КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

До захисту допущено
Завідувача кафедри ІСТ
_____ Олександр КУЧАНСЬКИЙ
(підпис) (ім'я, ПРІЗВИЩЕ)
"___"_____2022р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

спеціальності 126 «Інформаційні системи та технології»

освітньої програми «Програмні технології інтернет речей»

на тему: IoT система керування розумним будинком на базі децентралізованої мережі

Виконав: студент 4 курсу, групи ІР-41

(шифр групи)

Віталій БІДОЧКА

(Ім'я, ПРІЗВИЩЕ) (підпис)

Керівник к.т.н., доцент Сергій ПАЛІЙ

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

Консультант нормо контроль к.т.н., доцент Ростислав ЛІСНЕВСЬКИЙ

(назва розділу) (посада, вчене звання, науковий ступінь, Ім'я, ПРІЗВИЩЕ) (підпис)

Рецензент

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

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

Здобувач освіти _____

(підпис)

Київ – 2022 року

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

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

ЗАВДАННЯ

НА ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

Здобувач освіти: Віталій БІДОЧКА

Група: ІР-41

1. **Тема кваліфікаційної роботи бакалавра:** "IoT система керування розумним будинком на базі децентралізованої мережі"

   Затверджена протоколом засідання кафедри ІСТ №05/21_22 від 03.12.2021 року

2. **Строк подання студентом готової роботи -** "22" червня 2022 р.

3. **Вихідні дані до роботи:** Дослідження існуючих систем керування розумними будинками. Програмне забезпечення для керування розумним будинком на базі децентралізованої мережі.

4. **Зміст роботи:** РОЗДІЛ 1 ОГЛЯД ВИМОГ ДО СИСТЕМ КЕРУВАННЯ РОЗУМНИМИ БУДИНКАМИ (аналіз та опис сфери застосування, поняття децентралізованої мережі, постановка задачі, економічне обґрунтування); РОЗДІЛ 2 ПРОЄКТУВАННЯ СИСТЕМИ (декомпозиція задачі проєктування системи, опис допоміжних сервісів, опис вузла, опис модуля датчиків, алгоритм роботи системи); РОЗДІЛ 3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ (вибір програмного забезпечення та інструментарію для розробки системи,

розробка бібліотеки класів з використанням мови програмування Python, розробка бібліотеки компонентів для графічного інтерфейсу з використанням фреймворку Vue 3); РОЗДІЛ 4 ОПИС АПАРАТНОЇ РЕАЛІЗАЦІЇ (вибір елементної бази та її опис, побудова схеми, програмування мікроконтролера).

5. **Перелік графічного матеріалу** (з точним позначенням обов'язкових креслень) структурна схема системи, узагальнена схема роботи системи.

6. **Календарний план виконання роботи:**

Етапи виконання кваліфікаційної роботи бакалавра	**Термін виконання**	**Результат виконання**
1. Вибір тематики кваліфікаційної роботи бакалавра	01.09.2021-01.10.2021	виконано
2. Наказ про затвердження тем кваліфікаційної роботи бакалавра та призначення керівників	03.12.2021	виконано
3. Розробка плану кваліфікаційної роботи бакалавра і його погодження з керівником	25.12.2021	виконано
4. Написання I розділу кваліфікаційної роботи	19.03.2022	виконано
5. Написання II розділу кваліфікаційної роботи	25.04.2022	виконано
6. Написання III розділу кваліфікаційної роботи	29.04.2022	виконано
7. Підготовка висновків і пропозицій	30.04.2022	виконано
8. Попередній захист кваліфікаційної роботи	12.05.2022	виконано
9. Перевірка на плагіат	13.05.2022-15.06.2022	виконано
10. Нормоконтроль	02.06.2022-06.06.2022	виконано
11. Рецензування кваліфікаційної роботи бакалавра і представлення роботи на кафедру в друкованому вигляді	15.06.2022	виконано
12. Захист кваліфікаційної роботи бакалавра	23.06.2021 -24.06.2021	

Дата видачі завдання «__»_____2022 р.

Керівник роботи: к. т. н., доцент Сергій ПАЛІЙ (підпис)

Завдання прийняв до виконання:

Здобувач освіти на освітньому рівні «бакалавр» 4-го курсу групи ІР-41

Віталій БІДОЧКА

(Власне Ім'я, ПРІЗВИЩЕ) (підпис)

АНОТАЦІЯ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра Інформаційних систем та технологій

Освітня програма “Програмні технології інтернет речей”

Кваліфікаційна робота бакалавра Віталія БІДОЧКИ

Тема роботи: “IoT система керування розумним будинком на базі децентралізованої мережі”

Мета кваліфікаційної роботи бакалавра - розробка IoT системи керування розумним будинком на базі децентралізованої мережі. Реалізація всіх необхідних внутрішніх компонентів системи. Розробка графічного інтерфейсу користувача.

Об’єкт розробки - система керування розумним будинком.

Предмет дослідження - складові системи керування розумним будинком на базі децентралізованої мережі (бази даних, графічний інтерфейс, ядро, NH сервіс, сертифікаційний сервіс, API).

Апробація результатів. Робота розглядалася та обговорювалась на VIII Міжнародній науково-технічній конференції “Сучасні методи, інформаційне, програмне та технічне забезпечення систем керування організаційно-технічними та технологічними комплексами”, що відбулась 26 листопада 2021 року, Міжнародній науково-технічній конференції молодих учених, аспірантів і студентів “Наукові здобутки молоді - вирішенню проблем харчування людства у XXI столітті”, що відбувалась 15-26 квітня 2021 року, VIII міжнародній конференції “Information Technology and Interactions (Satellite)”, що відбувалась 1-3 грудня 2021 року та в IV Міжнародній науково-технічній конференції “Сучасні тенденції розвитку інформаційних систем і телекомунікаційних технологій”, що відбувалась 1-2 лютого 2022 року.

Кваліфікаційна робота бакалавра складається зі змісту, вступу, основної частини, яка включає 4 розділи, висновків, списку використаних джерел, додатків.

Всього 69 сторінок.

КЛЮЧОВІ СЛОВА: ІНТЕРНЕТ РЕЧЕЙ, РОЗУМНИЙ БУДИНОК, IOT, МЕРЕЖІ, ДЕЦЕНТРАЛІЗОВАНА МЕРЕЖА, PYTHON, DJANGO, VUE, ЗАХИСТ ДАНИХ.

Власні публікації:

1. Бідочка В. А., Палій С. В. Застосування децентралізованих мереж у IoT-системах. “Сучасні методи, інформаційне, програмне та технічне забезпечення систем керування організаційно-технічними та технологічними комплексами” : VIII Міжн. наук.-тех. конф., 26 лист. 2021 р. Київ. : НУХТ. С. 222-223.
2. Бідочка В. А., Палій С. В. Поняття децентралізованих мереж, їх використання та порівняння з централізованими мережами. “Наукові здобутки молоді - вирішенню проблем харчування людства у XXI столітті” : Міжн. наук.-тех. конф. молодих учених, аспірантів і студентів 15-26 квіт. 2021 р. Київ. : НУХТ. С. 367.
3. V. Bidochka, S. Paliy. Decentralized network in IoT systems. Information Technology and Interactions (Satellite) : Conference Proceedings, December 01, 2021, Kyiv, TSNUK. P.
4. Бідочка В. А., Палій С. В. Проєктування IoT системи керування розумним будинком на базі децентралізованої мережі. “Сучасні тенденції розвитку інформаційних систем і телекомунікаційних технологій” : IV Міжн. наук.-тех. конф., 1-2 лют. 2022 р. Київ. : НУХТ. С. 37-39.

ABSTRACT

TARAS SHEVCHENKO NATIONAL UNIVERSITY OF KYIV

Faculty of Information Technologies

Department of Information Systems and Technologies

Educational Program “Software Technologies of the Internet of Things”

Qualification work of master Vitalii BIDOCHKA.

Work topic: “IoT smart home management system based on a decentralized network”.

The purpose of the bachelor's qualification work is development of IoT smart home management system based on a decentralized network. Implementation of all necessary internal components of the system. Development of a graphical user interface.

The object of research is a smart home management system.

The subject of research is the components of a smart home management system based on a decentralized network (database, graphical interface, kernel, NH service, certification service, API).

Approbation of results. The work was reviewed and discussed at the VIII International Scientific and Technical Conference "Modern Methods, Information, Software and Technical Support of Management Systems of Organizational, Technical and Technological Complexes", held on November 26, 2021, the International Scientific and Technical Conference of Young Scientists, Postgraduates and Students - Solving the Problems of Human Nutrition in the XXI Century ”, held on April 15-26, 2021, VIII International Conference“ Information Technology and Interactions (Satellite) ”, held on December 1-3, 2021 and in the IV International Scientific and Technical Conference“ Modern Development Trends information systems and telecommunication technologies ”, which took place on February 1-2, 2022.

The bachelor's qualification work consists of the content, introduction, main part, which includes four sections, conclusions and a list of sources used. Total 69 pages.

KEY WORDS: Internet of Things, Smart Home, IoT, Networks, Distributed Networks, Python, Django, Vue, Data Security.

Own publications:

1. Бідочка В. А., Палій С. В. Застосування децентралізованих мереж у IoT-системах. “Сучасні методи, інформаційне, програмне та технічне забезпечення систем керування організаційно-технічними та технологічними комплексами” : VIII Міжн. наук.-тех. конф., 26 лист. 2021 р. Київ. : НУХТ. С. 222-223.
2. Бідочка В. А., Палій С. В. Поняття децентралізованих мереж, їх використання та порівняння з централізованими мережами. “Наукові здобутки молоді - вирішенню проблем харчування людства у XXI столітті” : Міжн. наук.-тех. конф. молодих учених, аспірантів і студентів 15-26 квіт. 2021 р. Київ. : НУХТ. С. 367.
3. V. Bidochka, S. Paliy. Decentralized network in IoT systems. Information Technology and Interactions (Satellite) : Conference Proceedings, December 01, 2021, Kyiv, TSNUK. P.
4. Бідочка В. А., Палій С. В. Проєктування IoT системи керування розумним будинком на базі децентралізованої мережі. “Сучасні тенденції розвитку інформаційних систем і телекомунікаційних технологій” : IV Міжн. наук.-тех. конф., 1-2 лют. 2022 р. Київ. : НУХТ. С. 37-39.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Умовне позначення	**Розшифрування**
IoT	Internet of Things
API	Application Programming Interface
MQTT	Message Queuing Telemetry Transport
HTTP	HyperText Transfer Protocol
NH	Next Hop
P2P	Peer To Peer
ПЗ	Програмне Забезпечення
АЗ	Апаратне Забезпечення
B2C	Business To Customer
B2B	Business To Business
QA	Quality Assurance
SPA	Single Page Application
MVT	Model-View-Template
E2E	End To End
DNS	Domain Name Space
HTML	Hyper Text Markup Language
CSS	Cascading Style Sheet

ВСТУП

Ми живемо в сучасному світі, в якому важко обійтись без новітніх технологій, одна з найбільш поширених це IoT системи, які дозволяють робити речі “розумними”, тим самим спрощувати життя людей.

Інтернет речей (IoT) описує мережу фізичних об’єктів — «речей», які вбудовані з датчиками, програмним забезпеченням та іншими технологіями з метою підключення та обміну даними з іншими пристроями та системами через Інтернет. Системи IoT стають все більш популярними в наші дні. Вони використовуються по-різному і виконують різні завдання. Зараз багато галасу про Інтернет речей (або IoT) та його вплив на все: від того, як ми подорожуємо та робимо покупки, до того, як виробники відстежують інвентар. Через стрімке зростання їх популярності в системах IoT виникає маса проблем і недоліків.

Забезпечення анонімності даних та їх розподілення у розмовному будинки одне з найважливіших складових даних систем, адже від достовірних даних з датчиків залежить правильне функціонування всього розумного будинку, а при забезпеченні анонімності даних телеметрії та власника - робить оселю по справжньому безпечну. Тому саме ці основні фактори являються основними при виборі IoT системи.

На даний момент при реалізації систем для керування розумного будинку використовуються загальноприйняті практики забезпечення правильного, зашифрованого та безпечного каналу передачі даних на контролер. Як правило для цього використовують шифровані канали передачі даних такі, як https для реалізації передачі даних між API, шифрування передачі даних з датчика на контролер через протокол MQTT.

Звичайно ці підходи являються правильними і забезпечують базовий рівень безпеки, який захищає від певних видів атак, але при поглибленому вивчені наведених вище технологій та протоколів можна знайти певні мінуси та вразливості в безпеці.

Метою роботи є розробка IoT системи керування розумним будинком на базі децентралізованої мережі. Реалізація всіх необхідних внутрішніх компонентів системи. Розробка графічного інтерфейсу користувача.

Об'єктом дослідження є децентралізована мережа та система розумного будинку на її основі.

Предмет дослідження - складові системи керування розумним будинком на базі децентралізованої мережі (бази даних, графічний інтерфейс, ядро, NH сервіс, сертифікаційний сервіс, API).

Основна гіпотеза полягає в тому, що використовуючи децентралізовану систему IoT для керування розумним будинком вдасться досягти максимального рівня анонімності даних, а також цілісності даних власника системи та даних телеметрії, що забезпечить надійний рівень безпеки.

На основі визначеної мети розроблено і обґрунтовано наступні *завдання роботи*:

- вивчення та наукове обґрунтування доцільності використання децентралізованої мережі;
- вивчення та наукове обґрунтування необхідних фактів побудови IoT системи на базі децентралізованої мережі;
- проєктування кожного окремого компонента та системи в цілому;
- створення програмного забезпечення для вузла мережі;
- створення веб-додатку для керування системою та його тестування;
- створення апаратного забезпечення для модуля датчиків збору даних розумного будинку;

Теоретико-методологічною основою роботи є літературні джерела та Internet - джерела.

РОЗДІЛ 1. АНАЛІЗ ТА ОПИС СФЕРИ ЗАСТОСУВАННЯ ДЕЦЕНТРАЛІЗОВАНОЇ МЕРЕЖІ. ПОСТАНОВКА ЗАДАЧІ

1.1. Аналіз та опис сфери застосування

IoT системи все більше приходять в наше життя, ми можемо побачити на вулицях розумні світлофори, освітлення та точки продажу товарів [5]. Також чи мало років IoT індустрія активно розвивається і для приватного сектора, тобто активно розробляється і втілюється в життя концепт розумного будинку та всього що з ним пов’язано.

IoT концепція розумного будинку передбачає реалізацію різних датчиків та приладів, які можуть спілкуватись між собою та взаємодіяти між собою. Тобто за допомогою IoT системи можна зробити повну автоматизацію будинку, починаючи з автоматичного освітлення і закінчуючи цілою системою опалення на основі двох різних видів котлів (до прикладу, переключення котлів в залежності від години).

З ростом популярності IoT екосистем на ринку з’явились та активно розвиваються багато компаній, чиї продукти використовують мільйони користувачів по цілому світу. До популярних IoT систем можна віднести наступні:

- ThingsBoard Inc. - українська компанія, яка розробляє однойменний продукт. ThingsBoard - це IoT платформа, яка вперше була випущена на використання всім охочим у 2016 році. Платформа дозволяє керувати пристроями, збирати дані, обробляти їх та візуалізовувати. Також продукт компанії дозволяє дуже гнучко налаштовувати інтерфейс під індивідуальні вподобання. Платформа має дуже потужний конструктор дашбордів, де за допомогою готових віджетів можна налаштувати відображення даних, які надходять з пристроїв, керування девайсами, аналітику даних. Також до системи можна підключити власні плагіни та віджети, якщо потрібного готового функціоналу недостатньо. ThingsBoard - це продукт з відкритим кодом, який на GitHub отримав 10000 зірок від користувачів, тому до його підтримки та розвитку може долучитись будь-який бажаючий.

- Ajax Systems - міжнародна технологічна компанія, яка заснована в 2011 році. Головний офіс і виробничі потужності розташовані в Києві. Компанія розробляє бездротові та дротові системи безпеки, має власне виробництво повного циклу. Основний продукт компанії - це система безпеки, яка складається з 36 пристроїв та забезпечує безпеку від пограбувань, затоплень і пожеж в будинку. Також компанія має розроблені власні протоколи: Jeweller - це пропрієтарний двосторонній радіо протокол, радіус дії якого до 2000 метрів, а швидкість передачі тривоги становить ~0.15 секунди; Wings - двосторонній радіо протокол за допомогою якого передаються фото з камер відеонагляду, його дальність дії становить в радіусі 1700 метрів, а передача фото становить близько 9 секунд на перший знімок. Продукція компанії продається більш ніж у 120 країнах світу та має 1 млн користувачів.
- Apple HomeKit - платформа для керування пристроями розумного будинку розроблена компанією Apple. Платформа дозволяє користувачам налаштовувати, взаємодіяти та керувати приладами за допомогою пристроїв Apple. Платформа дуже легка та зручна у використанні та дозволяє автоматично під’єднювати виявлені датчики та сенсори. Користувач може керувати будинком з любого сумісного пристрою з любої точки світу. Платформа має сучасний, простий та інтуїтивний інтерфейс додатку в найкращих традиціях компанії Apple. Також слід зазначити, що платформа має дуже хорошу інтеграцію з іншими існуючими системами. До недоліків можна віднести підтримку тільки платформ IOS та MacOS.
- Xiaomi Smart Home - платформа розроблена китайською компаніює Xiaomi Corporation. Вона дозволяє налаштовувати, підключати та керувати пристроями, які випущені сторонніми компаніями, які співпрацюють з Xiaomi. Всі пристрої керуються з додатку, який називається Mi Home. Інтерфейс програми зручний та зрозумілий для використання навіть користувачу початківцю. На відміну від Apple HomeKit, програмний додаток можна використовувати на будь-якій мобільній платформі, включно з

Android та IOS. Платформа налічує багато сумісних пристроїв, зокрема: пристрої для відслідковування витоків води; камери відеонагляду; розумне освітлення; маршрутизатори; головні керуючі пристрої; датчики якості повітря; розумні розетки, виключателі; освіжувачі повітря та інші. Також за традиціями Xiaomi всі споріднені товари мають досить низький діапазон цін, що дозволяє за мінімальні кошти облаштувати розумний будинок.

Відповідно до росту популярності IoT систем виникають проблеми та недоліки централізованих мереж, з основних можна виділити наступні:

- проблема з цілісністю даних;
- анонімності даних;
- масштабованість;
- доступності даних.

Для вирішення даних проблем була розроблена система, яка працює на базі децентралізованої мережі. Також був проведений аналіз ринку на наявність вже існуючої системи, яка б працювала на основі децентралізації. За результатами дослідження не вдалось знайти аналогів до створеної системи.

1.2. Поняття децентралізованої мережі

Поняття децентралізованої мережі (або p2p мережі) було введено ще у 1984 році Парбауелом Йонугуйтсманом, який розробляв складні типи мереж для компанії IBM [6]. Дана мережа являє собою зв’язок рівноправних вузлів, які можуть звертатись один до одного для отримання сервісів і, відповідно, надавати їх.

Вперше для комерційного використання децентралізована мережа успішно була застосована в 1999 році компанією Napster, яка розробила сервіс для прослуховування музики. Компанія використала гібридну децентралізовану мережу, яка містила центральний сервер на якому зберігались ip-адреса хостів та музики, яка на ньому зберігається і могла бути прослухана іншими вузлами.

Наступна компанія це BitTorrent, яка розробила однойменний протокол та програмне забезпечення µTorrent. Сервіс базується на шарингу файлів, які можуть скачати будь-які користувачі, якщо мають відповідний файл конфігурації для підключення до вузла.

Dropbox - компанія, яка заснована в 2007 році та розробила однойменний продукт. Dropbox використовується як сервіс файлообмінник та синхронізатор файлів для легкого доступу до файлів користувача з будь-якого пристрою. Продукт працює на основі децентралізованої мережі та привніс багато нових принципів реалізації та організації даних типів мереж, зокрема покращення безпеки передачі, зберігання та розподілення даних на вузлах [7].

Не можна обійти без уваги тему криптовалюти, де більшість валют працює на основі децентралізованої мережі, а саме всі вузли мережі діляться своїми потужностями для обчислення частин однієї одиниці валюти, які потім складаються в блок - у цьому випадку транзакція рахується успішною та усім учасникам в якості оплати за їхні потужності надається певний процент грошової винагороди. В якості прикладу валют, які працюють на основі децентралізованої мережі, можна навести досить популярні та широковідомі Bitcoin, Ethereum, Dogecoin монети.

Також даний тип мережі широко використовується у наукових цілях, а саме:

- Штучний інтелект.
- Біоінформатика.
- Грід системи.

У вищезгаданих областях, децентралізована мережа використовується за таким самим принципом, як і в блокчейн сфері, тобто користувачі надають свої потужності комп’ютерів для обчислення складних розрахунків. На даний момент існують наступні проєкти, які активно підтримуються та розвиваються:

- Folding@home - проєкт розподілених обчислень, що проводиться під егідою Стенфордського університету. Суть проєкту полягає в моделюванні процесу згортання білків з метою виявлення потенційних помилок у природній

конформації, які спричиняють ряд хвороб, зокрема: хвороба Альцгеймера, хвороба Паркінсона, діабет типу II та інші.

