувати багаторазово, тому вони можуть бути застосовані до різних проектів для надання готових рішень спільних проблем.

Проектування програмного забезпечення також надає засоби для оцінки та включення атрибутів якості, необхідних для програмних систем. Таким чином, такі питання, як продуктивність, зручність використання, портативність і безпека можуть бути вирішені на ранній стадії проекту розробки. Ці переваги переносяться на всі інші наступні етапи життєвого циклу розробки програмного забезпечення та мають безпосередній вплив на фазу впровадження, тестування та супроводу.

Для дизайну та моделювання продукту була використана діаграма варіантів використання. Діаграма варіантів використання - використовується для представлення динамічної поведінки системи. Вона інкапсулює функціональність системи шляхом включення варіантів використання, користувачів та їхніх взаємозв'язків. Вона моделює завдання, сервіси та функції, необхідні системі/підсистемі додатку. Вона відображає високорівневу функціональність системи, а також показує, як користувач працює з системою[10].

Основною метою використання діаграми – є відображення динамічного аспекту системи, основних вимог до системи та як користувач може взаємодіяти з системою.

Варіанти використання продукту визначаються згідно раніше визначених вимог до нього і представлені на рис.2.1. Для створення діаграми був використаний онлайн сервіс draw.io [12].

Рисунок 2.1 – Діаграма варіантів використання

В діаграмі варіантів показано основні взаємодії користувача за додатком. Користувач може виконувати такі основні функції, як реєстрацію, має можливість змінювати налаштування оповіщень, додавання пристрою трекінгу світла в свій акаунт, видалення пристрою трекінгу світла з акаунту та перегляд інформації наданої пристроєм трекінга світла.

Додаток повинен мати сторінку з списком усіх доданих пристроїв акаунту. Сервер обробки даних пристрою повинен мати велику продуктивність та відмовостійкість при виконанні задач користувача.

Вибір архітектури додатку також є важливою частиною життєвого циклу розробки програмного забезпечення.

Архітектура програмного забезпечення розкриває структуру системи, приховуючи деталі реалізації. Архітектура також фокусується на тому, як елементи та компоненти системи взаємодіють один з одним. Дизайн програмного забезпечення заглиблюється в деталі реалізації системи. Проблеми дизайну включають вибір структур даних та алгоритмів, або деталі реалізації окремих компонентів.

Питання архітектури та дизайну часто перетинаються. Замість того, щоб використовувати жорсткі і швидкі правила для розмежування архітектури і дизайну, має сенс об'єднати їх. У деяких випадках рішення є більш архітектурними за своєю природою. В інших випадках рішення значною мірою зосереджені на дизайні та тому, як він допомагає реалізувати архітектуру.

З точки зору інженерії програмного забезпечення, різні архітектурні проекти можуть допомогти декомпонувати програмне забезпечення та визначити основні структурні компоненти системи, визначити інтерфейси між компонентами, зіставити вимоги до них, оцінити проблеми паралелізму та надати загальне уявлення про проектне рішення.

Основною перевагою архітектурних проектів є їхня здатність оцінювати високорівневі інтереси зацікавлених сторін, які мають справу здебільшого з нефункціональними вимогами (наприклад, продуктивність, зручність використання, безпека). Для цих цілей архітектурні проекти слугують важливим

засобом комунікації, міркувань та інструментів аналізу, які підтримують розвиток і зростання систем.

Загалом виділяють три основних типи архітектури додатків:

- моноліт;
- мікросервіси;
- серверлес.

Для вибору архітектури додатку були проаналізовані основні типи архітектури [5], їх переваги та недоліки, результати аналізу наведені в таблиці 2.1.

Таблиця 2.1 – Зведені результати порівняння популярних типів архітектури програмного забезпечення

Визначення	Переваги	Недоліки
Тип архітектури: моноліт		
Традиційна модель програмного забезпечення, яка будується як єдиний блок, самодостатній і незалежний від інших додатків. Щоб внести зміни в такий додаток, потрібно оновити весь стек, отримавши доступ до бази коду, а також створити і розгорнути оновлену версію інтерфейсу на стороні сервісу.	Легке розгортання; Розробка - коли додаток побудовано з однією базою коду, його легше розробляти; Продуктивність; Спрощене тестування; Легке налагодження.	Повільніша швидкість розробки; Масштабованість; Надійність; Бар'єр для впровадження технологій; Відсутність гнучкості; Розгортання.

Продовження таблиці 2.1 – Зведені результати порівняння популярних типів архітектури програмного забезпечення

Визначення	Переваги	Недоліки
Тип архітектури: мікросервіс		
Архітектурний метод, який спирається на низку незалежних сервісів, що розгортаються незалежно один від одного. Ці сервіси мають власну бізнес-логіку та базу даних з конкретною метою. Оновлення, тестування, розгортання та масштабування відбуваються всередині кожного сервісу.	Надає гнучкості для роботи в команді та при масштабуванні проекту; Безперервне та незалежне розгортання; Високий рівень підтримки та тестування; Висока надійність.	Складність розробки; Експоненціальне зростання витрат на інфраструктуру; Додаткові організаційні витрати; Проблеми з налагодженням; Відсутність стандартизації та чіткої відповідальності.
Тип архітектури: безсерверна		
Безсерверна архітектура означає, що хмарний провайдер - наприклад, Google, Amazon Web Services (AWS) або Microsoft Azure - надає та володіє безсерверною платформою та внутрішньою інфраструктурою, що має свої переваги та недоліки. Безсерверну архітектуру іноді також називають функцією як послугою	Відсутність власної внутрішньої інфраструктури, на якій працюють ваші додатки; Вартість також є однією з переваг безсерверної	Через природу безсерверної архітектури додатки, що працюють на ній, майже неможливо контролювати;

(FaaS) або бездержавними, безсерверними обчисленнями.	архітектури (хоча це може бути і недоліком), оскільки іноді вона може бути економічно вигідним варіантом.	Безпека - ще один потенційний недолік безсерверної архітектури. Обмеженість ресурсів;

Проаналізувавши вимоги до продукту і існуючі види клієнт-серверної архітектури додатків для продукту доцільно буде обрати монолітну архітектуру. Діаграма розгортання додатку у відповідності до обраної архітектури наведена на рис.2.2.

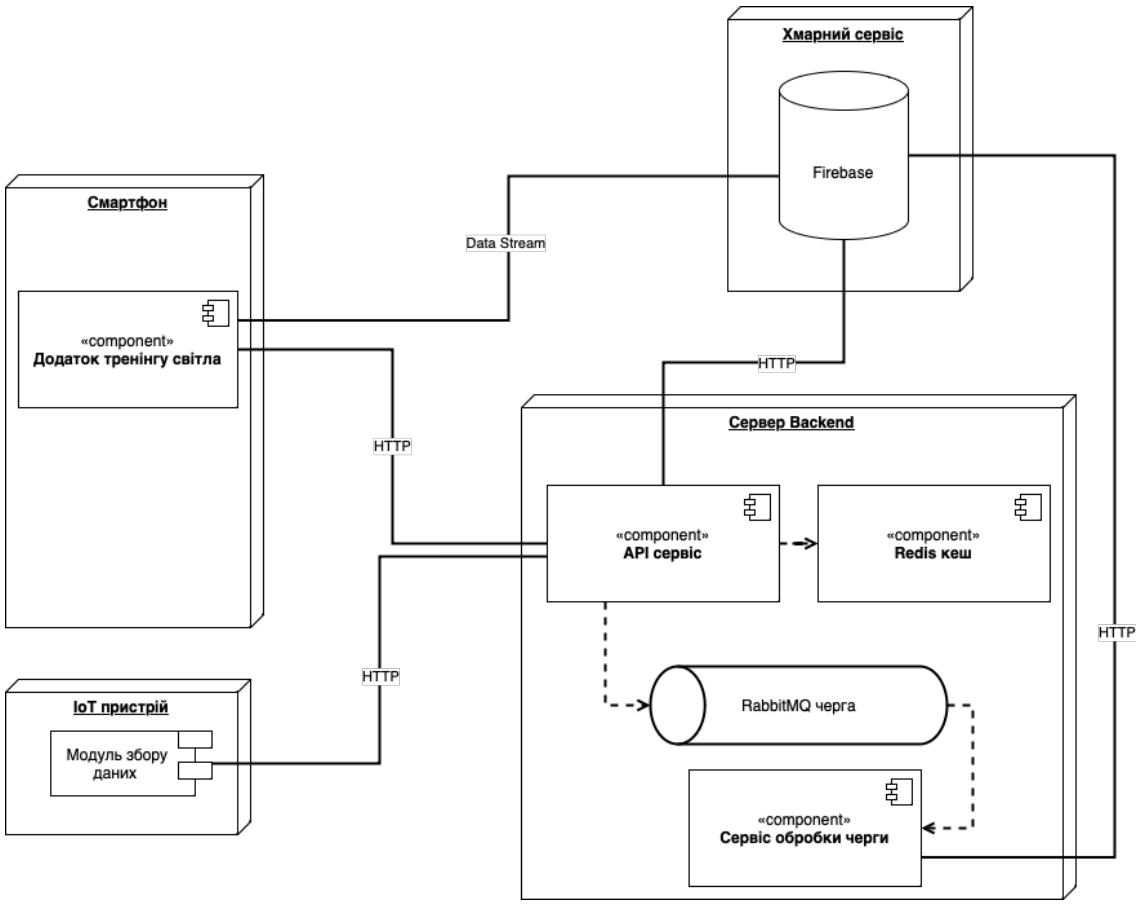


Рисунок 2.2 – Діаграма розгортання

Згідно діаграми на рис.2.2., всі процеси виконуються за допомогою декількох сервісів та бази даних, сервіси працюють незалежно один від одного, що надає надійність продукту.

