НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
ім. Ігоря Сікорського

Навчально-науковий комплекс «Інститут прикладного системного аналізу»
(повна назва інституту/факультету)

Кафедра Системного проектування
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
“___”_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

з напряму підготовки 6.050101 Комп’ютерні науки
(код і назва)

на тему: Система оптимізації енергоспоживання для розумного будинку на основі технології Bluetooth Low Energy

Виконав (-ла): студент (-ка) __4__ курсу, групи __ДА-32___
(шифр групи)

_____Войтех Дмитро Володимирович_____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____ст. викладач Британов О. А._____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант економічний розділ _____доц. Рощина Н.В_____ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Нормоконтроль _____старший викладач Британов О.А._____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2017 року

Національний технічний університет України
«Київський політехнічний інститут»
ім. Ігоря Сікорського

Інститут (факультет) ННК «Інститут прикладного системного аналізу
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки 6.050101 Комп'ютерні науки
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
«___»_____20__ р.

ЗАВДАННЯ
на дипломну роботу студенту
Войтеху Дмитру Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи Система оптимізації енергоспоживання для розумного будинку на основі технології Bluetooth Low Energy

керівник роботи Бритов Олексій Анатолійович, ст. викладач
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від 10 травня 2017 р. № 1477-с

2. Термін подання студентом роботи 10.06.2017

3. Вихідні дані до роботи _____

Система оптимізації енергоспоживання для розумного будинку яка:

- реалізує основні концепції у сфері розумного будинку;
- має певний специфічний функціонал, що відсутній в інших системах;
- є простою, дешевою і надійною за рахунок мінімалістичного інтерфейсу;
- є легкою для розгортання для будь-якого користувача;
- за замовчуванням працює в автономному режимі, але може бути налаштована на роботу з хмарою.

Smart Home, розумні розетки, протокол зв’язку Bluetooth Low Energy, Bluetooth LE модуль RN4020, мікроконтролер DSPIC33FJ16MC102, NodeJS, база даних PostgreSQL.

4. Зміст роботи

1. Розглянути предметну область, проаналізувати протоколи зв’язку, що використовуються у сфері Smart Home;
2. Розробити пристрій моніторингу та керування електроприладами;
3. Спроектувати архітектуру системи, що інтегрує дані пристрої. Розробити її програмну реалізацію. Описати сценарії роботи системи;
4. Провести функціонально-вартісний аналіз розробленого програмного продукту.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

1. Пристрій моніторингу та керування електроприладами – плакат;
2. Архітектура системи – плакат;
3. Потоки даних системи – плакат;
4. Розгортання системи – плакат.

6. Консультанти розділів роботи·

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н.В, доц.		

7. Дата видачі завдання 01.02.2017

· Консультантом не може бути зазначено керівника дипломної роботи.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	01.02.017	
2	Збір інформації	15.02.2017	
3	Дослідження предметної області	28.02.2017	
4	Розробка архітектури системи	10.03.2017	
5	Розробка пристрою моніторингу і контролю електроприладами	10.04.2017	
6	Виконання програмної реалізації системи оптимізації енергоспоживання	05.05.2017	
7	Розгортання системи	10.05.2017	
8	Оформлення дипломної роботи	31.05.2017	
9	Отримання допуску до захисту та подача роботи в ДЕК	10.06.2017	

Студент _____ Войтех Д.В
(підпис) (ініціали, прізвище)

Керівник роботи _____ Бритов О.А
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

бакалаврської дипломної роботи Войтеха Дмитра Володимировича на тему: «Система оптимізації енергоспоживання для розумного будинку на основі технології Bluetooth Low Energy»

Дана робота присвячена дослідженню Smart Home систем, що базуються на основі різних протоколів бездротового зв'язку, та розробці системи оптимізації енергоспоживання для розумного дому.

У роботі розглядається побудова системи домашньої диспетчеризації на основі новітнього протоколу Bluetooth Low Energy з використанням сучасних технологій. Також пропонується пристрій моніторингу і контролю електроприладами, що є елементом даної системи. Запропоновано архітектуру, що має переваги перед існуючими системи в таких аспектах як надійність, швидкодія, простота розгортання та керування. Система є гнучкою завдяки можливості вибору режимів роботи (автоматичного або ручного) та зміни різноманітних налаштувань, що впливають на роботу алгоритму оптимізації.

В роботі наведено приклади роботи системи в різних режимах та за різних значень налаштувань алгоритму, що складається з сервера, розгорнутого на персональному комп'ютері, та двох розроблених макетних зразків пристроїв моніторингу і контролю.

Загальний обсяг роботи: 91 сторінка, додаток на 9 сторінок, 35 рисунків, 8 таблиць, 29 посилань.

Ключові слова: розумний будинок, домашня диспетчеризація, моніторинг і контроль електроприладів, розумні розетки, Bluetooth Low Energy, RN4020 BT LE module, DSPIC33FJ16MC102, NodeJS.

ABSTRACT

for the bachelor's thesis of Voitekh Dmytro Volodymyrovych
"Smart Home system for power consumption optimization based on Bluetooth Low Energy"

This thesis is devoted to the study of Smart Home systems based on different wireless communication protocols and development of the household system for power consumption optimization.

In the course of this thesis the implementation of home automation system based on Bluetooth Low Energy protocol and other modern technologies is described. Also device for for monitoring and control of electrical appliance's power consumption is proposed as an element of the described system. Devised system's architecture has advantages over existing available solutions in terms of durability, performance, ease of deployment and support. Proposed system is flexible enough due to ability to switch system mode (automatic or manual) and adjust the optimization algorythm's settings.

System was deployed on laptop computer, integrating two developed working samples of the proposed monitoring and control device. Various scenarios for system's functioning were provided, describing different interaction types depending on current mode and chosen settings' values.

The thesis contains 91 pages, 35 figures, 8 tables, 29 references and 1 appendix (9 pages).

Key words: Smart Home, home automation, electrical appliances' monitoring and control, smart sockets, Bluetooth Low Energy, RN4020 BT LE module, DSPIC33FJ16MC102, NodeJS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 9

ВСТУП 10

1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ. МЕТА РОБОТИ 12

1.1 Актуальність роботи 12

1.2 Існуючі рішення. Протоколи зв'язку Smart Home 14

1.2.1 Протокол ZigBee 19

1.2.2 Wi-Fi 23

1.2.3 Bluetooth 26

1.2.4 Bluetooth Low Energy 29

1.3 Висновок 33

1.4 Мета роботи 34

2. РОЗРОБКА ПРИСТРОЮ МОНІТОРИНГУ І КЕРУВАННЯ ЕЛЕКТРИЧНИМИ ПРИЛАДАМИ 35

2.1 Вимоги до пристрою 35

2.2 Bluetooth LE модуль Microchip RN4020 36

2.3 Мікроконтролер Microchip dsPIC33FJ16 39

2.4 Проектування схеми 41

2.5 Програмування мікроконтролера 42

3. АРХІТЕКТУРА СИСТЕМИ. ПРОГРАМНА РЕАЛІЗАЦІЯ 48

3.1 Сервер Bluetooth LE 50

3.1.1 Бібліотека Noble 50

3.1.2 Реалізація 52

3.2 Веб-сервер 54

3.2.1 ExpressJS 54

3.2.2 Реалізація 55

3.3 Приклад роботи системи 61

4. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОДУКТУ 65

4.1 Постановка задачі 66

4.1.1 Обґрунтування функцій програмного продукту 67

4.1.2 Варіанти реалізації основних функцій 67

4.2 Обґрунтування системи параметрів ПП 70

4.2.1 Опис параметрів 70
4.2.2 Кількісна оцінка параметрів 71
4.2.3 Аналіз експертного оцінювання параметрів 75
4.3 Аналіз рівня якості варіантів реалізації функцій 79
4.4 Економічний аналіз варіантів розробки ПП 80
4.5 Вибір кращого варіанта ПП техніко-економічного рівня 85
4.6 Висновки 85
ВИСНОВКИ 87
ПЕРЕЛІК ПОСИЛАНЬ 89
ДОДАТОК А 92

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Bluetooth LE – Bluetooth Low Energy (Bluetooth з низьким енергоспоживанням);
PLC (Power Line Communication) – використання електромережі для передачі сигналів;
RF (Radio Frequency) – радіочастоти;
Smart Switch – розумна розетка, пристрій для моніторингу та контролю електроприладів;
Smart Dispatcher – сервер, що збирає дані від Smart Switch пристроїв та на їх основі підтримує/змінює конфігурацію системи для зменшення небажаних витрат електроенергії та уникнення аварійних ситуацій;
Smart Grid - електрична мережа, яка включає в себе різноманітні оперативні та енергозберігаючі заходи, такі як розумні лічильники, відновлювані джерела енергії та ресурси забезпечення енергоефективності;
Self-healing – властивість мережі самовідновлюватися у випадку обриву основних зв’язків шляхом вибору інших маршрутів;
ВДЕ – відновлювані джерела енергії.

ВСТУП

Smart Home (домашня автоматизація, розумний будинок) є одним з найперспективніших напрямків розвитку інформаційних та комунікаційних технологій. За допомогою систем даного типу функціонально пов’язуються між собою усі електроприлади будівлі, якими можна керувати централізовано – користувачем з пульта-дисплею, або автоматично за допомогою певних алгоритмів.

Завдяки постійному зростанню тарифів на електроенергію, проблемам електробезпеки побутових приладів, оптимізація енергоспоживання на сьогоднішній день є однією з ключових цілей Smart Home.

Кількість різновидів систем у даній сфері невпинно зростає, з’являються нові бездротові технології для інтеграції приладів у єдину мережу, нові види сенсорів. В той же час, зростання попиту на продукцію Smart Home робить надзвичайно актуальними такі проблеми:

- недостатній рівень стандартизації та сумісності різних протоколів;
- надійність системи;
- безпека та захищеність системи від стороннього доступу;
- дороговизна та складність розгортання системи для користувача.

У даній роботі досліджуються основні існуючі підходи до побудови Smart Home систем, аналізуються протоколи та мережеві технології, що використовуються для побудови локальних мереж, виокремлюються їх недоліки і переваги.

Вивчаються шляхи вдосконалення існуючих рішень у даній сфері. Обґрунтовується вибір протоколів, технологій та підходів до побудови Smart Home систем, що дозволили хоча б частково вирішити проблеми, описані раніше.

Метою роботи є побудова системи оптимізації енергоспоживання для розумного будинку, що складається з елементів двох видів:

- пристрій моніторингу і контролю (далі - *Smart Switch*), що відправляє дані про енергоспоживання/генерацію відповідного приладу домогосподарства, дозволяє здійснювати його комутацію;
- сервер (далі - *Smart Dispatcher*), що збирає дані від усіх пристроїв локальної мережі та приймає рішення про зміну конфігурації мережі, відключення/підключення приладів на основі її поточного стану.

У роботі продемонстровано роботу побудованої системи на основі створених макетних зразків розумних розеток, що відправляють на сервер дані про стан підключеного навантаження а саме: силу струму, напругу і температуру. Показано основні типи взаємодії цих пристроїв с сервером в залежності від отриманих даних.

1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ. МЕТА РОБОТИ

1.1 Актуальність роботи

Кількість електроприладів в домогосподарстві невпинно збільшується. Загальна їх потужність може сягати позначки у 15 кВт [1]. В той же час, пропускна здатність мережі обмежена і зазвичай не перевищує 10 кВт [2]. Тобто, при одночасному підключенні більшої частини електроприладів відбувається аварійне відключення домогосподарства від мережі. Більше того, споживання електроенергії в домогосподарствах нерівномірно розподілене протягом доби. Ця проблема призводить до дисбалансу в державній енергосистемі, необхідності маневрування генерованими потужностями (будівництво нових ГАЕС, регулювання потужності існуючих ТЕС та ГЕС) а також до відключень великих споживачів у пікових режимах, в результаті - зниження ефективності і надійності всієї енергосистеми, мільярдні витрати для держави, приватних виробників і споживачів. Для того, щоб стимулювати часткове збланансування енергоспоживання в домогосподарствах, державою вводяться багатозонні тарифи електроенергії (двозонні та тризонні). Проте у домогосподарстві неможливо використати цей стимул у в повній мірі без простої та надійної Smart Home системи, що дозволила б автоматизовано керувати електроприладами в залежності від поточного стану домашньої енергосистеми.

На сьогоднішній день з’являється величезна кількість засобів контролю та моніторингу роботи побутових пристроїв, таких як розумні розетки, розумні лічильники та інші. Однак вони не мають системного підходу до розв’язання описаних вище проблем: рішення про включення/відключення конкретного електроприладу найчастіше приймається користувачем в ручному режимі, не надається можливостей для автоматизованої підтримки надійної роботи енергосистеми домогосподарства у випадку високого рівня навантаження.

Також існуючі Smart Home системи в більшості випадків потребують додаткового мережевого обладнання, такого як шлюзи, маршрутизатори тощо, що вносить додаткові вразливості, зменшує надійність.

Крім того, у рамках четвертого (Зимового) енергопакету ЕС до 2030 року доля домогосподарств та кооперативів, що будуть учасниками енергоринку відновлюваних джерел енергії (ВДЕ) досягне 50% від всього населення ЕС, а їх загальний внесок у виробництво електроенергії складе майже 20%. При цьому передбачено збільшення частки відновлюваної енергетики у загальному виробництві до 50%. Одним з важливих елементів нової енергетичної політики в Європі є енергокооперативи - об'єднання громадян, підприємств і організацій, метою яких є, як правило, реалізація різних локальних проектів в сфері ВДЕ. Найчастіше такі об'єднання спрямовують свої зусилля на децентралізоване, екологічне та незалежне від компаній і концернів виробництво енергії.

Згідно Зимового енергопакету, енергокооперативи отримають переваги повноцінного підключення до електромережі на рівних з іншими учасниками ринку. Вони зможуть більш ефективно продавати вироблену електроенергію споживачам в різних регіонах на прозорих умовах в необхідних кількостях. Така політика буде грати на руку споживачеві, захищаючи його, і дозволяючи йому повністю контролювати своє споживання енергії та її постачання.

При цьому споживачі, які виробляють електроенергію з відновлюваних джерел енергії для власних потреб, зможуть продавати надлишки електроенергії без втрати своїх прав як споживачів. Приватна особа зможе поставляти до 10 МВт, а юридична - до 500 МВт без набуття статусу постачальника [3].

Україна зволікає з виконанням своїх зобов'язань, взятих на себе в рамках ще Третього енергопакету, чим викликає критику з боку своїх партнерів в ЄС. На альтернативні джерела в загальному обсязі виробництва енергії як і раніше доводиться частка трохи більше 1% (частина ВДЕ, сонячна і вітрова генерація,

були втрачені в Криму). Проте, наша країна цілком може декларувати прихильність ідеям, зафіксованим в Зимовому пакеті. Тим більше, що в рамках пропозицій Зимового енергопакету ЄС має намір посилити енергетичну співпрацю зі своїми східними сусідами в сфері енергоефективності.

У контексті нових енергетичних правил в Європі, які можуть стати реальністю через кілька років, в Україні істотно можуть зміцнитися дрібні і середні виробники електроенергії з ВДЕ. Вже на даний момент в нашій країні на ринку ВДЕ працює понад 200 компаній. З масовою появою енергокооперативів (законопроект, який регламентує їх діяльність, вже готується) і прийняттям прозорих правил гри для всіх учасників ринку відповідно до Зимового енергопакету, український споживач зміг би відчути істотне поліпшення якості послуг в енергетичній сфері. Крім того, з 2016 року європейські банки все активніше видають кредити для створення потужностей ВДЕ в Україні. Особливим фактором є український «зелений тариф», один з найвищих в Європі, який може бути знижений лише в 2030 році.

Разом з тим при реалізації планів Зимового енергопакету виникають нові проблеми пов’язані зі стійкістю локальних енергосистем у кооперативах та загальної енергосистеми при великій кількості генераторів і споживачів, що практично не контролюються диспетчерською службою енергосистеми. Необхідно забезпечувати баланс генерації-споживання електроенергії, а для цього потрібно мати можливість моніторингу стану споживачів та генераторів, їх комутації у разі необхідності. Така система має бути бездротовою, захищеною від завад та небажаного зовнішнього втручання, надійною, енергоефективною, мати невисоку вартість та можливість керування від комп’ютера, або смартфона.

1.2 Існуючі рішення. Протоколи зв’язку Smart Home

Розумний дім (Smart Home, Digital House, Home Automation) – будинок або приміщення комерційного призначення (магазин, офіс, будь-яка установа),

електроприлади якого функціонально пов'язані між собою. Вони можуть бути під’єднані до комп’ютерної мережі, що дозволяє керувати ними за допомогою ПК та надає віддалений доступ до них через Інтернет. Завдяки інтеграції інформаційних технологій у домашні умови, усі системи та прилади узгоджують виконання функцій між собою, порівнюючи задані програми та зовнішні показники.

Розумний дім створюється за допомогою професійного проектування та програмування компаніями, що займаються розробкою проектів smart-home. Програми, що вводяться до алгоритмів multi-room розумного дому, розраховані на певні потреби мешканців та ситуації, пов’язані із зміною середовища або безпекою. Особливістю smart-home є керування з пульта, на котрому людина може натиснути одну-єдину клавішу з метою створення певної обстановки. При цьому, сама система аналізує навколишню ситуацію та параметри усередині приміщення, та, керуючись власними висновками, виконує задані користувачем команди із відповідними налаштуваннями. Окрім цього, побутові прилади, встановлені у розумному будинку, можуть бути об’єднані у домашню Universal Plug’n’Play-мережу із виходом в Інтернет [4].

Основні сфери застосування розумного дому:

- Системи контролю освітлення;
- Автоматизований контроль стану системи: збір даних з різноманітних сенсорів для корекції стану системи (контроль температури, вологості, задимленості, витоку газу, води тощо) [5];
- Системи опалення, вентиляції та кондиціонування повітря (HVAC, Heating, ventilation and air conditioning): дистанційне керування всіма пристроями-споживачами електроенергії через Інтернет за допомогою простого і зручного користувацького інтерфейсу [6];
- Інтеграція побутових приладів зі Smart Grid, з метою, наприклад, використання електроенергії, генерованої сонячними панелями в середині дня, щоб запустити пральну машину [7];

- Система безпеки, інтегрована з системою автоматизації будинку може надавати додаткові послуги, такі як дистанційний доступ до записів камер відеоспостереження через Інтернет, або централізований контроль всіх дверей і вікон [8].

Однак, не зважаючи на актуальність вирішуваних проблем та потенціальну вигоду від використання технологій розумного дому, дана технологія має ряд вагомих недоліків і ризиків. Головними серед них є:

- Системи під'єдані до Інтернету є вразливими до стороннього доступу;
- Технологія ще досі знаходиться на початкових стадіях формування. Поширена ситуація, коли користувачі можуть інвестувати в системи, що закидаються виробниками. Так, наприклад, Google у 2014 році придбав компанію, що виробляла систему домашньої автоматизації Revolv Hub, інтегрував її з платформою Nest, а у 2016 році закрив усі сервери компанії [9];
- Історично Smart Home системи в основному розповсюджувалися як повністю завершені рішення, де користувач повністю покладається на виробника в усіх аспектах: апаратна частина, протокол зв'язку, центральний сервер та користувацький інтерфейс. Лише нещодавно почали з'являтися системи з відкритим кодом, що можуть бути використані з пропрієтарним апаратним забезпеченням.

Існує широкий спектр спеціалізованих платформ та протоколів, що розроблені саме під побудову локальних мереж для Smart Home систем. Кожен з них, по суті, є окремою мовою. Кожна з цих мов по-різному спілкується з підключеними пристроями та керує ними для виконання певних функцій. Ці протоколи базуються на дротовому підключенні, використовують електромережу, гібридне бездротове підключення або класичне бездротове. На жаль, більшість з цих протоколів не є відкритими. Однак кожен з них має свій API. Короткий опис основних таких протоколів наведено у табл. 1.1.

Таблиця 1.1 – Спеціалізовані протоколи Smart Home

Протокол	Тип передачі	Швидкість передачі	Зауваження
KNX	PLC, RF, вита пара, Ethernet	9,6 кбіт/сек	Стандартизовано на міжнародному рівні (ISO/IEC), Канада (CSA-ISO), Європа (CENELEC/CEN), Китай (GB/T)
Universal Powerline Bus	PLC	480 біт/сек	Технологія 2-сторонньої комунікації, що дозволяє використовувати в якості середовища передачі сигналів існуючу електромережу
X10	PLC, RF	20 біт/сек	Система віддаленого контролю для ламп та побутових приладів розроблена на основі X10 Wireless Technology що використовує домашню електромережу для передачі керуючих сигналів
Zigbee	RF	20 – 250 кбіт/сек	Профілі ZigBee засновані на специфікації IEEE 802.15.4, що описує радіо-мережі що працюють на частоті 2.4 ГГц та передають невеликі обсяги даних в будівлях на відстані до 100м
Z-Wave	RF	100 кбіт/сек	Протокол з низьким енергоспоживанням. В Пн. Америці використовується частота 908.42 МГц, інші використовують частоти до 1 ГГц

Бачимо, що існує велика кількість різних стандартів та специфікацій, створених саме для застосування у Smart Home. Головними недоліками усіх цих проколів зв’язку відносно низький рівень сумісності між різними версіями, необхідність наявності шлюзу з мережею типу Wi-Fi для ручного керування

системою користувачем [10]. Схематично типову схему інтеграції таких систем наведено на рис. 1.1.