- distributed.net - одна з найстаріших мережевих спільнот розподілених обчислень. Станом на 2009 рік в активі *distributed.net* 8 успішно завершених проєктів: 5 грошових криптографічних (злами стійких шифрів від *RSA* і *CS Communications*) і три науково-математичних (*OGR-24*, *OGR-25*, *OGR-26*).
- World Community Grid - це проєкт з розподілених розрахунків з громадянської науки. Проєкти WCG проаналізували дані, пов'язані з геномом людини, Мікробіомом людини, хворобами людини, моделюванням опадів, врожайністю рису, екологічно чистою енергією, очищенням води, та деякі інші дані.

Децентралізована мережа складається з автономних вузлів, які є повністю незалежними від інших вузлів і ця особливість являється основною у порівнянні з централізованою мережею.

У традиційній централізованій мережі поняття “клієнт” і “сервер” чітко розрізняються за цільовою метою, а саме: клієнт звертається до сервера для отримання сервісів, в той час як сервер не може звернутись до клієнта.

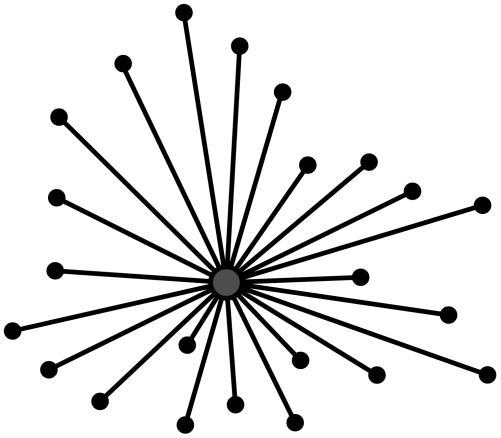


Рисунок 1.1. Централізована мережа

Всі клієнти є залежними від основного центрального серверу, який містить всі сервіси та дані необхідні для функціонування клієнтів (рис 1.1).

В загальному випадку централізована мережа складається з:

- сервера, які надають дані, інформацію, сервіси та послуги для клієнтів;
- клієнти, які зв’язуються з сервером для отримання інформації та послуг;
- мережа, яка забезпечує зв’язок між клієнтами та серверами.

Також слід зазначити, що сервера є незалежними між собою, так само як клієнти, які можуть працювати паралельно та не впливати один на одного.

У порівнянні, децентралізована мережа складається з незалежних вузлів, які являються і клієнтами, і серверами, це означає, що будь-який вузол може звернутись до “спорідненого” вузла, як до сервера та отримати запитувані послуги, і навпаки (рис. 1.2).

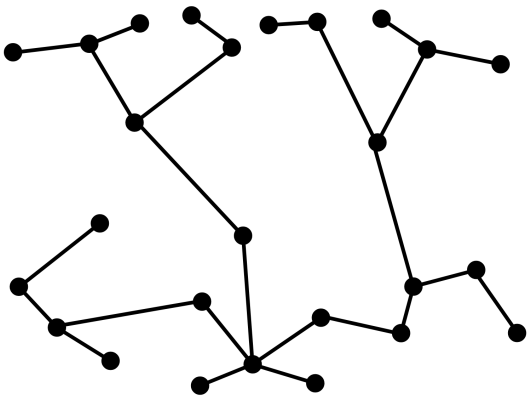


Рисунок 1.2. Децентралізована мережа

Слід зазначити, що чиста децентралізована мережа рідко використовується через складність її реалізації, тому на практиці, як правило, реалізують, так звану, гібридну децентралізовану мережу. Гібридні децентралізовані мережі містять сервери (не слід їх плутати з серверами з централізованих мереж), які можуть використовуватись для декількох цілей:

- у ролі маршрутизаторів, які містять окремі шляхи до інших таких серверів;
- у ролі інформаційних серверів, які містять інформацію про сервіси, які надають клієнти;
- у ролі службових серверів для виконання спеціальних задач (реєстрація вузлів, перевірка вузлів, перевірка ключів, а також оновлення програмного забезпечення, оновлення прошивки на вузлах).

При правильній реалізації децентралізованої мережі вона можна принести багато позитивних речей в продукт, а саме:

- Масштабованість. Це одна з ключових особливостей децентралізованої мережі. Через те що немає жорсткої централізації - немає центрального елемента, на який іде основне навантаження, так як він повинен обробляти запити від усіх клієнтів, які звертаються до нього. Відповідно не потрібно вручну масштабувати мережу у вигляді купівлі додаткових серверів або обчислювальних потужностей. Навантаження самостійно та органічно розподіляється по вузлах.
- Зменшення вартості обслуговування. Особливість випливає з масштабованості. Як вже зазначалось у попередньому пункті, через добру масштабованість цього типу мережі, не потрібно докуповувати додаткові сервери, відповідно вартість обслуговування мережі дуже зменшується у порівнянні з централізованою мережею. Також слід зазначити, що значна частка витрат зменшується на зберіганні даних, так як всі дані не зберігаються в центральному сховищі, а розподілені по вузлах мережі.
- Анонімність користувачів. Даний тип мережі забезпечує найбільшу анонімність користувачів у порівнянні з іншими типами мережі. Знову ж таки, дана особливість випливає з відсутності центрального сервера на якому зберігаються дані всіх користувачів. Децентралізована мережа розподіляє дані користувачів по всіх вузлах мережі.
- Автономність. Також одна з ключових особливостей даного типу мережі. Якщо у централізованій мережі знаходиться центральний сервер, навантаження на який перевищує його ліміт - він виходить з ладу, відповідно вся мережа стає паралізованою, так як немає кому надавати послуги клієнтам. У випадку з децентралізованою мережею такого не може статись, так як будь-який інший вузол може надавати послуги замість вузла, який вийшов з ладу.

З негативних сторін даного типу мережі можна віднести достатньо дорогу реалізацію та подальший розвиток продукту, оскільки кваліфікованих спеціалістів, які вміють розробляти, підтримувати та знати особливості децентралізованої мережі на ринку досить мало.

1.3. Постановка задачі

Результатом розробки повинна стати система, яка в першу чергу повинна гарантувати анонімність користувачів, а сама система повинна працювати на основі децентралізованої мережі, яка цю потребу забезпечує. Також слід забезпечити децентралізоване зберігання даних користувачів, тому система повинна мати написану логіку розподілення даних між вузлами мережі та їх керуванням.

IoT системи, як правило дуже динамічні та високонавантажені, саме тому система повинна мати хорошу швидкодію та забезпечувати неперервну і безперебійну роботу [15]. Для цих двох задач повинні бути використані новітні технології програмування та сучасні бібліотеки, фреймворки, які забезпечують найкращу швидкодію та є добре оптимізованими.

Також результатом роботи, повинні бути розроблені допоміжні сервіси, які:

- відповідають за надання, перевірку та вилучення сертифікатів користувачів;
- містять інформацію про доступні вузли та шляхи до них, а саме: сервіс повинен реєструвати та зберігати нові вузли; вираховувати метрику для кожного індивідуального вузла та на їх основі складати оптимальні маршрути для запитуваного вузла; видаляти вузол, якщо він неактивний.

Повинне бути написане програмне забезпечення для хабу, яке виступає в ролі агрегатора функціоналу та виконує наступні функції:

- керує рухом даних з датчиків, які до нього підключені;
- запускає веб-додаток;
- запускає базу даних;
- запускає MQTT брокер.

Система повинна давати можливість з’єднуватись з девайсами, які працюють через MQTT протокол, а також імплементувати всі можливості даного протоколу [19], а саме підключення до топіків, надсилання даних на топіки, підключення, відключення, повторна доставка даних у випадку невдачі, доставка даних у випадку, якщо девайс був неактивний.

Також система повинна мати зручний у використанні користувацький інтерфейс у вигляді веб-додатка. Веб-додаток повинен містити відповідні сторінки керуванням сутностями системи, перегляду даних телеметрії девайсів у режимі реального часу, а також можливість побудови індивідуальних, гнучко-налаштовуваних дашбордів.

Щодо апаратної частини, то вона повинна містити програмний код, який дозволяє керувати логікою роботи датчиків, які підключені до модуля, а також надсилати виміряну телеметрію на хаб.

1.4. Економічне обґрунтування

Окрім розробленої системи пропонується прототип компанії, організація та формат якої найкраще підходив би для подальшого розвитку та підтримки системи.

Прототипом підприємства, що займається розробкою програмного забезпечення та апаратних засобів для розумних будинків та іншого роду об’єктів, взято за основу ієрархічну структуру. В середині компанії всі процеси по поставці ПЗ та іншого роду комунікацій повністю автоматизовані та позбавлені інших видів бюрократії, де це можливо.

Підприємство використовує ієрархічну структуру (рис. 1.3).

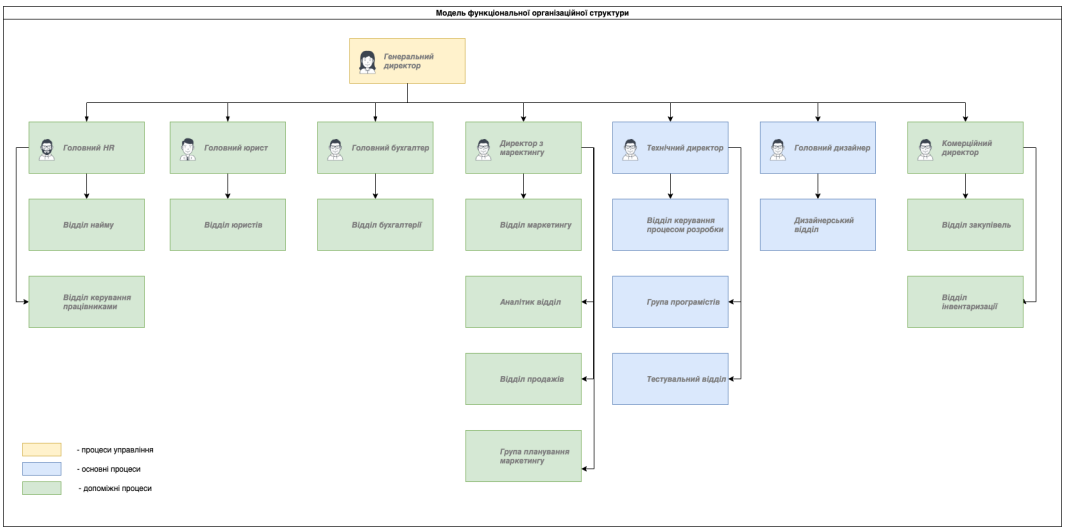


Рисунок 1.3. Структурна схема підприємства

Концепція ієрархічної структури була сформульована німецьким соціологом Максом Вебером, який розробив нормативну модель раціональної бюрократії, що ґрунтувалась на таких принципових положеннях: чіткий розподіл праці; ієрархічність управління; наявність формальних правил і норм; підбір кадрів для роботи на підприємстві у здійснюється відповідно до кваліфікаційних вимог до посади [24].

Використана лінійно-функціональна організаційна структура, яка є найефективнішою там, де апарат управління виконує рутинні функції, які часто повторюються і рідко змінюються. Її переваги проявляються в управлінні організаціями з масовим або серійним типом виробництва.

Підприємство поділяється на наступні відділи:

- Відділ менеджменту - Проєктний менеджер для кожного етапу розробки складає технічне завдання, набирає нову команду/доукомплектовує існуючу, вибір з двох позицій здійснюється по ситуації, якщо до прикладу задач багато і менеджер розуміє, що потрібно ще пару працівників або ж навпаки, коли працівників забагато на певний пулл задач. Якщо проєкт специфічний для клієнта, то менеджер проводить додаткові консультації з клієнтом. Також до до роботі в відділі менеджменту входять наступні задачі:
    - Створення та розподіл задач.

    - Моніторинг виконання задач.
    - Вибір стратегії виконання робіт.
- Відділ розробки - до робіт, які проводяться у відділі розробки відносяться: розробка серверного ПЗ, створення та продумування архітектури ПЗ, підтримка раніше створеного функціоналу, розробка нового функціоналу, розробка користувацького інтерфейсу, розробка користувацького інтерфейсу для мобільних пристроїв, а також створення та налаштування логіки взаємодії користувача з певним елементом.
- Відділ тестування - до робіт, які проводяться у відділі тестування відносяться: тестування розробленого функціоналу, виявлення багів, формування звітів про тестування.

За загальним принципом та алгоритмом роботи, підприємство працює наступним чином: після отримання розробником задачі від проєктного менеджера, він повинен розробити певний функціонал, який передбачений задачею, після закінчення розробки розробник повинен провести попереднє тестування нового функціоналу перш, ніж передавати його відділу тестування. Якщо після передачі готового функціоналу у відділ тестування, тестувальних повертає звіт з помилками для цього функціоналу, розробник повинен виправити помилки з звіту та передати виправлений функціонал на повторне тестування (рис. 1.4).

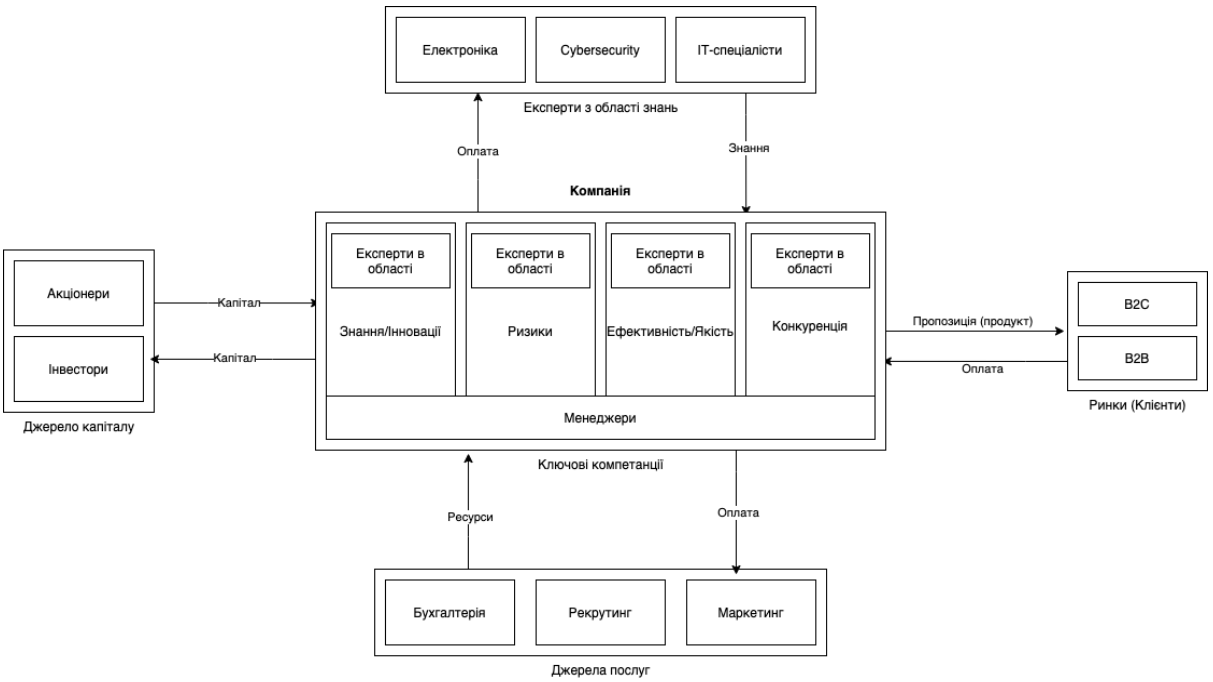


Рисунок 1.4. Бізнес модель підприємства

Основними ринками збуту продукції для підприємства є B2C та B2B ринки, тобто продукти компанії добре оптимізовані для обох ринків.

Джерелами послуг виступають 3 основні джерела, а саме бухгалтерія, яка надає послуги для роботи з грошима, рекрутинг - джерело працівників, а також маркетинг - джерело клієнтів.

Джерелами капіталу виступають акціонери та інвестори, які зацікавлені в розвитку продукта.

Сама компанія складається з 4 напрямків, кожен з яких керується менеджерами. Для кожного з 4 напрямків, а саме, знання/інновації, ризики, ефективність/якість, конкуренція, є експерти, які, власне, розвивають та підтримують певний напрямок.

Для більш чіткого уявлення про всі роботи, які будуть проводитись на виробництві була побудована контекстна діаграма в нотації IDEF0 (рис. 1.5).

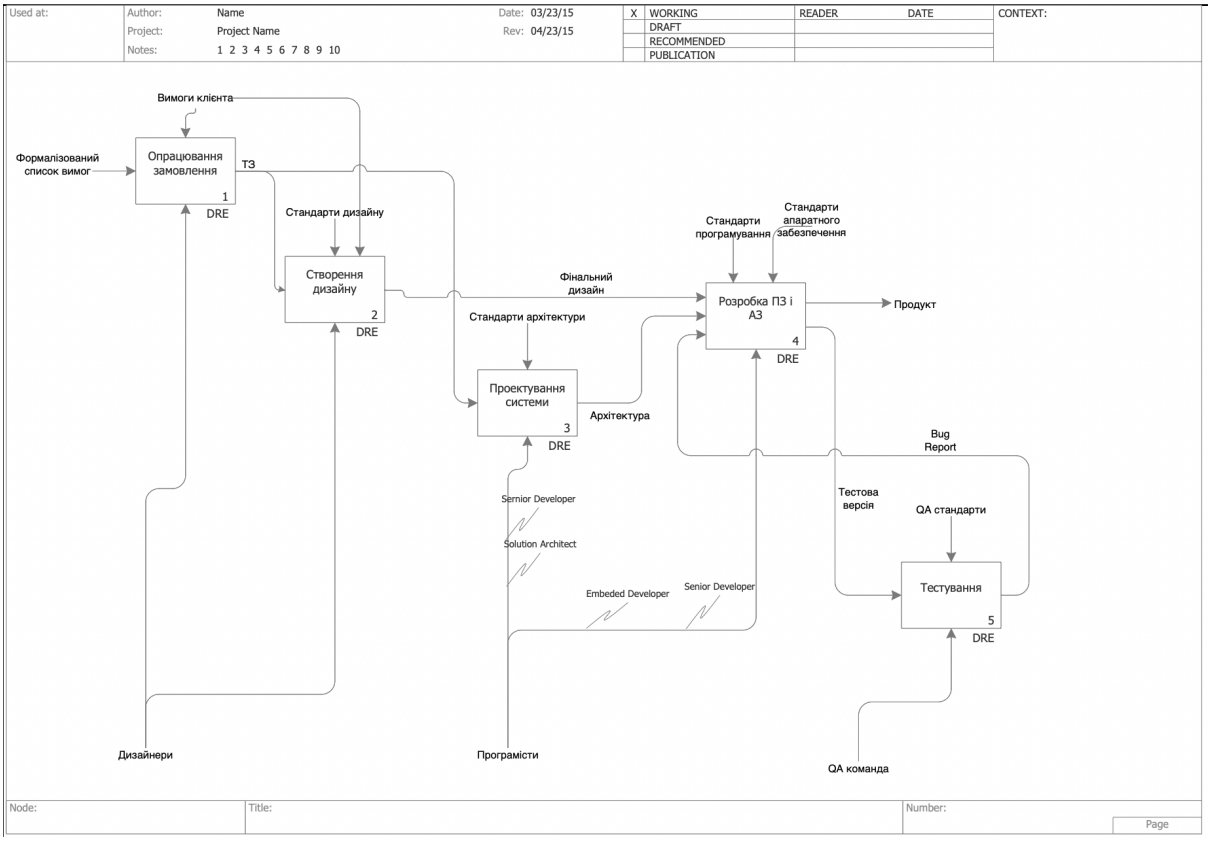


Рисунок 1.5. Контекстна діаграма підприємства в нотації IDEF0

Як можна бачити з діаграми, всього є 5 видів діяльності, а саме:

1. Опрацювання замовлення, де
   1.1. Виконують: дизайнер.
   1.2. Керуються: формалізованим списком вимог.
   1.3. Використовують: вимоги клієнта.
2. Створення дизайну, де:
   2.1. Виконують: дизайнер.
   2.2. Керуються: технічним завданням.
   2.3. Використовують стандарти дизайну, вимоги клієнта.
3. Проєктування системи, де:
   3.1. Виконують: програмісти, а саме Senior Developer, Solution Architect.
   3.2. Керуються: технічним завданням.
   3.3. Використовують: стандарти архітектури.
4. Розробка ПЗ і АЗ, де:

4.1. Виконують: програмісти, Senior Developer, Embedded Developer.

4.2. Керуються: фінальним дизайном, архітектурою.

4.3. Використовують: стандарти програмування, стандарти апаратного забезпечення.

4.4. Отримується: продукт.

5. Тестування, де:

5.1. Виконують: QA команда.

5.2. Керуються: тестовою версією.

5.3. Використовують: QA стандарти.

Також на етапі проєктування обраховано приблизну вартість створення проєкта для замовника, вартість та години роботи (рис. 1.6, 1.7).