Рисунок 2.3 відображає діаграму послідовності головних процесів.

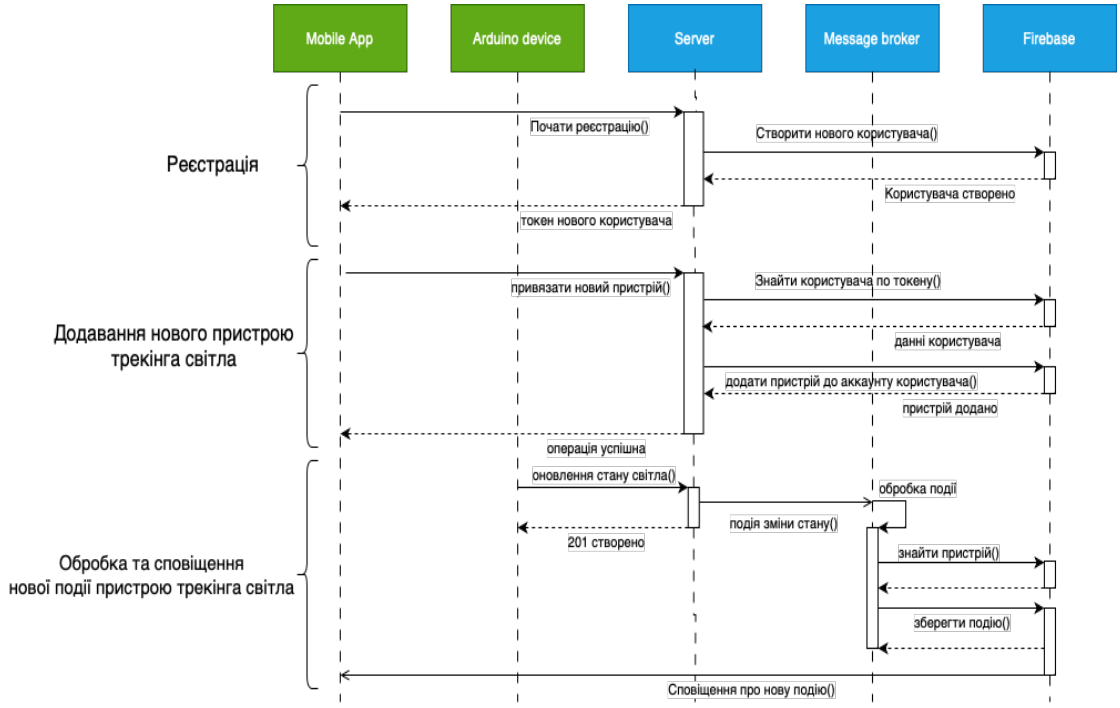


Рисунок 2.3 – Діаграма послідовності головних процесів

При реєстрації клієнт подає запит на реєстрацію на сервер, сервер запит на БД для створення нового користувача, після чого БД надсилає відповідь на сервер, а сервер на клієнта.

При обробці та сповіщенні нової події пристрою трекінгу можна побачити задіяними всі сервіси та БД.

Важливим елементом дизайну проекту також є архітектура БД продукту. Перед розробкою БД, необхідно визначити її структуру. Для цього була обрана ER-діаграма, яка є структурним дизайном бази даних. Вона діє як каркас, створений за допомогою спеціальних символів з метою визначення взаємозв'язку між сутностями бази даних. ER-діаграма створюється на основі трьох основних компонентів: сутності, атрибути та зв'язки.

Основними цілями використання ER-діаграми для розробки БД є:

- ER-діаграма допомагає концептуалізувати базу даних і дозволяє дізнатися, які поля потрібно вбудувати для конкретної сутності;
- ER-діаграма дає краще розуміння інформації, яка буде зберігатися в базі даних;
- вона зменшує складність і дозволяє розробникам баз даних швидко створювати бази даних;
- допомагає описувати елементи за допомогою моделей "сутність-зв'язок;
- дозволяє користувачам отримати попередній перегляд логічної структури бази даних.

Таким чином ER-діаграма є чудовим варіантом для описання БД проекту. Створена діаграма описує структуру БД, таблиці, поля, та зв’язки. Це надає можливість краще зрозуміти принцип роботи бази даних, а також те, як передані показники зберігаються всередині.

На рисунку 2.4 наведена схема бази даних проекту.

Сутність користувача User зберігає частину даних користувача та грає роль зв'язувальної ланки між 3 сутностями:

– TelegramSettings – дані потрібні для відправки повідомлень в месенджер телеграм;

– UserSettings – зберігає в собі налаштування користувача;

– Device – дані по пристроям, які користувач додав в додатку в свій профайл.

А також сутність події Event зберігає дані, зібрані з IoT пристрою для подальшого їх відображення в додатку.

Атрибути сутності User:

– id – унікальний ідентіфікатор сутності User;

– email – електрона адреса користувача;

– user_name – ім’я користувача для привітання та інших дій;

– token – токен користувача для авторизації на сервері;

– fcm_token – токен користувача для сервіса пуш нотифікацій;

– created_at – дата створення користувача.

Атрибути сутності UserSettings:

– id – унікальний ідентифікатор сутності UserSettings;
– user_id – зовнішній ключ для зв’язку з користувачем User;
– photo_url – посилання на фото користувача для профілю;
– send_push – налаштування відправки пуш повідомлення;
– send_tg – налаштування відправки повідомлення в телеграм;
– created_at – дата створення налаштувань;
– updated_at – дата зміни налаштувань.

Атрибути сутності TelegramSettings:

– id - унікальний ідентіфікатор сутності TelegramSettings;
– user_id - зовнішній ключ для звя’зку з користувачем User;
– telegram_token – токен для відправки повідомлення в телеграм;
– created_at - дата створення налаштувань телеграм.

Атрибути сутності Device:

– id - унікальний ідентіфікатор сутності Device;
– user_id - зовнішній ключ для звя’зку з користувачем User;
– device_number – номер пристрою для додавання в додаток;
– device_name – назва пристрою для відображення на інтерфейсі;
– device_address – адреса пристрою для відображення на інтерфейсі;
– active – пристрій активний чи видалений з додатка;
– created_at - дата створення пристрою.

Атрибути сутності Event:

– id - унікальний ідентифікатор сутності Event;
– device_id - зовнішній ключ для зв’язку з пристроєм Device;
– state – поточний стан світла;
– network_level – поточний стан рівня мобільного зв’язку;
– battery_level – поточний стан рівня заряду акумулятора;
– created_at – дата створення події.

Обраний тип архітектури дозволив декомпонувати програмне забезпечення та визначити основні структурні компоненти системи, визначити інтерфейси між компонентам та розподілити навантаження між компонентами, що дає змогу додатку мати підвищену продуктивність та відмовостійкість.