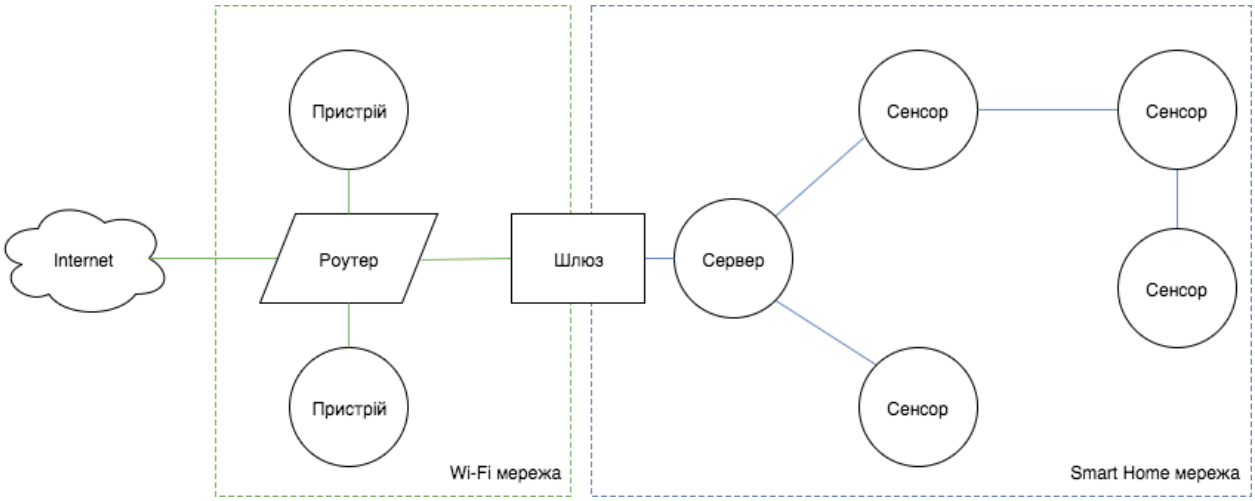

Рисунок 1.1 – Типова структура Smart Home системи

Крім описаних вище спеціалізованих протоколів іноді використовують такі загально розповсюджені технології, як Wi-Fi або Bluetooth. Їх використання вирішує проблему сумісності та ручного керування системою, спрощує архітектуру системи, проте ставить нові виклики безпеці, енергоспоживанню та надійності. Також важливим моментом є мережеві топології, що підтримуються даними протоколами, оскільки в умовах міської забудови або за умови наявності у будівлі товстих бетонних конструкцій, найбільш розповсюджена топологія “зірка” буде мати значні обмеження за радіусом дії.

Далі в роботі здійснюється детальний опис 3-ох найпопулярніших протоколів зв’язку, на основі яких на сьогоднішній день створюються більшість Smart Home систем: ZigBee, WiFi та Bluetooth, проводиться їх порівняльний аналіз.

1.2.1 Протокол ZigBee

ZigBee — бездротовий стандарт передачі даних. Один з найбільш популярних протоколів, використовуваних у системах Smart Home. Підтримується і розвивається однойменним альянсом ZigBeeTM, який був створений в 2002 році з метою об'єднання зусиль з розроблення найефективніших протоколів і забезпечення сумісності пристроїв різних виробників. В міру удосконалення стандарту, альянс публікує на своєму сайті специфікації стандарту, опис профілів програмного забезпечення та інші нормативні документи.

Мережі ZigBee є мережами з самоорганізовуванням та самовідновленням, оскільки ZigBee пристрої при вмиканні живлення, завдяки вбудованому програмному забезпеченню, вміють самі знаходити один одного й формувати мережу, а у разі виходу з ладу котрогось із вузлів можуть встановлювати нові маршрути для передачі повідомлень.

ZigBee – стандарт для набору високорівневих протоколів зв'язку, що використовують невеликі, малопотужні цифрові приймачі, заснований на стандарті IEEE 802.15.4-2006 для бездротових персональних мереж, таких як, наприклад, бездротові навушники, що з'єднані з мобільними телефонами за допомогою радіохвиль короткохвильового діапазону. Технологія визначається специфікацією ZigBee, яка розроблена з метою бути простішою та дешевшою, ніж інші персональні мережі, такі як Bluetooth. ZigBee призначений для мобільних пристроїв, де необхідна тривала робота від батарей і безпечність передачі даних у мережі.

Альянс ZigBee — є органом, який забезпечує і публікує стандарти ZigBee, він також публікує профілі додатків, що дозволяє OEM-виробникам створювати сумісні продукти. Поточний список профілів додатків, опублікованих, або вже знаходяться в роботі:

1) Домашня автоматизація;
2) Раціональне використання енергії (ZigBee Smart Energy 1.0/2.0);

3) Автоматизація комерційного будівництва;
4) Телекомунікаційні програми;
5) Персональне та лікарняне спостереження;
6) Дистанційне керування.

Відносини між IEEE 802.15.4 і ZigBee подібні до таких, що є між IEEE 802.11 і альянсом Wi-Fi. Специфікація ZigBee 1.0 була ратифікована 14 грудня 2004 і доступна для членів альянсу ZigBee. Порівняно недавно, 30 жовтня 2007 р., була розміщена специфікація ZigBee 2007. Про перший профіль програми — «Домашня автоматизація» ZigBee, було оголошено 2 листопада 2007. ZigBee працює в промислових, наукових і медичних (ISM-діапазон) радіодіапазонах: 868 МГц в Європі, 915 МГц у США та в Австралії, і 2.4 ГГц у більшості країн у світі (під більшістю юрисдикцій країн світу). Як правило, виробники чипів ZigBee, поєднують радіо- й мікроконтролер з розміром Flash-пам'яті від 60К до 128К. Радіомодуль також можна використовувати окремо з будь-яким процесором та мікроконтролером. Як правило, виробники радіомодулів пропонують також набір програмного забезпечення ZigBee, хоча доступні й інші незалежні стеки. Оскільки ZigBee може активуватися (тобто переходити від режиму сну до активного) за 30 мілісекунд або менше, затримка відгуку пристрою може бути дуже низькою, особливо в порівнянні з Bluetooth, для якого затримка, що утворюється при переході від сплячого режиму до активного, звичайно досягає трьох секунд. Оскільки ZigBee більшу частину часу перебуває в сплячому режимі, рівень споживання енергії може бути дуже низьким, завдяки чому досягається тривала робота від батарей. Перший випуск стека зараз відомий під назвою ZigBee 2004. Другий випуск має назву ZigBee 2006, і, в основному, замінює структуру MSG / KVP, що використовується в ZigBee 2004 разом з «бібліотекою кластерів». Стек 2004 зараз більш-менш вийшов з ужитку. Реалізація ZigBee 2007 в наш час є поточною, вона містить два профілі стека, профіль стека № 1 (який називають просто ZigBee) для домашнього і дрібного комерційного використання, і профіль стека № 2 (який

називають ZigBee Pro). ZigBee Pro пропонує більше функцій, таких як широкомовлення, маршрутизацію за схемою «багато-до-одного» і високу безпеку з використанням симетричного ключа (SKKE), в той час як ZigBee (профіль стека № 1) займає менше місця в оперативній і Flash-пам'яті . Обидва профілі дозволяють розгорнути повномасштабну мережу з комірчастою топологією і працюють зі всіма профілями додатків ZigBee.

Існують три різних типи пристроїв ZigBee.

I. Координатор ZigBee (ZC) — найвідповідальніший пристрій, який формує основну топологію дерева мережі і може зв'язуватися з іншими мережами. У кожній мережі має бути хоча б один координатор ZigBee, який повинен запустити мережу на початку. Він має зберігати інформацію про мережі, та виконувати функцію довіреного центра та сховища секретних паролів;
II. Маршрутизатор ZigBee (ZR) — даний маршрутизатор може виступати як проміжний маршрутизатор, передаючи дані з інших пристроїв. Він також може запускати додатки;
III. Кінцевий пристрій ZigBee (ZED) — його функціональна навантаженість дозволяє йому обмінюватися інформацією з вузлом вищого рівня (координатором або з маршрутизатором), але він не може передавати дані з інших пристроїв. Така функціональність дозволяє вузлу перебувати в сплячому стані левову частину часу, що дозволяє економити енергоресурс батарей. ZED потребує мінімальної кількості пам'яті, й тому може бути дешевшим у виробництві, аніж ZR чи ZC.

Приклад класичної Zig-Bee мережі з інтеграцією з дротовою мережею Ethernet показаний на рис. 1.2.

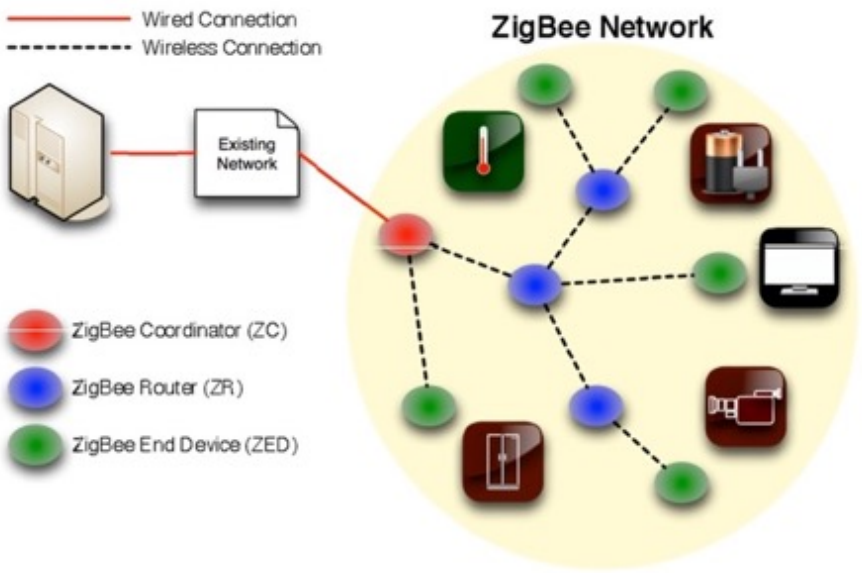

Рисунок 1.2 – Мережа ZigBee інтегрована з мережею Ethernet [11]

ZigBee 2007 повністю сумісний з пристроями ZigBee 2006. Пристрій ZigBee 2007 може підключатися і працювати з мережею ZigBee 2006, і навпаки. Але у зв'язку з наявністю відмінностей в опціях маршрутизації, пристрої ZigBee Pro можуть бути тільки кінцевими пристроями (ZEDs) мереж ZigBee 2006, і навпаки, пристрої ZigBee 2006 і ZigBee 2007 можуть бути тільки кінцевими пристроями в мережі ZigBee Pro. При цьому додатки, які запускаються на пристроях, працюють однаково, незалежно від реалізації профілю стека [11].

Підсумовуючи, сформулюємо основні переваги і недоліки технології ZigBee.

Переваги:

- Низьке енергоспоживання;
- Відносно великий радіус дії;
- Швидкий вихід зі сплячого режиму (близько 3 мс);
- Підтримка різних мережевих топологій, у тому числі мережевої, що здатна виконувати “self-healing” мережі у випадку відключення складових мережі;

Недоліки:

- Відсутність ZigBee адаптерів в існуючих комп’ютерах, смартфонах і т.д.;

- Необхідність встановлення вартісних адаптерів на сервері або створення шлюзу між ZigBee і Wi-Fi мережею;
- Недостатньо високий рівень стандартизації та єдиної програмно-апаратної платформи для розробки складних додатків;
- Часто занадто мала швидкість передачі даних. Більша частина пакетів витрачається на передачу пакетів, що містять адресну інформацію, пакети синхронізації і т.д. Корисна швидкість передачі складає близько 30 кбіт/сек.

1.2.2 Wi-Fi

Wi-Fi - загальновживана назва для стандарту IEEE 802.11 передачі цифрових потоків даних по радіоканалах. У Smart Home дана технологія використовується рідше у порівнянні з ZigBee, проте має деякі переваги. Поточні реалізації Wi-Fi дозволяють отримати швидкість передачі даних понад 100 Мбіт/с, при цьому користувачі можуть переміщуватися між точками доступу на території покриття мережі Wi-Fi, використовуючи пристрої, оснащені клієнтськими приймально-передавальними пристроями Wi-Fi та отримувати доступ в Інтернет [12]. Структура звичайної Wi-Fi мережі з виходом до Інтернету показана на рис. 1.3.

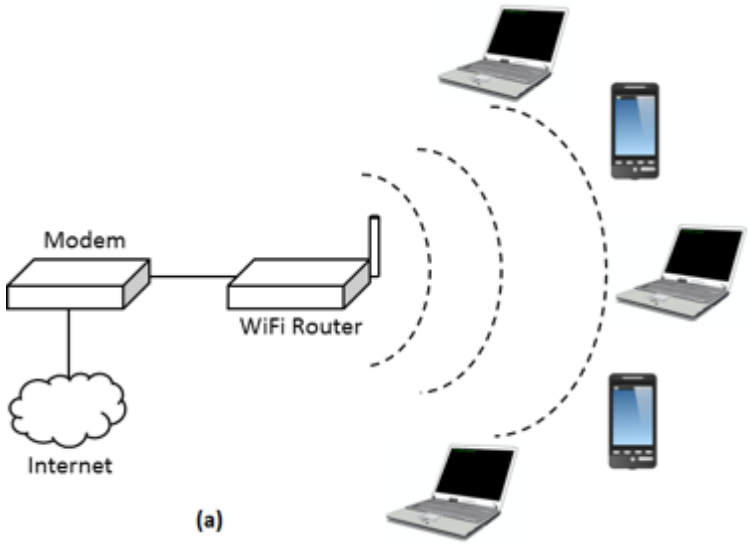

Рисунок 1.3 – Структурна схема звичайної Wi-Fi мережі [12]

Зазвичай схема Wi-Fi мережі містить не менш однієї точки доступу та може легко масштабуватись. Також можливо підключення двох клієнтів в режимі точка-точка (Ad-hoc), коли точка доступу не використовується, а клієнти з'єднуються за участю мережевих адаптерів «напряму». Продовженням цього напрямку стала технологія *Wi-Fi Direct*.

Wi-Fi Direct дозволяє комп'ютерам і портативним ґаджетам зв'язуватися один з одним безпосередньо за існуючим протоколом Wi-Fi без використання маршрутизаторів і точок доступу. Тобто з'єднання встановлюється так само просто, як через Bluetooth. Важливим моментом є те, що для організації прямого з'єднання досить, щоб тільки один з пристроїв відповідало стандарту Wi-Fi Direct. Іншими словами, до сертифікованої апаратури може бути підключено будь-яке сучасне обладнання з підтримкою Wi-Fi. Максимальна відстань передачі даних досягає 100 метрів. Організація Wi-Fi Alliance почала сертифікацію бездротових пристроїв відповідно до стандарту Wi-Fi Direct у жовтні 2010 [13]. Можливі типи з’єдань по протоколу Wi-Fi Direct показані на рис. 1.4.

Рисунок 1.4 – Типи з’єднань Wi-Fi Direct [13]

Точка доступу передає свій ідентифікатор мережі (SSID) з допомогою спеціальних сигнальних пакетів на швидкості 0,1 Мбіт/с кожні 100 мс. Тому 0,1 Мбіт/с — найменша швидкість передачі даних для Wi-Fi. Знаючи SSID мережі, клієнт може з’ясувати, чи можливо підключення до даної точки доступу. При потраплянні в зону дії двох точок доступу з ідентичними SSID приймач може вибирати між ними на основі даних про рівень сигналу. Стандарт Wi-Fi дає клієнту повну свободу при виборі критеріїв для з'єднання.

Наявність Wi-Fi-зон дозволяє користувачу підключитися до точки доступу, а також підтримувати з'єднання декількох комп'ютерів між собою. Дальність передавання інформації залежить від потужності передавача (яка в окремих моделях обладнання регулюються програмно), наявності та характеристики перешкод, типу антени.

Підсумовуючи, сформулюємо основні переваги і недоліки використання технології Wi-Fi у сфері Smart Home.

Переваги:

- Wi-Fi-пристрої широко поширені на ринку. Гарантована сумісність з усіма пристроями, що мають відповідний адаптер;
- Можливіть роботи пристроїв в Ad-hoc режимі та за стандартом Wi-Fi direct, що не вимагає наявності роутера та модема;
- Високий рівень стандартизації. Сумісність між пристроями різних виробників;

Недоліки:

- Вище енергоспоживання у порівнянні з такими технологіями як Bluetooth або ZigBee;
- Неможливість конфігурації mesh-топології без додаткових засобів, для цього використовують спеціальні вартісні повторювачі;
- Відносно велика вартість;
- Надлишкова, як для потреб Smart Home, швидкість передачі даних;

- Радіус дії та швидкість передачі залежить лише від потужності маршрутизатора або адаптера клієнта.

1.2.3 Bluetooth

Bluetooth - технологія бездротового зв'язку, створена у 1998 році групою компаній: Ericsson, IBM, Intel, Nokia, Toshiba. В Smart Home дана технолгія використовується рідше за Wi-Fi та ZigBee, проте має переваги у вигляді низької вартості, зручної мережевої топології та великої швидкості передачі даних.

Нині розробки в області Bluetooth ведуться групою Bluetooth SIG (Special Interest Group), до якої входять також Lucent, Microsoft та інші компанії, чия діяльність пов'язана з мережевими технологіями. Основне призначення Bluetooth — забезпечення економного (з точки зору споживаного струму) і дешевого радіозв'язку між різноманітними типами електронних пристроїв, таких як мобільні телефони та аксесуари до них, портативні та настільні комп'ютери, принтери та інші. Причому, велике значення приділяється компактності електронних компонентів, що дає можливість застосовувати Bluetooth у малогабаритних пристроях розміром з наручний годинник.

Інтерфейс Bluetooth дає змогу передавати як голос (зі швидкістю 64 Кбіт/с), так і дані. Для передачі даних можуть бути використані асиметричний (721 Кбіт/с в одному напрямку і 57,6 Кбіт/с в іншому) та симетричний (432,6 Кбіт/с в обох напрямках) методи. Працюючи на частоті 2.4 ГГц, прийомо-передавач (Bluetooth-chip) дає змогу встановлювати зв'язок у межах 10 або 100 метрів. Різниця у відстані, безумовно, велика, однак з'єднання в межах 10 метрів дає змогу зберегти низьке енергоспоживання, компактний розмір і досить невисоку вартість компонентів. Так, малопотужний передавач споживає всього 0.3 мА в режимі standby і в середньому 30 мА під час обміну інформацією.

У стандарті Bluetooth передбачене шифрування даних, що передаються з використанням ключа ефективної довжини від 8 до 128 біт і можливістю

вибору односторонньої або двосторонньої аутентифікації. Додатково, до шифрування на рівні протоколу, може бути використано шифрування на програмному рівні.

Технологія Bluetooth працює за принципом FHSS (Frequency-hopping spread spectrum). Коротко це можна пояснити так: передавач розбиває дані на пакети і передає їх за псевдовипадковим алгоритмом стрибкоподібної перебудови частоти (1600 разів в секунду), або шаблоном, складеному з 79 підчастот. «Зрозуміти» один одного можуть тільки ті пристрої, які налаштовані на один і той самий шаблон передачі — для сторонніх приладів передана інформація буде звичайним шумом. Основним структурним елементом мережі Bluetooth є так звана «пікомережа» (piconet) — сукупність від 2 до 8 пристроїв, що працюють на одному і тому ж шаблоні. Загальна схема пікомережі наведена на рис. 1.5.

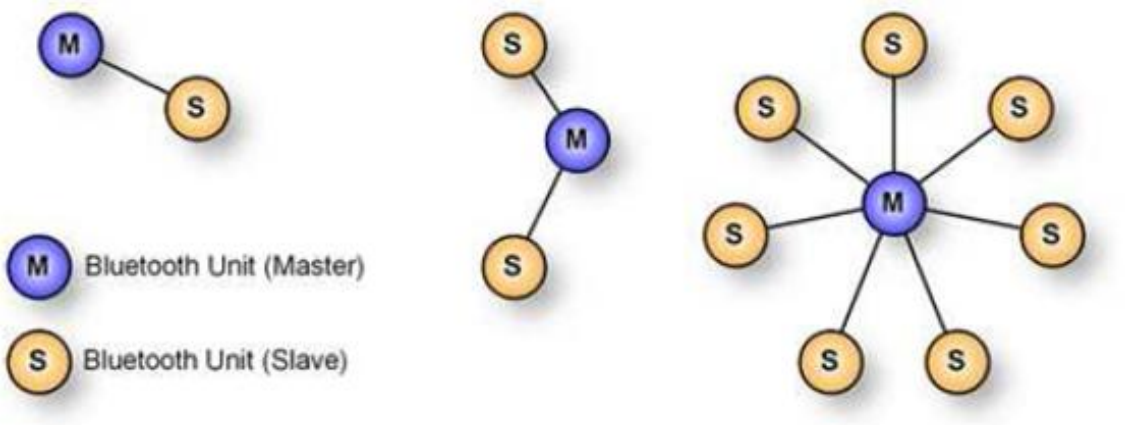

Рисунок 1.5 – Мережа Bluetooth Piconet [14]

У кожній пікомережі один пристрій працює як активний (master), а інші як пасивні (slave). Активний пристрій визначає шаблон, на якому працюватимуть усі пасивні пристрої його пікомережі, і синхронізує її роботу. Стандарт Bluetooth передбачає з'єднання незалежних і навіть не синхронізованих між собою пікомереж (до 10) в так звану «scatternet». Для

цього кожна пара пікомереж повинна мати як мінімум один спільний пристрій, який буде активним в одній і пасивним в іншій. Таким чином, у межах окремої scatternet з інтерфейсом Bluetooth може бути одночасно пов'язано максимум 71 пристрій, однак ніхто не обмежує застосування пристроїв-гейтів, які використовують той же Internet для більш далекого зв'язку. Загальна схема «scatternet» мережі наведена на рис. 1.6.