Resources

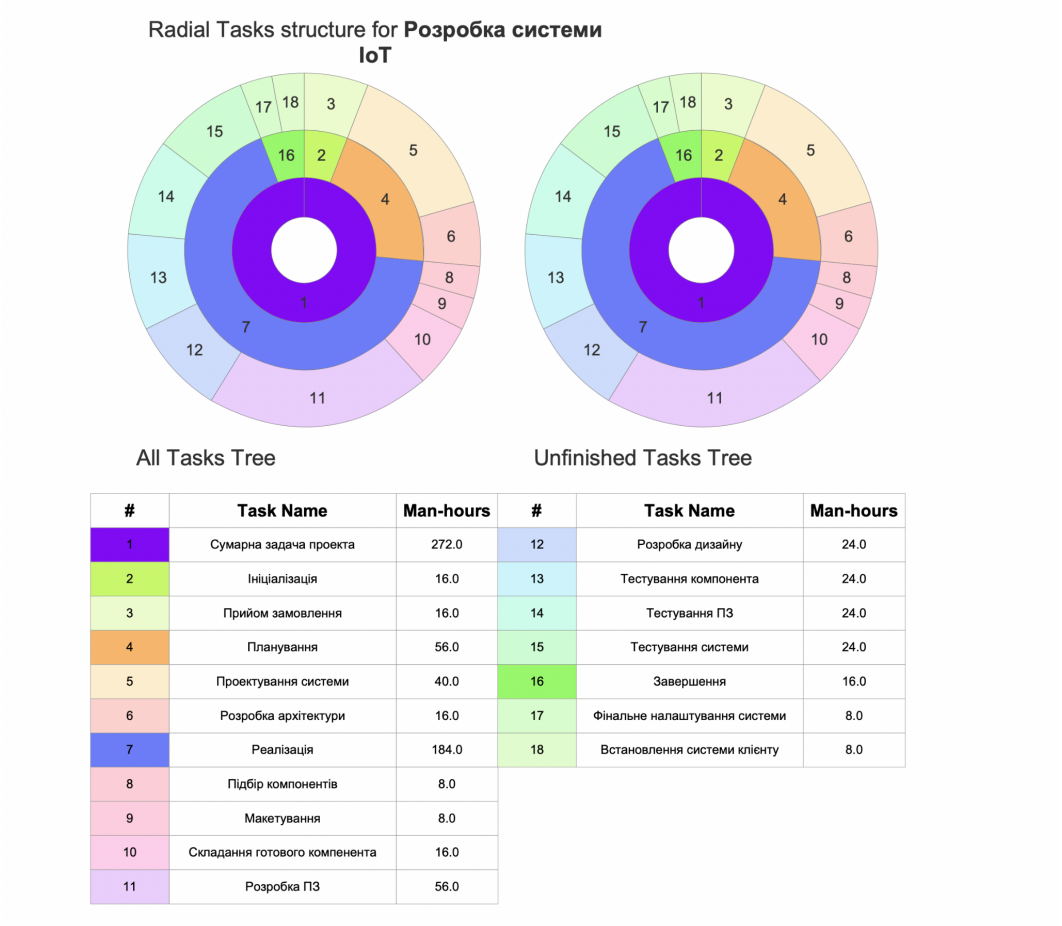
Work

Name	Man Hours	Cost	Cost per time unit
Tester	72	$ 720.00	$ 10/h
Senior Developer	8	$ 200.00	$ 25/h
Solution Architect	48	$ 1,200.00	$ 25/h
Designer	24	$ 360.00	$ 15/h
Developer	56	$ 840.00	$ 15/h
Спеціаліст прийому замовлень	16	$ 80.00	$ 5/h
Embeded Developer	32	$ 480.00	$ 15/h
Технічний спеціаліст	16	$ 80.00	$ 5/h

Material

Name	Usage	Cost	Cost per unit
		Total	
		$ 3,960.00	
		Year-to-Date	
		$ 0.00	

Рисунок 1.6. Вартість проєкта



#	Task Name	Man-hours	#	Task Name	Man-hours
1	Сумарна задача проекта	272.0	12	Розробка дизайну	24.0
2	Ініціалізація	16.0	13	Тестування компонента	24.0
3	Прийом замовлення	16.0	14	Тестування ПЗ	24.0
4	Планування	56.0	15	Тестування системи	24.0
5	Проектування системи	40.0	16	Завершення	16.0
6	Розробка архітектури	16.0	17	Фінальне налаштування системи	8.0
7	Реалізація	184.0	18	Встановлення системи клієнту	8.0
8	Підбір компонентів	8.0			
9	Макетування	8.0			
10	Складання готового компенента	16.0			
11	Розробка ПЗ	56.0			

Рисунок 1.7. Години роботи над проєктом

Отже, як можна бачити з рисунку 1.6, загальна вартість проєкта приблизно дорівнює 3960 доларів, потребує як мінімум 7 спеціалістів та в сумарному 272 години.

Побудова всіх видів вище описаних діаграм дали чітке представлення про проєкт та шляхи його реалізації.

1.5. Висновки до розділу

Отже, було описано поставлену задачу та виконаний аналіз всіх ключових елементів реалізації даної системи. Розглянуте поняття децентралізованої мережі, проведений аналіз даного типу мережі у порівнянні з централізованою мережею. Були описані сфери застосування децентралізованої мережі, включаючи сферу IoT.

РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ

2.1. Декомпозиція задачі проєктування системи

Децентралізована мережа має свою специфіку складових компонентів та процесу розробки. Для розробки даної системи була вибрана гібридна модель децентралізованої мережі, яка дозволяє використовувати допоміжні сервіси та має простіший алгоритм реалізації. Також таку модуль було вибрано з поглядом на подальший розвиток та підтримку системи з урахуванням можливості додавання до системи всіх компонентів, які б забезпечували кращу захищеність та підтримку сторонніх сервісів, наприклад, підключення технології Blockchain. Готова система може без труднощів використовуватись у бізнес цілях та ентерпрайз рішеннях.

IoT система для керування розумним будинком на основі децентралізованої системи, що проєктується, складатиметься з декількох допоміжних сервісів для менеджменту сертифікатів та вузлів, а також спеціального програмного коду, який реалізує клієнт/сервер на стороні вузла.

Загалом розроблену систему можна розбити на 3 основні програмні модулі (рис. 2.1):

- програмне забезпечення допоміжних сервісів;
- програмне забезпечення вузла;
- апаратне забезпечення модуля датчиків.

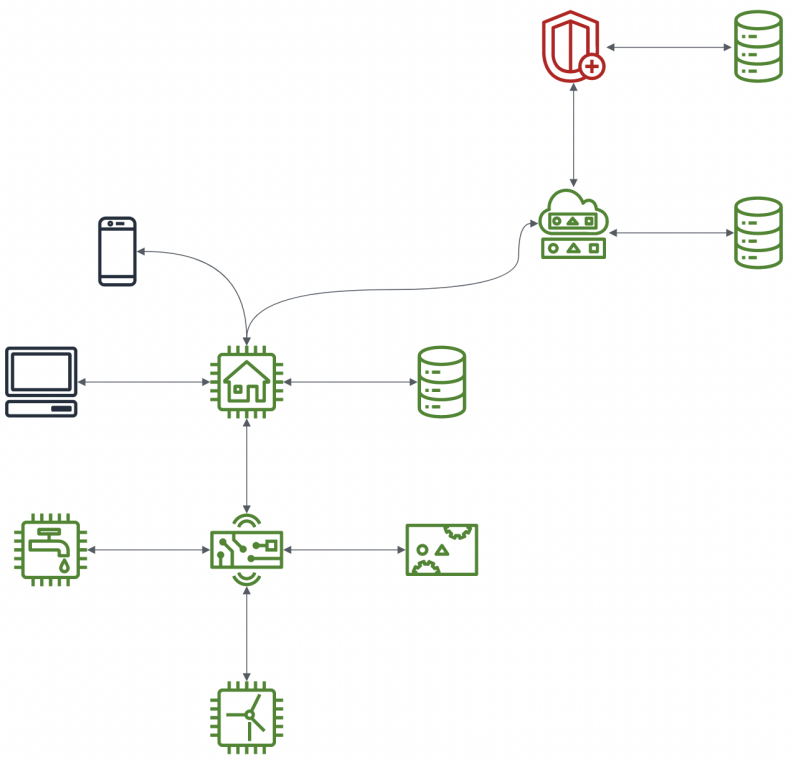


Рисунок 2.1. Схема підключення компонентів системи

Також за результатами використання децентралізованої мережі система отримала дуже хорошу масштабованість за допомогою якого компоненти можна групувати по кластерах.

Розбивка всіх компонентів та опис їх прямих задач дали чітке розуміння подальшого реалізації системи.

2.2 Опис допоміжних сервісів

Система, що проєктується, складатиметься з двох допоміжних сервісів, які забезпечують основні функції по організації мережі, а саме:

- сервіс сертифікації вузлів - даний сервіс використовується для генерації сертифікатів для нових користувачів та їхнім подальшим менеджментом. Сервіс був введений з оглядом на те, щоб до мережі не могли підключитись неавторизовані вузли, які могли б використовувати дані про знаходження інших вузлів, а також отримувати дані телеметрії користувачів.

Основна частина сертифікату складається з токена, який хешується за допомогою алгоритму SHA-256 та генерується на основі дати та часу створення сертифікату, назви та адреси користувача. Було обране саме хешування даних, а не шифрування, тому що захешовані дані не можна повернути в початковий вигляд, що забезпечує надійність конфіденційних даних користувача. Також сертифікат складається з дати та часу створення, ID, поля активації сертифікату, дати та часу активації сертифікату та поля деактивації сертифікату.

Сертифікаційний сервіс має захист від дублювання сертифікату для того щоб не можна було використовувати два одинакових сертифікати різними користувачами. Для прикладу роботи даного механізму можна навести наступну ситуацію: користувач отримав та активував сертифікат і помилково виклав його у відкритий доступ, сертифікат побачив зловмисник і вирішив також його використати, але при першій спробі включення вузла, він передає цей сертифікат на перевірку, де сертифікаційний сервіс знайде дублікат, після чого заблокує копію сертифіката і зловмисник не зможе підключитись до мережі.

Також даний сервіс дасть змогу в майбутньому монетизувати програмний продукт, а саме планується ввести кілька типів сертифікатів, які відрізнятимуться один від одного наборами різних доступних функцій, внутрішніми обмеженнями, вартістю та періодом дії;

- NHS (Next Hop Service) - допоміжний сервіс, який реєструє нові вузли, керує ними, а також видає оптимальні вузли для підключення та зберігання даних запитуваним вузлом на основі метрик. Даний сервіс буз розроблений у зв’язку з проблемою знаходження вузлів один одного, а також сервіс вирішує проблему оптимального вибору вузла для зберігання даних.

  Коли новий вузол включається, йому потрібно сповістити про це NH сервісу, після відправлення ним відповідного запиту NH сервіс зберігає дані вузла у локальній базі даних та вираховує метрику якості та швидкості

зв’язку, також вузлу потрібно надіслати свій сертифікат, який сервіс відправляє на перевірку до сертифікаційного сервісу, якщо наданий сертифікат проходить перевірку. Після цього вузлу потрібно надіслати запит на готовність отримати вузла (сервера) для зберігання телеметрії та іншого роду даних. NH сервіс по цьому запиту починає пошук з бази даних хост, який має найкращу метрику, якщо такий знаходиться - сервіс надсилає дані хоста до хоста відправника.

2.3. Опис вузла

Програмне забезпечення вузла відповідає за прийом даних з модуля датчиків через MQTT брокер та прийом даних з іншого вузла, та складається з кількох частин, а саме: ядра, API, графічного інтерфейса у вигляді WEB-додатка та бази даних.

Ядро - це основа вузла, де міститься вся логіка роботи з даними та запуску іншого програмного забезпечення. По своїй сутності ядро єдемоном, який слідкує за справністю роботи всіх частин хаба. Коли користувач запускає хаб, менеджер запуска попросить користувача вказати відповіді на певні конфігураційні питання, після чого ядро почне процедуру ініціалізації вузла в мережі та паралельний запуск MQTT брокера, WEB-додатку, PostgreSQL та API.

2.4. Опис модуля датчиків

Апаратне забезпечення модуля датчиків використовується для зняття телеметрії з підключених датчиків, а також їхнього керування. На момент розробки системи апаратне забезпечення було написане для контролера NodeMCU v.3 на базі чипа ESP8266, а в якості складових модуля були використані датчики: якості повітря, температури, детекції руху. Також вся комунікація модуля з хабом відбувається через протокол MQTT.

2.5. Алгоритм роботи системи

Передбачається, що система буде максимально легка у налаштуванні та не потребуватиме від користувача навичок програмування чи знань роботи мереж. Загальний алгоритм налаштування та роботи системи виглядає наступним чином:

1. Для початку роботи з системою, користувачу потрібно отримати сертифікат від адміністратора мережі, після чого користувач може продовжити подальше налаштування та використання системи.
2. Після отримання сертифікату, користувачу потрібно запустити розроблене програмне забезпечення та заповнити відповідні пункти у вікні “Налаштування” та запуску системи.
3. Якщо всі пункти були заповнені валідними даними, система починає запускати всі сторонні і внутрішні сервіси, які потрібні для правильної роботи системи, зокрема:
    a. Відправляє сертифікат на перевірку до сервіса сертифікації, якщо сертифікат проходить всі перевірки на валідність та дублікат, система переходить до наступного пункту або ж блокує запуск.
    b. Після перевірки сертифіката, система надсилає запит до NH сервісу для реєстрації нового вузла у мережі та отримання списку доступних вузлів, на яких зберігатимуться дані вузла відправника:
        i. Якщо доступні вузли не знайдені, сервіс надсилає спеціальну інструкцію для тимчасового зберігання даних на самому вузлі.
        ii. Якщо сервіс знайшов доступні вузли, він надсилає відповідний список, а вузол переходить на наступний крок.
    c. Ядро сервісу запускає MQTT брокер, базу даних PostgreSQL, API та веб-додаток на вузлі.
    d. Вузол відкриває канали для отримання даних з сторонніх вузлів та подальшого їх зберігання, а також канал для надсилання даних телеметрії з підключеного модуля.

4. Для подальшого використання, користувачу потрібно налаштувати інтерфейс системи для зручного моніторингу та керування розумним будинком, зокрема перейти за посиланням та виконати наступні кроки:
    a. Створити акаунт.
    b. Додати пристрої, з яких система буде зчитувати дані телеметрії.
    c. Створити дашборд.
    d. Додати на дашборд необхідні віджети для керування та перегляду даних.

Відповідно до описаного алгоритма роботи системи, користувачу необхідно виконати всього 4 кроки для повного налаштування системи.

2.6. Висновки до розділу

В даному розділі система була розбита на основні та допоміжні складові, був зроблений детальний опис кожного з компонентів, а також детально був описаний алгоритм роботи системи, що проєктується, в цілому.

РОЗДІЛ 3. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1. Вибір програмного забезпечення та інструментарію для розробки системи

Для створення системи керування розумним будинком на базі децентралізованої мережі було використано багато технологій та програмного забезпечення, які були особливо потрібні для якісної та відмовостійкої роботи.

Для фронтенду використані такі технології:

1. HTML5 - це мова розмітки документів, яка на сьогоднішній день широко використовується для розмітки веб-сторінок []. Дана мова розмітки буде використовуватись для розмітки всіх сторінок веб-додатка.
2. CSS3 - це мова, яка використовується для візуалізації окремих елементів веб-сторінки та сторінки в цілому. Саме на основі цієї мови реалізований фреймворк Tailwind CSS, який буде описаний нижче.
3. JavaScript - динамічна, об'єктно-орієнтована прототипна мова програмування. Найчастіше використовується для створення користувацьких сценаріїв поведінки веб-додатка та обробку дій користувача. В даній роботі, JavaScript буде використовуватись разом з фреймворком Vue 3, що дозволить використовувати високорівневе API для керування елементами веб-сторінки.
4. Vue 3 - це прогресивний фреймворк для створення інтерфейсів користувача. Він дозволяє створювати SPA додатки любої складності з нуля. Також по своїй архітектурі даний фреймворк може бути інтегрований у вже готовий додаток, що дозволяє поступово переписувати його на Vue. Даний фреймворк має багато зручного у використанні функціоналу у порівнянні з конкурентами. У цій роботі за допомогою Vue побудована вся графічна складова системи.
5. Vuex - це допоміжна бібліотека для керування глобальним станом додатку, який написаний на Vue 3. Дану бібліотеку зручно використовувати, коли

додаток має складні структури даних його стану та дозволяє централізовано керувати цим станом.

6. Tailwind CSS - фреймворк для CSS3, який дозволяє максимально пришвидшити та спростити процес розробки веб-додатку, зокрема його візуальної складової. Tailwind CSS містить вже готові класи імен, які можна використовувати як окремі компоненти для візуалізації певного елемента, також при використанні даних імен класу - фреймворк забезпечує максимальну сумісність з усіма веб-браузерами.

Для бекенду були використані такі технології:

1. Python 3 - інтерпретована об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією [8]. Розроблена в 1990 році Гвідо ван Россумом. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднання наявних компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій, так і у вихідній формі на всіх основних платформах [11]. В мові програмування Python підтримується кілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована.
2. Django v.3 - високорівневий відкритий Python-фреймворк для розробки веб-систем [9]. Названо його було на честь джазмена Джанго Рейнхардта (відповідно до музичних смаків одного із засновників проєкту). Даний фреймворк працює за моделлю MVT, що дозволяє максимально спростити та прискорити розробку веб-сервісу. Також він містить ORM систему, яка складається з описання класів та прив’язки їх до таблиць бази даних, таким чином розробник отримує максимальну зручний та простий інструмент керування сутностями системи. Для даної системи Django використовувався разом з Django Rest Framework, який дозволяє розробляти REST API.

3. PostgreSQL - об'єктно-реляційна система керування базами даних (СКБД). Є альтернативою як комерційним СКБД (Oracle Database, Microsoft SQL Server, IBM DB2 та інші), так і СКБД з відкритим кодом (MySQL, Firebird, SQLite) [10]. Дана база даних має максимальну швидкодію у порівнянні з конкурентами, а також має найбільший інструментарій для керування, створення, редагування баз-даних. Також PostgreSQL має систему розширень, яка дозволяє додати необхідний функціонал для максимально зручного та швидкого розгортання бази-даних.

Також з програмного забезпечення було використано:

1. Visual Studio Code (настроєний для розробки на Python-ні та Django) - засіб для створення, редагування та написання коду на різних мовах програмування. Даний текстовий редактор розроблений компанією Microsoft та підтримує всі платформи, зокрема Linux, Windows та MacOS. За допомогою системи плагінів дану програму можна з легкістю трансформувати у потужний ide, який не буде поступатись функціональністю платним конкурентам.
2. Arduino IDE - програмне забезпечення Arduino (IDE) з відкритим кодом, яке дозволяє розробляти та налагоджувати програми для плат Arduino. За допомогою системи плагінів та додаткових бібліотек список для програмування суттєво збільшується та охоплює найпопулярніші плати.
3. Figma - кросплатформний, векторний графічний редактор від компанії Figma. Даний сервіс максимально зручний для створення UI/UX дизайну програм, веб-додатків та мобільних додатків. Сервіс містить систему плагінів, яка дозволяє розширювати стандартний функціонал та зробити з Figm-и універсальний та єдиний інструмент для створення дизайнів.
4. MQTTX - це елегантний кросплатформний настільний клієнт MQTT 5.0 з відкритим кодом EMQ, який підтримує macOS, Linux, Windows. Дане програмне забезпечення має унікальний та максимально продуманий дизайн для зручного користування технологією MQTT. Вся унікальність

заключається у дизайні програми, яка нагадує чат між топіками, де одна сторона служить для відображення ваших повідомлень, а інша для відображення повідомлень від топіків на, які ви підписались.

Загальне програмне забезпечення:

1. Git - розподілена система керування версіями файлів та спільної роботи. Проєкт створив Лінус Торвальдс для управління розробкою ядра Linux, а сьогодні підтримується Джуніо Хамано. Git є однією з найефективніших, надійних і високопродуктивних систем керування версіями, що надає гнучкі засоби нелінійної розробки, що базуються на відгалуженні і злитті гілок.
2. GitHub - один з найбільших та найпопулярніших веб-сервісів для спільної розробки програмного забезпечення. Він активно розвивається та вносить зручніші інструменти для спільної розробки. Саме цей сервіс використаний для зберігання коду системи.

3.2. Розробка бібліотеки класів з використанням мови програмування Python

На етапі проєктування розроблені моделі класів та їхні взаємодії між собою для усіх компонентів системи.

Django дає дуже гнучку систему опису та керуванням моделей. Можна виділити три основних типи класів:

- model класи;
- form класи;
- generic view класи.

Django працює за моделлю MTV (Модель-Шаблон-Вид) [11], щоб пояснити принцип проєктування можна розглянути наступний приклад. Отже, допустимо, що потрібно створити сайт новин, на якому будуть відображатись статті. Для його реалізації потрібно створити модель статті, в якій будуть поля, що описують модель, також за необхідністю можна написати функції, які будуть розширювати функціональність моделі.