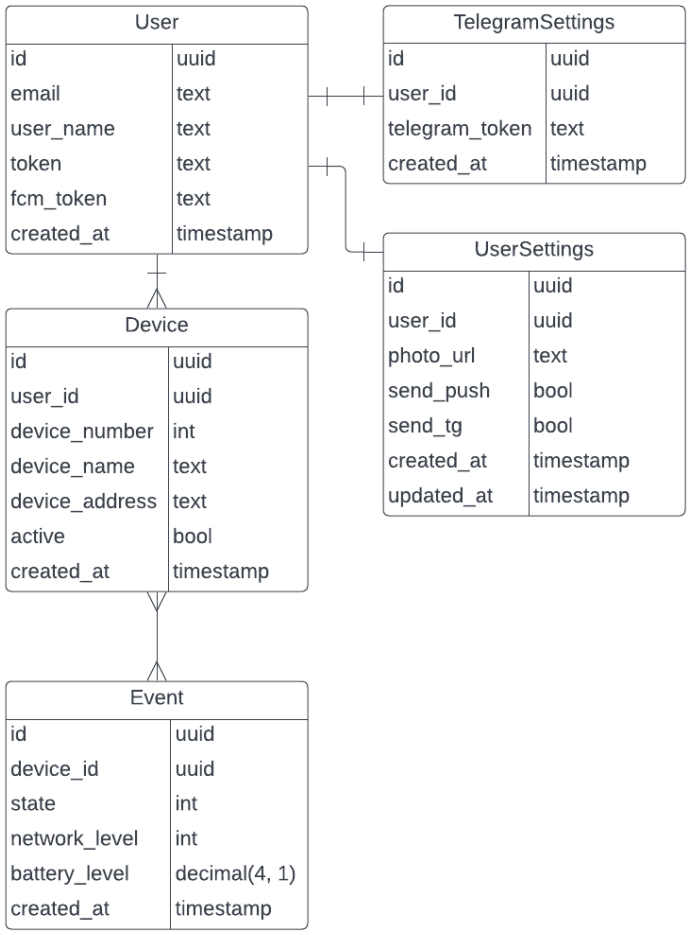


Рисунок 2.4 – ER-діаграма бази даних

2.3 Розробка embedded додатку для трекінгу світла

Важливим етапом розробки є планування та вибір методології розробки. Виділяють три основні типи планування розробки.

Підтвердження концепції (PoC) - допомагає переконатися, що технічні можливості, інструменти та ресурси, необхідні для реалізації вашої ідеї, є життєздатними. Оскільки PoC використовується для внутрішніх цілей, він

здебільшого нехтує найкращими практиками інтерфейсу, безпеки та розробки. Код зрідка використовується повторно на більш пізніх етапах розробки, тому він часто є жорстко закодованим, має імітацію API та базові елементи керування інтерфейсом користувача;

Prototype - це базовий макет продукту, який ви хочете створити. Це інтерактивний дизайн, який візуалізує, як кінцеві користувачі будуть взаємодіяти з вашим продуктом. Коли справа доходить до розробки нових та унікальних продуктів, прототипування є природним наступним кроком після етапу відкриття продукту. Процес зазвичай складається з розробки кількох макетів веб-сторінок/додатків, організованих у логічній послідовності.

MVP - це функціональна, але пуста версія продукту. Вона містить лише найнеобхідніші функціональні можливості та основні функції, необхідні для випуску на ринок. На відміну від прототипу, який бачить лише команда розробників та деякі зовнішні зацікавлені сторони, MVP має реальних користувачів. Таким чином, він забезпечує надійну основу для вивчення потреб клієнтів і розуміння того, які саме функції має включати готове рішення. Він спрямований на вирішення найбільших проблем користувачів і дає можливість продукту закріпитися на ринку.

Проаналізувавши існуючі типи планування було обрано MVP для розробки продукту. Для початку були виділені основні та найнеобхідніші функції продукту для реалізації. Були виділені та реалізовані вимоги для мінімально життєвого продукту, також вимоги до БД і додаткові вимоги для реалізації на майбутнє проекту.

Для методології розробки було обрано Kanban - одна з найпопулярніших систем гнучкого управління проектами. Методологія Kanban використовує візуальний підхід до управління проектами. З урахуванням переваг Kanban, було вирішено його викристовувати на проекті. Для створення Kanban-дошок було обрано додаток Trello.

Він має дуже простий інтерфейс і всі необхідні функції, такі як постановка завдання, пріоритет задачі, та постановка термінів, коли задача має бути виконана. Серед переваг Trello також можливо додати те, що Trello доступний на багатьох

пристроях, включаючи настільні комп'ютери, смартфони та планшети, через веб-браузери та спеціальні мобільні програми. Ця крос-платформна доступність гарантує, що користувачі можуть отримати доступ до своїх дощок і завдань з будь-якого місця та в будь-який час, має автоматизацію рутинних справ та різні види перегляду поточного статусу дошки такі як табличний перегляд, календарний перегляд та статистику. На рис. 2.5 наведено Kanban-дошка проєкту, рис.2.6 відображає деталізацію етапів розробки.

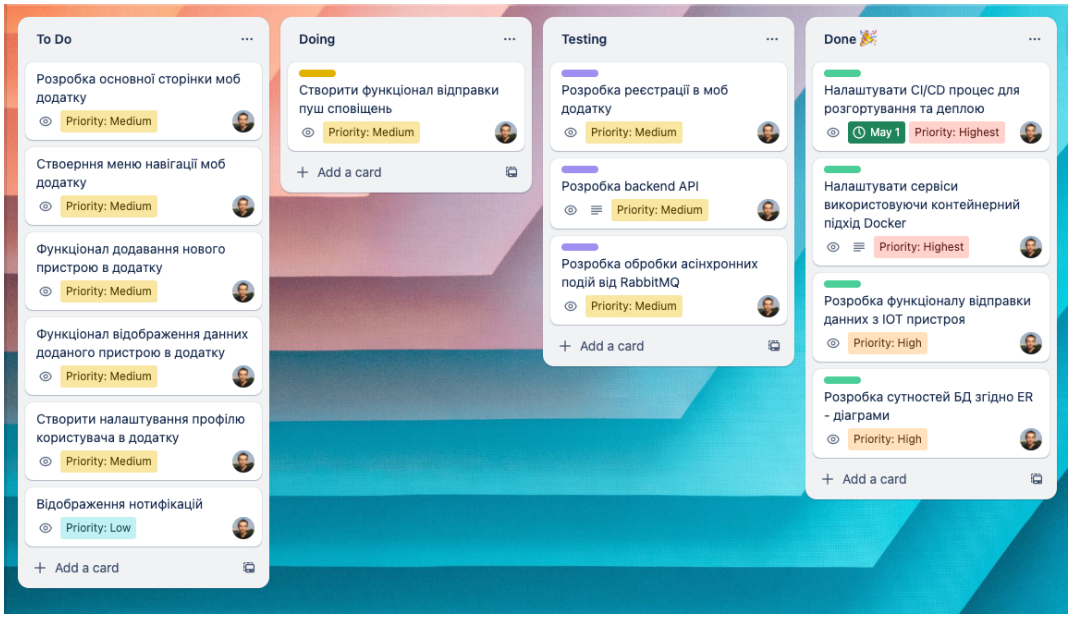


Рисунок 2.5 – Kanban дошка проекту

Card	List	Labels	Members	Due date
Розробка основної сторінки моб додатку	To Do			
Створення меню навігації моб додатку	To Do			
Функціонал додавання нового пристрою в додатку	To Do			
Функціонал відображення данних доданого пристрою в додат	To Do			
Створити налаштування профілю користувача в додатку	To Do			
Відображення нотифікацій	To Do			
Створити функціонал відправки пуш сповіщень	Doing			
Розробка реєстрації в моб додатку	Testing			
Розробка backend API	Testing			
Розробка обробки асінхронних подій від RabbitMQ	Testing			
Налаштувати CI/CD процес для розгортування та деплою	Done			May 1
Налаштувати сервіси використовуючи контейнерний підхід D	Done			May 3
Розробка функціоналу відправки данних з IOT пристроя	Done			May 7
Розробка сутностей БД згідно ER - діаграми	Done			May 10

Рисунок 2.6. – Табличний перегляд дошки

Поточний перегляд задач по статусам наведено на рис.2.7.

Рисунок 2.7. – Поточний перегляд задач по статусам

Під час розробки продукту використовувався GitHub для збереження коду. GitHub побудовано на Git, потужній розподіленій системі контролю версій. Це дозволяє відстежувати зміни у вашій кодовій базі, керувати різними версіями вашого проекту та безперешкодно співпрацювати з іншими. Git надає такі функції, як розгалуження, злиття та вирішення конфліктів, які є ключовими для командної співпраці та керування кодом.

GitHub служить централізованим сховищем вихідного коду вашого проекту. Він забезпечує простий спосіб зберігання, керування та обміну кодом з іншими розробниками. Один або кілька членів команди можуть співпрацювати над одним проектом, клонуючи репозиторій, вносячи зміни та пропонуючи модифікації за допомогою запитів на отримання. GitHub також дозволяє перевіряти код і обговорювати його, сприяючи ефективній співпраці та забезпечуючи якість коду.