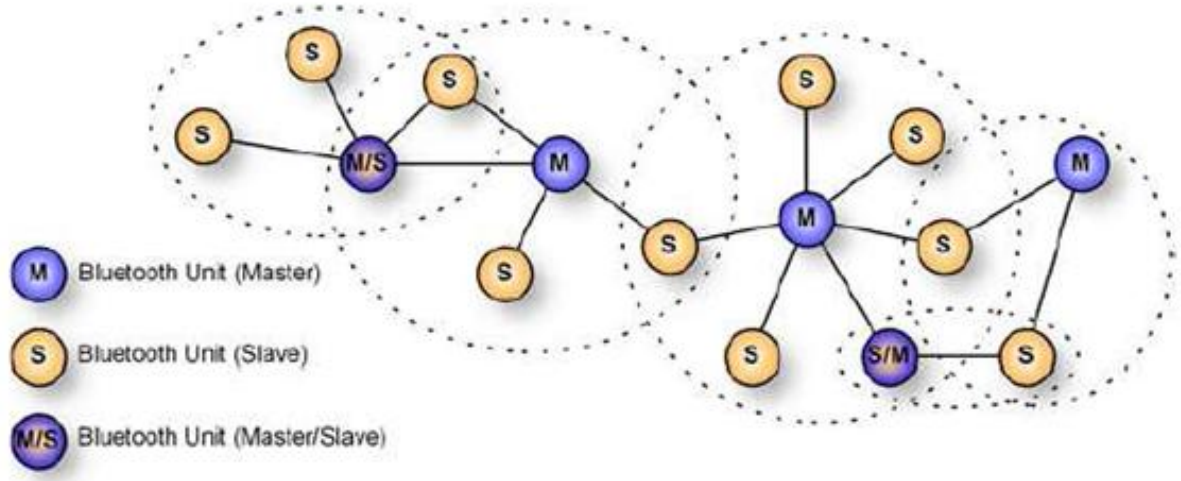

Рисунок 1.6 – Мережа Bluetooth Scatternet [14]

Ініціалізацією, щодо Bluetooth, прийнято називати процес встановлення зв'язку. Її можна розділити на три етапи:

- Генерація ключа Kinit;
- Генерація ключа зв'язку (він носить назву link key і позначається, як Kab);
- Аутентифікація.

Перші два пункти входять в так звану процедуру парінга. Парінг - процес зв'язку двох (або більше) пристроїв з метою створення єдиної секретної величини Kinit. Надалі до передаваних даних застосовується симетричне 128-бітне AES шифрування з використанням цього ключа.

Підсумовуючи, сформулюємо основні переваги і недоліки технології Bluetooth.

Переваги:

- Високий рівень стандартизації;
- Надійна система захисту даних за замовчуванням;
- Велике розмаїття модулів Bluetooth під різні задачі;
- Невелика вартість;
- Наявніть адаптерів у більшості гаджетів;
- Масштабована архітектура, заснована на принципі “scatternet”;

Недоліки:

- Відносно велике енергоспоживання;
- Невеликий радіус дії;
- Відсутність mesh-топології;

1.2.4 Bluetooth Low Energy

Bluetooth з низьким енергоспоживанням або Bluetooth смарт - технологія цифрової бездротової передачі даних з наднизьким енергоспоживанням, заснована на недорогих мікросхемах в передавальних пристроях. У сфері Smart Home дана технологія знайшла використання лише кілька років тому і на 2017 рік є одним з найбільш перспективних напрямків розвитку IoT та Smart Home.

Інтеграція Bluetooth з низьким енергоспоживанням в специфікацію ядра завершена на початку 2010 року. Першим пристроєм, що включає в себе цю технологію, був iPhone 4S, випущений в жовтні 2011. Інші виробники представили пристрої з Bluetooth Smart Ready в 2012 році.

Споживаючи менше енергії, технологія Bluetooth з низьким енергоспоживанням запропонує тривалий забезпечення зв'язку і з'єднує маленькі пристрої типу датчиків і мобільні пристрої в межах персональних мереж (PAN) [14].

Процес з’єднання в Bluetooth LE зображено на рис. 1.7.

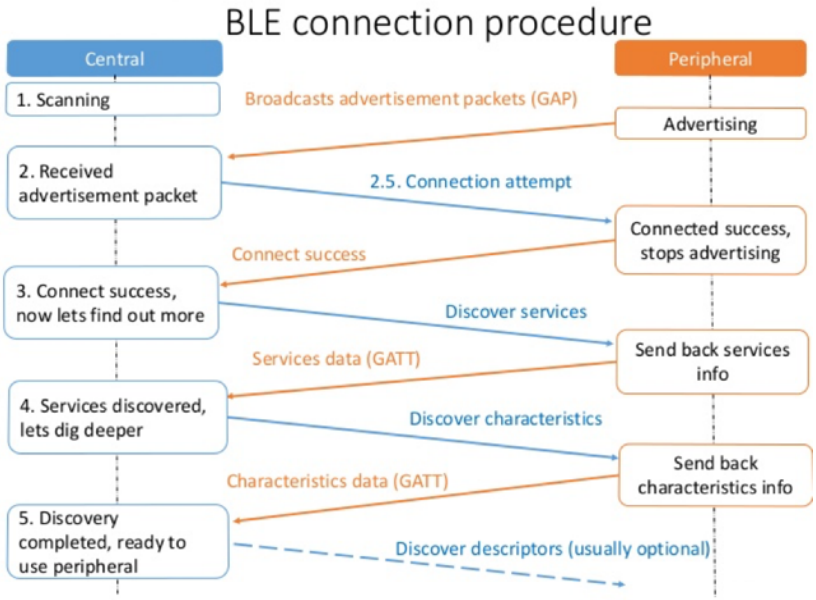

Рисунок 1.7 – Процес встановлення з’єднання Bluetooth [14]

Специфікація Bluetooth 4.0 (і більш пізні) фактично визначає дві бездротові технології: BR / EDR (класичний Bluetooth, що розвивається, починаючи з першої версії стандарту) і BLE (Bluetooth Low Energy).

Пристрої, в яких застосований BLE, можуть бути як дворежимні BR / EDR / BLE (називаються Bluetooth Smart Ready), сумісні з класичними Bluetooth-пристроями, так і однорежимні BLE (Bluetooth Smart) [15]. Особливості структури цих протоколів показані на рис. 1.8.

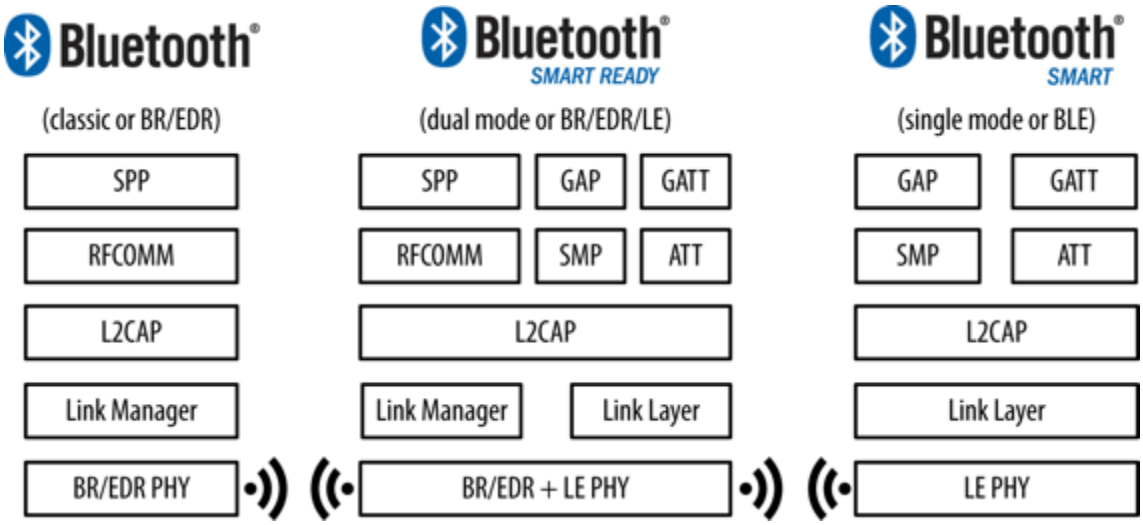

Рисунок 1.8 - Схема стандартів Bluetooth [14]

Основними блоками Bluetooth-пристроїв є:

a) Додаток - реалізує корисну для кінцевого користувача логіку роботи
b) Головний пристрій, хост - надає верхні рівні стека протоколів Bluetooth. Містить наступні протоколи:
    - GAP (Generic Access Profile) - профіль загального доступу;
    - GATT (Generic Attribute Profile) - профіль загальних атрибутів;
    - L2CAP (Logical Link Control and Adaptation Protocol) - протокол логічного з'єднання і адаптації;
    - ATT (Attribute Protocol) - протокол атрибутів;
    - SM (Security Manager) - менеджер безпеки;
    - HCI (Host Controller Interface) - інтерфейс хост - контролер, частина на стороні хоста.
c) Контролер - займається нижніми рівнями Bluetooth. Містить протоколи:
    - HCI - інтерфейс хост - контролер з боку контролера;
    - LL (Link Layer) - рівень з'єднання;
    - PHY - фізичний рівень.

Комерційні продукти зазвичай використовують один з наступних апаратних рішень:

- SoC - однокристальна система, що об'єднує в собі додаток, хост і контролер. Застосовується в компактних недорогих пристроях, таких як датчики;
- Рішення на двох мікросхемах, при якому додаток і хост з'єднані з контролером за допомогою UART, USB, SDIO. Така конфігурація може використовуватися, наприклад, в мобільних пристроях;
- Рішення на двох мікросхемах, в якому додаток з'єднується з пристроєм зв'язку (хост і контролер) за пропрієтарним протоколом.

Наприкінці 2016 року індустріальна група Bluetooth SIG затвердила характеристики нового стандарту Bluetooth 5. Дальність бездротового з'єднання збільшилася на цілих 400%, а швидкість передачі подвоїлася в порівнянні з

Bluetooth минулого покоління. Оновлений стандарт також більш стабільний і відрізняється високим ступенем захищеності. Компанія заявила, що Bluetooth 5 націлений на носяться пристрої, індустріальну і домашню автоматизацію, а також корпоративні ринки. Основною і найбільш очікуваною новиною було впровадження підтримки mesh-топології, що дозволятиме значно збільшити розміри мережі, заснованої на Bluetooth Low Energy [16].

Продукти з підтримкою Bluetooth 5 з'являться у продажу в найближчі кілька місяців. Згідно з даними компанії Bluetooth SIG, в Bluetooth 5 покращиться процес збору даних для "маячків" - маленьких пристроїв, які відправляють цільову інформацію, яку користувачі можуть переглянути зі смартфона або планшета під час своєї прогулянки по магазину. На початку 2017 року випущено перший пристрій оснащений Bluetooth 5.0 – смарфон Galaxy S8.

Підсумовуючи, сформулюємо основні переваги і недоліки технології Bluetooth LE.

Переваги:

- Високий рівень стандартизації та сумісність між різними протоколами;
- Низька вартість;
- Ультранизьке енергоспоживання;
- Швидкість передачі більше 1 мбіт/сек;
- Продуктивність модуля може налаштовуватись в залежності від потреб (збільшення швидкості передачі за рахунок зменшення радіусу дії або навпаки);
- Простота з’єднання;
- Захищеність зв’язку;
- Наявність уніфікованих API для роботи з периферією Bluetooth LE під більшість платформ: Android, iOS, desktop (Java, JS, Python, C++, C#, Ruby та інші);

Недоліки:

- Підтримка mesh-топології з’явилася лише в останній версії 5.0 та продовжує перебувати на етапі активної розробки.

1.3 Висновок

Підсумовуючи, перед Wi-Fi та ZigBee технологіями Bluetooth LE має декілька вагомих переваг:

- Можливість конфігурації модулів в залежності від потреб: (збільшення радіусу дії в 2 рази при максимальній швидкості 256 кбіт/сек, або в 4 рази при швидкості 128 кбіт/сек);
- Високий рівень стандартизації;
- Практично повна зворотна сумісність з попередніми версіями;
- Наявність адаптерів у всіх сучасних смартфонах та ноутбуках;
- Кількість вузлів у локальній мережі майже ніяк не обмежена (підключення може відбуватися лише у момент передачі даних);

Основні характеристики порівнюваних стандартів зв’язку показані на рис. 1.9.

	ZIGBEE	Bluetooth Low energy	Wi-Fi
Range	10-100 m	>60m (10m for Classic BT)	Depends on specification
Power	Low	Very Low (High for classic BT and medium for others)	High (variable for WiFi Direct)
Entries	254 (>64000 per network)	➢ 2 Billion ➢ (Classic: 7)	Depends on no. of IP addresses
Latency	Low	3 ms (compared to 100ms in classic BT)	Variable
Self healing	Yes	-	Yes
Topologies	Mesh, Star and Cluster-tree	Star	Star, Point-to-Point
Data transmission rate	Up to 250Kbps	1Mbps (BT v4.0: 25Mbps)	11Mbps & 54Mbps (250 Mbps: WiFi Direct)
Bandwidth	2.4GHz, 915MHz & 868 MHz	2.4 GHz only (BT + HS: 6-9 GHz)	2.4, 3.6 & 5 GHz
Transmission technique	DSSS DS/FA?	Adaptive FHSS (Classic BT: FHSS)	DSSS, CCK & OFDM

Рисунок 1.9 – Порівняльна характеристика стандартів бездротового зв’язку[15]

Описані вище переваги обумовили вибір Bluetooth Low Energy у якості основи для побудови домашньої системи оптимізації енергоспоживання.

1.4 Мета роботи

Метою роботи є побудова системи оптимізації енергоспоживання для розумного будинку на основі протоколу зв'язку Bluetooth Low Energy, а саме дві її основні складові:

- *Smart Switch* – розумна розетка з модулем Bluetooth LE, що відправляє дані про споживаний/генерований струм та напругу підключеного до неї електроприладу, поточну температуру, дозволяє здійснювати його комутацію за керуючим сигналом;
- *Smart Dispatcher* – сервер для комп'ютера, що забезпечує автоматизоване управління системою у з наступними функціями:
  1) Збирає дані від усіх пристроїв системи;
  2) Надає можливість користувачу переглядати статистику роботи електроприладів;
  3) Надає можливість в ручному режимі підключати/відключати їх від електромережі;
  4) На основі отриманих даних в автоматичному режимі приймає рішення про комутацію електроприладів за наступними показниками, значення яких встановлюються користувачем у додатку:
     - Відключення приладу від мережі у разі отримання від нього даних про підвищену температуру;
     - Відключення приладу від мережі у разі аномальної зміни струму;
     - Відключення приладів у разі перевищення встановленого ліміту загальної споживаної потужності домогосподарства;
     - Відключення/підключення приладу від електромережі в залежності від встановленого для нього добового графіку роботи.

2. РОЗРОБКА ПРИСТРОЮ МОНІТОРИНГУ І КЕРУВАННЯ ЕЛЕКТРИЧНИМИ ПРИЛАДАМИ

2.1 Вимоги до пристрою

Даний пристрій призначений для моніторингу та керування потоками електричної енергії, що споживається (або виробляється) електричним приладами.

Вимоги до пристрою:

- пристрій живиться від мережі (220В, 50Гц);
- пристрій повинен мати Bluetooth Low Energy модуль підвищеного радіусу дії для забезпечення з'єднання з сервером системи;
- пристрій повинен мати мікроконтролер, що забезпечує роботу Bluetooth LE модуля, вимірювання аналогових значень технологічних параметрів (струму, напруги і температури) та перетворення їх у 10-розрядний двійковий код, формування сигналів для включення або відключення електричних приладів;
- пристрій повинен мати зовнішню оптичну розв'язку для включення або відключення живлення електричних приладів.

На рис. 2.1 наведено структурну схему пристрою.

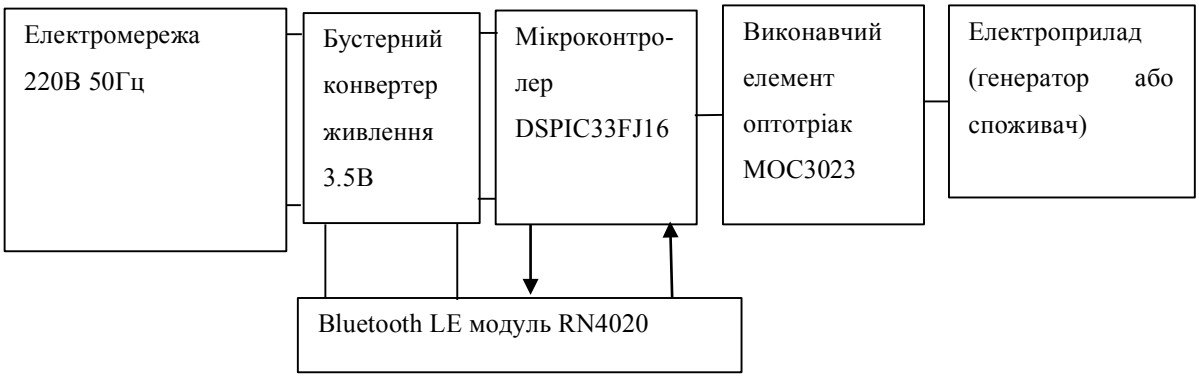

Рисунок 2.1 – Структурна схема пристрою.

Пристрій, побудований за наведеною структурною схемою, як вже було згадано раніше, живиться від електромережі (220В, 50Гц). Змінна напруга на вході пристрою за допомогою бустерного конвертера перетворюється у стабілізовану постійну напругу 3.5В, що живить мікроконтролер DSPIC33FJ16 та Bluetooth Low Energy модуль RN4020 (Microchip). RN4020 забезпечує бездротовий зв'язок пристрою з комп'ютером та передачу даних в обох напрямках. Мікроконтролер DSPIC33FJ16 взаємодіє з RN4020 за допомогою UART каналу, формує допоміжні сигнали, здійснює вимірювання аналогових значень необхідних параметрів роботи електричного приладу, таких як струм, напруга та температура та їх перетворення у цифровий формат. Крім цього мікроконтролер формує сигнали для виконавчого елементу - оптотріаку MOC3023, що виконує комутацію електроприладу. У електричному приладі встановлені датчики напруги та струму, що у вигляді аналогового сигналу (0-3.5В) подаються на входи АЦП мікроконтролера.

2.2 Bluetooth LE модуль Microchip RN4020

Компанія Microchip є світовим лідером на ринку мікроконтроллерів. Microchip пропонує кілька Bluetooth LE рішень - від окремих мікросхем до модулів. Однією з ключових переваг модулів є те, що зазвичай вони вже сертифіковані виробником і мають сертифікати FCC або CE / ETSI і пройшли випробування в Bluetooth SIG. Інша перевага модулів, це те, що вони мають гарантовано працюючу радіочастотну частину. Модулі як правило мають інтегрований стек і конфігуруються через прості команди, що значно скорочує час розробки пристроїв.

Одним з найбільш популярних модулів є RN4020, його структурна схема наведена на рис. 2.2.

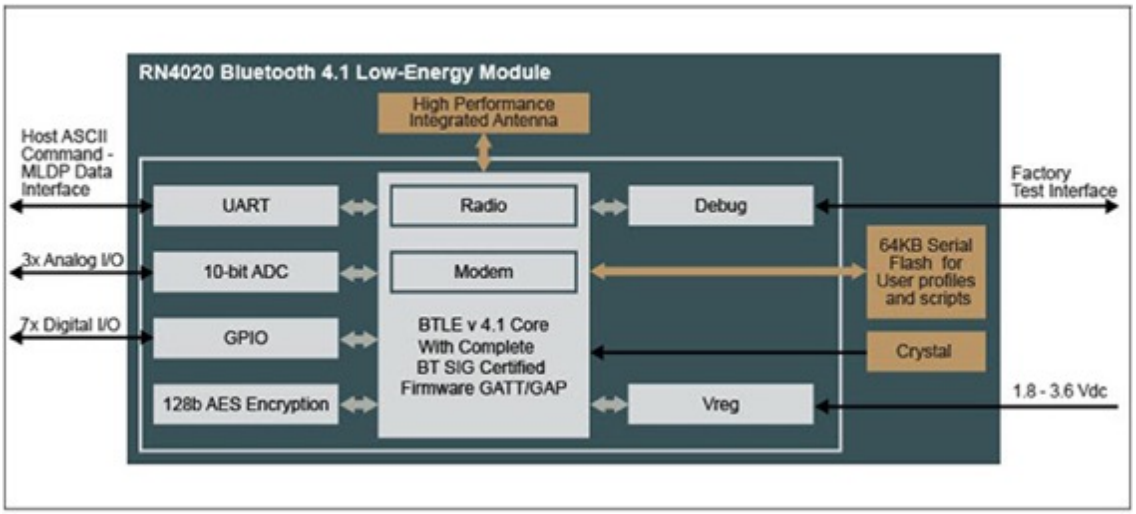

Рисунок 2.2 - Структурна схема модуля RN4020 [17]

Модулі RN4020 мають можливість виконання скриптів, записаних в пам'ять модуля, що дозволяє застосовувати їх в пристроях з керуючим мікроконтролером з достатньо простим програмним забезпеченням [17].