Наступний крок, це створення функції, яка буде керувати поведінкою моделі, а саме, прив’язуватись до конкретної моделі, робити запити до бази даних, а також якщо потрібно, щоб була можливість видаляти, обновляти та створювати екземпляр моделі, то прив’язку форми, яка може бути двох типів:

- форма, яка має прямий зв’язок з моделлю (саме ця потрібна в нашому випадку);
- форма, яка не відноситься до сторонніх структур даних (незалежна), вона використовується в більшості випадків для створення GET запитів.

Згідно офіційної документації View functions, потрібно старатись як можна менше навантажувати. Саме для цього були створені Generic View Templates, які значно спрощують роботу з View, а також економлять час написання коду.

Generic View являють собою python клас, в якому основні поля це:

- template_name;
- model;
- get_object_id() - використовується за необхідності, коли нам потрібно звернутись до конкретної моделі за її ID.

Також є велика різновидність Generic View, які налаштовані під будь-які операції з моделями. Варто зазначити, що можна створювати власні Generic View для більш екзотичних операцій.

Наступний крок, це реєстрація View функцій в файл під назвою “urls.py”, який займається реєстрацією, контролем та маршрутизацією наших View функцій. Також слід зазначити простір імен для полегшення конторелем маршрутизації, а саме для уникнення хардкодних посилань.

Далі потрібно створити шаблон, який буде відображатись на екранах користувачів. Шаблон являє собою HTML файл з static файлами, функціями, а також запитами до баз даних.

Django дає можливість користувачу створювати гнучку систему templating, яка дозволяє підвищити швидкодію сайту. Головною особливістю являється

наслідування html файлів, які слугують обгорткою для інших файлів. Таким чином можна створити лишень один файл, який буде унаслідуватись іншими сторінками, якщо у них використовуються функціонал нашої обгортки.

На мою думку, це дуже гнучка система, яка дає можливість економити час на написання фронтенду, а також для покращення реактивності веб-сайту.

Також всі створені класи були протестовані. Для тестування класів написані юніт тести, які дозволяють тестувати найменшу одиницю програми.

Модульне тестування (англ. Unit testing) — це метод тестування програмного забезпечення, який полягає в окремому тестуванні кожного модуля коду програми. Модулем називають найменшу частину програми, яка може бути протестованою. У процедурному програмуванні модулем вважають окрему функцію або процедуру. В об'єктно-орієнтованому програмуванні — метод.

Модульні тести, або unit-тести, розробляють в процесі розробки програмісти та, іноді, тестувальники білої скриньки (white-box testers).
Зазвичай unit-тести застосовують для того, щоб упевнитися, що код відповідає вимогам архітектури та має очікувану поведінку.

Django має свою систему створення та написання unit тестів, які мають потужну систему додаткових класів, які можна наслідувати в користувацьких unit тестах, а також дозволяють спростити процес їх написання.

Також для додаткового тестування кожен клас перевірявся на несумісні типи даних та їхню поведінку під час їх отримання. В результаті цього були написані exceptions, які запобігають отриманню сервером помилкових даних.

3.2.1. Сертифікаційний сервіс

Сертифікаційний сервіс важлива складова системи, яка відповідає за сертифікати користувача. Він складається з наступних класів:

- клас користувача - цей клас відповідає за зберігання та керування структурою даних користувача, для сертифікаційного сервісу користувачем

вважається адміністратор, який може створювати сертифікати. Всі поля, які має даний клас наведені в таблиці 1 додатку Ґ;

- клас сертифіката - відповідає за зберігання структури сертифіката та являється в даному сервісі основною сутністю, яка використовується для забезпечення правового доступу до основного сервісу. Всі поля, які має даний клас наведені в таблиці 2 додатку Ґ.

API сертифікаційного сервісу має всі необхідні кінцеві вузли для ефективного керування сертифікатами та користувачами, а саме:

- /certificates/ : GET;
- /certificates/<id> : GET;
- /certificates/<id> : DELETE;
- /certificates : POST;
- /certificates/activate-certificate : POST;
- /certificates/check-certificate : POST;
- /users/ : GET;
- /users/<id> : GET;
- /users/<id> : PATCH;
- /user/<id> : DELETE;
- /user/login : GET;
- /users/sign-up : POST.

3.2.2. NH сервіс

Next Hop сервіс відіграє центральну роль всієї мережі, керуючи вузлами та маршрутами до них. Він складається з наступних класів:

- клас користувача - відповідає за зберігання структури користувача, а також, як і сутність користувача в сертифікаційного сервісу, являється адміністратором. На відміну від адміністратора в сертифікаційного сервісу, дії адміністратора даного сервісу максимально обмежені. Всі поля, які має даний клас наведені в додатку Ґ, в таблиці 1;

- клас вузла - відповідає за зберігання структури вузла мережі та являється основною сутністю даного сервіса. Всі поля, які має даний клас наведені в додатку Ґ, в таблиці 3.

API Next Hop сервісу має всі необхідні кінцеві вузли для ефективного керування вузлами та користувачами, а саме:

- /hubs : GET;
- /hubs : POST;
- /hubs/<id> : GET;
- /hubs/update-hub : PATCH;
- /hubs/get-available-hubs : GET;
- /users : GET;
- /users/<id> : GET;
- /users/<id> : PATCH;
- /user/<id> : DELETE;
- /user/login : GET;
- /users/sign-up : POST.

3.2.3. Ядро

Ядро являється демоном, який слідкує за виконанням всіх базових функцій вузла, а також керує рухом даних з вузлів та від вузла. Воно складається з наступних класів:

- клас BlityNet - даний клас зберігає та керує запуском всіх додаткових та сторонніх сервісів, програмного забезпечення, а також слідкує за справною роботою вузла в мережі. Всі поля, які має даний клас наведені в додатку Ґ, в таблиці 4;
- клас Forwarder - відповідає за прийом даних від стороннього вузла, їх обробку та зберігання. Всі поля, які має даний клас наведені в додатку Ґ, в таблиці 5;

- клас Node - даний клас являється центральною сутністю всієї системи. Він містить всю логіку по отриманню даних інших вузлів, розподілення даних, керування даними, а також сценарії поведінки при аварійних ситуаціях. Всі поля, які має даний клас наведені в додатку Ґ, в таблиці 6.

3.2.4. API девайса

API девайса реалізує всі необхідні сутності та кінцеві вузли, які використовує WEB-додаток. API складається з наступних класів:

- клас Dashboard - відповідає за зберігання та керування структурою дашборда, а також керуванням віджетів, які підпорядковуються відповідній сутності класу. Всі поля, які має даний клас наведені в додатку Ґ, в таблиці 7;
- клас Widget - відповідає за зберігання та керування структурою віджета. Загалом даний клас має динамічну структуру, яка залежить від типу віджета, дана особливість дозволяє без проблем оновлювати та створювати нові користувацькі віджети. Всі поля, які має даний клас наведені в додатку Ґ, в таблиці 8;
- клас Device - відповідає за зберігання та керування структурою девайса. Даний клас містить всю необхідну інформацію для підключення до девайса та отримання з нього телеметрії. Всі поля, які має даний клас наведені в додатку Ґ, в таблиці 9;
- клас Value - відповідає за зберігання та керування структурою телеметрії девайса. Даний клас також може мати досить динамічну структуру, яка дозволяє для кожного окремого девайса отримувати унікальну та необхідну структуру даних телеметрії. Всі поля, які має даний клас наведені в додатку Ґ, в таблиці 10.

3.3. Розробка бібліотеки компонентів для графічного інтерфейсу з використанням фреймворку Vue 3

На етапі проєктування був розроблений дизайн майбутнього сайту з усіма елементами, функціоналом та інтерактивною картою, яка дозволяє визначити ергономіку та правильність майбутнього функціоналу.

Для розробки дизайну усіх сторінок використовувався веб сервіс Figma, який має безліч інструментів будь-якого рівня складності. Також як допоміжний веб сервіс використовувався Freepik, який має величезну базу даних безкоштовних іконок, ілюстрацій та векторних картинок.

Також були розроблені компоненти за допомогою фронтенд фреймворку Vue 3 та допоміжних бібліотек, які полегшують процес розробки складних та комплексних рішень.

Для опису специфіки та методів побудови компонентів для прикладу було взято систему побудови та налаштування дашбордів.

3.3.1. Діаграма компонентів

Основною особливістю веб-додатку являється система побудови гнучких дашбордів, яка дозволяє побудувати дашборд під потреби кожного унікального користувача.

З основних особливостей можна виділити наступні:

- система пересування віджетів;
- кілька типів віджетів;
- можливість додавання нових віджетів, за допомогою стандартного набору складових віджета.

Система пересування віджетів була написана з нуля та не використовує додаткові допоміжні бібліотеки. Вона використовує стандартні засоби розробки мови програмування JavaScript. Система складається з наступних компонентів (які взаємодіють між собою):

1. Board - даний компонент відповідає за побудову та відображення сітки, та складається з наступних складових:

   Пропси:

   - editable : Boolean;
   - widgets : Array.

   Дані:

   - boardElement : Class ;
   - boardHeight : Double;
   - boardWidth : Double;
   - itemCount : Integer;
   - boardGrid : Array;

   Методи:

   - calculateBoardItemCount();

   Життєві хуки:

   - mounted().
2. BaseWidget - даний компонент відповідає за обгортку, яка реалізує базовий функціонал віджета в загальному, та складається з наступних складових:

   Пропси:

   - id : Integer;
   - x : Integer;
   - y : Integer;
   - width : Integer;
   - height : Integer;
   - type : String;
   - errorMessage : String;
   - editable : Boolean.

   Дані:

   - element : Class ;
   - clicked : Boolean;

- startMouseMoveX : Boolean;
- startMouseMoveY : Boolean;
- dist : Integer.

Методи:

- handleMouseUp();
- handleMouseDown();
- handleMouseMove();
- draw();
- onDeleteWidget().

Життєві хуки:

- mounted().

3. MinimalWidget - базовий універсальний віджет, який підходить для відображення телеметрії, а також графічного зображення графіку її зміни, даний компонент складається з наступних складових:

   Пропси:

   - id : Integer;
   - type : String;
   - title : String
   - symbol : String
   - plot : Boolean;
   - fromValue : Boolean;
   - trending : Boolean;
   - device : Integer;
   - editable Boolean.

   Дані:

   - devicesSelect : Array;
   - client : MQTT;
   - values : Array;
   - value : Dynamic;

- pastValue : Dynamic;
- countReceivedMessage : Integer.

Методи:

- updateWidget();
- createClient();

Наглядачі:

- device().

Вираховувані пропси:

- percentage().

Життєві хуки:

- mounted().

3.3.2. Опис компонентів

Фронтенд фреймворк Vue 3 дає надзвичайно зручну та гнучку систему побудови компонентів, яка з коробки дає всі основні інструменти для реалізації найскладнішого функціоналу.

Загалом Vue 3 пропонує ділити компоненти на наступні частини:

- template - html розмітка, яка використовується для опису вигляду компонента, а також фреймворк додає спеціальний синтаксис для реалізації додаткових функцій, до прикладу інтерполяції змінних та багато іншого;
- style - блок CSS розмітки, який дозволяє налаштувати зовнішній вигляд компонента, для реалізації даної системи цей блок не використовувався, так як була використана бібліотека Tailwind CSS, яка базується на використанні назв класів для графічного оформлення елементів;
- script - блок коду, де необхідно описати поведінку компонента на ті чи інші події. Це основний блок компонента, який повністю відповідає за логіку роботи компонента. Vue 3 надає надзвичайно багатофункціональні складові цього блока. Для реалізації системи були використані наступні складові цього блока:

- props - це дані, які передаються до дочірнього компонента з батьківського на основі яких вираховується додатковий стан компонента або безпосередній стан. Також слід зауважити, що Vue не дозволяє змінювати пропси у рантаймі;
- data - це стан компонента, який може змінюватись за впливом внутрішніх або зовнішніх подій, на відміну від пропсів ці дані можуть змінюватись;
- methods - це методи, які виконуються на реакцію зовнішніх подій, до прикладу можна навести: клік мишки, рух мишки, підтвердження відправки форми, фокусу мишки і багато інших. Тобто цей блок використовується для опису дій компонента у відповідь на зовнішню подію;
- watch - це функція, яка виконується, коли змінюється змінна компонента. Даний функціонал зручно використовувати, коли потрібно відслідковувати зміну певного стану і викликати відповідну логіку;
- lifecycle hooks - це так звані хуки життєвого циклу компонента, які будуть виконуватись при певному життєвому етапі компонента. Це надзвичайно корисний інструмент, який використовується як правило для взаємодії компонента та API. Загалом можна виділити наступні життєві цикли компонента:
    - setup - використовується тільки з Composition API і в реалізації даній системі не використовувався;
    - beforeCreate - викливається перед тим, як компонент буде створений;
    - created - викликається, коли компонент створений;
    - beforeMount - викликається перед тим, як компонент буде вставлений та вдрендерений в дереві компонентів;

        - mounted - викликається, коли компонент включений в деремо компонентів;
        - beforeUnmount - викликається перед тим, як компонент буде видалений з дерева;
        - unmounted - викликається, коли компонент видалений з дерева компонентів.
    - computed - це функція, яка викликається, коли потрібно виконати додаткову логіку над пропсами компонента. В документації до фреймворка вказується, що саме їх потрібно використовувати при додаткових обрахуваннях, а не встроєну інтерполяцію в шаблоні.

Також через велику кількість компонентів та комплексності їхнього стану була використана допоміжна бібліотека Vuex, яка відповідає за централізоване керування станом веб-додатка. Основною ідеєю бібліотеки є створення централізованого сховища стану та реалізації методів, за допомогою яких та тільки них можна змінювати стан додатку.

3.3.3. Тестування бібліотеки компонентів

Розробляючи стратегію тестування програми Vue, потрібно використовувати такі типи тестування:

- Unit: перевіряє, що вхідні дані до певної функції, класу або компонування викликають очікуваний результат або побічні ефекти.
- Component: перевіряє, чи компонент монтується, відтворює, чи можна взаємодіяти з ним і веде себе належним чином. Ці тести імпортують більше коду, ніж модульні тести, є складнішими та потребують більше часу для виконання.
- End-to-end: перевіряє функції, які охоплюють декілька сторінок, і робить реальні мережеві запити щодо робочої програми Vue. Ці тести часто передбачають створення бази даних або іншого бекенда. Кожен тип тестування відіграє певну роль у стратегії тестування веб-додатку.

Для реалізованої системи були написані два типи тестування, це юніт тестування та end-to-end. Component тестування було опущене через складність та великі часові витрати реалізації.

Модульні тести написані для перевірки того, що невеликі ізольовані одиниці коду працюють належним чином. Модульний тест зазвичай охоплює одну функцію, клас, компонований або модуль. Модульні тести зосереджені на логічній коректності і займаються лише невеликою частиною загальної функціональності програми. Вони можуть висміювати великі частини середовища вашої програми (наприклад, початковий стан, складні класи, сторонні модулі та мережеві запити).

Зазвичай це звичайні модулі JavaScript / TypeScript, не пов'язані з Vue. Загалом, написання модульних тестів для бізнес-логіки в додатках Vue суттєво не відрізняється від програм, що використовують інші фреймворки [12]. Є два випадки, коли потрібно робити модульне тестування специфічних функцій Vue:

- Composition API.
- Component.

Хоча модульні тести надають розробникам певний ступінь впевненості, модульні та компонентні тести обмежені у своїх можливостях забезпечити цілісне охоплення програми під час розгортання у виробництві. В результаті наскрізні тести (E2E) забезпечують висвітлення того, що, можливо, є найважливішим аспектом програми: що відбувається, коли користувачі дійсно використовують ваші програми.

Наскрізні тести часто виявляють проблеми з вашим маршрутизатором, бібліотекою керування станом, компонентами верхнього рівня (наприклад, додатком чи макетом), загальнодоступними активами чи будь-якою обробкою запитів. Як зазначено вище, вони виявляють критичні проблеми, які неможливо зловити за допомогою модульних або компонентних тестів.

Наскрізні тести не імпортують код програми Vue, а натомість повністю покладаються на тестування програми шляхом навігації по цілих сторінках у реальному браузері.

Наскрізні тести перевіряють багато шарів у програмі. Вони можуть націлюватися на локально створену програму, або навіть на живе проміжне середовище. Тестування щодо проміжного середовища включає не тільки інтерфейсний код і статичний сервер, але й усі пов’язані серверні служби та інфраструктуру, що було досить важливе при розробці системи.

Перевіряючи, як дії користувачів впливають на веб-додаток, тести E2E часто є ключем до підвищення впевненості в тому, чи веб-додаток функціонує належним чином чи ні.

3.3.4. Огляд функціональності веб-додатка

Для доступу до веб-додатку необхідно запустити програмне забезпечення вузла і в браузері перейти за посиланням: http://127.0.0.1:8080 або додати до налаштувань локального DNS файлу доменне ім’я додатка. При відкритті сайту на вашому екрані з’явиться перша сторінка (рис. 3.1):

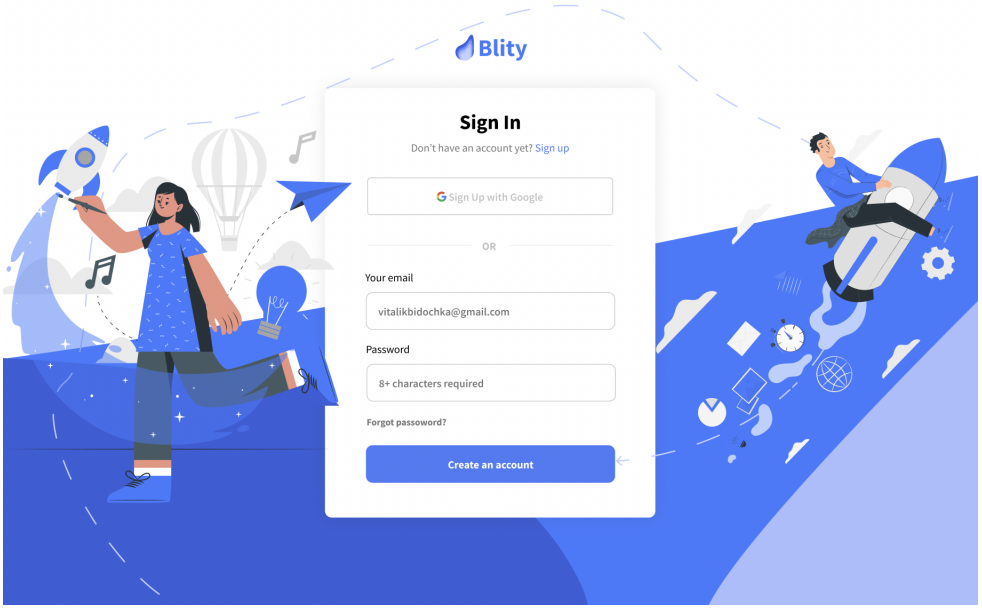


Рисунок 3.1. Сторінка авторизації

На сторінці авторизації знаходиться форма, яка містить наступні поля:

- Текстове поле Your email.
- Текстове поле Password.
- Кнопка Sign In.

Ввівши коректні дані від акаунта користувача і нажавши кнопку Sign In, ви перейдете на сторінку списку дашбордів, яка зображена на рисунку 3.2. Також слід зауважити, що користувач не може самостійно створити собі акаунт у системі, це робить адміністратор.

Кожна сторінка додатка складається з базових елементів: лівої бокової навігаційної панелі ти шапки.

Навігація по веб-додатку робиться за допомогою лівої бокової навігаційної панелі, яка складається з наступних посилань:

- Dashboards - сторінка списку дашбордів.
- Settings - сторінка налаштувань.
- Documentation - сторінка документації системи.
- зміна теми додатку - на даний момент доступна світла та темна тема додатку;
- зміна мови - на даний момент доступна тільки англійська мова;
- довідка.

Шапка додатка містить інформації про користувача, який знаходиться в системі, системні повідомлення, та налаштування профіля. При натисканні на фотографію користувача - з’явиться меню (рис. 3.2).

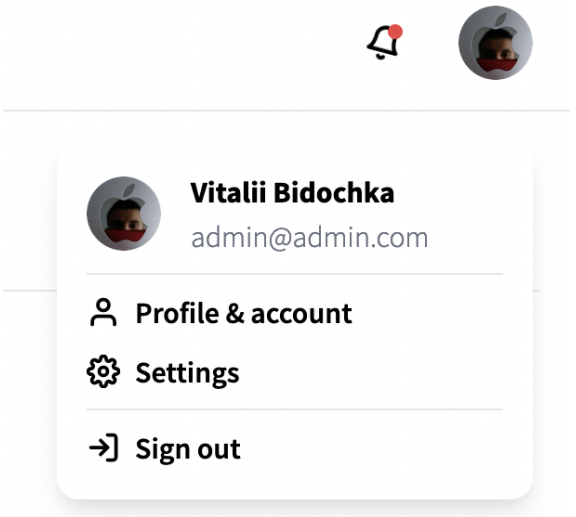


Рисунок 3.2. Спливаюче меню шапки додатку

Спливаюче меню містить наступні посилання:

- Profile & account.
- Settings.
- Sign out.

При натисканні на Sign out - сесія користувача буде завершена, а самого користувача перенесе на сторінку авторизації.