GitHub легко інтегрується з різними інструментами CI/CD, такими як GitHub Actions, Jenkins або Travis CI. Ці інтеграції дозволяють автоматизувати процеси тестування, створення та розгортання. Налаштувавши робочі процеси, ви можете автоматизувати такі завдання, як запуск тестів, створення артефактів і розгортання

програми, оптимізуючи процеси розробки та випуску. В якості CI/CD було обрано Woodpecker CI (рис.2.8). Woodpecker CI має на меті забезпечити просте та зручне налаштування. Він пропонує веб-інтерфейс для налаштування конвеєрів CI/CD, що дозволяє визначати етапи збірки та розгортання, визначати середовища та налаштовувати тригери для робочих процесів, також він має інтеграцію з GitHub та є абсолютно безкоштовним. Оскільки вихідний код Woodpecker є відкритим.

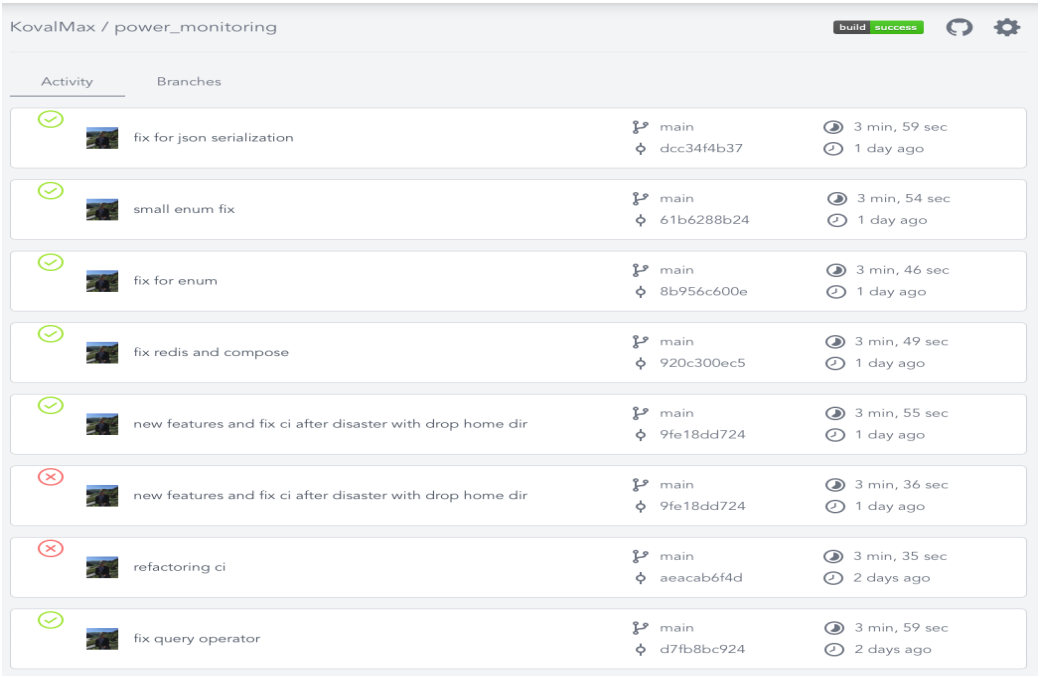


Рисунок 2.8 – Woodpecker CI/CD

2.4. Імплементація функціоналу передачі даних embedded пристрою

В цьому пункті розглянуто реалізацію функціоналу передачі даних з embedded пристрою. Збір даних пристроєм робиться за допомогою бібліотек роботи з різними модулями пристрою, в тому числі модуль стільникового зв’язку, модуль що контролює електропостачання пристрою та модуль заряду акумулятора. Принцип роботи полягає в наступному: кожен цикл роботи процесору виконується низка завдань - таких як перевірка чи є мобільний зв’язок наразі та його підключення, збір даних по стану джерела електропостачання, рівень мобільного

зв’язку та рівень заряду акумулятора, та передача цих даних кожні 15 секунд на сервер. Алогритм роботи відображено на блок схемі на рис.3.1.

Функціонал було імплементовано за допомогою платформи Arduino та мовою C++.

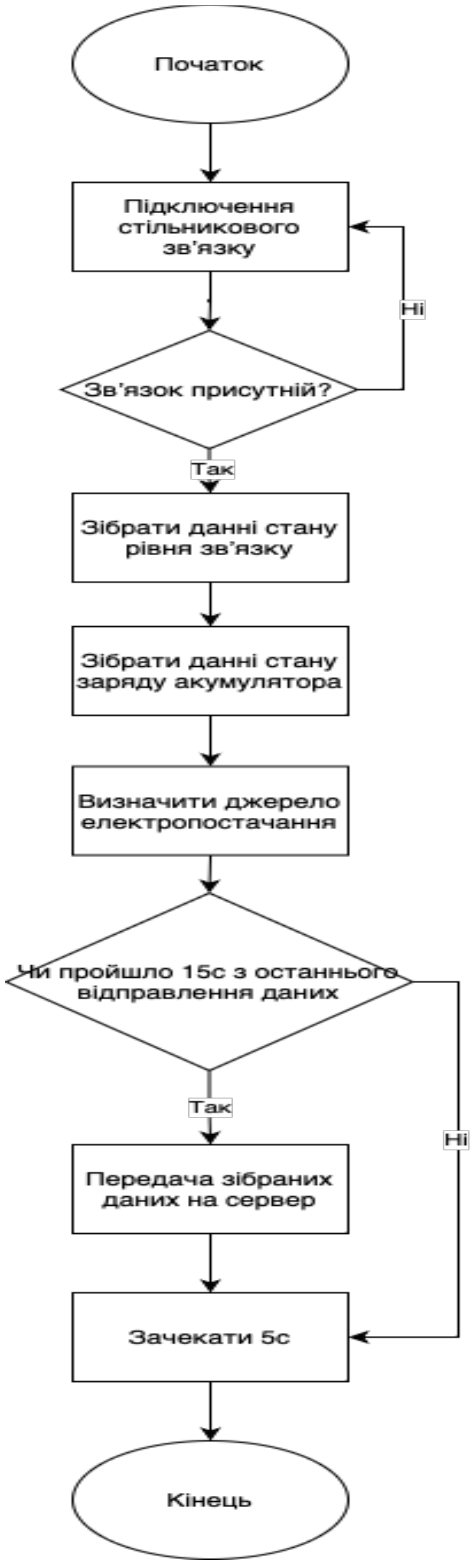


Рисунок 2.9. – Блок-схема алгоритму роботи пристрою

2.5. Функціонал обробки даних, переданих embedded пристроєм

Функціонал обробки даних складається з кількох етапів. Перший етап - це валідація отриманих даних, перевірка чи було отримано всі потрібні атрибути для успішної обробки, виконується швидка перевірка на заздалегідь визначений User-Agent в запросі і, якщо всі перевірки пройдено, то дані поміщуються в чергу брокера повідомлень RabbitMQ та відбувається відповідь кодом 201 «запит створено». Таким чином, пристрій не задіяний весь цей час, поки буде проходити обробка даних та всі потрібні дії. Після того як повідомлення потрапляє в чергу, вільний в цей час worker забирає повідомлення з черги та починає робити всі потрібні кроки: перевірка чи зареєстровний пристрій в системі, відбувається спроба отримати в кеш систему, у випадку, якщо данні відсутні, формується запит до БД, після цього дані поміщуються в кеш. Практика кешування даних зберігає велику кількість часу та знімає певну частину навантаження з БД. Після цього йде збереження даних в сховище. У випадку, якщо стан світла з останнього разу змінився, відправляється повідомлення тримачу пристрою згідно налаштувань сповіщення.

3 ТЕСТУВАННЯ EMBEDDED ПРОДУКТУ ДЛЯ ТРЕКІНГУ СВІТЛА В БУДИНКУ

3.1 Опис функціонування додатку

В цьому пункті розглянуто основна функціональність додатку та його взаємодія з користувачем, що допоможе його тестуванню та подальшій реалізації.

На головному екрані мобільного додатку відображено вікно з авторизацією (рис.3.1). В додатку можна авторизуватись за допомогою сервісу Google, що надає додаткову перевірку на реальність користувача за допомогою цього сервісу, тому що перед авторизацією потрібно, щоб користувач був попередньо зареєстрований на сервісі Google.