Основні його переваги:

- Інтегрований стек Bluetooth 4.1 Low-Energy (BLE);
- Завершене рішення для управління і контролю через малоспоживаюче бездротове з'єднання;
- Прості ASCII команди для управління модулем;
- Сумісність з іншими Bluetooth LE пристроями;
- Вбудована антена для підвищення радіусу дії, що дозволяє його використовувати на відстанях до 150 м.

Даний модуль має дуже компактні розміри, що робить його дуже привабливим для використання в пристроях, для яких вільний простір на платі є критичним показником. Зовнішній вигляд модуля RN4020 показано на рис. 2.3.

Рисунок 2.3 – Модуль Bluetooth LE RN4020 [17]

Модуль RN4020 може бути підключений до будь-якого мікроконтролера, що має інтерфейс UART. По UART передаються налаштування, команди і дані.

Команди та їх параметри вводяться в вигляді ASCII символів (великими чи малими літерами - не має значення). Команди та параметри розділяються комами. Ознакою закінчення команди є символ кінця рядка (<CR>).

Приклад команди:

SN, myDevice //задає ім'я модуля

GN //повертає ім'я модуля

На рис. 2.4 наведено функціональну схему взаємодії RN4020 з довільним мікроконтролером.

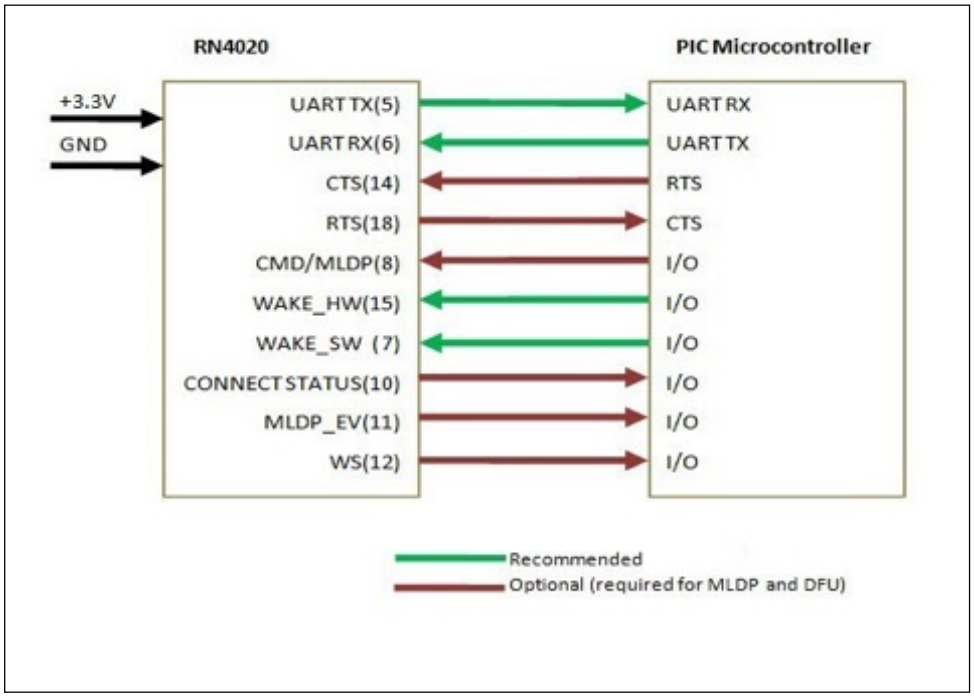

Рисунок 2.4 – Схема взаємодії RN4020 з мікроконтролером [17]

2.3 Мікроконтролер Microchip dsPIC33FJ16

В якості контролера використовується спеціалізований мікроконтролер компанії Microchip DSPIC33FJ16MC102, що дозволяє реалізувати весь описаний функціонал в однопроцесорній схемі із максимальним використанням переваг сучасних сигнальних процесорів. Структурна схема DSPIC33FJ16MC102 наведена на рис 2.5.

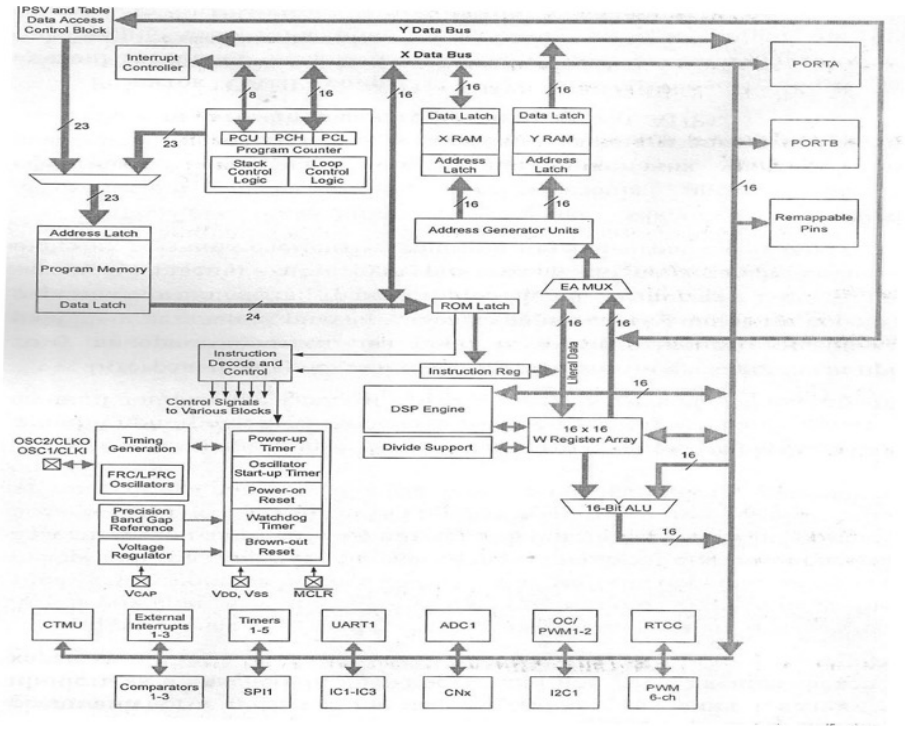

Рисунок 2.5 – Структурна схема мікроконтролера DSPIC33FJ16MC102[18]

Наведемо основні технічні характеристики даного мікроконтролера:

- Число виходів – 28;
- ПЗУ – 16 кбт;
- ОЗУ – 1 кбт;
- UART інтерфейс– 1;
- АЦП – 6 входів;
- Входи-виходи – 21 шт.;
- Багатофункціональні виходи – 16 шт.;
- 16-бітний таймер – 3 шт.;
- Компаратори – 3 шт.

Дана серія мікроконтролерів має 16 16-бітних регістрів. Кожен з регістрів може служити для збереження даних, адреси або адресного зсуву. 16-ий регістр (W15) представляє з себе Software Stack Pointer (SSP) для обслуговування переривань і викликів [18].

Схему входів/виходів мікроконтролера з позначеннями входів/виходів показано на рис. 2.6.

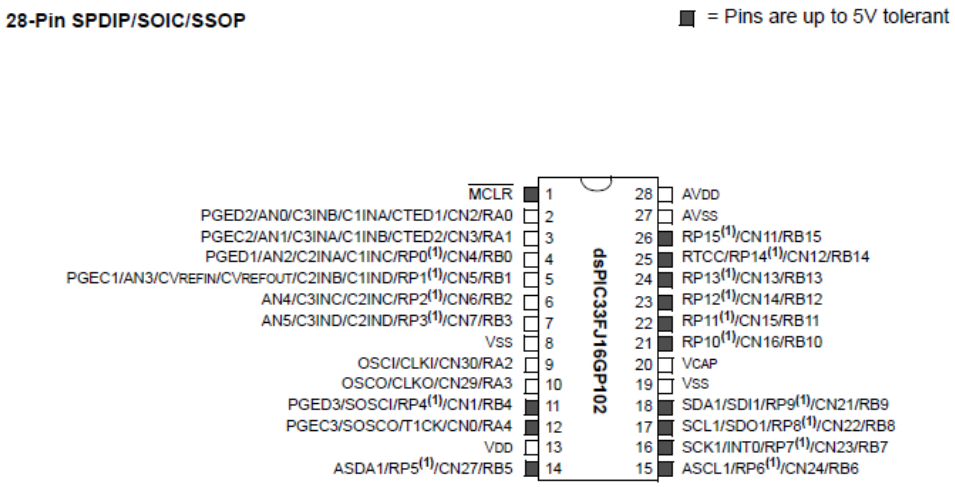

Рисунок 2.6 – Загальна структура мікроконтролера DSPIC33FJ16MC102[18]

2.4 Проектування схеми

На рис. 2.7 наведено електричну схему з’єднання RN4020 та мікроконтролера DSPIC33FJ16MC102, а також підключення до нього конектора.

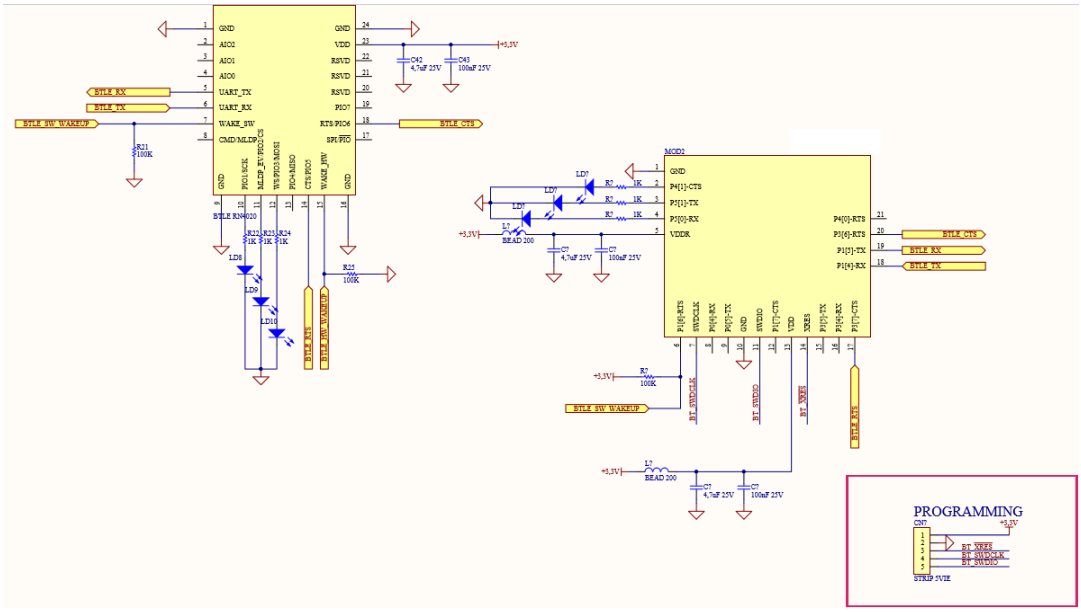

Рисунок 2.7 – Схема з’єднання RN4020 та мікроконтролера

Її було розроблено згідно вимогам структурної схеми, що зображена на рис. 2.1. Окрім наведених вище елементів вона має:

- трансформаторну розв'язку 220/70В для забезпечення безпеки демонстраційного зразка;
- оптотріак MOC3023 для включення (відключення) навантаження;
- вентильний двигун вентилятора у якості навантаження, струм живлення якого подається на шунт плати пристрою, напруга з якого підключена до входу АЦП мікроконтролера.

Згідно з розробленою схемою було розведено друковану плату у програмі PCAD і виготовлено її на підприємстві "Електронмаш" у м. Київ. На рис. 2.8 наведено вигляд проекту розробленої друкованої плати.

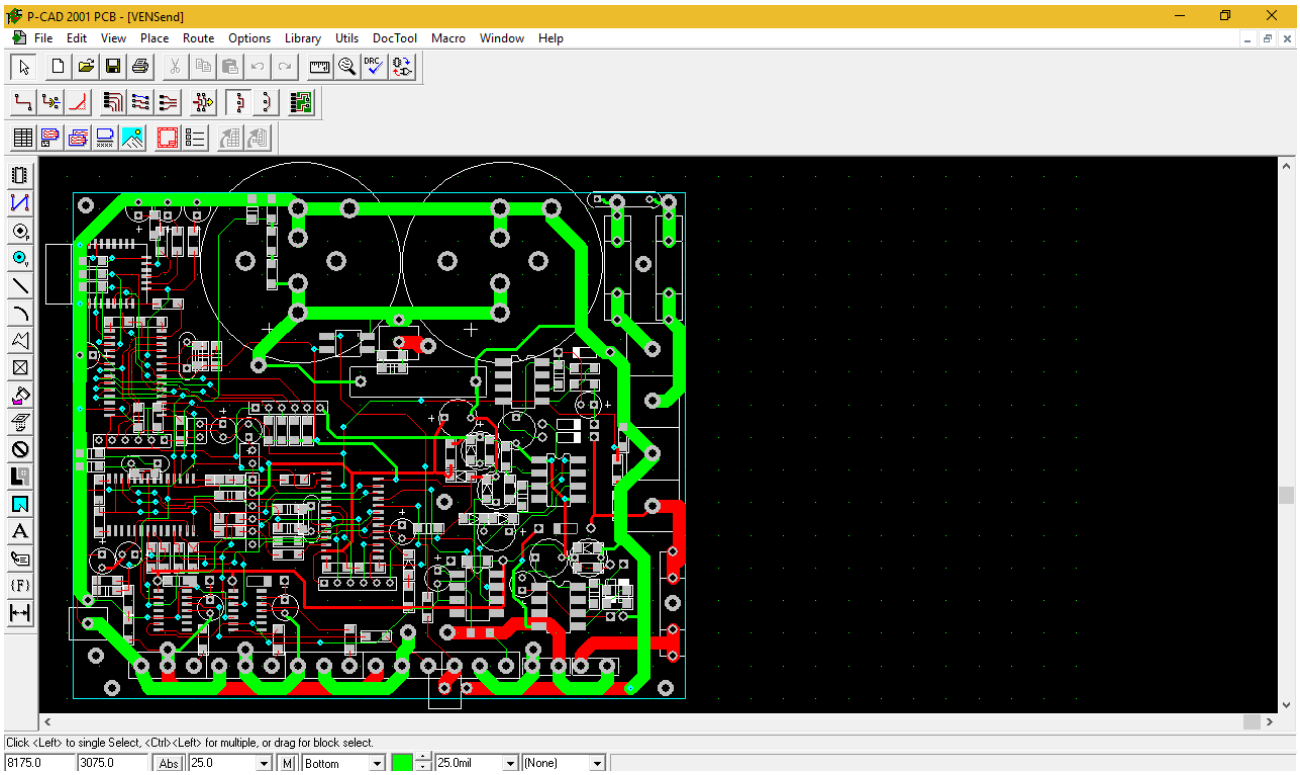

Рисунок 2.8 – Проект друкованої плати пристрою в програмі PCAD

2.5 Програмування мікроконтролера

Програмне забезпечення для мікроконтролера виконано у середовищі MPLAB X IDE із апаратними засобами відладки та програмування PICkit3.

У якості апаратного засобу відладки та програмування використано PICkit 3 рис. 2.9.

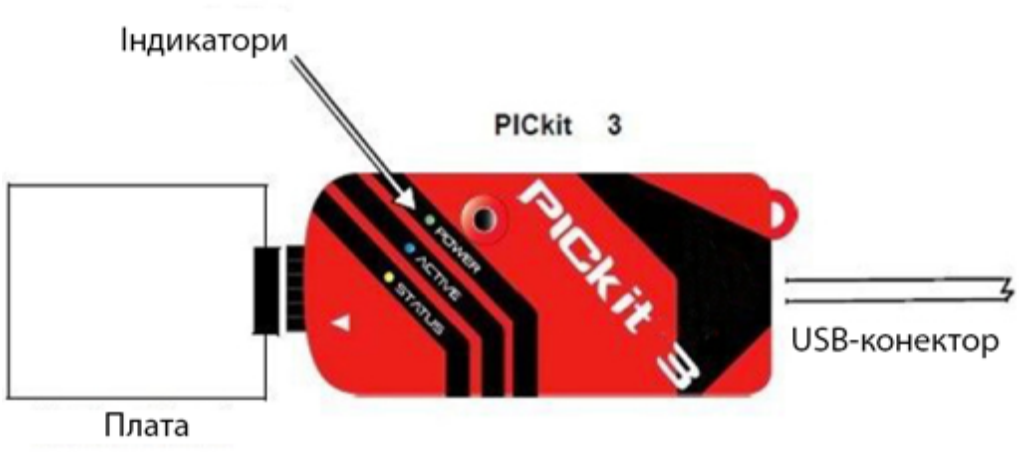

Рисунок 2.9 – відладчик-програматор PICkit 3

PICkit 3 являє собою простий, недорогий відладчик-програматор, керований за допомогою ПК з встановленим програмним забезпеченням MPLAB X IDE. PICkit 3 є невід'ємною частиною набору інструментів інженера розробника для апаратного і програмного забезпечення Microchip PIC мікроконтролерів (MCU), і контролерів dsPIC Digital Signal (DSP), які засновані на внутрішньосхемній системі послідовного програмування (ICSP) через 2-провідні послідовні інтерфейси [19].

Особливості PICkit 3:

- підтримка USB високої швидкості за допомогою стандартних драйверів;
- виконання команд в режимі реального часу;
- процесори працюють на максимальній швидкості;
- вбудований індикатор високої напруги / короткого замикання;
- низька напруга живлення, до 5В (діапазон 1.8 – 5В);
- індикатори (живлення, активний, статус);
- читання / запис програм, доступ до пам’яті мікроконтролера;
- очистка всіх типів пам'яті (EEPROM, ID конфігурації і програми).

Рекомендується ставити резистор (10 кОм) між пінами MCLR до VDD щоб забезпечити RESET пристрою.

MPLAB X IDE це програма, яка працює у різних операційних системах на комп'ютері (Windows, Mac OS, Linux) для розробки додатків для мікроконтролерів і цифрових сигнальних процесорів компанії Microchip. Вона називається інтегрованим середовищем розробки (IDE), оскільки забезпечує зручний інтерфейс для розробки коду для мікроконтролерів на основі відкритого вихідного коду NetBeans IDE від Oracle.

Весь програмний проект розподіляється на декілька файлів та папки за їх функціональним призначенням, що показано на рис. 2.10.

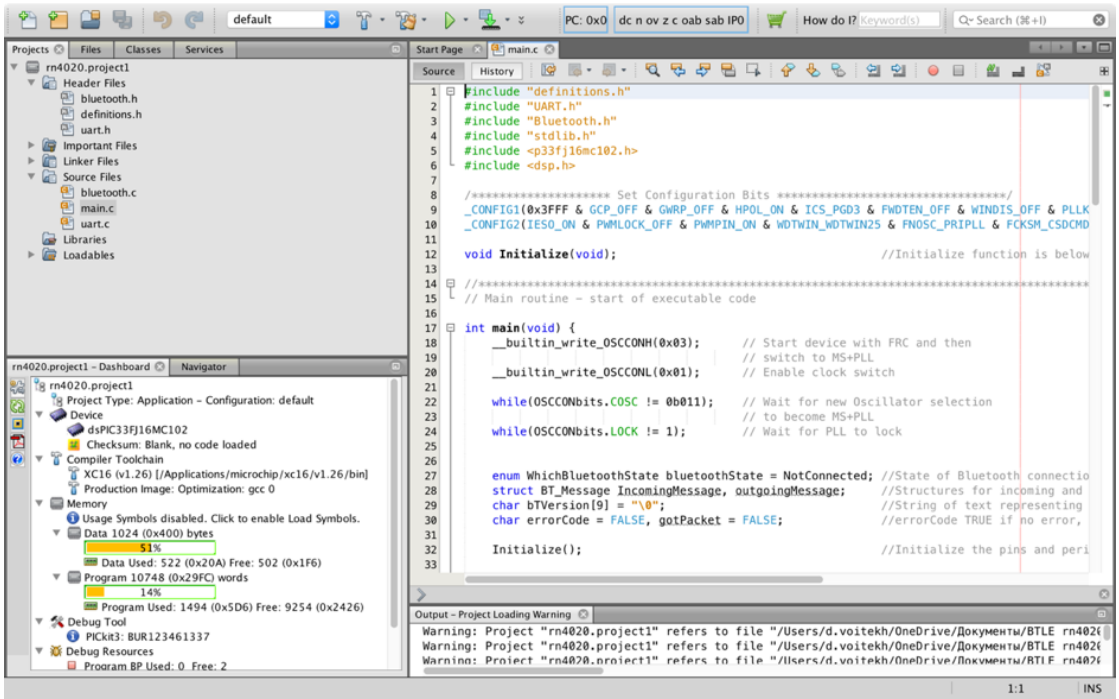

Рисунок 2.10 – Проект у програмному середовищі MPLAB X IDE

Далі наведено основні фрагменти коду проекту.

Файл “definitions.h”.

Тут відбувається присвоєння програмним змінним значень портів, задається розмір буферів, формат повідомлень тощо. Одна з частин коду даного файлу:

```
#define BLE_CONNECTED PORTBbits.RB1 //Bluetooth module is
connected to central
#define SIZE_RxBuffer 256 //UART_RX buffer size in bytes
#define SIZE_TxBuffer 256 //UART_TX buffer size in bytes
enum WhichBluetoothState {NotConnected, Connected};
struct BT_Message //form of message
{
   char Command;
   char Data;
};
```

Файл “bluetooth.c”.