При натисканні на посилання Profile & account, користувача перенесе на сторінку налаштування акаунта користувача (рис. 3.3).

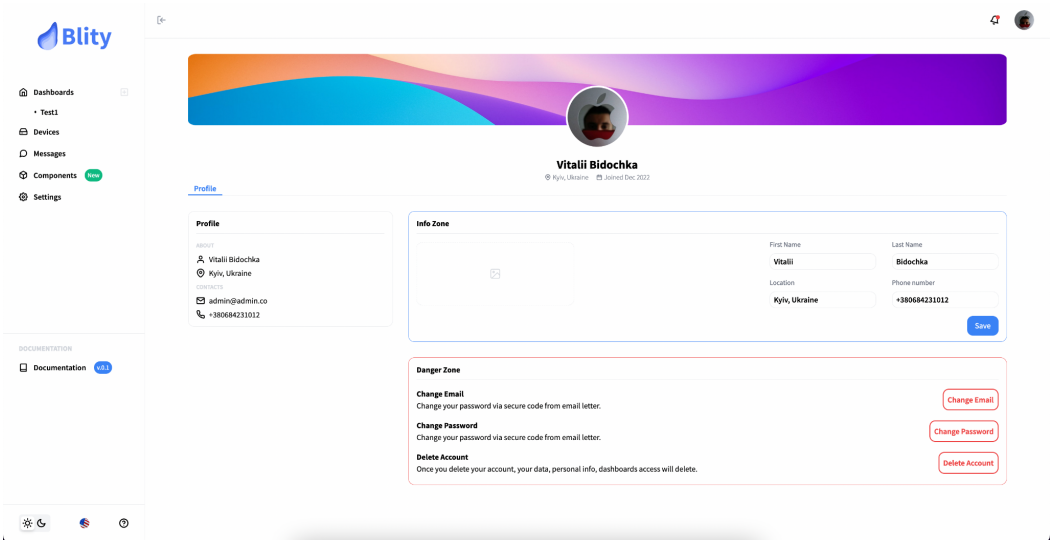


Рисунок 3.3. Налаштування акаунта та профіля користувача

Сторінка налаштування акаунта та профілю користувача складається з 4 основних блоків:

- короткий інформаційний блок про профіль користувача;
- форма зміни інформації про користувача, яка складається з наступних полів:
  - поле завантаження фотографії;
  - First Name.
  - Last Name.
  - Location.
  - Phone number.
- небезпечна зона, яка складається з наступних кнопок:
  - Change Email.
  - Change Password.

- Delete Account.

Сторінка спику девайсів, містить список всіх доданих девайсів до система, а також має інструменти для їхнього керування (рис. 3.4).

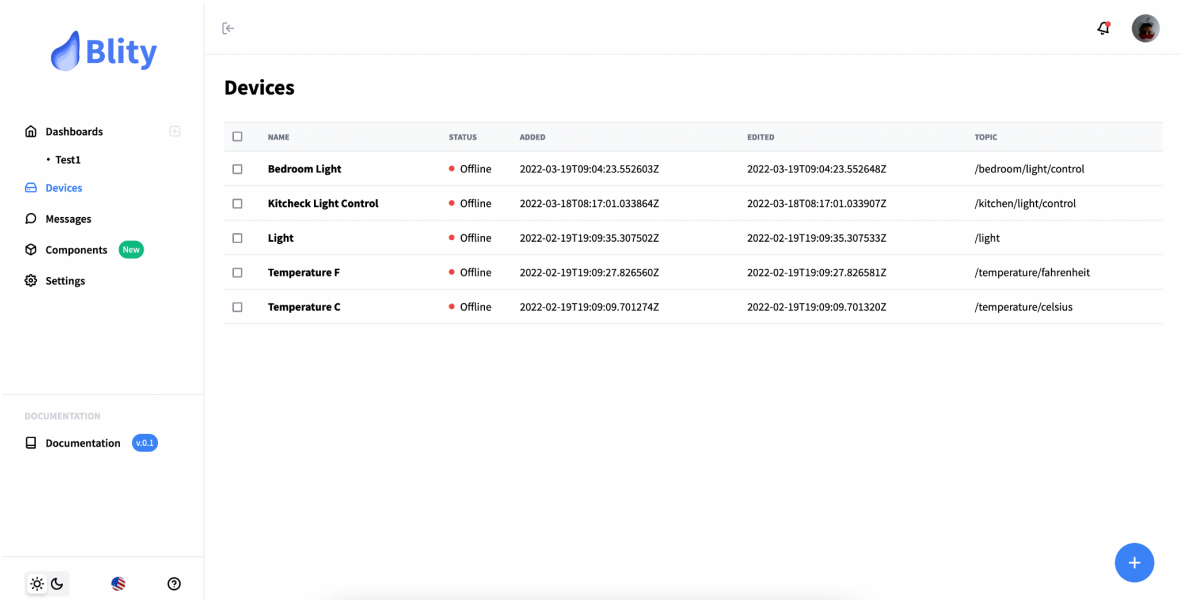


Рисунок 3.4. Сторінка девайсів

Сторінка девайсів складається з панелі керування та списку девайсів. Для додавання нового девайса, користувачу потрібно натиснути на зелену кнопку у нижньому правому куті екрана, після чого з’явиться модальне вікно з формою, яка міститься наступні поля:

- Текстове поле Device Name.
- Текстове поле Topic.

Після заповнення всіх необхідних полів, девайс з’явиться у списку девайсів.

Для видалення девайса, потрібно поставити галочку у чекбоксі навпроти відповідного девайса, після чого з’явиться червона кнопка Delete у верхньому правому куті.

В майбутньому планується додати автоматичне додавання девайсів у систему.

На сторінці списку дашбордів знаходять всі дашборди, які створив користувач, а також коротка інформація по кожному з них (рис. 3.5).

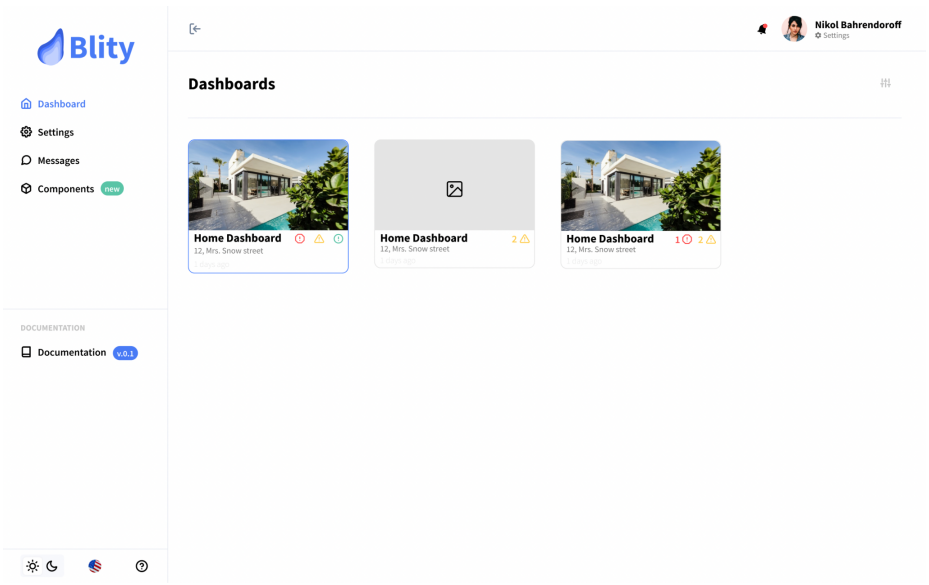


Рисунок 3.5. Сторінка списку дашбордів

З головних елементів на цій сторінці можна виділити наступні:

- блок керування дашбордами;
- список дашбордів.

При натисканні на іконку налаштування дашбордів у верхньому правому куті, користувача перенесе на сторінку налаштування дашбордів (рис. 3.6).

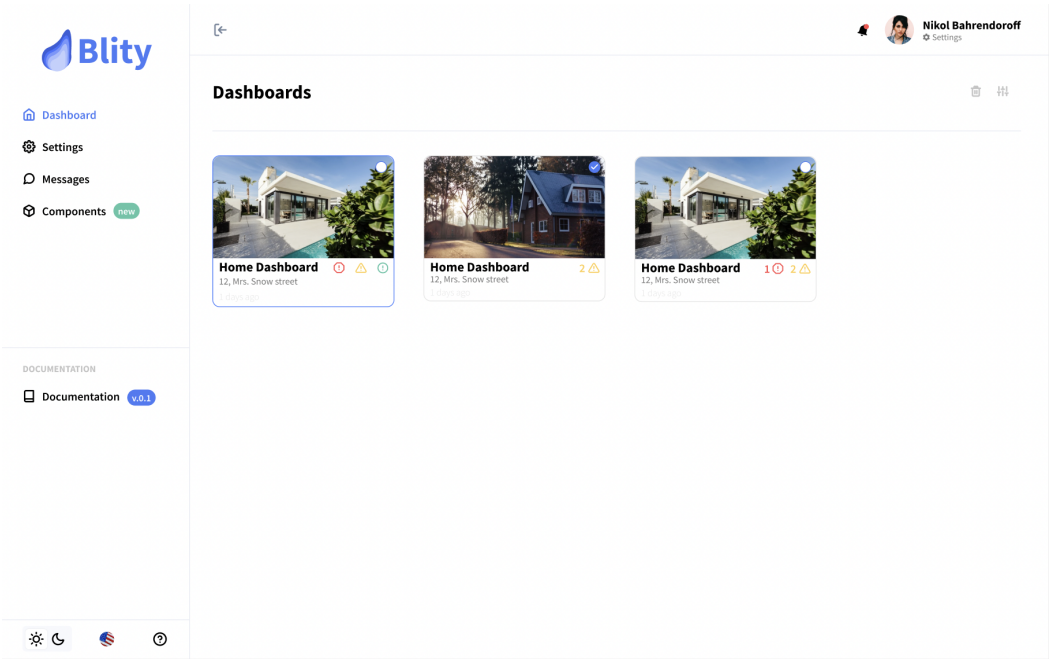


Рисунок 3.6. Сторінка налаштування дабордів

На сторінці керування дашбордами, користувач може видалити, редагувати або додати дашборд.

Для видалення дашборда користувачу необхідно вибрати його, натиснувши на круг у верхньому правому куті блока дашборда, після чого в ньому з’явиться галочка, а також у верхньому правому куті з’явится кнопка видалення.

Для додавання нового дашборда користувача потрібно натиснути на синю кнопку у нижньому правому куті екрану, після чого з’явиться модальне вікно, яке містить форму, яка складається з наступних полів:

- поле завантаження картинки;
- Dashboard Name.
- Dashboard Description.

Після заповнення всіх необхідних полів, користувачу потрібно натиснути на кнопку Add, якщо всі поля заповнені вірно - на сторінці списку дашбордів з’явиться щойно створений дашборд.

Для переходу безпосередньо на самий дашборд, користувачу потрібно два рази натиснути на картку дашборд після чого його перенесе на сторінку деталей дашборда (рис. 3.7).

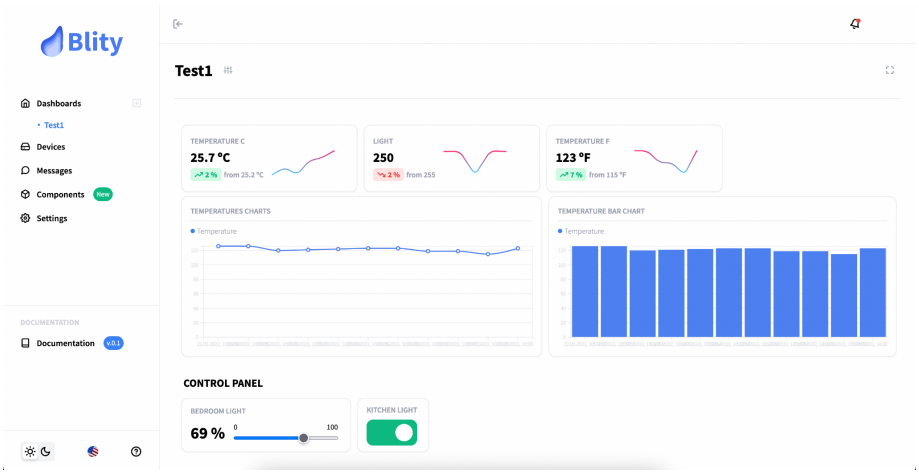


Рисунок 3.7. Сторінка дашборда

Сторінка дашборда складається з панелі керування та віджетів. Панель керування містить кнопку налаштування віджетів та їхнього положення, а також кнопку, при натисканні на яку, додаток перейде у повноекранний режим.

При натисканні на кнопку налаштування дашборда, режим сторінки зміниться з перегляду на редагування та буде мати вигляд (рис. 3.8).

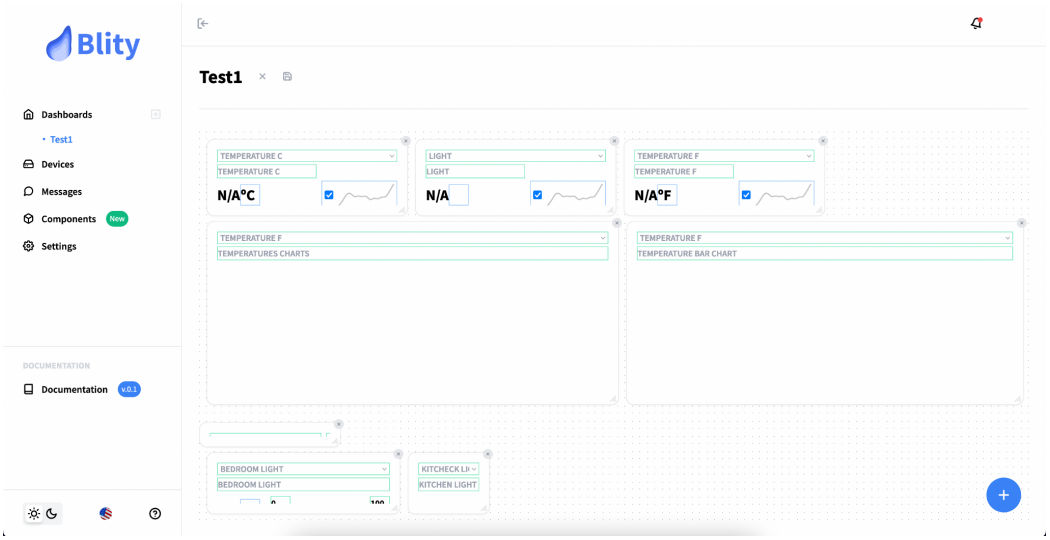


Рисунок 3.8. Сторінка дашборда (режим редагування)

У цьому режимі користувач може додавати нові віджети, налаштовувати існуючі, а також пересувати віджети по заданій області. Таким чином користувач може створювати максимально гнучкі та персоналізовані дашборди під власні потреби.

На даний момент в системі існує два типи віджетів:

- для відображення даних, до них відносяться:
  - Minimal Widget.
  - Line Chart Widget.
  - Bar Chart Widget.
- для керування девайсами, до них відносяться:
  - Slider Widget.
  - Checkbox Widget.
- для декорування, до них відносяться:
  - Text Widget.

Для того щоб додати новий віджет, користувач повинен натиснути на синю кнопку у правому нижньому куті екрана, після чого з’явиться модальне вікно, яке міститься форму з вибором типу віджета (для зручності кожен тип віджета був зображений у вигляді відповідного прев’ю). Після вибору типу віджета, він з’явиться на вільному місці дашборда і буде містити форму з полями, варто

уточнити, що кожен віджет містить індивідуальні поля в залежності від його типу. Для прикладу взяти набір полів форми налаштування Minimal Widget:

- Поле вибору Device.
- Текстове поле Widget Title.
- Текстове поле Special Symbol.
- Чекбокс Plot.
- Чекбокс Percentage Change.
- Чекбокс From Value.

Після заповнення всіх необхідних полів, користувачу потрібно зберегти зміни в дашборді, нажавши на кнопку Save. Якщо системи виявить певні проблеми по валідації полів, то внизу відповідного віджета з’явиться червоне поле з описом помилки. Якщо таких проблем не виявлено, щойно доданий віджет почне процес підключення до девайса та відображення телеметрії.

3.3.5 Дослідження ефективності обчислень

При написанні веб-додатка були обрані оптимальні технології програмування, які є досить оптимізовані та дозволяють в процесі написання коду додатково його оптимізовувати.

Як інструмент вимірювання ефективності використовувався веб-браузер Mozilla Firefox, який має багато засобів для виміру швидкодії.

Для відкриття інструменту аналізу швидкодії потрібно натиснути клавішу F12, після чого внизу екрана відкриється вікно, де будуть знаходитись всі інструменти. Для запуску інструменту Аналізу швидкодії, потрібно натиснути значок секундоміра на панелі інструментів (рис. 3.9).

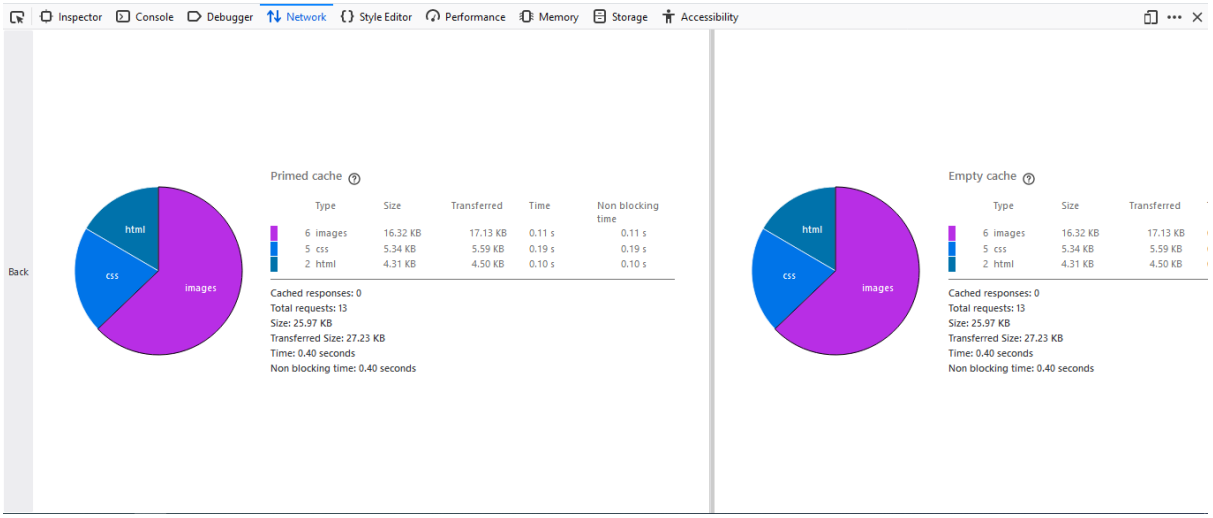


Рисунок 3.9. Швидкодія передачі даних

Після цього Монітор мережі завантажує сайт двічі: один раз з пустим кешем браузера і один раз із запущеним кешем браузера. Це імітує роботу першого разу, коли користувач відвідує веб-додаток, і наступні відвідування. Результати для кожного виконання підсумовуються в таблиці та круговій діаграмі.

Таблиці групують ресурси за типом і показують загальний розмір кожного ресурсу і загальний час, необхідний для їх завантаження. Супровідна до таблиць кругова діаграма показує відносний розмір кожного типу ресурсу.

Наступний інструмент Performance (рис. 3.10). Інструмент дає зрозуміти загальну віддачу веб-додатка, JavaScript і загальне уявлення про розмітку. За допомогою інструменту продуктивності можна створити запис або профіль веб-додатку за певний проміжок часу. Потім, інструмент покаже дії браузера і графік зміни частоти кадрів, поверх профілю, рендеру веб-додатку.

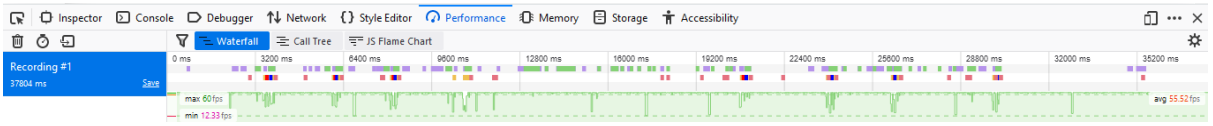


Рисунок 3.10. Швидкодія

За допомогою цього інструмента можна отримати чотири набори інструментів для більш докладного вивчення аспектів профілю:

- Waterfall (Водоспад) показує різні операції браузера, наприклад, виконує макет, JavaScript, перемальовує і збирає сміття.

- Call Tree (Дерево виклику) показує функції JavaScript, в яких браузер провів більшу частину свого часу.
- Flame Chart (Полум'яний Графік) показує стек викликів JavaScript над кінцевою записом.
- Allocations. У цьому поданні відображаються розподілу купи, зроблені вашим кодом під час запису. Це уявлення з'являється тільки в тому випадку, якщо ви відзначили «Записати виділення» в настройках інструменту «Продуктивність».