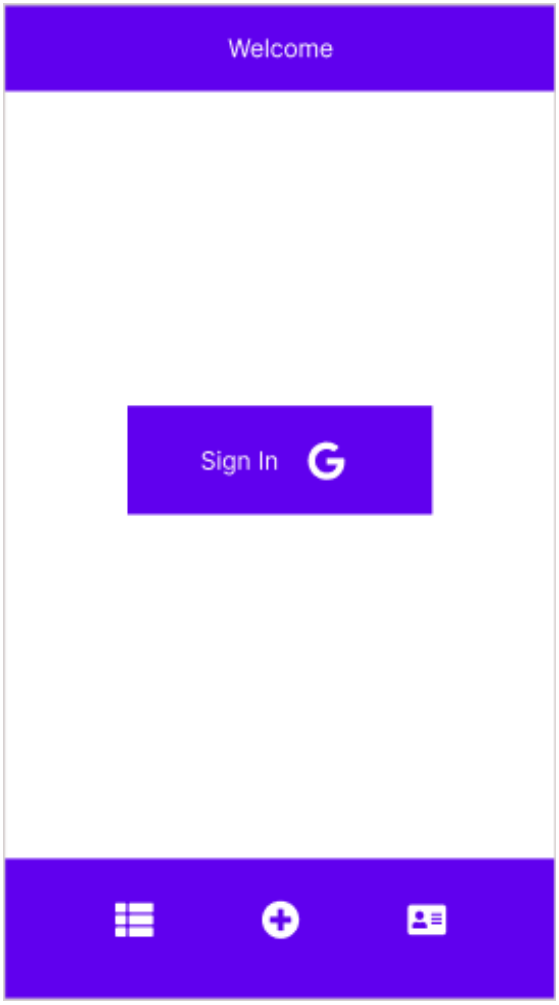


Рисунок 3.1 – Головний екран додатку з формою авторизації

Після авторизації користувачу будуть доступні наступні функції додатку – додавання, перегляд існуючих девайсів та перегляд профілю.

Для роботи з девайсами користувач може їх додавати у мобільному додатку, а саме – ID девайсу, локацію та ім’я девайсів після чого буде можливість їх використання та перегляду.

Також у цьому меню користувач може переглядати девайси та повну інформацію про них – включене чи відключене світло, ім’я девайсу, його локація, статус включення, рівень батареї, рівень мережі, та проміжок часу між відключеннями. Всі цю інформацію можна побачити на рисунку 3.2.

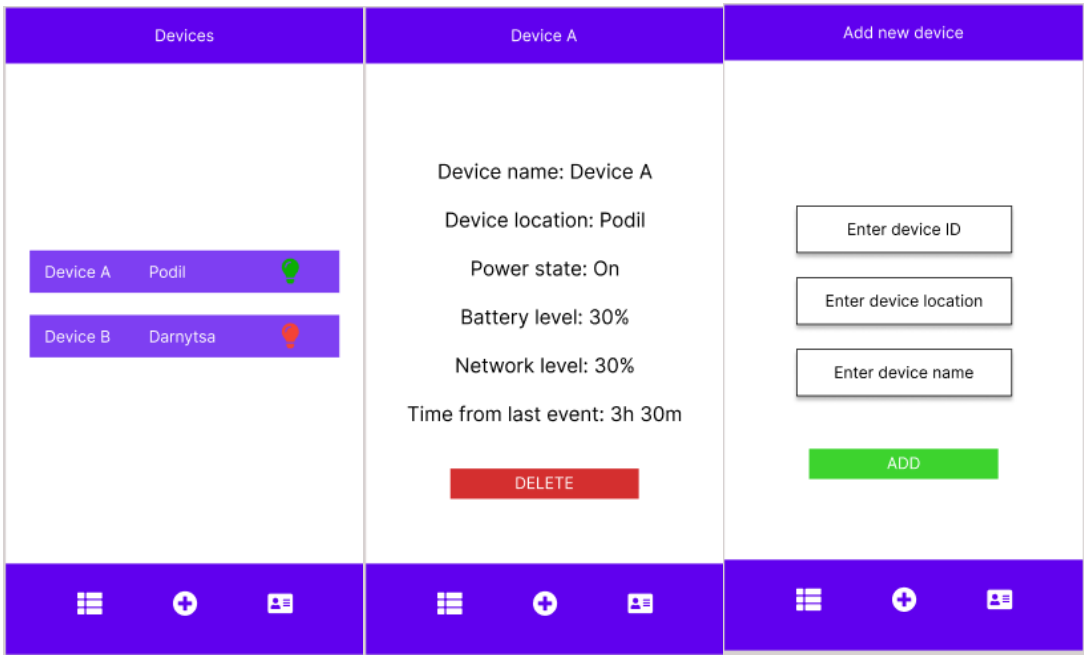


Рисунок 3.2 – Екрани меню для роботи з девайсами

Ще одна функціональність, яка доступна клієнту у додатку – це перегляд та налаштування профілю. В цьому меню користувач може переглянути наступну інформацію про профіль – повну адресу, E-mail, за допомогою якої була авторизація у додатку, ім’я користувача.

Також у даному меню доступні наступні налаштування – вибір способу отримання повідомлень про статус світла. При виборі користувачем налаштування Push, повідомлення будуть приходити у додатку. При виборі чек-боксу Telegram користувач буде отримувати повідомлення у обраному месенджері.

Меню профілю представлено на рис. 3.3.

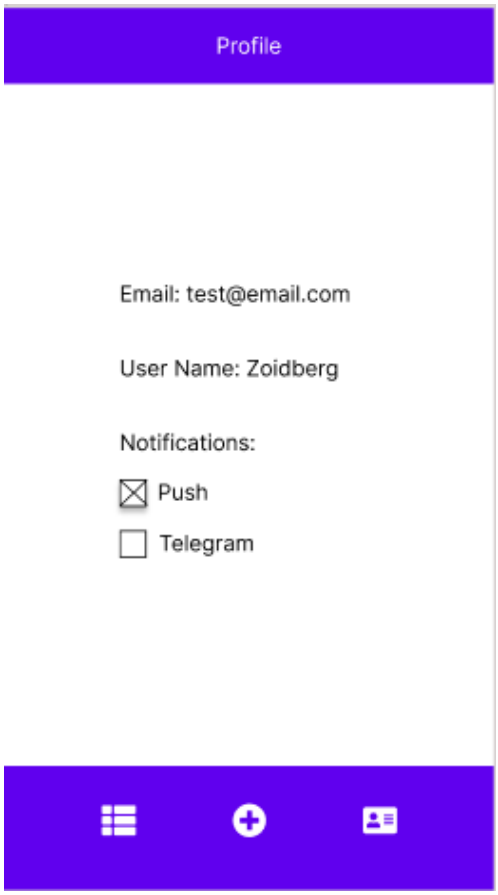


Рисунок 3.3 – Екран меню профілю

Описані функціональні можливості додатку дозволяють провести його тестування та презентацію кінцевим користувачам.

3.2 Розробка тест-кейсів та аналіз результатів тестування

Однією із запорук успіху тестування є визначення тестової стратегії. Для тестування додатку було визначену стратегію тестування, згідно якої основними задачами тестування є досягнення цілей проекту, перевірка реалізованих вимог і досягнення стабільності додатку.

Під час реалізації додатку на ранніх етапах було проведено тестування API, а саме інтеграції компонентів між додатком та сервером.

Також було реалізовано тестування БД для перевірки записів у БД.

Було проведено системне та інтеграційне тестування додатку, після реалізації всіх компонентів продукту. Було проведено, як позитивне, так і негативне тестування. Тестування було проведено мануально за допомогою чек-листа з основними перевірками (табл.3.1).

Таблиця 3.1 – Чек-лист системного тестування додатку.

Перевірка	Результат	Коментарі
Меню авторизації		
Вхід з поштою Google	Pass	
Вхід як не зареєстрований користувач Google	Pass	
Меню Device		
Додати девайс	Pass	
Видалити девайс	Pass	
Редагувати інформацію про девайс	Pass	
Додати неіснуючий девайс	Pass	
Перевірити інформацію про статус світла у меню	Pass	
Перевірити інформацію про статус батареї у меню	Pass	
Перевірити інформацію про статус мережі у меню	Pass	
Меню Профіль		
Перевірити інформацію користувача	Pass	
Вибрати Push-повідомлення	Pass	
Вибрати Telegram-повідомлення	Pass	
Обрати 2 види нотифікації	Pass	
Не обирати жодного виду нотифікації		

При тестуванні були використані такі додатки, як Excel – для описання тестових варіантів у формі чек листа, Postman – для тестування API продукту, Trello – для відслідковування прогресу тестування та DataGrip для тестування БД. Тестування проводилось на девайсі з ОС Андроід.