Тут оголошено функції, що відповідають за передачу та прийом даних.

Далі наведено функцію що ініціалізує модуль набором команд, що запускають у ньому необхідний сервіс та активує його характеристики:

```
char BT_SetupModule()
{
      BT_SendCommand("sf,1\r"); //reset module settings
      BT_SendCommand("ss,30000000\r");
      BT_SendCommand("sr,32000000\r");
      BT_SendCommand("sn, Ener1 \r"); //set name
      BT_SendCommand("r,1\r"); //reload module to enable changes
      return TRUE
}
```

Далі наведено функцію, що передає повідомлення через UART інтерфейс на RN4020:

```
char BT_SendPacket(struct BT_Message *MessageOut)
{
   WriteTxBuffer(BT_SOF_1);
   WriteTxBuffer(BT_SOF_2);
```

```
        WriteTxBuffer((int)MessageOut->Command);
        WriteTxBuffer((int)MessageOut->Data);
        WriteTxBuffer('\r'); //Load carriage return
        WriteTxBuffer('\n'); //Load line feed
        UART_TxStart(); //Start the transmission
        return TRUE;
    }
```

Далі наведено функцію що приймає повідомлення від RN4020:

```
char BT_ReceivePacket(struct BT_Message *Message)
{
        char messageChar;
        if (IsNewRxData())
        {
                messageChar = ReadRxBuffer();
                if(messageChar == '\n')
                        return TRUE;
        }
        return FALSE;
 }
```

Файл “main.c”.

Виклик основної функції main(). Наведено фрагмент нескінченного циклу в якому відбувається підключення до пристрою з подальшою пересилкою даних та перевіркою наявності вхідних повідомлень для комутації навантаження:

```
if(bluetoothState == Connected)
{
        sendADCvalues();
        gotPacket = BT_ReceivePacket(&IncomingMessage);
        if(gotPacket == TRUE)
```

```
        {
                if(IncomingMessage.Command == SWITCH)
                {
                        TRIAC_1 = (TRIAC_1 + 1) % 2;
                }
        }
}
```

Отже дана програма передбачає наступну взаємодію з Мастером (сервером): при підключенні по Bluetooth до пристрою сервером починається передача даних (струм, напруга, температура). В свою чергу сервер може надсилати на пристрій команду **SWITCH**, після якої мікроконтролер здійснить комутацію свого електроприладу через оптотріак.

Для якісного тестування роботи системи було розроблено 2 макетних зразки описаних пристроїв. Розміщено їх на одному стенді як показано на рис. 2.11.

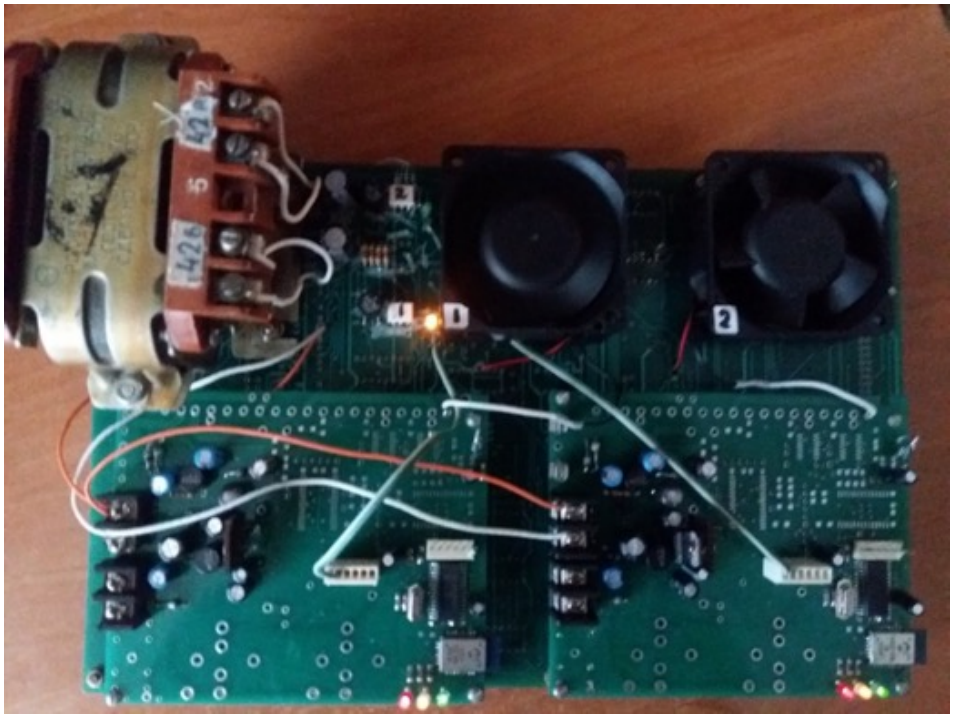

Рисунок 2.11 – Два макетних зразки пристрою моніторингу і контролю

3. АРХІТЕКТУРА СИСТЕМИ. ПРОГРАМНА РЕАЛІЗАЦІЯ

Для реалізації системи, що об’єднує описані раніше пристрої в єдину мережу, було спроектовано серверну архітектуру. Вона містить наступні процеси:

- Сервер Bluetooth Low Energy, що виконує підключення, отримання даних від SmartSwitch пристроїв мережі за протоколом Bluetooth Low Energy. В режимі реального часу передає отримані дані на веб-сервер та очікує від нього відповіді у вигляді результату виконання алгоритму оптимізації, на основі якого передає на відповідний SmartSwitch керуючий сигнал;
- Веб-сервер. Зберігає отримані від сервера Bluetooth LE дані, на їх основі за допомогою алгоритму оптимізації приймає рішення про підключення/відключення побутових приладів до мережі з метою обмеження навантаження електромережі або уникнення аварійних ситуацій. Надає API для ручного керування системою користувачем з графічного інтерфейсу браузера.

  На рис. 3.1 наведено початкову контекстну DFD діаграму системи.

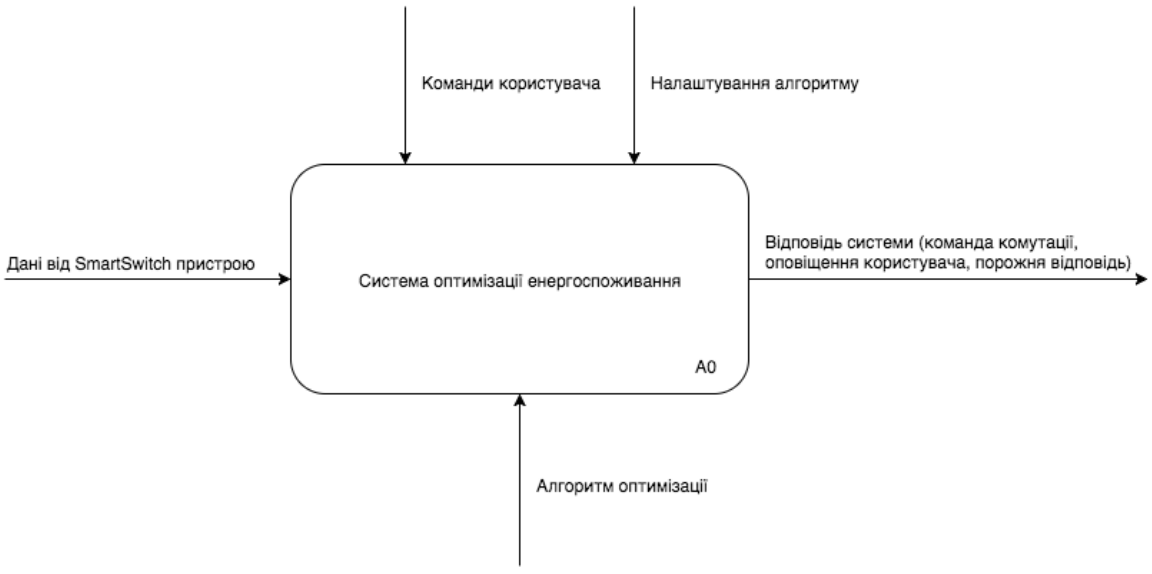

Рисунок 3.1 – Початкова контекстна DFD діаграма

На рис. 3.2 блок A0 деталізовано відповідно до архітектури системи, що складається з двох серверів, що взаємодіють за протоколом WebSocket, та користувацького інтерфейсу, призначеного для моніторингу та керування системою.

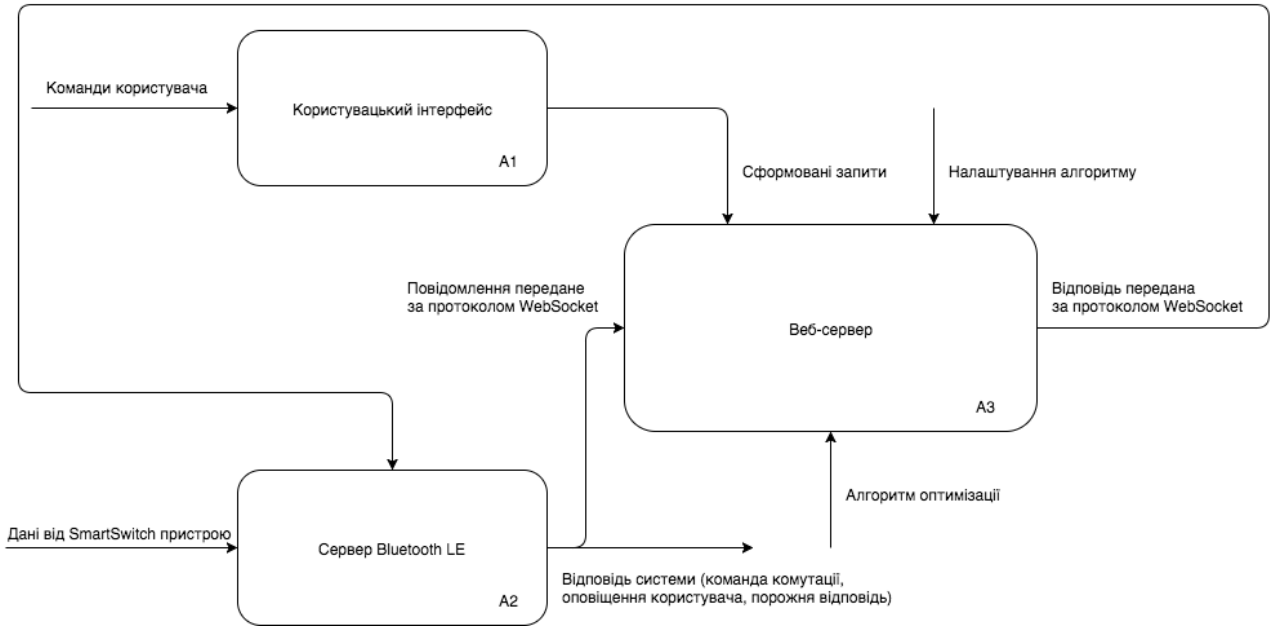

Рисунок 3.2 – Деталізована контекстна DFD діаграма

Для програмної реалізації системи, було обрано мову програмування Javascript та програмну платформу NodeJS. У якості бази даних було обрано реляційну базу даних PostgreSQL, для кешування даних та налаштувань системи – нереляційну базу даних типу “ключ-значення” Redis. Для обміну даними в режимі реального часу між веб-сервером та сервером Bluetooth LE було обрано протокол WebSocket. Програмний код системи наведено в додатку. В ньому були використані такі бібліотеки:

- **Noble** – написана на Javascript кросплатформена бібліотека для роботи з Bluetooth Low Energy на серверній стороні (MacOS, Linux, Windows);
- **ExpressJS** - програмний каркас розробки веб-додатків та API для Node.js. Мінімалістичний веб-сервер з можливістю підключення великої кількості плагінів;

- **SocketIO** – реалізація протоколу WebSocket на мові програмування Javascript для обміну даними між процесами в режимі реального часу. Крім WebSocket може використовувати JSONP або AJAX запити в залежності від можливостей клієнта та сервера;
- **pg-promise** – Javascript API для роботи з СУБД PostgreSQL;
- **redis** – Javascript API для роботи з нереляційною базою даних Redis;
- **VueJS** – мінімалістичний фронтенд фреймворк для побудови користувацьких інтерфейсів.

3.1 Сервер Bluetooth LE

3.1.1 Бібліотека Noble

За основу сервера взято бібліотеку Noble, що надає зручний серверний API для усіх необхідних типів взаємодії за протоколом Bluetooth Low Energy у event-driven парадигмі, притаманній мові Javascript:

1) Сканування активних пристроїв мережі:

```
noble.on('stateChange', function(state) {
    if (state === 'poweredOn') {
        noble.startScanning();
    } else {
        noble.stopScanning();
    }
});
```

2) Колбек (callback) для обробки події виявлення нового пристрою внаслідок сканування. В середині тіла даного колбеку відбувається уся робота з самим пристроєм (підключення, передача даних):

```
noble.on('discover', function(peripheral) {
    console.log('peripheral: ', peripheral.advertisement.localName);
});
```

3) Підключення до виявленого пристрою. Ідентифікаторами за якими розрізняють різні пристрої є: uuid – унікальний ідентифікатор та MAC-адреса Bluetooth адаптера:

```
peripheral.connect(function(error) {
    console.log('connected to peripheral: ', peripheral.uuid);
});
```

4) Відключення:

```
peripheral.disconnect(function(error) {
    console.log('disconnected from peripheral: ', peripheral.uuid);
});
```

5) Отримання усіх сервісів пристрою. Сервіси є своєрідними додатками Bluetooth LE. Кожен сервіс має своє власне призначення. Кількість сервісів одного пристрою є необмеженою:

```
peripheral.discoverServices(null, function(error, services) {
    console.log('discovered the following services:', services);
});
```

6) Отримання усіх характеристик певного сервісу. Характеристики є інтерфейсами взаємодії для інших пристроїв з даним сервісом. Поділяються за призначенням на read, write та indicate характеристики:

```
service.discoverCharacteristics(null, function(error, characteristics) {
    console.log('discovered characteristics:', characteristics);
});
```

7) Зчитування даних характеристики пристрою (для характеристик типу read):

```
characteristic.read(function(error, data) {
    console.log(data.toString('utf8'));
});
```

8) Підписка на нові дані від характеристики (для характеристик типу indicate):

```
characteristic.on('data', function(data, isNotification) {
    console.log(data.toString('utf8'));
});
```

9) Надсилання даних до характеристики пристрою (для характеристик типу write):

```
characteristic.write(new Buffer([0x01]), true, function(error) {
    console.log('sent');
});
```

Слід зазначити, дана бібліотека накладає певні обмеження на кількість одночасних підключень до сервера в залежності від швидкодії адаптера Bluetooth Low Energy, що може бути як вбудованим так і зовнішнім. Зазвичай це близько 10 одночасно під'єднаних пристроїв. Проте бібліотека надає чудові можливості для почергового підключення пристроїв тільки на момент отримання чи передачі даних, що фактично прибирає дане обмеження [22].

3.1.2 Реалізація

Слід зазначити, що в процесі розробки Smart Switch пристроїв для них створювались сервіси та характеристики з однаковими ідентифікаторами та властивостями а саме:

SERVICE_UUID = '00035b0358e607dd021a08123a000300'

CHARACTERISTIC_UUID = '00035b0358e607dd021a08123a000301'

Це дозволяє отримати універсальний інтерфейс для роботи з усіма пристроями системи. Характеристика має властивості indicate (для отримання даних пристрою від пристрою) та write (для передачі команд до пристрою), що покриває увесь необхідний для системи функціонал.

Далі наведено послідовність сценаріїв роботи Bluetooth LE сервера:

- створюється логічне з'єднання з веб-сервером за допомогою SocketIO бібліотеки. З бази даних Redis дістаються закешовані веб-сервером ідентифікатори пристроїв системи для подальшої роботи з ними у JSON форматі:

smart_dispatcher:<device.uuid>: {

id: device.id

}

, де **uuid** – унікальний ідентифікатор адаптера пристрою, **id** – первинний ключ пристрою з таблиці реляційної БД;

- перевіряється активність Bluetooth LE адаптера комп'ютера, у разі успіху починається сканування пристроїв мережі;
- у разі відповідності одному з отриманих у попередньому пункті uuid ідентифікаторам відбувається встановлюється з'єднання з даним пристроєм;
- з отриманих при з'єднанні даних пристрою виокремлюється сервіс та його характеристика з описаними вище ідентифікаторами;
- для активації пристрою за допомогою методу write на нього передається сигнал у вигляді шістнадцяткового коду 0x01;
- сервер підписується на отримання даних від даної характеристики у вигляді строки, що містить значення з АЦП про напругу, струм та температуру розділені між собою комою;
- відбувається розбір строки та формування з отриманих значень повідомлення у форматі JSON наступного вигляду:

{

device_id: 1,

current: 67,

voltage: 110,

temperature: 20

}

, де **device_id** є первинним ключем відповідного пристрою з БД, **current**, **voltage** та **temperature** – значення з відповідних АЦП мікроконтролера;

- сформоване повідомлення через логічне з'єднання передається на веб сервер де його вміст нормалізується та на основі нього створюється новий

запис в БД в таблиці Logs. Далі відбувається виклик алгоритму оптимізації з оновленими даними та перевірка на наявність невиконаних ручних команд користувача. В результаті цього формується відповідь, що відправляється до сервера Bluetooth LE у наступному форматі:

```
{
   switch: true/false,
   uuid: device.uuid
}
```

, де **switch** – рішення про комутацію пристрою, яка відбувається при вказаному значенні **true**. При значенні **false** не відбувається нічого;

- Bluetooth LE сервер на основі отриманого повідомлення або, надсилає на пристрій команду у разі необхідності комутації, або очікує на наступні дані.

3.2 Веб-сервер

3.2.1 ExpressJS

За основу веб-сервера було взято Javascript фреймворк ExpressJS, що де-факто є стандартним для більшості сучасних NodeJS додатків. Написаний він був у дусі фреймворка для Ruby Sinatra – основною метою є простота та швидкість обробки запитів.

В якості прикладу далі наведено код тестового додатку:

```
var express = require('express')
var app = express()
app.get('/', function (req, res) {
      res.send('Hello World')
})
app.listen(3000)
```

Даний код створює екземпляр додатка та запускає сервер на 3000 порту. У разі переходу у браузері за url-адресою http://localhost:3000 буде отримано повідомлення ‘Hello world’ [23].

ExpressJS має методи для створення будь-яких типів роутів:

- **.get()** – GET запит (отримання даних, перехід на сторінку тощо);
- **.patch()** – PATCH запит (оновлення певної сутності, відправка edit-форми);
- **.put()** –PUT запит (оновлення певної сутності, відправка edit-форми);
- **.post()** – POST запит (створення сутності, відправка create-форми);
- .**delete()** – DELETE запит (видалення сутності);
- **.all()** – універсальний роут для перехоплення будь-якого типу запиту.

Крім цього ExpressJS надає ряд зручних методів для реалізації відповідей у ендпойнтах дадатка, наприклад:

- **res.download()** – передача файла для завантаження;
- **res.end()** – завершення процесу відповіді;
- **res.json()** – відповідь у форматі JSON;
- **res.jsonp()** – відповідь у форматі JSON з підтримкою JSONP;
- **res.redirect()** – перенаправлення;
- **res.render()** – рендеринг шаблону;
- **res.send()** – метод для відповіді у довільному форматі;
- **res.sendStatus()** – встановлення статусу відповіді.

3.2.2 Реалізація

У даному розділі наведено опис архітектури розробленого веб-сервера а також алгоритму оптимізації енергоспоживання побутових приладів системи.

Функції веб-сервера умовно можна розподілити на 2 групи:

- Обробка запитів з користувацького інтерфейсу;
- Обробка даних що надходять за протоколом WebSocket від сервера Bluetooth LE.

База даних системи складається з 2-ох таблиці – пристроїв системи (Devices) та записів про стан пристрою (Logs). Записи створюються по мірі отримання даних від сервера Bluetooth LE. На їх основі формується висновок про поточний стан кожного побутового приладу та усієї системи вцілому. Entity Relations діаграму бази даних наведено на рис. 3.3.

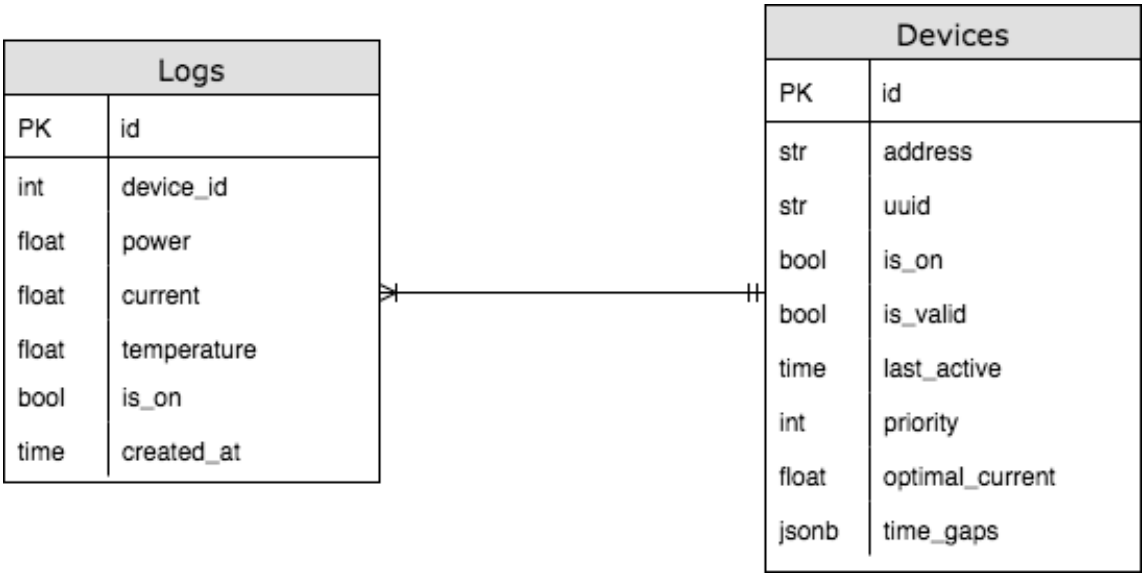

Рисунок 3.3 – ER діаграма бази даних

Далі наведено деталі реалізації алгоритму оптимізації системи.