Наступний інструмент Data Storage (рис. 3.11). Інспектор сховища дозволяє перевірити різні типи сховищ, які веб-додаток може використовувати. В даний час він може бути використаний для перевірки наступних типів зберігання:

Кеш сховищ (нове від версії Firefox 47) - будь-яких DOM кешей створених за допомогою Cache API.

- Cookies - Всіх cookies, створених для сторінок або елементів iframe всередині сторінок.
- Cookies, створених як частина відповіді для Інтернет-дзвінків лише на час роботи ресурсу.
- Local Storage - Всіх local storage елементів сторінок або елементів iframe всередині сторінок.
- Session Storage - Всіх session storage елементів сторінок або елементів iframe всередині сторінок.
- IndexedDB - Всіх IndexedDB баз даних, створених за сторінці, будь-яких елементів iframe всередині сторінок, їх об'єктів і елементів.

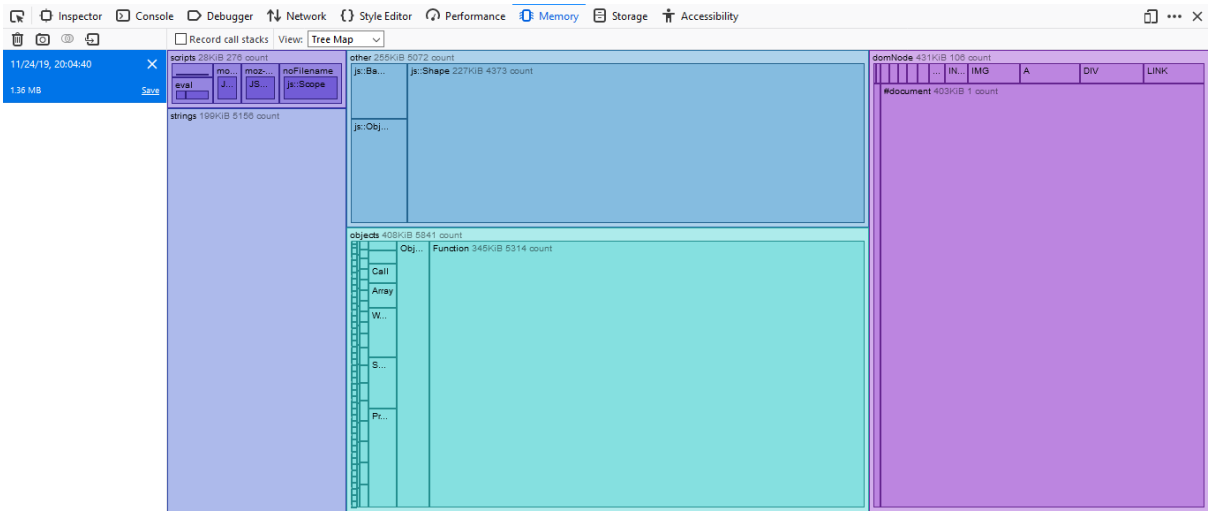


Рисунок 3.11. Швидкодія пам’яті

На даний момент, інспектор зберігання дає уявлення тільки для читання пам'яті.

Спеціальні можливості в веб-розробці (web-доступність) - це забезпечення можливості використання сайтів якомога більшою кількістю людей, включаючи тих, чиї здібності будь-яким чином обмежені. "Інтернет принципово створений для всіх людей на Землі, незалежно від їх апаратних, програмних, мовних, культурних, територіальних, фізичних або розумових здібностей." W3C - Accessibility. Отже, останній інструмент це Accessibility (рис. 3.12), який включає в себе багато додаткових інструментів виміру доступності.

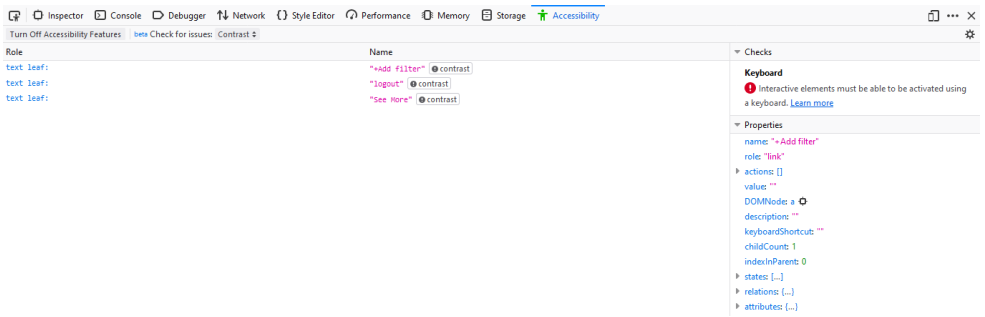


Рисунок 3.12. Доступність

Результати наведені вище дають чітку картину про доступність веб-додатка та дають зрозуміти, де саме потрібно провести доопрацювання.

3.4. Вибір елементної бази та її опис

Контролер - це основний прилад, який використовується для встановлення зв’язку між датчиками та віддаленим MQTT брокером. Тому до вибору контролера потрібно підійти відповідально та прискіпливо, адже через неправильний вибір контролери IoT система може неправильно працювати [13], зокрема можуть виникнути наступні критичні ситуації:

- якщо система стає високонавантаженою, а контролер має малу швидкодію, пакети з інформацією від датчиків, можуть надсилатись з запізненням або взагалі не надсилатись;
- якщо датчик немає реалізації всіх готових бібліотек для побудови повноцінної IoT систем;
- контроллер не містить всіх потрібних портів для коректної роботи з датчиками або має їх дуже мало, що також напряму впливає на масштабованість системи.

На даний момент на ринку існує велике різноманіття мікроконтролерів з різних цінових діапазонів, під різні види задач та з різною швидкодією.

Найбільш популярним мікроконтролером на сьогоднішній день вважається Arduino Uno R3 (рис. 3.13), який використовується у більшості випадків початківцями, але контролер може використовуватись для великих проектів. Виробляється даний контролер в Італії на чипі ATmega328P, з частотою 16 МГц, флеш пам’яттю 32 Кб. З плюсів даної плати можна відмітити багату кількість входів та виходів, а саме: 14 цифрових (6 з яких може працювати в режимі ШІМ); 8 аналогових. З недоліків можна зазначити низьку швидкодію у порівнянні з платами конкурентів, великі габарити (68 х 53 х 15 мм), що не дозволяє використовувати її для реалізації невеликих пристроїв, та високу ціну.

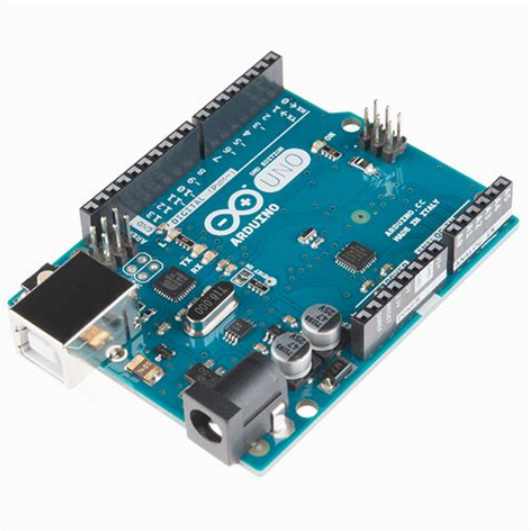


Рисунок 3.13. Arduino Uno R3

Наступний по популярності контролера це Arduino Nano (рис. 3.14), який використовується як початківцями так і досвідченими розробниками та дозволяє реалізовувати проекти будь-якої складності. Виробляється даний контролер в Італії на чипі ATmega328P з частотою 16 МГц, флеш пам’яттю 32 Кб. Як і Arduino Uno, контролер має 14 цифрових (6 з яких може працювати в режимі ШІМ) та 8 аналогових входів/виходів.

Рисунок 3.14. Arduino Nano

В загальному дана плата відрізняється від Arduino Uno тільки форм фактором, дозволяючи використовувати її для девайсів з невеликим розміром. З недоліків можна зазначити високу ціну та відсутність стандартного роз’єму живлення, замість якого використовується Mini-B USB.

Менш популярним являється контролер NodeMCU (рис. 3.15), реалізація якого можлива на двох типах чипів: esp8266 і esp32. Даний контролер виробляється в Китаї. В більшості випадків плату використовують саме для реалізації IoT проектів через ультра низьке споживання електроенергії та

розпаяного “з коробки” WiFi модуля. Також плата може працювати в доволі екстремальних діапазонах температур, а саме -40 ~ +125 ° C. З плюсів також можна віднести невеликі габарити плати та малу вагу.

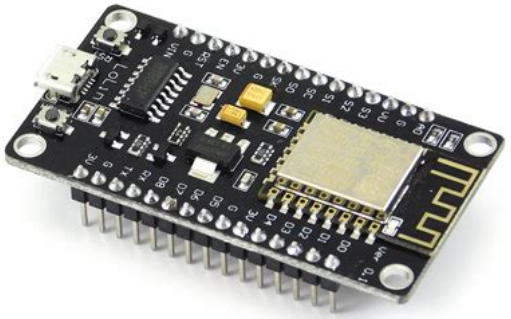


Рисунок 3.15. NodeMCU v3 на платі esp8266

Також обидва чипа мають підтримку різного типу методів завантаження прошивки: UART/GPIO інтерфейси для передачі даних прошивка з хмари або через USB.

Щодо відмінностей між чипами esp8266 та esp32, то контролер на базі першого має тільки один аналоговий вхід/вихід, що дуже обмежує варіативність датчиків, які можуть бути підключені. Також esp8266 має меншу швидкодію та відсутній Bluetooth модуля.

Отже, виходячи з характеристик 4 мікроконтролерів, які були описані вище, був обраний мікроконтролер NodeMCU v. 3 на платі esp8266, який у порівнянні з платами Arduino зразу має розпаяний WiFi модуль для підключення до мережі Інтернет, має більшу місткість постійної пам’яті, більшу швидкість зчитування та запису інформації та його цінова характеристика значно нижча від плат Arduino. Слід зауважити, що даної швидкодії, яку нам дає мікроконтролер NodeMCU v. 3 на платі esp8266 повністю задовольняє навантаженість IoT системи розумного будинку [14].

Також крім контролера, модуль складається з 3 датчиків, а саме:

- Інфрачервоний датчик руху. Він дозволить модулю відслідковувати рух в кімнаті та зразу оновлювати дані на сервері. Був обраний HC-SR501, який дозволяє реагувати на рух в радіусі 7-ми метрів, а кут

спрацювання 110 градусів. Датчик має мінімальну затримку реагування, яка становить 0.3 секунди. Робоча температура становить від -20 до +50 градусів Цельсія. Також датчик має два режима роботи: при першому режимі (встановлений в положення H), датчик буде подавати високий рівень на вихід увесь час поки він вловлює рух; при другому режимі (встановлений в положення L), датчик буде подавати низький рівень з високого на вихід весь час поки він вловлює рух. Для модуля, який використовується в даній роботі датчик був налаштований в режим H.

- Датчик температури. Ще один елемент реалізованої системи. Для данного модуля був обраний аналоговий датчик LM35DZ призначений для вимірювання температури в діапазоні від -55 до + 150 ° C. Хороша лінійність, економічність, швидкодія та точність перетворення дозволяють використовувати його як гідну альтернативу цифровим датчикам температури.
- Датчик якості повітря MQ135. Даний датчик дозволить моніторити якість повітря в реальному часі в кімнаті, де буде розташований модуль. Він дозволяє вимірювати наступні гази: NH3, NO3, пари алкоголю, бензину, диму, CO2, ацетону та інші. Також, важливий момент при використанні даного датчика, що його показники будуть неточні приблизно перші 24 години (період прогріву).

3.5. Побудова схеми та програмування мікроконтролера

Метою першого етапу роботи, було правильне складання схеми та отримання коректних даних з датчиків.

Схема підключення всіх датчиків до контролера NodeMCU на базі ESP8266 (рис. 3.16).

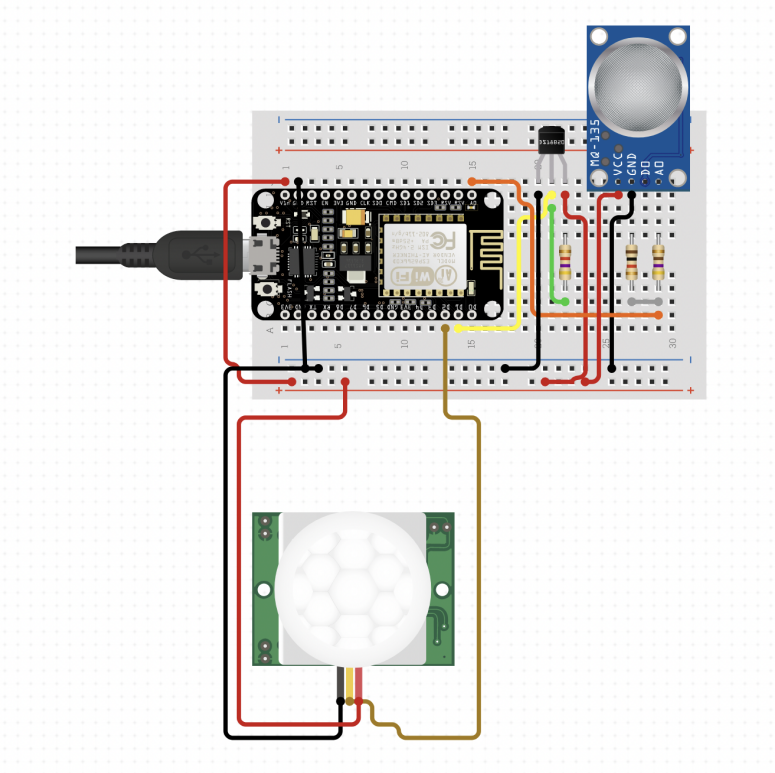


Рисунок 3.16. Схема підключення датчиків до контролера

Контролер підключений до джерела живлення за допомогою роз'єму Micro USB. Після чого позитивний кінець джерела живлення був підключений до першого входу датчика температури. Негативний контакт джерела живлення був підключений до третього контакту датчика, а також другий контакт датчика був приєднаний до аналогову виходу плати.

Інфрачервоний датчик руху має три контакти: для підключення позитивного полюса джерела струму, негативного полюса джерела струму, а також цифровий вихід, з якого в залежності від режиму подається відповідний сигнал при детекції руху. Позитивний та негативний контакти були підключені до відповідних входів на платі, а цифровий вихід підключений до входу D2.

Датчик якості повітря має 4 контакти: для підключення позитивного полюса струму, негативного полюса струму, цифровий вихід, на який подається високий рівень, якщо перевищений рівень концентрації газів (рівень налаштовується окремо), аналоговий вихід з якого можна зчитувати точні показники концентрації газів датчика. Позитивний та негативний контакти були підключені до

відповідних входів на платі, а також був використаний аналоговий вихід. який підключений до входу A0 на платі.

Код, наведений в додатку Г, використовується для збору показників з усіх датчиків підключених до контролера.

Для використання API протоколу MQTT була використана бібліотека для мови програмування C++ MQQTClient.h, яка має весь необхідний функціонал для повної реалізації всіх функцій протоколу MQTT [15]. Таким чином на вище наведеному фрагменті коду, програма підключається до MQTT брокера та кожну секунду надсилає йому показники виміру температури.

Після тестування підключення схеми на навчальному макеті, перевірки правильності вичитування показників, формату даних пакету на відправку до брокера - схема була розпаяна на макетну плату (рис. 3.17).

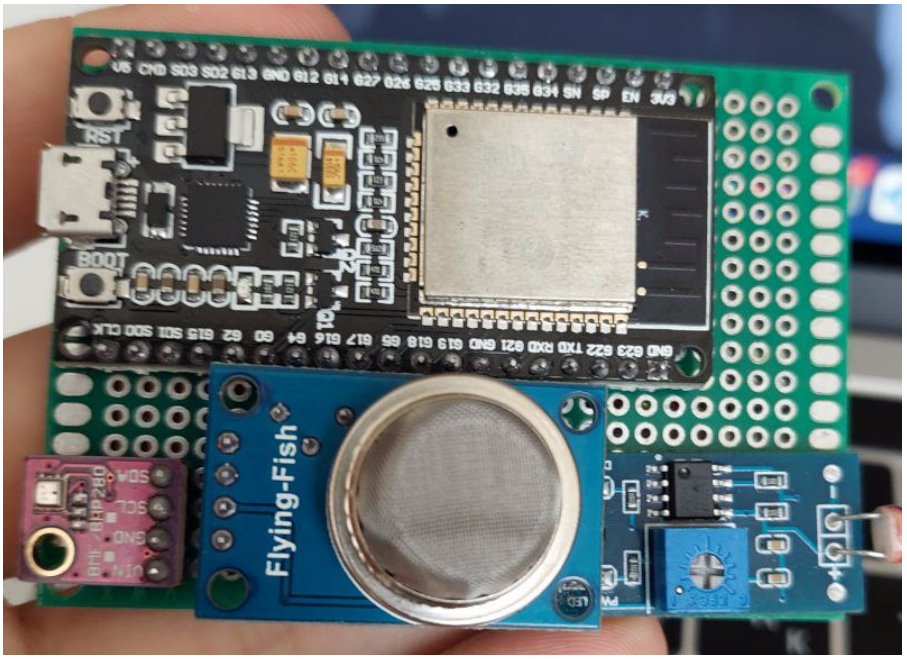


Рисунок 3.17. Готовий модуль

3.6. Висновки до розділу

В даному розділі були описані обрані технології, а також програмне забезпечення за допомогою яких побудована система. Описано реалізований веб-інтерфейс та його функціонал. Також приведено методи тестування

веб-додатка та власне його тестування. Отже, використавши ці всі інструменти аналізу швидкодії, можу прийти до висновку, що обраний мною інструментарій для написання системи є правильний і досить оптимізований.

Також, було реалізоване програмне забезпечення для мікроконтролера, яке знімає показники температури, якості повітря, а також детекція руху з відповідних датчиків на платі кожної секунди, після чого надсилає їх через транспортний протокол MQTT до брокера, який виступає в ролі маршрутизатора та надсилає ці пакети дальше до підписаних клієнтів. Також було намальовано та складено схему на навчальному макеті, після чого програмне забезпечення протестувалось. На фінальному етапі складена схема була розпаяна на макетну плату.

В результаті всіх вище описаних етапів був отриманий готовий для використання багатофункціональний модуль, який можна розміщувати в кімнаті розумного будинку.

ВИСНОВКИ

За короткий час в Україні відбулися глибокі зміни в різних сферах життя. З нововведень, що змінили стиль життя та інтереси осіб різного віку, можна відокремити появу інтернету, який можна підключити будь-кому. Саме інтернет докорінно змінив ставлення людей до комп'ютеризація та появи сучасних технологій.

Інтернет, який зараз проник в усі сфери життя людини, став останнім часом одним з видів проведення дозвілля дітей, підлітків та дорослих людей. Інтернет та інтернет технології настільки наповнені різноманітної, весь час змінною інформацією, що утримують користувача перед екраном цілі години.

З розвитком інтернету, у наше життя приходять нові галузі технологій, які полегшують його та автоматизовують. Одним з таких прикладів інновацій є IoT системи, які можна використовувати та впроваджувати у всі види людської зайнятості.

В Україні ця галузь інтернет технологій тільки зароджується, як в прогресивних країнах світу, IoT системи стали невід'ємною частиною життя багатьох людей. Вони автоматизовують наше життя, роблять прогулянки по місту, дорожній рух та ситуацію в місцях великого скупчення людей безпечнішими. Дозволяють більш ефективно використовувати ресурси для здійснення різного роду фермерських занять.

Саме ці факти спонукали мене до написання цієї роботи. Для цього, довелось опрацювати значний обсяг теоретичного матеріалу і набути багатьох навиків у програмуванні і графічному дизайні.

Виконуючи цю роботу, я створив систему керування розумним будинком на основі децентралізованої мережі, яка дозволяє керувати будинком, а також гарантувати анонімність даних користувача та даних телеметрії. Для створення цієї системи було обране сучасне середовище розробки Visual Studio Code, яке пропонує готові рішення для аналізу і редагування коду, призначені для роботи з новітніми технологіями Python, Django та Vue.

Створена мною системи, може служити позитивною мотивацією для учнів та програмістів-початківців вивчати мови програмування. Адже тоді, коли рядок за рядком програмного коду виростає повноцінний веб-сайт, це викликає багато позитивних емоцій і жаги до створення чогось власного.

Підбиваючи підсумки, мені вдалось виконати мету, написавши надійний та простий у використанні веб-додаток з легким налаштуванням. Також система получилась максимально масштабованою та малозатратною за рахунок усіх плюсів, які дає децентралізована мережа.

Використовуючи об’єктно орієнтований принцип програмування та об'єктно-орієнтовані шаблони проєктування, а також принципи функціонального програмування, вдалось створити просту та зрозумілу логіку взаємодії об’єктів. Також, вважаю, що технології програмування вибрані правильно, про що свідчить швидкодія, відмовостійкість та масштабованість створеної системи.