Можна виділити наступні ризики продукту – проблеми на стороні API Google або Telegram, або відмова хмарного сховища FireBase.

Отже, усі важливі функції додатку були протестовані, не зважаючи на можливі ризики, продукт був реалізований вчасно та доступний для користування.

3.3 Аналіз результатів практичного використання та перспективи подальшого розвитку продукту

В ході розробки та тестування продукту було виявлено низку переваг та недоліків системи. Також визначено вектор можливого розвитку цього проєкту.

З переваг цього проєкту можу визначити наступне:

- швидкодія та відмовостойкість проєкту завдяки обраним підходам та технічним рішенням;
- усі данні доступні в режимі реального часу;
- є кілька каналів сповіщення користувача в разі зміни стану світла.

З недоліків маю зазначити наступне:

- додаток існує тільки для платформи Android;
- аутентіфікація можлива тільки для користувачів з Google аккаунтом.

На основі цього було створено наступні пункти для подальшого розвитку проєкту:

- використання фреймворку для розробки кроссплатформенних додатків ios/android;
- додати інші методи аутентіфікації в додаток: Facebook/Apple/LinkedIn та інші;
- додати візуалізацію статистики з групуванням по різним проміжкам часу по даним стану світла в оселі;
- додати можливість встановлення та збір даних з додаткових сенсорів, наприклад, температура в оселі та рівень вологи;

ВИСНОВКИ

1. Проведено аналіз існуючих додатків та систем для трекінгу світла в будинку, популярних на ринку України. Визначено їх ключові переваги та недоліки

2. Проведено аналіз існуючих мікроконтролерів для збору та передачі телеметрії. В якості апаратної платформи обрано Arduino MKR GSM. Значною перевагою Arduino MKR GSM перед іншими пристроями є вбудований в плату модуль мобільного зв’язку.

3. На основі результатів огляду існуючих аналогів та з урахуванням обраної апаратної платформи були визначені ключові вимоги до embedded продукту та сформовано пункти технічного завдання.

4. Розроблено функціонал пристрою для трекінгу світла згідно з технічним завданням з використанням платформи Arduino мовою C++.

5. Розроблено архітектуру бази даних, використовуючи хмарний сервіс Google Firebase.

6. Розроблено архітектуру серверу обробки даних за допомогою Python та фреймворку FastApi, обробка асинхронних завдань з використанням брокеру повідомлень RabbitMQ.

7. Розроблено дизайн графічного інтерфейсу мобільного додатку за допомогою редактора Figma.

8. Розроблено архітектуру мобільного додатку на базі платформі Android мовою Kotlin з використанням фреймворку Jetpack Compose що дає змогу всі UI елементи описати кодом, без використання класичного XML файлу з описом елементів та їх стилів. Окрім пуш-нотифікації про події безпосередньо в мобільному додатку, реалізовано додаткове сповіщення користувача засобами Telegram API.

9. Розроблено текст-кейси та виконано тестування embedded продукту для трекінгу світла в будинку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Android Developer guides [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/docs.

2. Arduino for Beginners: How to Choose, Purchase, and Program an Arduino Board to Create Amazing Projects Step by Step, 2020. – 150 с.

3. Augmented Reality Market by Device Type (Head-mounted Display, Head-up Display), Offering (Hardware, Software), Application (Consumer, Commercial, Healthcare), Technology, and Geography - Global Forecast to 2026.

4. Build better apps faster with Jetpack Compose [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/jetpack/compose.

5. Building Microservices: Designing Fine-Grained Systems 2nd Edition, 2021. – 612 с.

6. Finances Online research. Number of Internet of Things (IoT) Connected Devices Worldwide 2022/2023: Breakdowns, Growth & Predictions [Електронний ресурс] – Режим доступу до ресурсу: https://financesonline.com/number-of-internet-of-things-connected-devices/.

7. Gartner Research. Forecast: IT Services for IoT, Worldwide, 2019-2025. [Електронний ресурс] – Режим доступу до ресурсу: https://www.gartner.com/en/documents/4004741

8. ISO/IEC/IEEE 12207:2017, Systems and software engineering, Software life cycle processes. https://www.iso.org/standard/63712.html

9. The Ultimate Guide to the internet of things [Електронний ресурс] – Режим доступу до ресурсу: https://iotsources.com/.

10. UML Use Case Diagram. https://www.javatpoint.com/uml-use-case-diagram.

11. Unleashing the Power of Data with IoT and Augmented Reality. By Zia Yusuf, Vladimir Lukic, James Heppelmann, Craig Melrose, Neeru Ravi, Usama Gill, and Andres Rosello. [Електронний ресурс] – Режим доступу до ресурсу: https://www.bcg.com/publications/2020/unleashing-the-power-of-data-with-iot-and-augmented-reality

12. Сервіс для створення діаграм. [Електронний ресурс] – Режим доступу до ресурсу: https://app.diagrams.net/.

13. Телеграм-бот SvetaTyDeBot. [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/zd333/electro_bot

ДОДАТОК А
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (ПРЕЗЕНТАЦІЯ)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка embedded продукту для трекінгу світла в будинку з використанням Arduino, Python, Telegram API

Виконав студент 5 курсу
групи ПП3-51
Коваль Максим Сергійович
Керівник роботи
К.т.н, доц, доцент кафедри ІПЗ Золотухіна Оксана Анатоліївна

Київ – 2023

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – спрощення процесу перевірки стану світла в будинку в режимі реального часу та підвищення рівня поінформованості користувача про зміну стану світла в будинку за рахунок використання embedded продукту на основі Arduino, Python, Telegram API.

Об'єкт дослідження – процес трекінгу світла в будинку.

Предмет дослідження – embedded продукти для трекінгу світла в будинку.

2

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Огляд та аналіз існуючих додатків та систем для трекінгу світла в будинку.
2. Огляд та аналіз існуючих мікроконтролерів для збору та передачі телеметрії.
3. Аналіз вимог до embedded продукту для трекінгу світла в будинку та формування технічного завдання до проекту.
4. Проектування та розробка пристрою для трекінга світла.
5. Проектування та розробка архітектури бази даних.
6. Проектування та розробка архітектури серверу обробки даних.
7. Проектування графічного інтерфейсу мобільного додатку.
8. Проектування та розробка архітектури мобільного додатку.
9. Тестування embedded продукту для трекінгу світла в будинку.

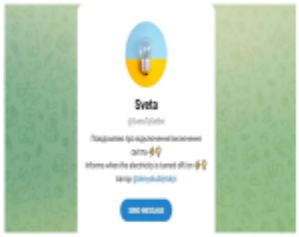
3

АНАЛОГИ

Додаток Світло

Ajax Security System

SvetaTyDeBot



4

ТАБЛИЦЯ ПОРІВНЯНЬ АНАЛОГІВ

Назва	Переваги	Недоліки
Додаток "Світло"	• Можливість додати декілька місць • Графіки відключень • Сповіщення за 30 хв. до відключення	• Працює виключно згідно графіка відключень • Графік не завжди відповідає реальності • Сповіщення тільки через додаток
Ajax Security System	• Повноцінна система розумного будинку • Мобільний додаток для ios/android • Має вбудовану батарею живлення	• Велика ціна • Не має сенсу використовувати тільки для моніторингу мережі • Сповіщення тільки через додаток
Телеграм бот SvetaTyDeBot	• Аналітика відключень за добу • Сповіщення при зміні стану мережі • Нагадування поповнити запаси води та їжі під час тривалого відключення • Сповіщення засобами Telegram	• Домашній інтернет повинен мати статичну IP адресу • Домашній інтернет повинен працювати без ДБЖ
strumOK	• Можливість додати декілька місць • Данні в режимі реального часу • Сповіщення при зміні стану мережі через додаток та Telegram • Має вбудовану батарею живлення	• Додаток тільки під Android • Реєстрація тільки через Google

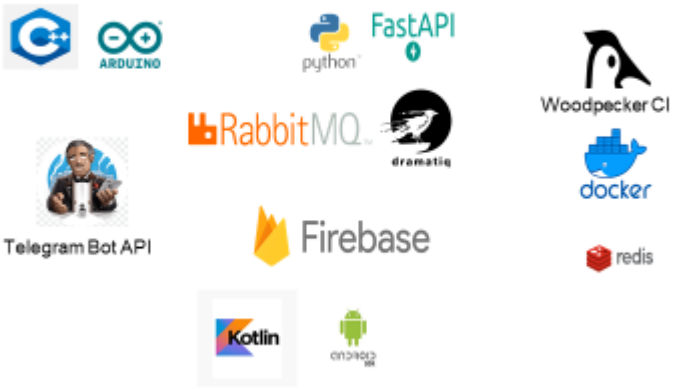
5

ВИМОГИ ДО ДОДАТКУ

- Реєстрація нового користувача.
- Можливість змінювати налаштування оповіщень.
- Додавання пристрою трекінгу світла в свій аккаунт.
- Видалення пристрою трекінгу світла з аккаунта.
- Перегляд інформації наданої пристроєм трекінга світла.
- Додаток повинен мати сторінку з списком усіх доданих пристроїв аккаунта.
- Сервер обробки данних пристрою повинен мати велику продуктивність та відмовостійкість.