Налаштування системи (Settings) являють собою пари “ключ-значення”, характеризують певні важливі для системи показники. Зберігаються в Redis для прискорення швидкості доступу. Список використовуваних налаштувань системи:

- **AUTO_MODE = <bool>**. Задає режим роботи системи. При встановленому значенні **false** керування системою відбувається тільки в ручному режимі з користувацького інтерфейсу, тобто, усі наведені далі перевірки не виконуються. При встановленному значенні **true** система працює в автоматичному режимі, тобто рішення про необхідність комутації пристроїв приймає алгоритм;

- **POWER = <float>**. Налаштування алгоритму. Задає максимальне допустиме загальне навантаження системи у ватах. При перевищенні даного значення відбувається відключення приладу від електромережі. Надалі з певною періодичністю здійснюються спроби підключення приладу;
- **TEMPERATURE = <float>**. Налаштування алгоритму. Задає максимальне допустиме значення температури в градусах за Цельсієм. При його перевищенні відбувається відключення відповідного приладу від електромережі. Також відповідний прилад помічається як **is_valid = false**, що забороняє підключати його до прийняття рішення користувачем, чи може він продовжувати роботу.

На рис. 3.4 показано користувацький інтерфейс, призначений для додавання, редагування та видалення налаштувань.

Рисунок 3.4 – Користувацький інтерфейс для управління налаштуваннями

Крім описаних налаштувань алгоритм також конфігурується за допомогою полів самого пристрою, а саме:

- **priority = <int>**. Поле задає пріоритет приладу по відношенню до інших. Цей показник впливає на послідовність в якій прилади будуть підключатись або відключатись від електромережі;
- **optimal_current = <float>**. Значення оптимального споживання струму приладом в амперах. В алгоритмі використовується наступним чином: у разі перевищення споживання струму більш ніж у 2 рази відбувається відключення приладу з відміченням його як **is_valid = false**;
- **time_gaps = [[<time>, <time>], …]**. Масив з часових проміжків протягом яких даний прилад має працювати. У разі потрапляння поточного часу в один з проміжків відбувається підключення приладу і навпаки – у разі відсутності збігів відключається. Дозволяє розподілити споживання електроприладів найбільш оптимальним для користувача чином в залежності від дії багатозоних тарифів на електроенергію, або ж добового піку генерації відновлюваних джерел живлення тощо.

Для керування пристроями в ручному режимі з користувацького інтерфейсу передбачено окрему кінцеву точку веб-сервера PATCH /devices/:id. При виконанні запиту на неї у Redis створюється запис наступного вигляду:

```
smart_dispatcher:<device.uuid>: {
    id: device.id,
    switch_pending: true
}
```

При наступному циклі обробки даних, що надходять від Bluetooth LE сервера за протоколом WebSocket, перевіряється наявність запису в Redis з таким ідентифікатором. У випадку його наявності на Bluetooth LE сервер відправляється команда про комутацію даного пристрою без подальшого виконання алгоритму. Сам запис у базі даних Redis видаляється,

У додатку є ряд кінцевих точок (endpoints), які забезпечують усі необхідні функції щодо моніторингу та керування пристроями, управління налаштуваннями системи а також відображення усіх необхідних сторінок

користувацького інтерфейсу. Список усіх наявних кінцевих точок розробленого API з коротким описом їх призначення наведено в табл. 3.1.

Таблиця 3.1 – Кінцеві точки API веб-сервера

Тип запиту	Відносна адреса	Опис
GET	/devices	повертає масив пристроїв системи
POST	/devices	створює новий пристрій
GET	/devices/:id	повертає пристрій з первинним ключем id
PATCH	/devices/:id	команда комутації пристрою
DELETE	/devices/:id	видалення пристрою
GET	/settings	повертає список налаштувань сервера
POST	/settings	створює нове налаштування
DELETE	/settings/:id	видалення налаштування

Для реалізації користувацького інтерфейсу веб-сервера було обрано фронтенд фреймворк VueJS, що надає можливість зручної шаблонізації всередині HTML розмітки на кшталт Angular, двустороннього зв’язку даних, їх оновлення [24].

Весь обмін даними з веб-сервером виконується за допомогою AJAX запитів у форматі JSON. Для цього використано зручний AJAX API бібліотеки jQuery. Завдяки цьому оновлення сторінки для отримання актуальних даних від веб-сервера є непотрібним – вони оновлюються автоматично з інтервалом в 1

секунду, що дозволяє здійснювати моніторинг системи в режимі реального часу.

На рис. 3.5 наведено вигляд домашньої сторінки веб-сервера, де розміщено інформацію про пристрої системи.

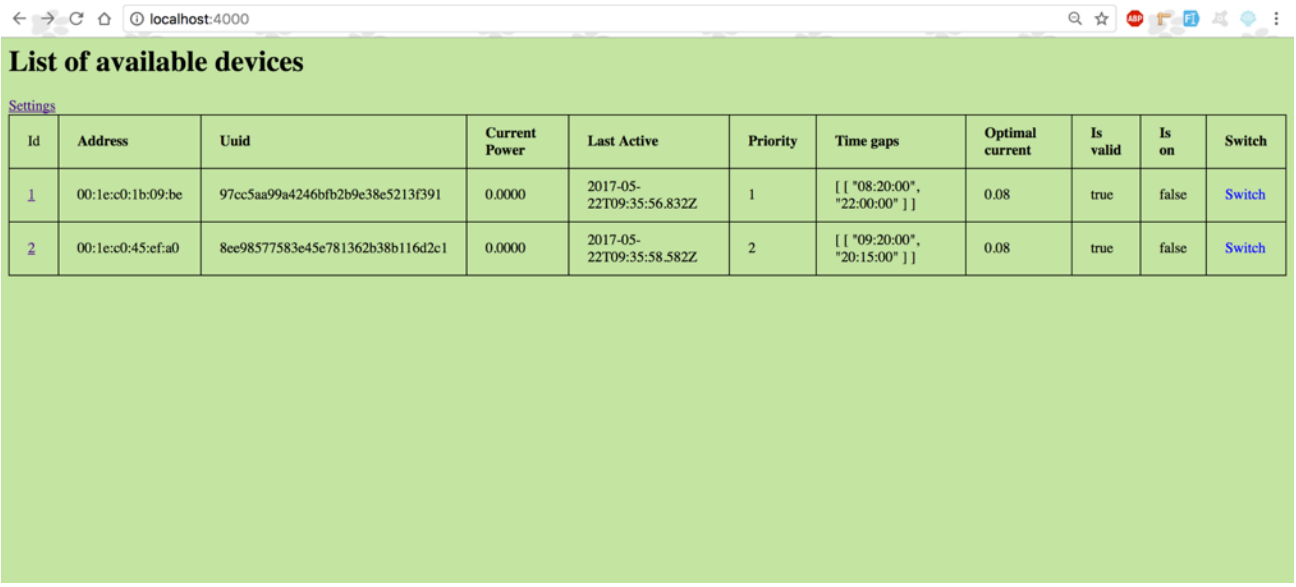

Рисунок 3.5 – Домашня сторінка додатка

На рис. 3.6 наведено вигляд деталізованої сторінки приладу, де користувач може побачити його параметри, історію споживання в графічному вигляді та журнал записів. Крім цього можна здійснити комутацію приладу, натиснувши кнопку **“Switch”**.

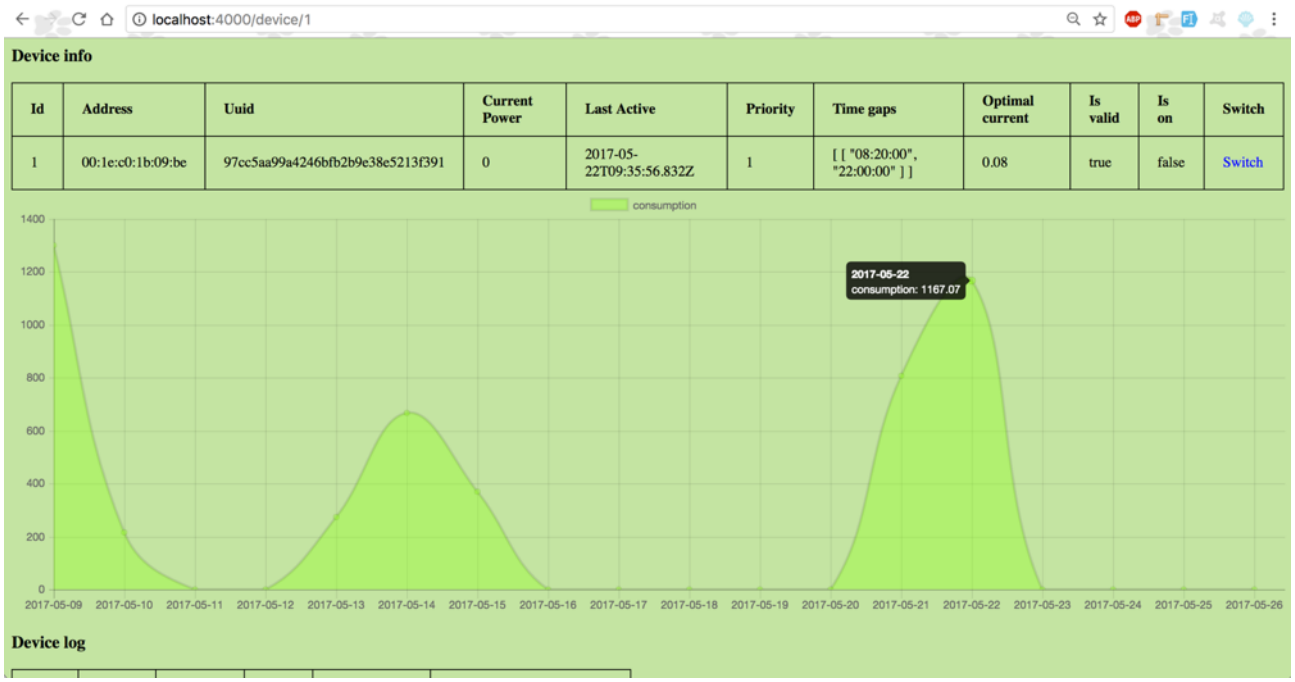

Рисунок 3.6 – Сторінка приладу

3.3 Приклад роботи системи

Для ілюстрації роботи системи розглянемо основні сценарії, що описують деякі особливості розробленого алгоритму оптимізації, деталі ручного керування та управління налаштуваннями:

1. Підключимо живлення до Smart Switch пристроїв;
2. за допомогою команди **npm start** запустимо обидва сервери. Логи підключення пристроїв до сервера Bluetooth LE та обмін повідомленнями з веб-сервером наведено на рис. 3.7;

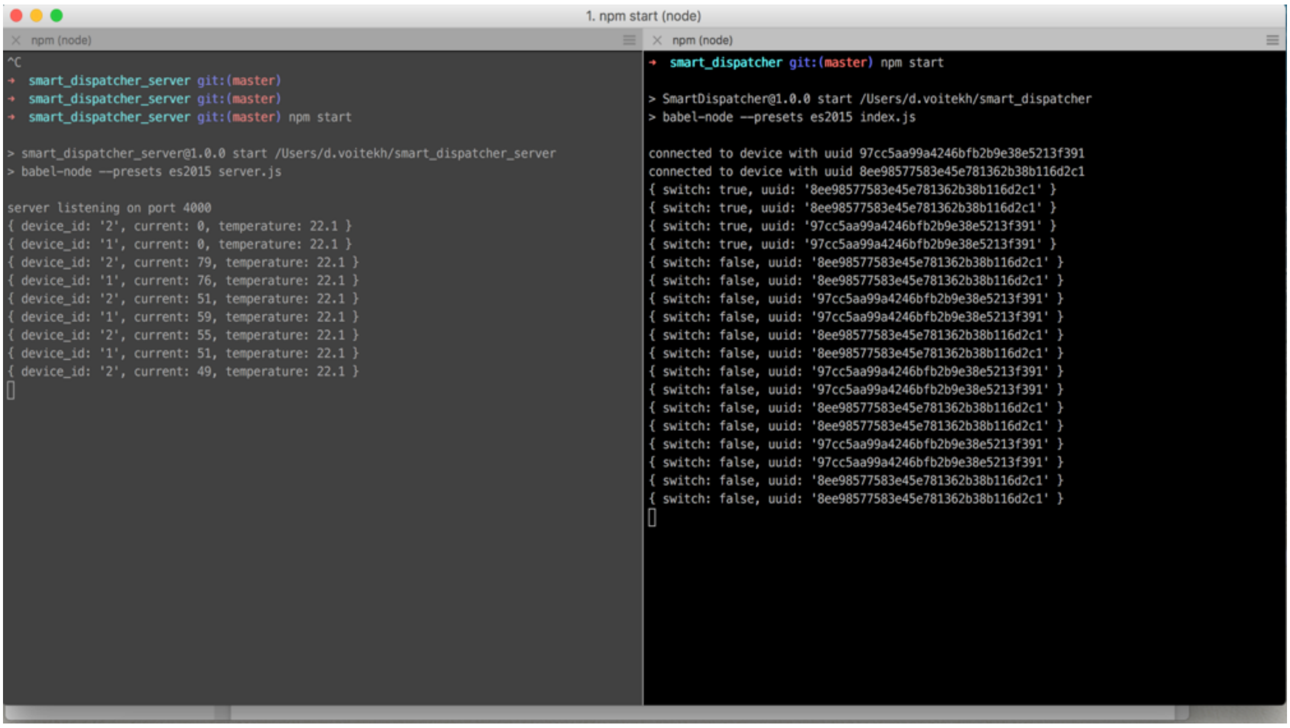

Рисунок 3.7 – Процес запуску системи

3. далі перейдемо в браузері за посиланням http://localhost:3000/device/1 щоб переглянути поточний стан пристрою. На рис 3.8 показана дана сторінка. Бачимо, що дані про про поточний стан та енергоспоживання пристрою постійно логуються. На даний момент двигун споживає 2.1120 вати потужності і стан його роботи оцінюється як задовільний. Бачимо зміну агрегованих даних з таблиці Logs на графіку про добове споживання приладу;

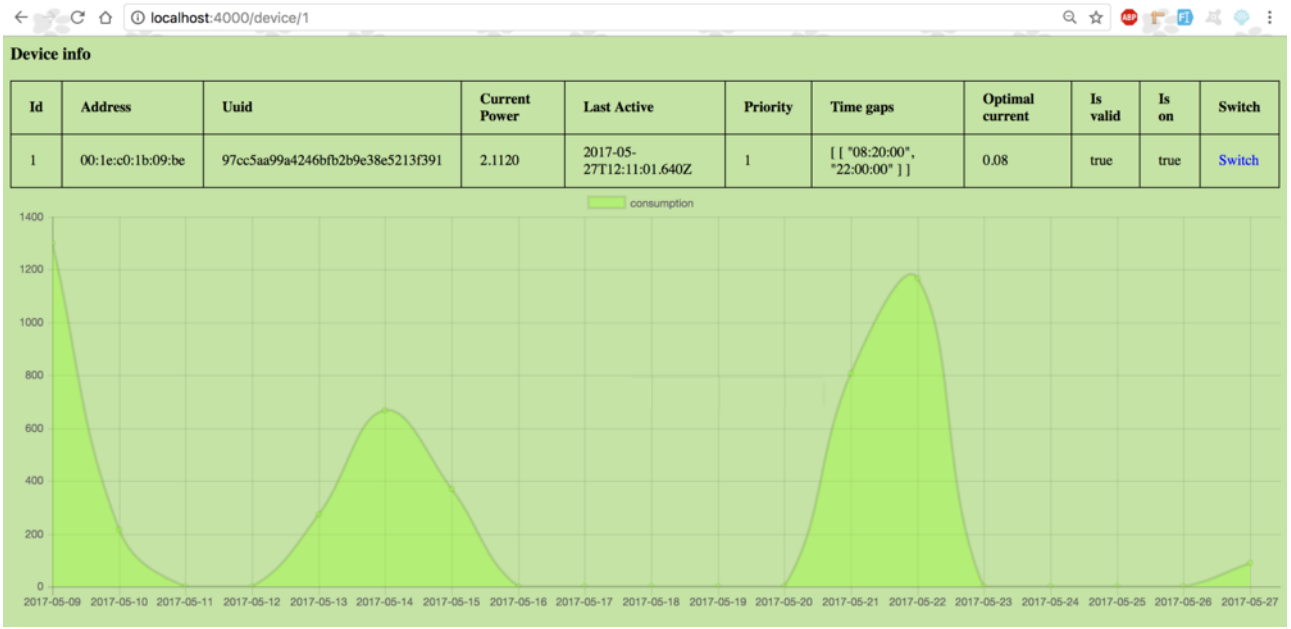

Рисунок 3.8 – Дані про споживання приладу

4. тепер здійснимо комутацію навантаження приладу, натиснувши кнопку **“Switch”**. Як наслідок, двигун відключився. Інформація на екрані оновилася – пристрій став **is_on = false**, значення **current_power** змінилося на 0;
5. перейдемо в автоматичний режим роботи, на сторінці http://localhost:3000/settings_page задавши значення налаштування **AUTO_MODE = true**;
6. бачимо, що відключений раніше двигун включився, оскільки алгоритмом було знайдено збіг за часовим проміжком [’08:20:00’, ’22:00:00’]. Так само якщо змінити даний проміжок так, аби поточний час не потрапляв у нього двигун буде вимкнено;
7. перевіримо частину алгоритму щодо пошуку несправності приладу. У даного приладу вказано **optimal_current** як 0.08A. Рукою призупинимо двигун для того щоб споживана потужність зросла. При перевищені споживаного струму позначки 0.16A двигун відключився, поле відповідного пристрою **is_valid = false**. Отже для подальшої роботи пристрою користувачу необхідно змінити це значення;

8. перевіримо частину алгоритму щодо балансування навантаження. На сторінці налаштувань зазначимо максимальну потужність системи як **POWER = 5**, що відповідає 5 ватам. Включимо обидва двигуни приладів. Загальна споживана потужність на даний момент складає 4.3 вати. Призупинимо рукою двигун першого пристрою для збільшення споживаного струму до 0.12А. Відбулося перевищення загальної потужності позначки 5 ват. Даний двигун відключився. В наступний цикл отримання даних від цього пристрою двигун знову буде включено.

Отже у макеті даної системи задоволені усі визначені раніше вимоги. Зображення макетного зразка системи наведено на рис. 3.9.

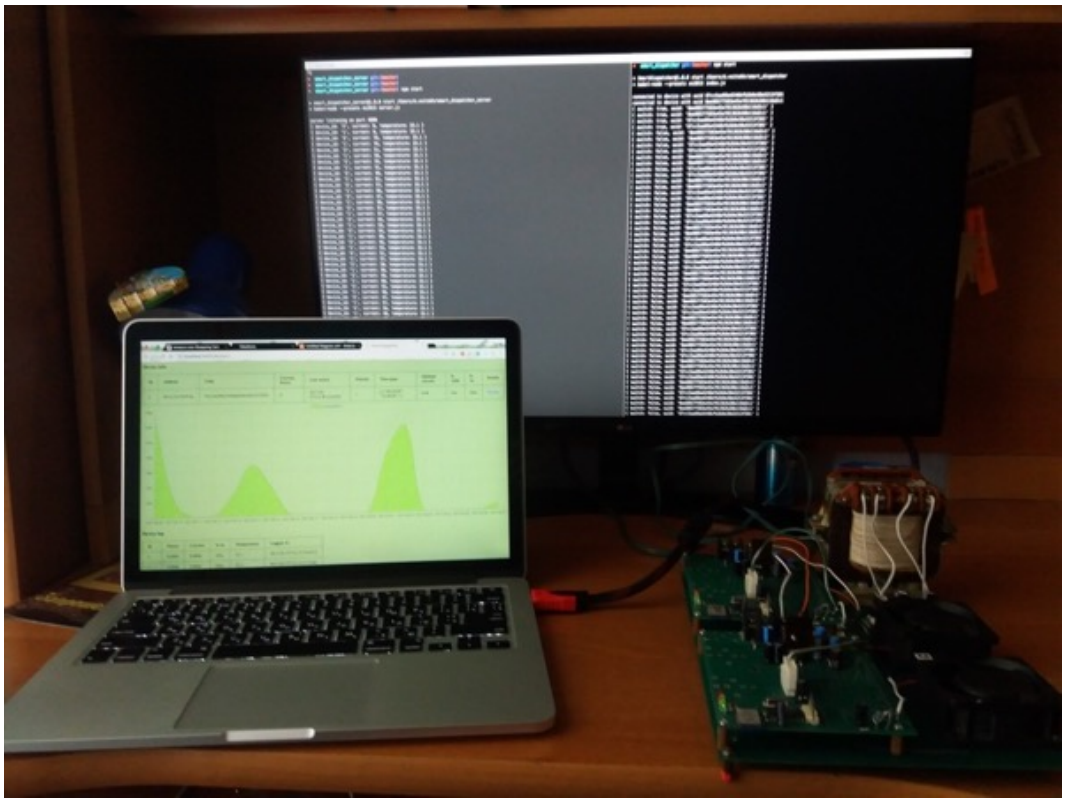

Рисунок 3.9 – Макет системи

4. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОДУКТУ

У даному розділі проводиться оцінка основних характеристик системи оптимізації енергоспоживання для розумного будинку на основі протоколу зв'язку Bluetooth LE. Система складається з розумних розеток - пристроїв для моніторингу і контролю роботи побутових приладів, та серверного додатку. Додаток був розроблений мові програмування Javascript та працює на платформі Node.js. Розумні розетки створені на базі сімейства мікросхем компанії Microchip.