Мета кваліфікаційної роботи бакалавра досягнута.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Бідочка В. А., Палій С. В. Застосування децентралізованих мереж у IoT-системах. “Сучасні методи, інформаційне, програмне та технічне забезпечення систем керування організаційно-технічними та технологічними комплексами”: VIII Міжн. наук.-тех. конф., 26 лист. 2021 р. Київ. : НУХТ. С. 222-223.
2. Бідочка В. А., Палій С. В. Поняття децентралізованих мереж, їх використання та порівняння з централізованими мережами. “Наукові здобутки молоді - вирішенню проблем харчування людства у XXI столітті”: Міжн. наук.-тех. конф. молодих учених, аспірантів і студентів 15-26 квіт. 2021 р. Київ. : НУХТ. С. 367.
3. V. Bidochka, S. Paliy. Decentralized network in IoT systems. Information Technology and Interactions (Satellite): Conference Proceedings, December 01, 2021, Kyiv, TSNUK. P.
4. Бідочка В. А., Палій С. В. Проєктування IoT системи керування розумним будинком на базі децентралізованої мережі. “Сучасні тенденції розвитку інформаційних систем і телекомунікаційних технологій” : IV Міжн. наук.-тех. конф., 1-2 лют. 2022 р. Київ.: НУХТ. С. 37-39.
5. What is IoT? URL: https://www.oracle.com/internet-of-things/what-is-iot/ (Дата звернення: 21.02.2022)
6. What is Peer to Peer network? URL: https://www.cspsprotocol.com/p2p-network/ (Дата звернення: 17.01.2022)
7. Introduction to Decentralized P2P Apps. URL: https://www.youtube.com/watch?v=oCS05QSQ-1k (Дата звернення: 17.01.2022)
8. Вивчаємо Python, том 1, 5-е видання, Марк Лутц, ISBN 978-5-907144-52-1, NY
9. Django 2 Web Development Cookbook (3rd ed.) by Jake Kronika, Aidas Bendoraitis, ISBN-13: 978-1788837682, ISBN-10: 1788837681, NY.

10. PostgreSQL. URL: https://www.postgresql.org/ (Дата звернення: 09.09.2021)
11. Django Framework. URL: https://docs.djangoproject.com/en/3.0/ (Дата звернення: 21.09.2021)
12. Testing Vue 3. URL: https://vuejs.org/guide/scaling-up/testing.html (Дата звернення: 24.03.2022)
13. NodeMCU MQTT Tutorial. URL: https://www.teachmemicro.com/nodemcu-mqtt-tutorial/ (Дата звернення: 10.10.2021)
14. Beginners Guide To The MQTT Protocol. URL: http://www.steves-internet-guide.com/mqtt/ (Дата звернення: 10.10.2021)
15. What is the Internet of Things, or IoT. URL: https://www.iotforall.com/what-is-internet-of-things (Дата звернення: 10.02.2022)
16. MQTT Essentials. URL: https://www.hivemq.com/blog/mqtt-essentials (Дата звернення: 10.02.2022)
17. Internet of Things. URL: https://en.wikipedia.org/wiki/Internet_of_things (Дата звернення: 09.09.2021)
18. MQTT Protocol. URL: https://mqtt.org/ (Дата звернення: 09.09.2021)

ДОДАТОК А

Текст програми ядра

482.IoT.КНУШ.211290-01 ПЗ

Листів 3

Розробник Віталій БІДОЧКА

Керівник Сергій ПАЛІЙ

Київ - 2022

```
class BlityNet(Thread):
    def __init__(self):
        super().__init__()
        self._log = ColoredLogger(name='BlityNet')

        self._log.info(colored(f.renderText('BLITY'), 'blue'))
        self._log.info(colored('BlityNet', 'blue'))
        self._log.info(colored('v.0.1 Author: Vitalii Bidochka', 'green'))

        self._config = self._make_configuration()

        self._api_abs_path = os.path.abspath('../api')
        self._ui_abs_path = os.path.abspath('../ui')

        self.name = 'BlityNet'
        self.daemon = True

        self._log.info('Initializing...')

        self._log.info('[1/5] Starting MQTT Broker')
        self._log.debug(os.system(self._config['mqtt_broker_command']))

        self._log.info('[2/5] Starting Node')
        self.node = Node(certificate_path=self._config['certificate'],
name=self._config['hub_name'])
        self.node.start()

        self._log.info('[3/5] Starting database')
        self._log.debug(os.system(self._config['db_command']))
```

```
        self._log.info('[4/5] Starting API')
        self._api_thread = Thread(target=self._run_api,
args=(f'{self.node.ipaddress}:8081',), daemon=False, name='API')
        self._api_thread.start()

        self._log.info('[5/5] Starting UI')
        self._ui_thread = Thread(target=self._run_ui, daemon=False, name='UI')
        self._ui_thread.start()

    @staticmethod
    def _make_configuration():
        answers = prompt(GLOBAL_CONFIG_QUESTIONS,
style=PY_INQUIRER_STYLE)
        if len(answers) != len(GLOBAL_CONFIG_QUESTIONS):
            raise RuntimeError('Make sure you provide all config information!')

        return answers

    def _run_api(self, socket):
        p = subprocess.Popen(['cd', self._api_abs_path], stdout=subprocess.PIPE,
universal_newlines=True)
        manage_py_file_path = self._api_abs_path + '/manage.py'
        subprocess.run(['python', manage_py_file_path, 'runserver', socket], check=True,
stdin=p.stdout,
                       universal_newlines=True)

    def _run_ui(self):
```

```
        p = subprocess.Popen(['cd', self._ui_abs_path], stdout=subprocess.PIPE,
universal_newlines=True)
        subprocess.run(['npm', '--prefix', self._ui_abs_path, 'run', 'serve'], check=True,
stdin=p.stdout,
                       universal_newlines=True)
```

ДОДАТОК Б

Текст сертифікаційного сервісу

482.IoT.КНУШ.211290-01 ПЗ

Листів 3

Розробник Віталій БІДОЧКА

Керівник Сергій ПАЛІЙ

Київ - 2022

```
class CertificateViewSet(ViewSet):
        queryset = Certificate.objects.all().order_by('-id')
        permission_classes = [permissions.IsAuthenticated]

        def list(self, request):
        queryset = Certificate.objects.all().order_by('-id')
        serializer = CertificateSerializer(queryset, many=True, context={'request':
request})
        return Response(serializer.data)

        def retrieve(self, request, pk=None):
        certificate = Certificate.objects.filter(pk=pk).first()
        serializer = CertificateSerializer(certificate, context={'request': request})
        return Response(serializer.data)

        def create(self, request):
        dt = datetime.now()
        random_str = generate_random_string(30)
        token = generate_token(str(dt) + random_str)

        serializer = CertificateCreateSerializer(data={'token': token})
        if serializer.is_valid():
        serializer.save()
        return Response(serializer.data, status=status.HTTP_201_CREATED)

        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

        @action(detail=False, methods=['POST'], url_path='activate-certificate',
name='Activate Certificate',
```

```
        permission_classes=[])
        def activate_certificate(self, request):
        token = request.data.get('token')

        if token:
        certificate = Certificate.objects.filter(token=token).first()

        if certificate.banned:
                return Response('Certificate is banned',
status=status.HTTP_401_UNAUTHORIZED)

        certificate.activated = True
        certificate.activated_at = datetime.now()
        certificate.save()

        serializer = CertificateSerializer(certificate, context={'request': request})
        return Response(serializer.data)

        return Response('Invalid token', status=status.HTTP_401_UNAUTHORIZED)

        @action(detail=False, methods=['POST'], url_path='check-certificate',
name='Check Certificate',
        permission_classes=[])
        def check_certificate(self, request):
        certificate = Certificate.objects.filter(pk=request.data['id']).first()

        if certificate.banned:
        return Response('Certificate banned',
status=status.HTTP_401_UNAUTHORIZED)
```

```
if CertificateCheckSerializer(certificate).data == request.data:
return Response('ok')

return Response(status=status.HTTP_401_UNAUTHORIZED)
```

ДОДАТОК В

Текст NH сервісу

482.IoT.КНУШ.211290-01 ПЗ

Листів 3

Розробник Віталій БІДОЧКА

Керівник Сергій ПАЛІЙ

Київ - 2022

```
class HubViewSet(ViewSet):
    queryset = Hub.objects.all().order_by('-id')

    def list(self, request):
    queryset = Hub.objects.all().order_by('-id')
    serializer = HyperlinkedHubSerializer(queryset, many=True, context={'request': request})
    return Response(serializer.data)

    def create(self, request):
    certificate = request.data.pop('certificate')
    if certificate and is_authorized(certificate):
    serializer = HubCreationSerializer(data=request.data)

    if serializer.is_valid():
        serializer.save()
        return Response(serializer.data, status=status.HTTP_201_CREATED)

    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

    return Response(status=status.HTTP_401_UNAUTHORIZED)

    def retrieve(self, request, pk=None):
    hub = Hub.objects.filter(pk=pk).first()
    serializer = HyperlinkedHubSerializer(hub, context={'request': request})
    return Response(serializer.data)

    @action(detail=False, methods=['PATCH'], url_path='update-hub', name='Update Hub')
```

```
        def update_hub(self, request):
        name = request.data.pop('name')
        hub = Hub.objects.filter(name=name).first()
        serializer = HubSerializer(hub, data=request.data, partial=True)

        if serializer.is_valid():
        serializer.save()
        return Response(serializer.data)

        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

        @action(detail=False, methods=['GET'], url_path='get-available-hubs',
name='Get Available Hubs',
        permission_classes=[])
        def get_available_hubs(self, request):
        certificate = request.data.pop('certificate')
        if certificate and is_authorized(certificate):
        name = request.data['name']
        hubs = Hub.objects.filter(active=True).exclude(name=name)

        if not len(hubs):
                hub = Hub.objects.filter(name=name).first()
                serializer = HubSerializer([hub], many=True)
                return Response(serializer.data)

        serializer = HubSerializer(hubs, many=True)
        return Response(serializer.data)

        return Response(status=status.HTTP_401_UNAUTHORIZED)
```

ДОДАТОК Г

Текст мікроконтролера

482.IoT.КНУШ.211290-01 ПЗ

Листів 5

Розробник Віталій БІДОЧКА

Керівник Сергій ПАЛІЙ

Київ - 2022

```
#include <ESP8266WiFi.h>
#include "EspMQTTClient.h"
#include <OneWire.h>
#include <MQUnifiedsensor.h>

#ifndef STASSID
#define STASSID "SSID"
#define STAPSK  "PASSWORD"
#endif

#define FIRE_PIN 4
bool is_fire;
void ICACHE_RAM_ATTR fireDetect ();

#define PIR_PIN 0
bool is_moution;
void ICACHE_RAM_ATTR moutionDetect();

#define LIGHT_PIN 14

#define placa "ESP8266"
#define Voltage_Resolution 3.3
#define pin A0
#define type "MQ-135"
#define ADC_Bit_Resolution 12
#define RatioMQ135CleanAir 3.6
double CO2 = (0);
MQUnifiedsensor MQ135(placa, Voltage_Resolution, ADC_Bit_Resolution, pin, type);
```

```
const char* ssid     = STASSID;
const char* password = STAPSK;

EspMQTTClient client(
  "SSID",
  "PASSWORD",
  "192.168.1.198",  // MQTT Broker server ip
  "TestClient1111" // Client name that uniquely identify your device
);

void IRAM_ATTR fireDetect(void) {
  is_fire = digitalRead(FIRE_PIN);
  Serial.println(is_fire);
  if (is_fire) {
        client.publish("/fire", "Fire!");
  } else {
        client.publish("/fire", "No fire");
  }
}

void IRAM_ATTR moutionDetect(void) {
  is_moution = digitalRead(PIR_PIN);

  if (is_moution) {
        client.publish("/moution", "Detect");
  } else {
        client.publish("/moution", "Not detected");
  }
}
```

```
void setup() {
        Serial.begin(115200);
  delay(10);
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  delay(2000);

  pinMode(FIRE_PIN, INPUT);
  attachInterrupt(digitalPinToInterrupt(FIRE_PIN), fireDetect, CHANGE);

  pinMode(PIR_PIN, INPUT);
  attachInterrupt(digitalPinToInterrupt(PIR_PIN), moutionDetect, CHANGE);

  pinMode(LIGHT_PIN, OUTPUT);
  digitalWrite(LIGHT_PIN, HIGH);
```

```
  //Set math model to calculate the PPM concentration and the value of constants
        MQ135.setRegressionMethod(1); //_PPM =  a*ratio^b
        MQ135.setA(110.47);
        MQ135.setB(-2.862);
        // Configurate the ecuation values to get NH4 concentration
        MQ135.init();
        Serial.print("Calibrating please wait.");
        float calcR0 = 0;
        for(int i = 1; i<=10; i ++)  {
        MQ135.update(); // Update data, the arduino will be read the voltage on the analog pin
        calcR0 += MQ135.calibrate(RatioMQ135CleanAir);
        Serial.print(".");
        }
        MQ135.setR0(calcR0/10);
        Serial.println("  done!.");
        if(isinf(calcR0)) { Serial.println("Warning: Conection issue founded, R0 is infite (Open circuit detected) please check your wiring and supply"); while(1);}
        if(calcR0 == 0){Serial.println("Warning: Conection issue founded, R0 is zero (Analog pin with short circuit to ground) please check your wiring and supply"); while(1);}
        /*****************************  MQ CAlibration ************************/
        MQ135.serialDebug(false);
}

void onConnectionEstablished() {
  client.subscribe("/light", [] (const String &payload)  {
```

```
        Serial.println(payload);
        if (payload == "true") {
        digitalWrite(LIGHT_PIN, HIGH);
        } else {
        digitalWrite(LIGHT_PIN, LOW);
        }
  });
}

void loop() {
  client.loop();

  MQ135.update(); // Update data, the arduino will be read the voltage on the analog pin
        CO2 = MQ135.readSensor(); // Sensor will read CO2 concentration using the model and a and b values setted before or in the setup
        Serial.print("CO2: ");
        Serial.println(CO2);
        client.publish("/air/CO2", (String)CO2);
        delay(5000);

}
```

ДОДАТОК Ґ

Діаграми класів

482.IoT.КНУШ.211290-01 ПЗ

Листів 5

Розробник Віталій БІДОЧКА

Керівник Сергій ПАЛІЙ

Київ - 2022

Таблиця Ґ. 1

Поля класу “User”

Назва	**Тип даних**
1	2
id	Integer
email	EmailField
first_name	CharField
last_name	CharField
middle_name	CharField
password	HashField
is_superuser	Boolean
is_stuff	Boolean

Таблиця Ґ. 2

Поля класу “Certificate”

Назва	**Тип даних**
1	2
id	Integer
token	TextField
created_at	DateTime
activated_at	DateTime
activated	Boolean
banned	Boolean

Таблиця Ґ. 3

Поля класу “Node”

Назва	**Тип даних**
1	**2**
id	Integer
name	CharField
ipaddress	IPAddressField
port	Integer
added_date	DateTime
activate	Boolean
memory	Integer
quality	Integer

Таблиця Ґ. 4

Поля класу “BlityNet”

Назва	**Тип даних**
1	2
log	Logger
config	Dict
api_abs_path	str
name	str
daemon	Boolean
node	Node
api_thread	Thread
ui_thread	Thread

Таблиця Ґ. 5

Поля класу “Forwarder”

Назва	**Тип даних**
1	2
log	Logger
name	str
daemon	Boolean
stopped	Boolean
api_url	str
queue	Queue

Таблиця Ґ. 6

Поля класу “Node”

Назва	**Тип даних**
1	2
log	Logger
name	str
daemon	Boolean
stopped	Boolean
active	Boolean
mqtt_client	MqttClient
certificate_path	str
certificate	Certificate
ipaddress	str
forwarder	Forwarder

Таблиця Ґ. 7

Поля класу “Dashboard”

Назва	**Тип даних**
1	2
id	Integer
name	CharField
description	TextField
created	DateTime
warnings	Integer
cautions	Integer
notices	Integer
owner	ForeignKey

Таблиця Ґ. 8

Поля класу “Widget”

Назва	**Тип даних**
1	2
id	Integer
type	ChoiceField
title	CharField
x	Integer
y	Integer
width	Integer
height	Integer
props	JsonField

Таблиця Ґ. 9

Поля класу “Device”

Назва	**Тип даних**
1	2
id	Integer
name	CharField
added	DateTime
edited	DateTime
topic	CharField
owner	ForeignKey
status	ChoiceField

Таблиця Ґ. 10

Поля класу “Value”

Назва	**Тип даних**
1	2
id	Integer
device	ForeignKey
value	JsonField
ts	DateTime

ДОДАТОК Д

Інструкція користувача

482.IoT.КНУШ.211290-01 ПЗ

Листів 8

Розробник Віталій БІДОЧКА

Керівник Сергій ПАЛІЙ

Київ - 2022

АНОТАЦІЯ

Інструкція містить відомості про користування програмним інтерфейсом інформаційної системи “Blity”, його налаштування, а також інформацію, необхідну для розуміння функцій розробленої програми і її експлуатації. Керування таблицями БД кінцевими користувачами системи здійснюється за допомогою натискання необхідних кнопок у вікнах програмних форм, введенням відповідної інформації в поля вводу під час заповнення та редагування таблиць бази даних. Докладному опису перелічених питань і присвячена ця інструкція.

ЗМІСТ

РОЗДІЛ 1. НАЛАШТУВАННЯ ПРОГРАМИ

Цей розділ охоплює запуск системи, налаштування початкових параметрів, вхід в систему, створення девайсів, створення та налаштування дашборда, додавання віджета та його налаштування.

Розроблена інформаційна система є робоча та може бути запущена на кожному комп'ютері користувача.

Для того щоб зареєструвати особистий кабінет, користувач повинен звернутись до адміністратора системи, щоб він згенерував дані для входу. Після цього адміністратор повідомить користувачу згенеровані дані і останній отримає змогу зайти в особистий кабінет.

Для успішного запуску системи, на комп’ютері користувача повинне бути встановлене наступне програмне забезпечення:

- PostgreSQL.
- NodeJS.
- MQTT Broker.
- Vue 3.
- Python від версії 3.7.

Після успішного встановлення всього необхідного ПЗ, потрібно запустити виконуваний файл запуску системи, після чого повинне з’явитись вікно налаштування параметрів запуску, після цього потрібно заповнити всі обов’язкові поля, як показано на рисунках Г.1.1-1.2.

Рисунок Г.1.1 Вікно налаштування запуску системи

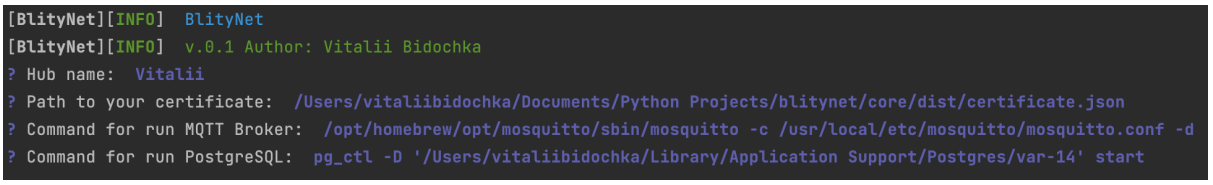


Рисунок Г.1.2 Заповнена форма налаштування запуску системи

Якщо всі введені дані вірні, система успішно запуститься та проведе всі дії, які описані в розділі 2.2 основної частини. Інакше система виведе всі помилки в логи системи.

Після успішного запуску системи, користувач може перейти на сторінку веб-додатка системи за адресою https://localhost:8080, де його перенесе на сторінку входу в систему, де йому потрібно ввести свої дані для авторизації (сторінка входу наведена на рисунку Г.1.3.).

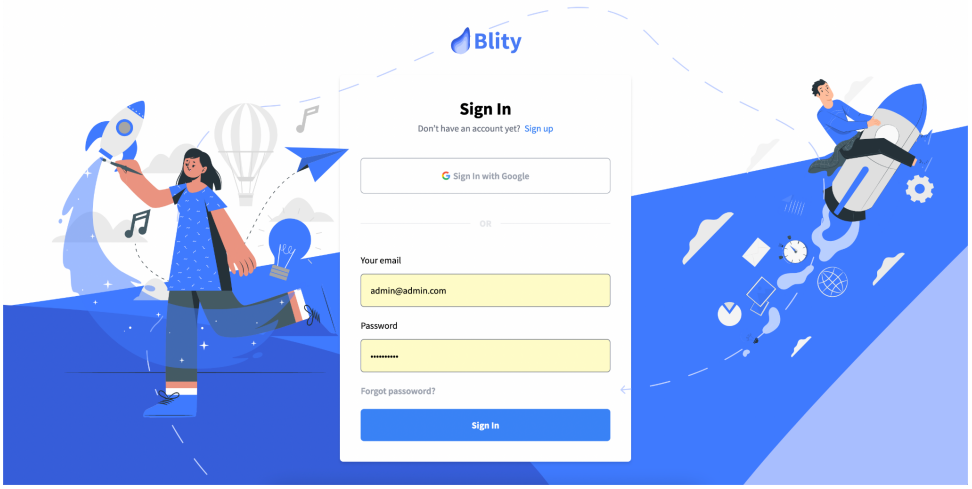


Рисунок Г.1.3. Сторінка входу в систему

Якщо користувач ввів валідні дані для входу - його перенесе на сторінку дашбордів. Якщо користувач перший раз увійшов у систему, відповідно у нього буде порожній список дашбордів.