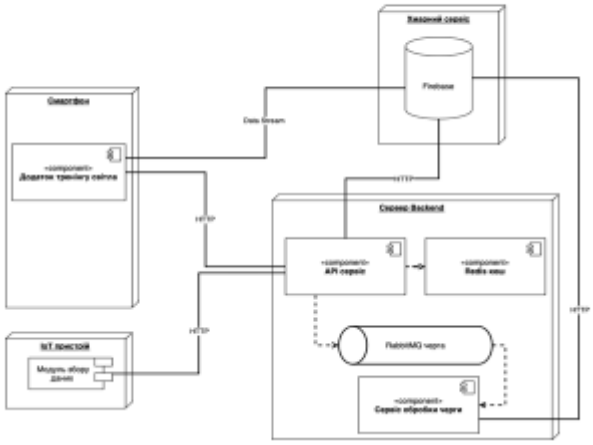
6

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



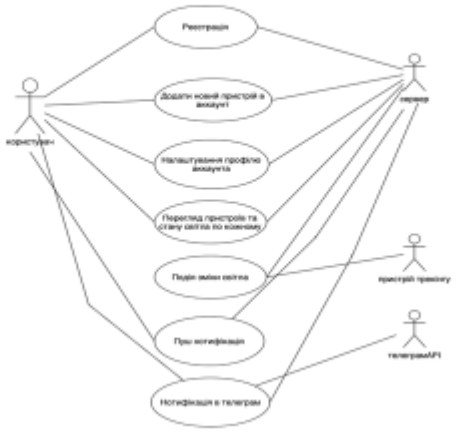
7

ДІАГРАМА РОЗГОРТАННЯ



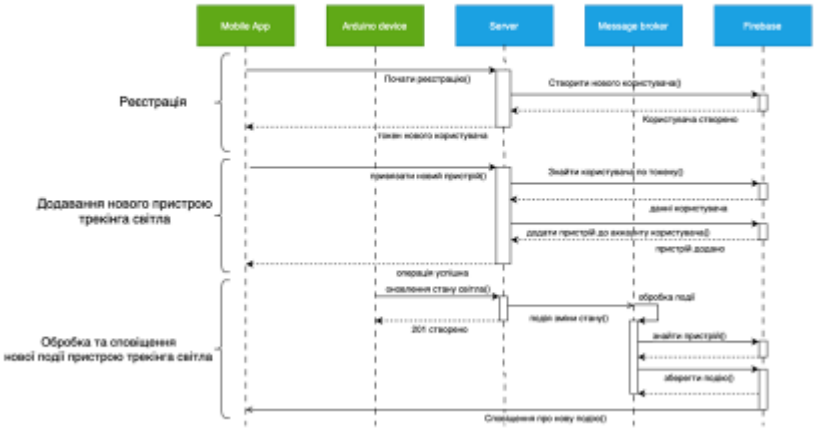
8

ДІАГРАМА ПРЕЦЕДЕНТІВ



9

ДІАГРАМА ПОСЛІДОВНОСТІ ГОЛОВНИХ ПРОЦЕССІВ



10

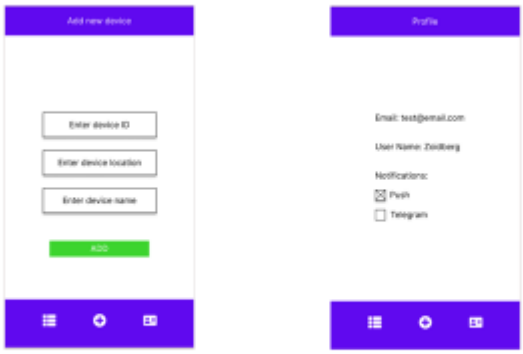
СХЕМА БАЗИ ДАНИХ

11

ЕКРАННІ ФОРМИ

12

ЕКРАННІ ФОРМИ



13

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Коваль М.С. Інтернет речей (IoT) та додаткові реалістичні технології в діджиталізованому світі / М.С. Коваль, О.А.Золотухіна// Сучасні аспекти діджиталізації та інформатизації в програмній та комп'ютерній інженерії Сучасні аспекти діджиталізації та інформатизації в програмній та комп'ютерній інженерії. 1-3 червня 2023 р., ДУТ, м. Київ — К.: ДУТ, 2023. — Подано до друку.

14

ВИСНОВКИ

1.Проведено аналіз існуючих додатків та систем для трекінгу світла в будинку, популярних на ринку України. Визначено їх ключові переваги та недоліки

2.Проведено аналіз існуючих мікроконтролерів для збору та передачі телеметрії. В якості апаратної платформи обрано Arduino MKR GSM. Значною перевагою Arduino MKR GSM перед іншими пристроями є вбудований в плату модуль мобільного зв'язку.

3.На основі результатів огляду існуючих аналогів та з урахуванням обраної апаратної платформи були визначені ключові вимоги до embedded продукту та сформовано пункти технічного завдання.

4.Розроблено функціонал пристрою для трекінгу світла згідно з технічним завданням з використанням платформи Arduino мовою C++.

5.Розроблено архітектуру бази даних, використовуючи хмарний сервіс Google Firebase.

6.Розроблено архітектуру серверу обробки даних за допомогою Python та фреймворку FastApi, обробка асинхронних завдань з використанням брокеру повідомлень RabbitMQ.

7.Розроблено дизайн графічного інтерфейсу мобільного додатку за допомогою редактора Figma.

8.Розроблено архітектуру мобільного додатку на базі платформі Android мовою Kotlin з використанням фреймворку Jetpack Compose що дає змогу всі UI елементи описати кодом, без використання класичного XML файлу з описом елементів та їх стилів. Окрім пуш-нотифікації про події безпосередньо в мобільному додатку, реалізовано додаткове сповіщення користувача засобами Telegram API.

9.Розроблено текст-кейси та виконано тестування embedded продукту для трекінгу світла в будинку.