Розроблений програмний продукт є кросплатформеним, а отже може працювати на усіх комп'ютерах під управлінням Windows, Linux та MacOS, що мають вбудований або зовнішній адаптер Bluetooth LE.

Нижче наведено аналіз різних варіантів реалізації модулю з метою вибору оптимальної, з огляду при цьому як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. Як прямі, так і побічні витрати розподіляються по продуктам та послугам у залежності від потрібних на кожному етапі виробництва обсягів ресурсів. Виконані на цих етапах дії у контексті метода ФВА називаються функціями.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Фактично цей метод працює за таким алгоритмом:

- визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На цьому ж етапі оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат.
- для кожної функції визначаються повні річні витрати й кількість робочих часів.
- для кожної функції на основі оцінок попереднього пункту визначається кількісна характеристика джерел витрат.
- після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

4.1 Постановка задачі

У роботі застосовується метод ФВА для проведення техніко-економічного обґрунтування розробки системи оптимізації енергоспоживання для розумного будинку на основі протоколу зв’язку Bluetooth LE. Основні проектні рішення визначаються відповідно до вимог до всієї системи, тому її підсистеми також мають їм задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для моніторингу та контролю роботи побутових приладів, оснащених розробленими розумними розетками.

Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

- програмний продукт повинен функціонувати на звичайних персональних комп’ютерах та серверах з адаптерами Bluetooth LE;
- забезпечувати високу швидкість обробки даних від багатьох пристроїв у режимі реального часу;

- забезпечувати зручність і простоту взаємодії з користувачем або з розробником програмного забезпечення у випадку використовування його як модуля;
- передбачати мінімальні витрати на впровадження програмного продукту.

4.1.1 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка автоматизованої системи, що оптимізує роботу побутових приладів, на основі даних про їх енергоспоживання. Відповідно до цієї мети, можна виділити такі основні функції ПП:

F_1 – вибір апаратної платформи для розумних розеток;

F_2 – вибір мови програмування для реалізації серверного додатка;

F_3 – вибір СУБД;

Кожна з основних функцій може мати декілька варіантів реалізації:

Функція F_1:

а) розробка власного пристрою на основі сімейства мікросхем Microchip;

б) використання готової платформи Arduino;

Функція F_2:

а) мова програмування Javascript;

б) мова програмування Java;

Функція F_3:

а) СУБД PostgreSQL;

б) СУБД MySQL.

4.1.2 Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1). На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (таблиця 4.1).

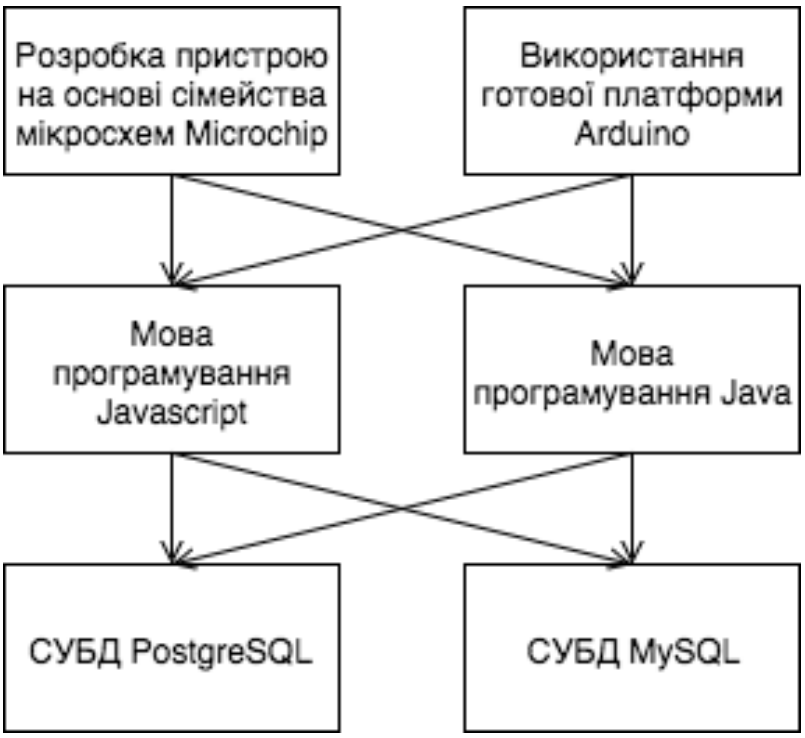

Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

Таблиця 4.1 – Позитивно-негативна матриця

Основні функції	Варіанти реалізації	Переваги	Недоліки
F_1	А	На основі макетного зразка можна створити реальний промисловий пристрій, повна свобода у розробці нового функціоналу	Складніша та довша розробка, що включає в себе як апаратну так і програмну частину
	Б	Швидка розробка, необхідною є лише програмна частина, наявність готових бібліотек	Платформа накладає певні апаратні обмеження, нижча швидкодія
F_2	А	Найбільша кількість бібліотек та фреймворків. Вища швидкість розробки коду	Нижча швидкодія та читабельність коду через асинхронний стиль мови
	Б	Вища швидкодія, менше помилок завдяки статичній типізації	Нижча швидкість розробки коду, дещо менша кількість бібліотек та API
F_3	А	Підтримка найбільш сучасних стандартів SQL, краща продуктивність та масштабованість, детальніша документація	Складніше адміністрування, нижча продуктивність при повільних каналах передачі даних
	Б	Набагато зручніше адміністрування, більш розповсюджена на ринку	Повільне впровадження новітніх стандартів, гірша масштабованість та продуктивність

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути тому, що вони не відповідають поставленим перед програмним продуктом задачам.

Функція F1:

Оскільки основною вимогою до системи є універсальність та простота інтерфейсу, можливість конфігурувати пристрої та змінювати їх функціонал, варіант А є набагато більш привабливим. Тому Б було відкинуто.

Функція F2:

Обидва варіанти є привабливими, перший - з точки зору швидкості розробки коду та різноманітних сторонніх бібліотек, другий – з точки зору швидкодії та надійності коду. Тому жоден варіант не можна однозначно відкинути.

Функція F3:

Варіант А однозначно переважає Б за рахунок кращої масштабованості, автоматичної оптимізації запитів, можливості їх паралельної обробки та детальнішої документації. Тому варіант Б відкидається.

Таким чином, будемо розглядати такі варіанти реалізації ПП:

- F1A – F2A – F3A
- F1A – F2Б – F3A

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.2 Обґрунтування системи параметрів ПП

4.2.1 Опис параметрів

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня:

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

X_1 – швидкодія мови програмування;

X_2 – об’єм пам’яті для збереження даних;

X_3 – час обробки даних алгоритмом;

X_4 – потенційний об’єм програмного коду;

X_5 – потенційний час розробки програмного коду.

X_1: Відображає швидкодію виконання атомарних операцій в залежності від обраної мови програмування;

X_2: Відображає швидкодію операцій залежно від обраної мови програмування.

X_3: Відображає час, необхідний для отримання даних від пристрою, їх обробку та формування алгоритмом на їх основі команди-відповіді;

X_4: Відображає розмір програмного коду проекту у рядках, що необхідно написати розробнику;

X_5: Відображає час в людино-днях, необхідний на розробку програмного коду.

4.2.2 Кількісна оцінка параметрів

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у табл. 4.2.

Таблиця 4.2 – Основні параметри ПП

Назва параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X_1	оп/мс	19000	11000	2000
Об’єм пам’яті для збереження даних	X_2	мб	32	16	8
Час обробки даних алгоритмом	X_3	мс	800	400	60
Потенційний об’єм програмного коду	X_4	кількість рядків коду	2000	1500	1000
Потенційний час розробки програмного коду	X_5	людино-дні	200	50	20

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.6.

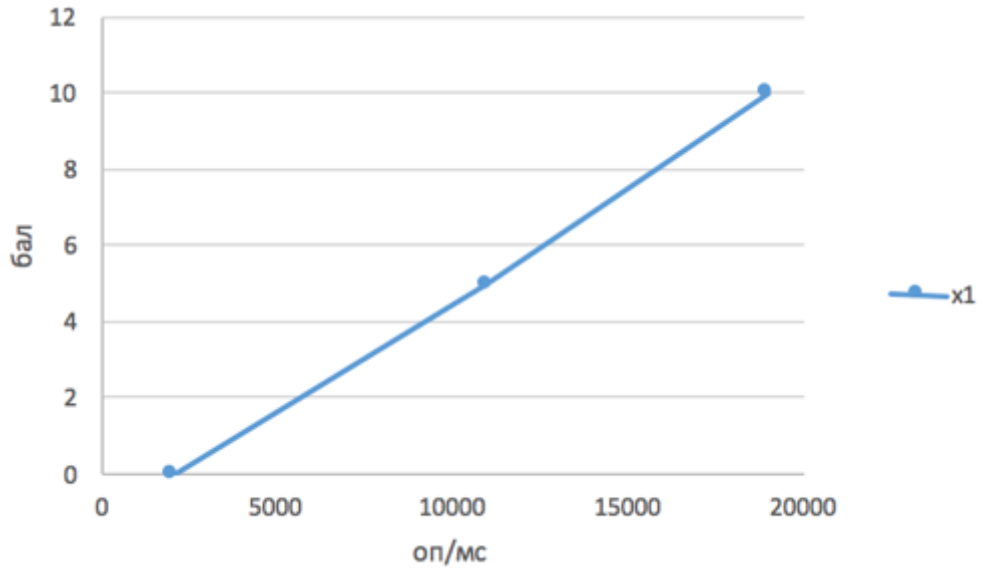

Рисунок 4.2 – X1, швидкодія мови програмування

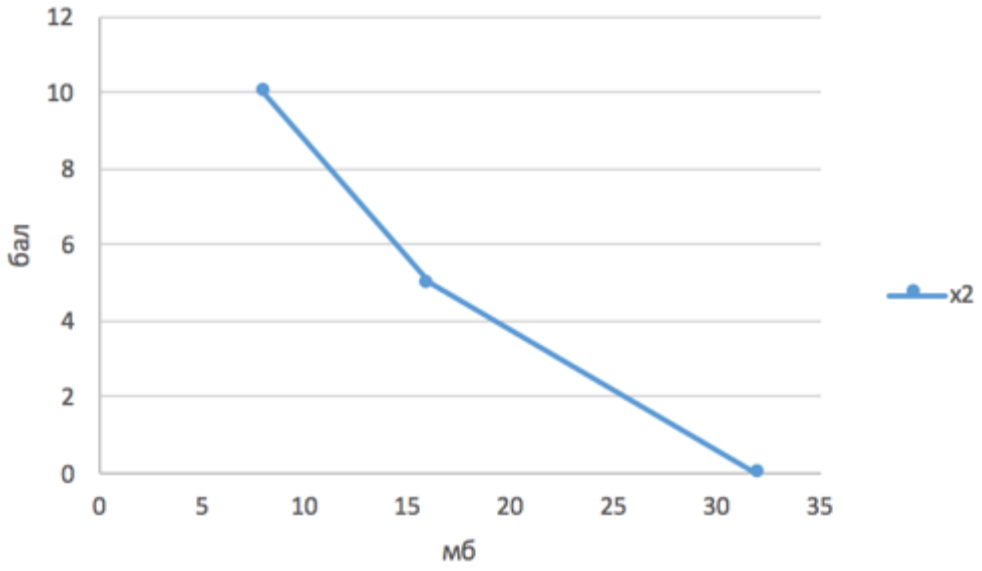

Рисунок 4.3 – X2, об’єм пам’яті для збереження даних

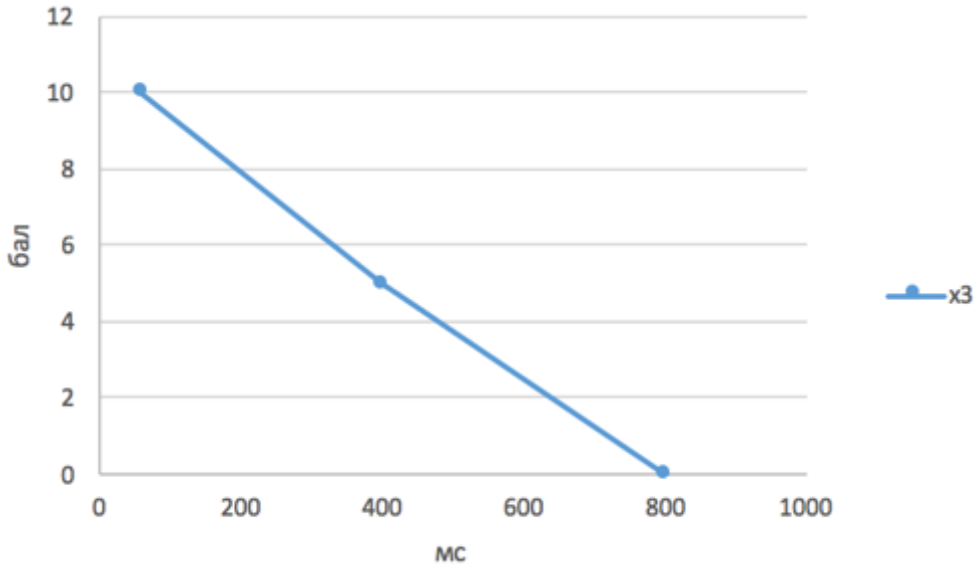

Рисунок 4.4 – X3, час обробки даних алгоритмом

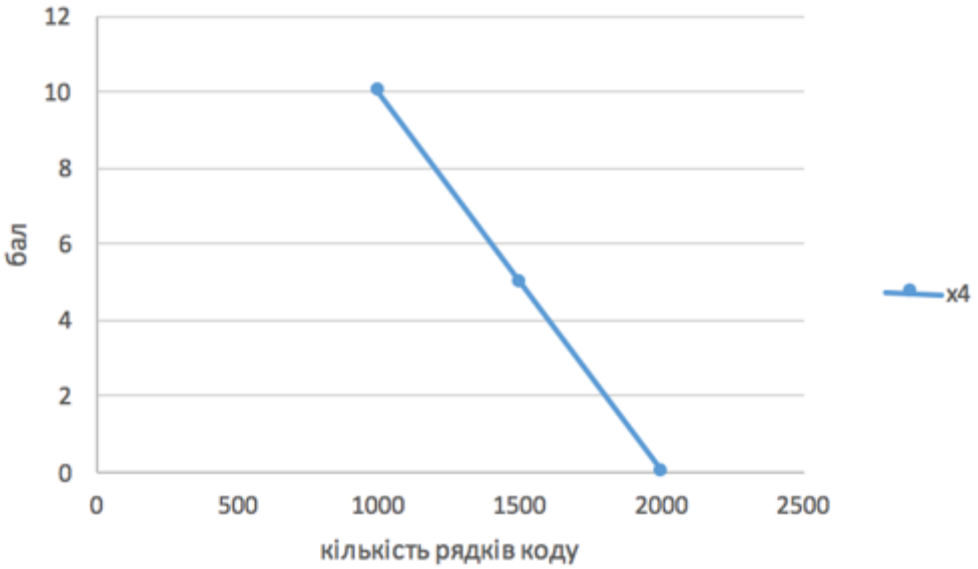

Рисунок 4.5 – X4, потенційний об'єм програмного коду

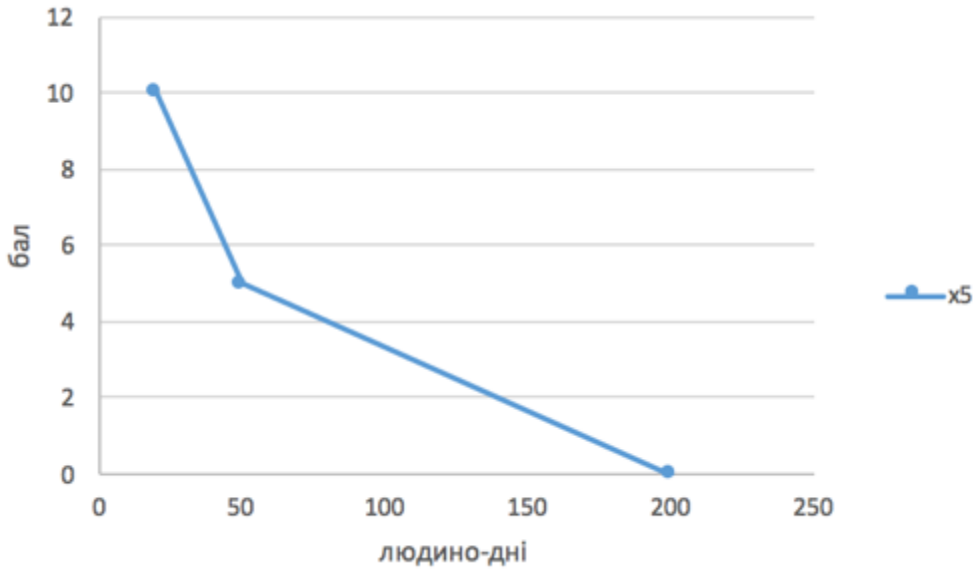

Рисунок 4.6 – X5, потенційний час розробки програмного коду

4.2.3 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка системи оптимізації енергоспоживання для розумного будинку.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

1) визначення рівня значимості параметра шляхом присвоєння різних рангів;
2) перевірку придатності експертних оцінок для подальшого використання;
3) визначення оцінки попарного пріоритету параметрів;
4) обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці вимірру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	оп/мс	4	3	4	4	3	4	3	25	-2,6	6,76
X2	Об’єм пам’яті для збереження даних	мб	3	4	3	3	4	3	4	24	-3,6	12,96
X3	Час обробки даних алгоритмом	мс	2	2	1	2	1	2	2	12	-15,6	243,36
X4	Потенційний об’єм програмного коду	к-сть рядків коду	6	5	5	6	5	5	5	37	9,4	88,36
X5	Потенційний час розробки програмного коду	людн	5	6	6	5	6	6	6	40	12,4	153,76
	Разом		21	21	21	21	21	21	21	138	0	505,2

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

1) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 138$$

де N – число експертів, n – кількість параметрів;

2) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 27.6$$

3) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T$$

Сума відхилень за всіма параметрами має дорівнювати 0;

4) загальна сума квадратів відхилення

$$S = \sum_{i=1}^{n} \Delta^2{}_i = 505.2$$

Обчислимо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)},$$

$$W = \frac{12 * 505.2}{7^2(5^3 - 5)} = 1.03 > W_k = 0.67$$

Ранжирування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю.

Таблиця 4.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	<	>	>	<	>	<	>	1.5
X1 і X3	>	>	>	>	>	>	>	>	1.5
X1 і X4	<	<	<	<	<	<	<	<	0.5
X1 і X5	<	<	<	<	<	<	<	<	0.5
X2 і X3	>	>	>	>	>	>	>	>	1.5
X2 і X4	<	<	<	<	<	<	<	<	0.5
X2 і X5	<	<	<	<	<	<	<	<	0.5
X3 і X4	<	<	<	<	<	<	<	<	0.5
X3 і X5	<	<	<	<	<	<	<	<	0.5
X4 і X5	>	<	<	>	<	<	<	<	0.5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначають за формулою:

$$a_{ij} = \begin{cases} 1{,}5 \text{ за } X_i > X_j \\ 1.0 \text{ за } X_i = X_j \\ 0.5 \text{ за } X_i < X_j \end{cases}$$

З отриманих числових оцінок переваги складемо матрицю A=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за такою формулою:

$$\hat{E}_{â^3} = \frac{b_i}{\sum_{i=1}^{n} b_i}, äå\ b_i = \sum_{i=1}^{n} a_{ij}.$$

Відносні оцінки розраховують декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2 %). На другому і наступних кроках відносні оцінки розраховують за такою формулою:

$$\hat{E}_{â^3} = \frac{b'_i}{\sum_{i=1}^{n} b'_i}, äå \quad b'_i = \sum_{j=1}^{n} a_{ij} b_j$$

Таблиця 4.5 - Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j					Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	X5	b_i	$К_{вi}$	b_i^1	$\hat{E}_{â^3}^1$	b_i^2	$\hat{E}_{â^3}^2$
X1	1,0	0,5	0,5	1,5	1,5	5	0,2	22	0,191	100	0,190
X2	1,5	1,0	0,5	1,5	1,5	6	0,24	27,5	0,239	124,75	0,238
X3	1,5	1,5	1,0	1,5	1,5	7	0,28	34	0,296	155,5	0,296
X4	0,5	0,5	0,5	1,0	1,5	4	0,16	17,5	0,152	80,25	0,153
X5	0,5	0,5	0,5	0,5	1,0	3	0,12	14	0,122	64,5	0,123
Всього:	25					1	115	1	525	1	

Як видно з таблиці, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

4.3 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів X1 (швидкодія мови програмування) та X2 (об’єм пам’яті для збереження даних) відповідають технічним вимогам умов функціонування даного ПП.