Для подальшого налаштування, користувачу необхідно створити всі девайси, які будуть надсилати дані в систему або ж якими користувач хоче керувати. Для цього йому потрібно перейти на сторінку девайсів та нажати на синю кнопку в нижньому правому куті сторінки, після чого з'явиться модальне вікно з формою, яку потрібно заповнити валідними даними, форма наведена на рисунку Г.1.4.

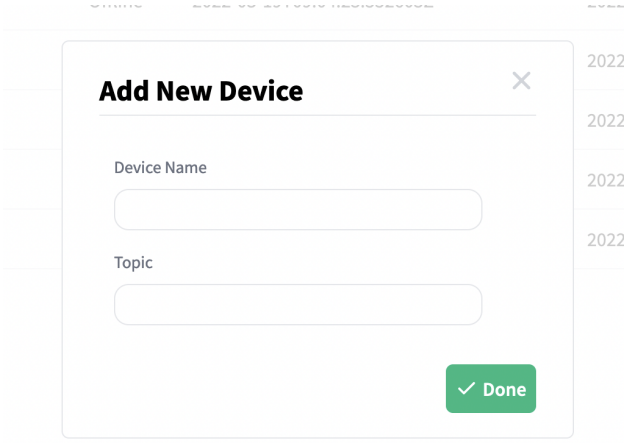


Рисунок Г.1.4 Форма створення девайса

Відповідно до прикладу, який буде налаштований, було створені наступні девайси, які наведені на рисунку Г.1.5.

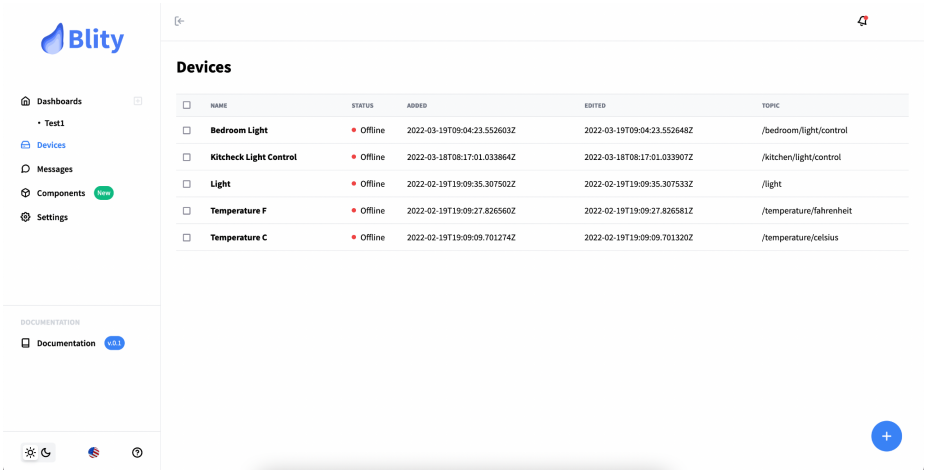


Рисунок Г.1.5. Список створених девайсів

Після того як користувач додав всі девайси, можна почати процес створення та налаштування дашборда. Для цього користувачу потрібно перейти на сторінку дашбордів та натиснути кнопку в правому верхньому куті панелі керування. Далі потрібно натиснути на синю кнопку у нижньому правому куті вікна. Після чого з’явиться модальне вікно з формою, яку потрібно заповнити валідними даними, форма наведена на рисунку Г.1.6.

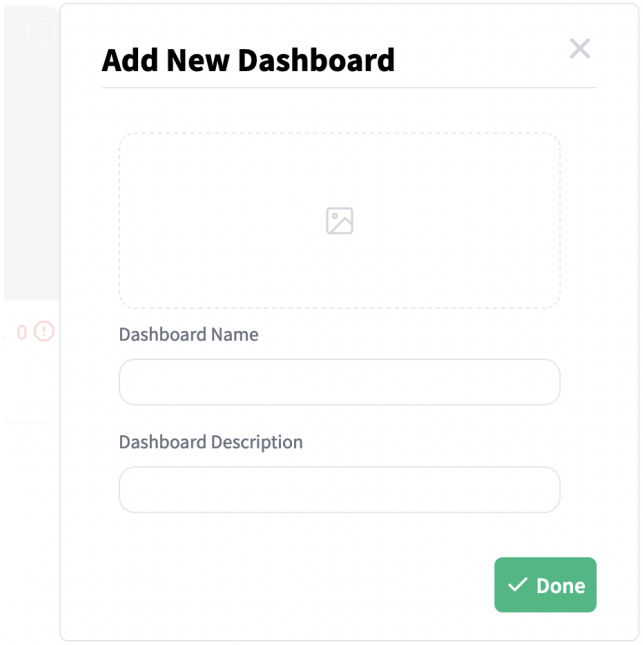


Рисунок Г.1.6. Форма створення дашборда

Після заповнення всіх полів валіднимим даними, у списку дашбордів з’явиться щойно створений. На нього потрібно натиснути два рази і користувача перенесе на сторінку дашборда. Наступний крок - потрібно додати віджет, для цього потрібно перейти в режим налаштування дашборда і нажати на синю кнопку в нижньому правому куті сторінки, після чого з’явиться модальне вікно з усіма типами віджетів, які присутні в даній версії системи. Користувачу потрібно вибрати бажаний тип віджета та натиснути кнопку “Done”, після чого він з’явиться на дашборді. Модальне вікно та вид щойно доданого віджета наведені на рисунках Г.1.7-1.8.

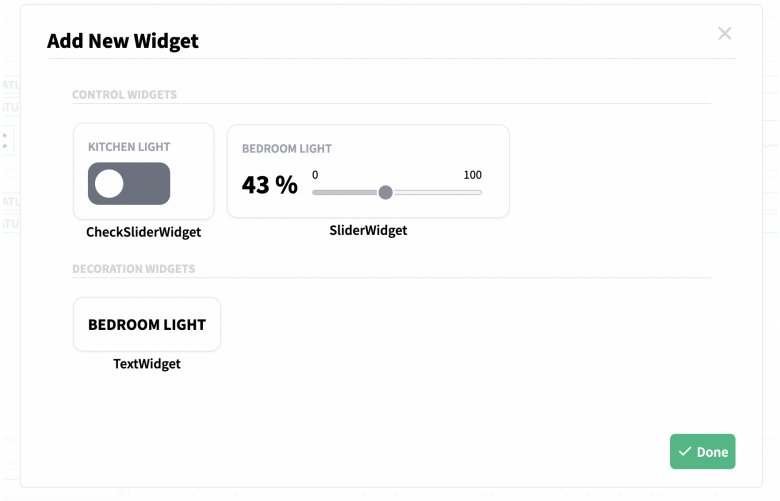


Рисунок Г.1.7 Модальне вікно вибору типу віджета

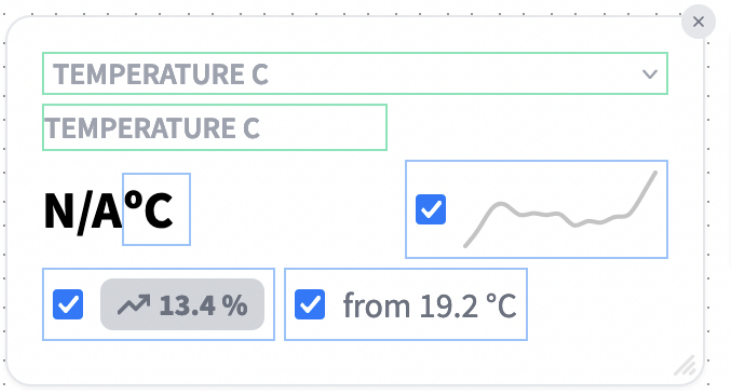


Рисунок Г.1.8. Доданий віджет

Далі користувачу потрібно заповнити всі необхідні поля налаштування віджета та налаштувати його розміщення і розмір.

В даному прикладі був налаштований дашборд, який містить всі типи віджетів та наведений на рисунку Г.1.9.

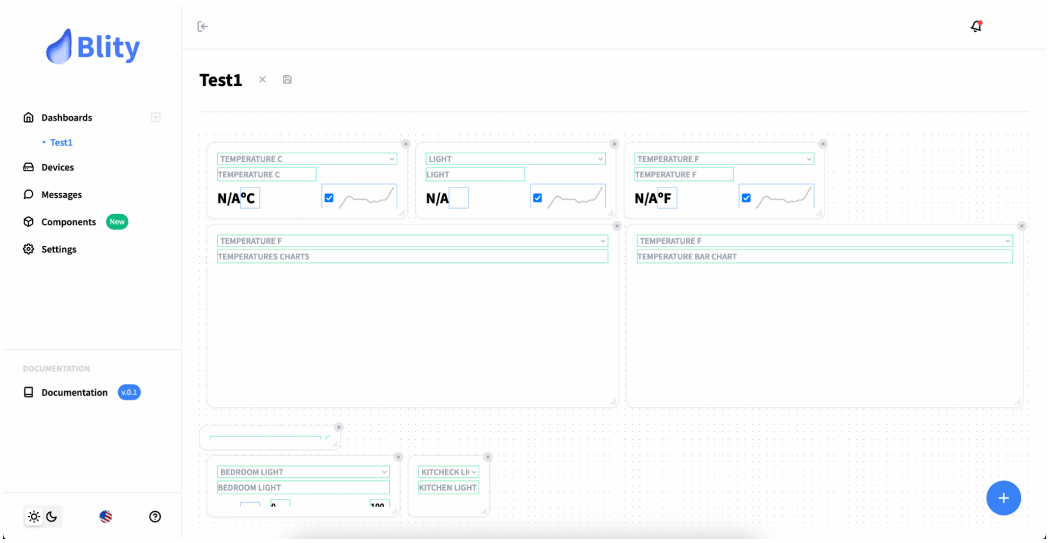


Рисунок Г.1.9 Налаштований дашборд

Після усіх налаштувань пов’язаних з типами віджетів, їх розмірами та розташуванням, користувачу потрібно зберегти дашборд нажавши на кнопку в лівій верхній частині панелі керування.

Після цього система запустить MQTT клієнт та почне реагувати на вхідні та вихідні дані телеметрії.

В даному прикладі повністю налаштована система та яка отримує дані має вигляд, який наведений на рисунку Г.1.10.

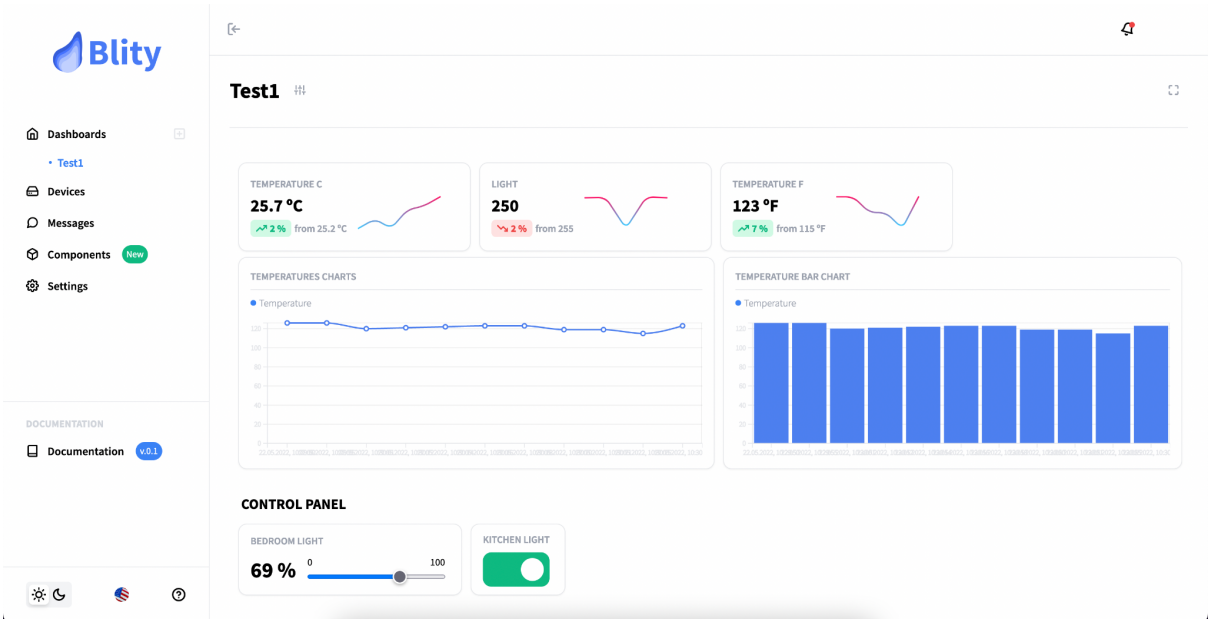


Рисунок Г.1.10 Готовий дашборд

Також після усіх налаштувань рекомендується перейти в повноекранний режим, для цього потрібно натиснути на кнопку у правій верхній частині панелі керування, тоді система буде мати вигляд наведений на рисунку Г.1.11.

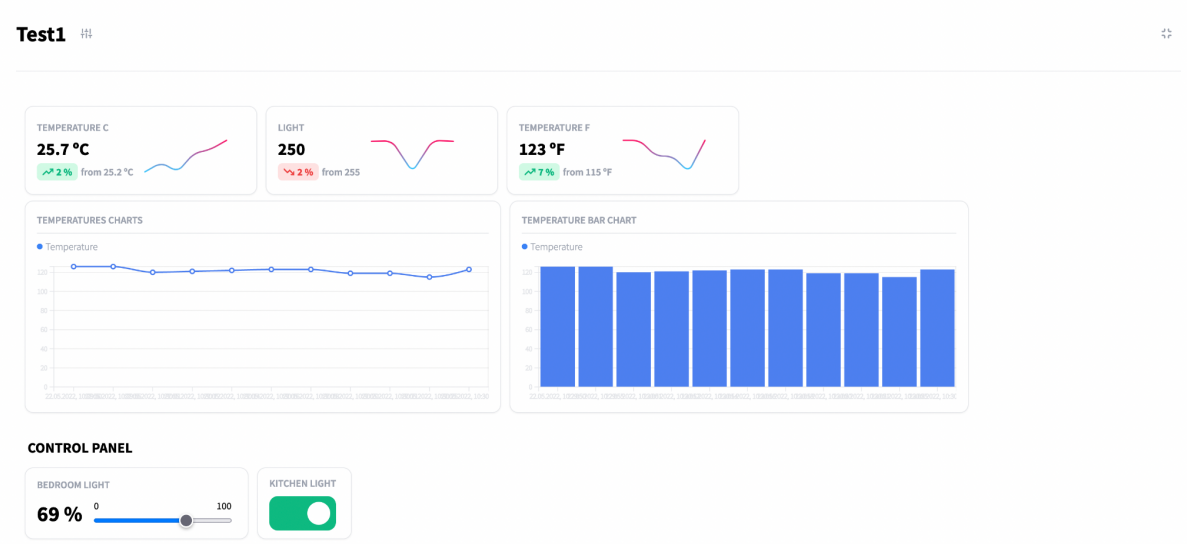


Рисунок Г.1.11 Повноекранний режим

ВИСНОВКИ

В розділі було розглянуте базове налаштування інформаційної системи, а саме запуск системи, вхід в особистий обліковий запис, додавання девайсів, додавання дашбордів та їх налаштування.

ДОДАТОК Е

Основні слайди презентації

482.IoT.КНУШ.211290-01 ПЗ

Листів 2

Розробник Віталій БІДОЧКА

Керівник Сергій ПАЛІЙ

Київ - 2022

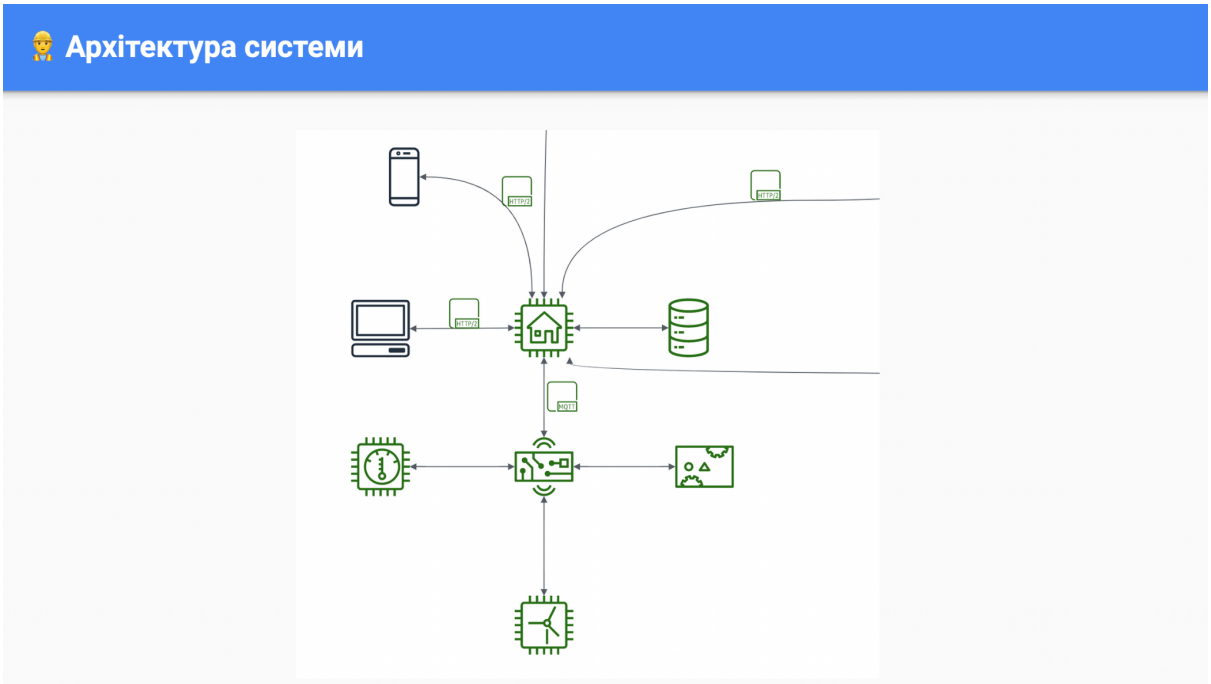


Рисунок 1 - Архітектура системи

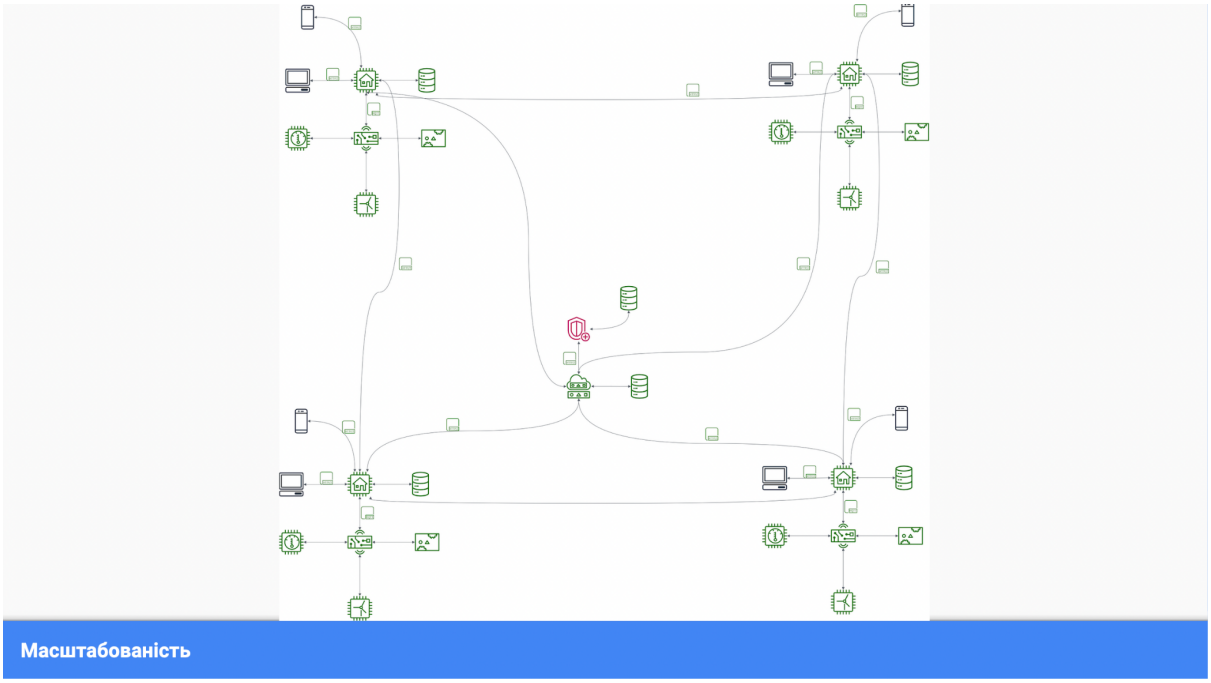


Рисунок 2 - Масштабованість системи

Рисунок 3 - Принцип роботи системи