15

ДОДАТОК Б
ЛІСТИНГИ ГОЛОВНИХ МОДУЛІВ

Модуль відправки даних пристрою

```
#include <Arduino_CRC32.h>
#include <ArduinoJson.h>
#include <ArduinoHttpClient.h>
#include <MKRGSM.h>
#include <Arduino_PMIC.h>
#include "ArduinoLowPower.h"
#include "arduino_secrets.h"

enum POWER_STATE {
  POWER_ON = 1,
  POWER_OFF,
};

const char pinnumber[] = SECRET_PINNUMBER;
const char gprs_apn[] = SECRET_GPRS_APN;
const char gprs_login[] = SECRET_GPRS_LOGIN;
const char gprs_password[] = SECRET_GPRS_PASSWORD;
const char backend_host[] = BACKEND_HOST;
const int backend_port = BACKEND_PORT;
const char user_agent[] = USER_AGENT;
const char api_path[] = BACKEND_API_PATH;

const char content_type[] = "application/json";
const int cycles_between_gsm_recconect = 15;
const int send_request_retries = 5;
const unsigned long send_interval = 15000L;
```

```
Arduino_CRC32 crc32;
const uint32_t device_crc_id = crc32.calc((uint8_t const *)DEVICE_ID, strlen(DEVICE_ID));

static int cycles = 1;
static bool gprs_connected = false;
static unsigned long time_since_last_update = millis();

GSMClient client;
GPRS gprs;
GSM gsmAccess;
GSMScanner scannerNetworks;
HttpClient httpClient = HttpClient(client, backend_host, backend_port);

void setup() {
  Serial.begin(9600);
  delay(8000);

  println_info("Starting setup");

  scannerNetworks.begin();

  if (!PMIC.begin()) {
    println_info("Failed to initialize PMIC!");
  }

  if (!PMIC.setInputCurrentLimit(2.0)) {
    println_info("Error in set input current limit");
  }
```

```
  if (!PMIC.setInputVoltageLimit(3.88)) {
    println_info("Error in set input voltage limit");
  }

  if (!PMIC.setMinimumSystemVoltage(3.5)) {
    println_info("Error in set minimum system volage");
  }

  if (!PMIC.setChargeVoltage(4.2)) {
    println_info("Error in set charge volage");
  }

  if (!PMIC.setChargeCurrent(1)) {
    println_info("Error in set charge current");
  }

  println_info("Finished setup");
}

void loop() {
  println_info("Starting_loop...");

  while (!connect_gprs());

  int signal_strength = scannerNetworks.getSignalStrength().toInt();
  if (signal_strength > 33) {
    signal_strength = 0;
  }
```

```
  float battery_voltage = analogRead(ADC_BATTERY) * 3.3f / 1023.0f / 1.2f * (1.2f + 0.33f);
  float battery_percentage = floor(((battery_voltage - 3.5) / 0.7) * 100);
  int signal_strength_percentage = map(signal_strength, 0, 33, 0, 100);

  show_telemetry(signal_strength, signal_strength_percentage, battery_voltage, battery_percentage, cycles);

  check_power_data(battery_percentage, signal_strength_percentage);

  if (cycles >= cycles_between_gsm_recconect) {
    cycles = 0;
    shutdown_gprs();
  }


  cycles += 1;
  println_info("Ending_loop...");
  wait_for_delay();
}

void wait_for_delay() {
  if (cycles == 1) {
    LowPower.sleep(6500);
  } else {
    delay(6500);
  }
}

void check_usb_mode(const float battery_percentage, const int network_level) {
```

```
  int usb_mode = PMIC.USBmode();
  switch (usb_mode) {
    case ADAPTER_PORT_MODE:
    case BOOST_MODE:
    case USB_HOST_MODE:
      if ((millis() - time_since_last_update) >= send_interval) {
        try_update_state(POWER_ON, battery_percentage, network_level);
      }

      break;
    default:
      if ((millis() - time_since_last_update) >= send_interval) {
        try_update_state(POWER_OFF, battery_percentage, network_level);
      }

      break;
  }
}

void check_power_data(const float battery_percentage, const int network_level) {
  int charge_state = PMIC.chargeStatus();
  switch (charge_state) {
    case NOT_CHARGING:
      if ((millis() - time_since_last_update) >= send_interval) {
        try_update_state(POWER_OFF, battery_percentage, network_level);
      }

      break;
    case PRE_CHARGING:
    case FAST_CHARGING:
```

```
    case CHARGE_TERMINATION_DONE:
      if ((millis() - time_since_last_update) >= send_interval) {
        try_update_state(POWER_ON, battery_percentage, network_level);
      }

      break;
    default:
      check_usb_mode(battery_percentage, network_level);

      break;
  }
}

void try_update_state(const POWER_STATE state, const float battery_percentage, const
int network_level) {
  int attempts = send_request_retries;
  while (attempts > 0) {
    if (send_post(state, battery_percentage, network_level)) {
      time_since_last_update = millis();

      break;
    }

    attempts -= 1;
  }
}

bool connect_gprs() {
  if (gprs_connected) {
    return true;
```

```
  }

  println_info("Starting_gprs_connection");
  bool connected = false;
  while (!connected) {
    if ((gsmAccess.begin(pinnumber) == GSM_READY) &&
(gprs.attachGPRS(gprs_apn, gprs_login, gprs_password) == GPRS_READY)) {
      connected = true;
    } else {
      println_info("Waiting_for_gprs_connection...");
      delay(500);
    }
  }

  gprs_connected = true;

  println_info("GRPS_connected");

  return true;
}

void shutdown_gprs() {
  println_info("GRPS_shutting_down");
  gsmAccess.shutdown();
  gprs_connected = false;

}

bool send_post(const int state, const float battery_percentage, const int network_level) {
  if (!httpClient.connect(backend_host, backend_port)) {
```

```
  println_info("Failed_to_connect_to_backend");
  httpClient.stop();
  delay(500);

  return false;
}

println_info("Preparing_POST_request");

StaticJsonDocument<128> json;
json["device_id"] = device_crc_id;
json["power_state"] = state;
json["battery_level"] = battery_percentage;
json["network_level"] = network_level;

String buffer;
serializeJson(json, buffer);

httpClient.beginRequest();
httpClient.post(api_path);
httpClient.sendHeader(HTTP_HEADER_CONTENT_TYPE, content_type);
httpClient.sendHeader(HTTP_HEADER_USER_AGENT, user_agent);
httpClient.sendHeader(HTTP_HEADER_CONTENT_LENGTH, buffer.length());
httpClient.beginBody();
httpClient.print(buffer);
httpClient.endRequest();

int status_code = httpClient.responseStatusCode();
String response = httpClient.responseBody();
```

```
  print_info("ResponseCode:");
  println_info(status_code);
  print_info("Response:");
  println_info(response);

  return status_code == 201 || status_code == 200;
}

void show_telemetry(const int signal_strength, const int network_level, const float battery_level, const float percentage, const int cycles) {
  print_info("BatteryVoltage:");
  println_info(battery_level);

  print_info("BatteryCapacity:");
  println_info(percentage);

  print_info("ChargeStatus:");
  println_info(PMIC.chargeStatus());

  print_info("Cycles:");
  println_info(cycles);

  print_info("NetworkSignal:");
  println_info(signal_strength);

  print_info("NetworkSignal%:");
  println_info(network_level);
}

void println_info(const char info[]) {
```

```
  if (Serial) {
    Serial.println(F(info));
  }
}

void println_info(const String &info) {
  if (Serial) {
    Serial.println(F(info.c_str()));
  }
}

void println_info(const int info) {
  if (Serial) {
    Serial.println(info);
  }
}

void println_info(const float info) {
  if (Serial) {
    Serial.println(info);
  }
}

void println_info(const bool info) {
  if (Serial) {
    Serial.println(info);
  }
}

void print_info(const char info[]) {
```

```
  if (Serial) {
    Serial.print(F(info));
  }
}
```

Модуль обробки подій

```
from os import environ
from typing import Annotated

from fastapi import APIRouter, Header, HTTPException
from starlette import status

from app.model.request.create_event_request import CreateStateEventRequest
from app.queue.worker import power_state_update

router = APIRouter(
    prefix="/api",
    tags=["events"],
    responses={403: {"description": "Forbidden"}, 201: {"description": "Created"}},
)


@router.post("/events", status_code=status.HTTP_201_CREATED)
async def create_state_event(req: CreateStateEventRequest, user_agent: Annotated[str |
None, Header()] = None):
    if user_agent != environ.get("DEVICE_AGENT"):
        raise HTTPException(status_code=403, detail="Forbidden")

    power_state_update.send(req.dict())
```

```
    return {}


import dramatiq

from app.dependencies.dependecies import get_event_service, get_device_service, get_rabbitmq
from app.model.domain.event_model import EventModel
from app.model.event.power_state_update import PowerStateUpdateEvent
from app.service.device_service import DeviceService
from app.service.event_service import EventService

rabbitmq_broker = get_rabbitmq()
dramatiq.set_broker(rabbitmq_broker)


@dramatiq.actor
def power_state_update(update_event: dict[str, str | float], event_service: EventService = get_event_service(),
                       device_service: DeviceService = get_device_service()):
    e = PowerStateUpdateEvent(**update_event)
    if not device_service.device_exists(e.device_id):
        return

    model = EventModel(device_id=e.device_id, power_state=e.power_state, network_level=e.network_level,
                       battery_level=e.battery_level, created_at=e.fired_at, updated_at=e.fired_at)

    event_service.create_new_event(model)
```

```
from google.cloud.firestore_v1 import FieldFilter

from app.firebase.firebase_client import FireBaseClient
from app.redis.redis_client import RedisClient


class DeviceService:
    DEVICE_CACHE_PREFIX = 'app:devices'

    def __init__(self, redis_client: RedisClient, firebase_client: FireBaseClient):
        self.redis_client = redis_client
        self.firebase_client = firebase_client

    def device_exists(self, device_id: str) -> bool:
        cache_key = self.__get_key(device_id)
        if self.redis_client.get(cache_key):
            return True

        device_filter = FieldFilter('device_id', '==', device_id)
        query = self.firebase_client.client.collection('Devices').where(filter=device_filter).limit(1)
        snapshot = query.get()

        exists = sum(1 for _ in snapshot) > 0
        if exists:
            self.redis_client.set(cache_key, 1)

        return exists

    def __get_key(self, key_value: str) -> str:
```

```
return f'${self.DEVICE_CACHE_PREFIX}:${key_value}'
```