Варіант а) більш простий для реалізації: при його використанні потрібно 1000 рядків коду (X4) та 25 людино-днів (X5), у той час як варіант б) вимагає 1600 рядків та 50 людино-днів. Але за умови вибору варіанту а) система повільніше оброблює запити: час обробки даних (X3) становитиме 400 мс, тоді як варіант б) забезпечує обробку даних за 200 мс.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{в\,i,j} B_{i,j},$$

де n – кількість параметрів; $K_{в\,i}$– коефіцієнт вагомості i–го параметра; B_i – оцінка i–го параметра в балах.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1(X1)	А	11000	5	0.190	0.95
F2(X3)	А	400	5	0.296	1.48
	Б	200	8		2.36
F2(X4)	А	1000	10	0.153	1.53
	Б	1600	4		0.61
F2(X5)	А	25	9.2	0.123	1.13
	Б	50	5		0.61
F3(X2)	А	16	5	0.238	1.19

За даними з таблиці 4.6 за формулою

$$K_K = K_{T\,У}[F_{1k}] + K_{T\,У}[F_{2k}] + \ldots + K_{T\,У}[F_{zk}],$$

визначаємо рівень якості кожного з варіантів:

K_{K1} = 0.95+ 1.480 + 1.53 + 1.13 +1.19 = 6.28

K_{K2} = 0.95+ 2.36 + 0.61 + 0.61 +1.19 = 5.72

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.4 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної реалізації системи;

При реалізації варіанту а) з’являється додаткова задача до завдання 2:

3. Оптимізація використання пам’яті програмою.

Варіант б) вимагає виконання такої підзадачі у контексті виконання задачі 2:

4. Створення зручного базового API для роботи з Bluetooth LE.

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 2.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Загальна трудомісткість обчислюється як

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М}, \quad (4.1)$$

де T_P – трудомісткість розробки ПП; $K_П$ – поправочний коефіцієнт; $K_{СК}$ – коефіцієнт на складність вхідної інформації; K_M – коефіцієнт рівня мови програмування; $K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм; $K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_P = 90$ людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_П = 1.7$. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх чотирьох завдань рівний 1: $K_{СК} = 1$. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{СТ} = 0.8$. Тоді, за формулою 4.1, загальна трудомісткість програмування першого завдання дорівнює:

$T_1 = 90 \cdot 1.7 \cdot 0.8 = 122.4$ людино-днів.

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм 2-ї групи складності, степінь новизни Б), тобто $Т_Р = 27$ людино-днів, $К_П = 0.9$, $К_{СК} = 1$,$К_{СТ} = 0.8$:

$Т_2 = 27 \cdot 0.9 \cdot 0.8 = 19.44$ людино-днів.

Для третього завдання (використовується алгоритм третьої групи складності, ступінь новизни Г), тобто ТР = 8 людино-днів, КП = 0,3, КСК = 1, КСТ = 0,8:

$Т_3 = 8 \cdot 0.3 \cdot 0.8 = 1.92$ людино-днів.

Для четвертого завдання (використовується алгоритм другої групи складності, степінь новизни В), тобто ТР = 19 людино-днів, КП = 0,6, КСК = 1, КСТ = 0,8:

$Т_4 = 19 \cdot 0.6 \cdot 0.8 = 9.12$ людино-днів.

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$Т_I = (122.4 + 19.44 + 1.92) \cdot 8 = 1150.08$ людино-годин;

$Т_{II} = (122.4 + 19.44 + 9.12) \cdot 8 = 1207.68$ людино-годин;

Найвищу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 7000 грн., один розробник апаратного забезпечення з окладом 9500 грн. Визначимо зарплату за годину за формулою:

$$СЧ = \frac{М}{Т_m \cdot t} \ грн.,$$

де М – місячний оклад працівників; $Т_m$ – кількість робочих днів у місяць; t – кількість робочих годин в день.

$$СЧ = \frac{7000 + 7000 + 9500}{3 \cdot 21 \cdot 8} = 46{,}62 \ грн.$$

Тоді, розрахуємо заробітну плату за формулою

$$СЗП = С_ч \cdot Т_i \cdot КД,$$

де $С_Ч$ – величина погодинної оплати праці програміста; T_i – трудомісткість відповідного завдання; $К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 46.62 \cdot 1150.08 \cdot 1.2 = 64340.08$ грн.

I. $С_{ЗП} = 46.62 \cdot 1207.68 \cdot 1.2 = 67562.45$ грн.

Відрахування на єдиний соціальний внесок в залежності від групи професійного ризику (II клас) становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 64340,08 \cdot 0.22 = 14154.82$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 67562,45 \cdot 0.22 = 14884.54$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 7000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 7000 \cdot 0,2 = 16800$ грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г \cdot (1+ К_З) = 16800 \cdot (1 + 0.2) = 20160$ грн.

Відрахування на єдиний соціальний внесок:

$С_{ВІД} = С_{ЗП} \cdot 0.22 = 20160 \cdot 0.22 = 4435.2$ грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 10000 грн.

$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 10000 = 2875$ грн.,

де $К_{ТМ}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; $К_А$ – річна норма амортизації; $Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 10000 \cdot 0.05 = 575$ грн.,

де $К_Р$ – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$Т_{ЕФ}$ =($Д_К$ – $Д_В$ – $Д_С$ – $Д_Р$) · $t_З$· $К_В$ = (365 – 104 – 8 – 16) · 8 · 0.9 = 1706.4 годин,

де $Д_К$ – календарна кількість днів у році; $Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів; $Д_Р$ – кількість днів планових ремонтів устаткування; t –кількість робочих годин в день; $К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН}$ =1706,4 · 0,156 · 0,2 · 1,94177 = 103.379 грн.,

де $N_С$ – середньо-споживча потужність приладу; $К_З$– коефіцієнтом зайнятості приладу; $Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$С_Н = Ц_{ПР} \cdot 0.67$ = 10000 · 0,67 = 6700 грн.

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н$$

$С_{ЕКС}$ = 20160 + 4435,2 + 2875 + 575 + 103.379 + 6700 = 34848.58 грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$С_{М-Г} = С_{ЕКС} / Т_{ЕФ}$ = 34848.58 / 1706,4 = 20.42 грн/год.

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М-Г} \cdot Т$$

I. $С_М$ = 20.42 · 1150.08 = 23484.63 грн.;

II. $С_М$ = 20.42 · 1207.68 = 24660.82 грн.;

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67$$

I. $С_Н$ = 64340.08 · 0.67 = 43107.85 грн.;

II. $С_Н$ = 67562.45 · 0.67 = 45266.84 грн.;

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н$$

I. $С_{ПП} = 64340.08 + 14154.82 + 23484.63 + 43107.85 = 145087.38$ грн.;

II. $С_{ПП} = 67562.45 + 14884.54 + 24660.82 + 45266.84 = 152374.65$ грн.;

4.5 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕРj} = К_{Кj} / С_{Фj},$$

$К_{ТЕР1} = 6.28 / 145087.38 = 4.328 \cdot 10^{-5}$;

$К_{ТЕР2} = 5.72 / 152374.65 = 3.754 \cdot 10^{-5}$;

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 4.328 \cdot 10^{-5}$.

4.6 Висновки

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП.

Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного

комплексу оптимальним є другий варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 4.328 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

- апаратна платформа на основі сімейства мікросхем Microchip;
- мова програмування Javascript;
- СУБД PostgreSQL.

ВИСНОВКИ

У ході виконання даного дипломного проекту було досліджено та побудовано систему оптимізації енергоспоживання для розумного будинку на основі технології Bluetooth Low Energy.

Було сформульовано основні вимоги до системи, досліджено існуючі підходи до вирішення схожих задач та виокремлено шляхи їх покращення.

Запропоновано та реалізовано архітектуру системи, що збирає дані про стан електроприладів, на їх основі дозволяє автоматизовано приймати рішення про їх включення або відключення з метою балансування навантаження внутрішньої електромережі, попередження несправностей та мінімізації фінансових витрат на електроенергію.

Система є дешевою, надійною та легкою для розгортання завдяки мінімізації кількості необхідного обладнання – вимагається лише наявність комп’ютера з вбудованим або зовнішнім адаптером Bluetooth 4.0 або вище.

Розроблена система повністю задовольняє усім поставленим вимогам та завданню дипломного проекту.

В якості елементів системи було спроектовано та розроблено на основі сімейства мікросхем Microchip два діючі макетні зразки пристроїв моніторингу і контролю електроприладів, що також відповідають усім поставленим вимогам.

Спроектований інтерфейс приладу “мікроконтролер-модуль Bluetooth LE” є універсальним і може бути використаний для розробки будь-яких приладів, для яких необхідна можливість їх моніторингу та керування з мінімальними енерговитратами за допомогою смартфона або комп’ютера (побутові прилади, промислове устаткування, інвертори відновлюваних джерел живлення тощо).

В подальшому планується вдосконалювати алгоритм розробленої системи шляхом впровадження технологій аналізу даних, таких як класифікація і

регресія. Це дозволить зменшити час відповіді системи та автоматизувати вибір значень деяких налаштувань, які наразі формуються тільки вручну. Крім цього, в майбутньому, з появою промислових зразків модулів Bluetooth 5.0, буде оновлено модуль розробленого пристрою, що дозволить отримати значний приріст радіусу дії а також можливість для конфігурації mesh-топології Bluetooth мережі, що значно розширить її потенційні розміри та забезпечить більш надійний зв’язок пристроїв за умови наявності у приміщенні товстих стін та перекриттів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Офіційний сайт компанії СТЕМ - продаж та монтаж сучасного електрообладнання. - Режим доступу: http://www.sutem.com.ua. - Дата доступу: 26.05.2017.
2. Объединение «Коммунар». Счетчики электроэнергии. Режим доступу: http://www.tvset.com.ua/products/meters. - Дата доступу: 26.05.2017.
3. Clean energy for all. The Revised renewable energy directive / European Commission. - 2016. - P. 1-6;
4. Smart Home concepts. - Режим доступу: http://cctvinstitute.co.uk/smart-home. - Дата доступу: 26.05.2017.
5. Jin Ming. Occupancy Detection via Environmental Sensing / Jin M., Bekiaris-Liberis N., Weekly K., Spanos C. J., Bayen A. M. // IEEE Transactions on Automation Science and Engineering. - 2016. - P. 1–13.
6. Control Your Castle. The Latest in HVAC Home Automation. - Режим доступу: http://www.achrnews.com/articles/124160-control-your-castle-the-latest-in-hvac-home-automation. - Дата доступу: 26.05.2017.
7. Lars T. Berger. Smart Grid Applications, Communications, and Security / Lars T. Berger, Krzysztof Iniewski. // Devices, Circuits, and Systems. CRC Press. - 2014. - P. 1-50.
8. Melanie Griffiths. Smart Home Security / Melanie Griffiths. // Homebuilding & Renovating. - 2008. - P. 1-4.
9. Google's parent company is deliberately disabling some of its customers' old smart-home devices. - Режим доступу: http://uk.businessinsider.com/googles-nest-closing-smart-home-company-revolv-bricking-devices-2016-4. - Дата доступу: 26.05.2017.
10. Fahmy H. M. Wireless Sensor Networks: Concepts, Applications, Experimentation and Analysis / Fahmy H. M. // Singapore: Springer. - 2016. - P. 10-55;

11. ZigBee Wireless Technology Architecture and Applications. - Режим доступу: https://www.elprocus.com/what-is-zigbee-technology-architecture-and-its-applications. - Дата доступу: 26.05.2017;
12. Сторінка з Вікіпедії. Wi-Fi. - Режим доступу: https://en.wikipedia.org/wiki/Wi-Fi. - Дата доступу: 26.05.2017;
13. Wi-Fi Direct: Eliminates Access Point / Sam Churchill. - Режим доступу: http://www.dailywireless.org/2010/10/25/wi-fi-direct-eliminates-access-point. - Дата доступу: 26.05.2017;
14. Сторінка з Вікіпедії. Bluetooth. - Режим доступу: https://uk.wikipedia.org/wiki/Bluetooth. - Дата доступу: 26.05.2017.
15. Обзор современных технологий беспроводной передачи данных в частотных диапазонах ISM (Bluetooth, ZigBee, Wi-Fi). - Режим доступу: http://www.wireless-e.ru/articles/technologies/2011_4_6.php. - Дата доступу: 28.05.2017.
16. How Bluetooth 5 and Ilumi MeshTek create the perfect IoT platform for 2017 and beyond. - Режим доступу: https://ilumi.co/blogs/pulse/how-will-bluetooth-5-impact-bluetooth-mesh-networks. - Дата доступу: 28.05.2017.
17. Data Sheet of RN4020 BLE module / Microchip Technology Inc., 2015. – P. 2-40;
18. Data Sheet of dsPIC33F family digital signal controller / Microchip Technology Inc., 2015. – P. 2-65;
19. PICkit 3 In-Circuit Debugger/Programmer User's Guide For MPLAB X IDE / Microchip Technology Inc., 2013. - P. 2-88;
20. PostgreSQL official website. - Режим доступу: https://www.postgresql.org. - Дата доступу: 28.05.2017.
21. Redis official website. - Режим доступу: https://redis.io. - Дата доступу: 28.05.2017.

22. Official documentation for Bluetooth LE Javascript library Noble. - Режим доступу: https://github.com/sandeepmistry/noble/wiki/Getting-started. - Дата доступу: 28.05.2017.
23. ExpressJS official website. - Режим доступу: https://expressjs.com. - Дата доступу: 28.05.2017.
24. VueJS official website. - Режим доступу: https://vuejs.org. - Дата доступу: 28.05.2017.
25. NodeJS official website. - Режим доступу: https://nodejs.org. - Дата доступу: 28.05.2017.
26. NPM official website. - Режим доступу: https://docs.npmjs.com. - Дата доступу: 28.05.2017.
27. PG-PROMISE Github page. - Режим доступу: https://github.com/vitaly-t/pg-promise. - Дата доступу: 28.05.2017.
28. Node Redis Github page. - Режим доступу: https://github.com/NodeRedis/node_redis. - Дата доступу: 28.05.2017.
29. Socket IO official website. - Режим доступу: https://socket.io. - Дата доступу: 28.05.2017.

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

package.json

```
{
  "name": "smart_dispatcher_server",
  "version": "1.0.0",
  "description": "SmartDispatcher web-server for home appliances control",
  "repository": "https://github.com/dvoitekh/smart_dispatcher_server",
  "main": "server.js",
  "scripts": {
    "start": "babel-node --presets es2015 server.js"
  },
  "author": "Dmitry Voitekh",
  "license": "ISC",
  "dependencies": {
    "babel-cli": "^6.24.1",
    "babel-preset-es2015": "^6.24.1",
    "body-parser": "^1.17.1",
    "express": "^4.15.2",
    "pg-promise": "^5.6.7",
    "redis": "^2.7.1",
    "socket.io": "^1.7.3"
  }
}
```

server.js

```
import express from 'express';
import path from 'path';
import redis from 'redis';
import automaticSwitch from './app/utils/automatic_switch';
import { setItem, getItem, setSetting, getSetting, deleteSetting } from
'./app/utils/redis_utils';

const SERVER_PORT = 4000;

const databaseConfig= {
  'host': 'localhost',
  'port': 5432,
  'database': 'smart_dispatcher',
  'user': ''
};

let bodyParser = require('body-parser');
let app = express();
const server = require('http').createServer(app);
const io = require('socket.io')(server);
const pgp = require('pg-promise')({});
const db = pgp(databaseConfig);
let client = redis.createClient();

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
app.use(express.static(__dirname + '/app'));

// HTML. root page. devices list
```

```
app.get('/', (request, response) => {
  response.sendFile(path.join(__dirname + '/app/views/index.html'));
});

// JSON. root page. devices list
app.get('/devices', (request, response) => {
  db.any('SELECT DISTINCT ON (devices.id) * FROM devices JOIN logs ON logs.device_id =
devices.id ORDER BY devices.id ASC, logs.created_at DESC', [true])
    .then(data => response.json(data))
    .catch(error => response.status(500));
});

// HTML. device show page
app.get('/device/:id', (request, response) => {
  response.sendFile(path.join(__dirname + '/app/views/show.html'));
});

// JSON. device show page. device + logs
app.get('/devices/:id', (request, response) => {
  db.any('SELECT * FROM logs WHERE device_id = $1 ORDER BY created_at DESC',
[request.params.id])
  .then((logs) => {
    db.any('SELECT DISTINCT ON (devices.id) * FROM devices JOIN logs ON logs.device_id =
devices.id WHERE devices.id = $1 ORDER BY devices.id ASC, logs.created_at DESC',
[request.params.id])
    .then(device => response.json([...device, ...logs]))
    .catch(error => console.error(error.stack));
  }).catch(error => console.error(error.stack));
});

// switch device
app.patch('/devices/:id', (request, response) => {
  db.one('SELECT * FROM devices WHERE id = $1', [request.params.id])
    .then((device) => {
      setItem(client, `smart_dispatcher:${device.uuid}`, JSON.stringify({ id: device.id,
switch_pending: true }));
      response.status(200);
    }).catch(error => response.status(500))
});

// HTML. settings list page
app.get('/settings_page', (request, response) => {
  response.sendFile(path.join(__dirname + '/app/views/settings.html'));
});

// JSON. settings list page
app.get('/settings', (request, response) => {
  client.keys('*settings*', (error, keys) => {
    Promise.all(
      keys.map(key => getSetting(client, key.split(':').pop()))
    ).then(arr => response.json(arr));
  });
});

// create setting
app.post('/settings', (request, response) => {
  setSetting(client, request.body.key, request.body.value)
    .then(() => response.status(200))
    .catch(() => response.status(500))
});

// delete setting
app.delete('/settings', (request, response) => {
  deleteSetting(client, request.body.key)
    .then(() => response.status(200))
    .catch(() => response.status(500))
});

// socket callback for bluetooth service
```

```
io.on('connection', (socket) => {
  socket.on('deviceLog', (data) => {
    console.log(data);
    let current = 0.22 / 130 * data.current; // max motor current - 0.22A, max ADC - 130
units
    let power = current * 24; // motor voltage - 24V

    db.one('SELECT * FROM devices WHERE id = $1', [data.device_id])
      .then((device) => {
        db.none(
          'INSERT INTO logs(device_id, current, power, temperature, is_on, created_at)
VALUES($1, $2, $3, $4, $5, $6)',
          [data.device_id, current, power, data.temperature, current > 0, new
Date(Date.now() - 2 * new Date().getTimezoneOffset() * 60000).toISOString()]
        ).catch(error => console.log(error));
        db.none(
          `UPDATE devices SET is_on = $1, last_active = $2 WHERE id = $3`,
          [current > 0, current > 0 ? new Date(Date.now() - 2 * new
Date().getTimezoneOffset() * 60000).toISOString() : device.last_active, data.device_id]
        ).then(() => {
          getItem(client, `smart_dispatcher:${device.uuid}`).then((response) => {
            let parsedResponse = JSON.parse(response);

            if (parsedResponse.switch_pending) {
              socket.emit('deviceLog', JSON.stringify({ switch: true, uuid: device.uuid
}));
              setItem(client, `smart_dispatcher:${device.uuid}`, JSON.stringify({ id:
device.id, switch_pending: false }));
            } else {
              automaticSwitch(db, client, data).then((automaticSwitch) => {
                socket.emit('deviceLog', JSON.stringify({ switch: automaticSwitch, uuid:
device.uuid }));
              });
            }
          });
        }).catch(error => console.log(error));
      }).catch(error => console.log(error));
  });
});

// server boots here
server.listen(SERVER_PORT, () => {
  console.log('server listening on port', SERVER_PORT);

  db.any('SELECT * FROM devices', [true])
    .then((devices) => {
      devices.forEach((device) => {
        setItem(client, `smart_dispatcher:${device.uuid}`, JSON.stringify({ id:
device.id, switch_pending: false }));
      });
    })
});
```

automatic_switch.js

```
import { getSetting } from './redis_utils';

export default (db, redis, params) => {
  return new Promise((resolve) => {
    redis.keys('*settings*', (error, keys) => {
      Promise.all(
        keys.map(key => getSetting(redis, key.split(':').pop()))
      ).then((arr) => {
        // collect all settings from redis
        const SETTINGS = arr.reduce((acc, el) => {
          acc[el.key] = el.value;
```

```
          return acc;
        }, {});

        // do not perform switch if AUTO_MODE is disabled
        if (SETTINGS.AUTO_MODE === 'false') {
          resolve(false);
          return;
        }

        db.any('SELECT DISTINCT ON (device_id) * FROM logs JOIN devices ON devices.id =
logs.device_id ORDER BY device_id ASC, created_at DESC')
          .then((logs) => {
            let deviceLog = logs.find(log => log.device_id === params.device_id);

            // do not perform switch if device was marked as invalid
            if (!deviceLog.is_valid) {
              resolve(false);
              return;
            }

            // turn device off if current is twice bigger than optimal
            if (deviceLog.current > deviceLog.optimal_current * 2) {
              Promise.all([
                db.none(
                  `UPDATE devices SET is_on = $1, is_valid = $2, last_active = $3 WHERE
id = $4`,
                  [false, false, new Date(Date.now() - 2 * new Date().getTimezoneOffset()
* 60000).toISOString(), params.device_id]
                ),
                db.none(
                  'INSERT INTO logs(device_id, current, power, temperature, is_on,
created_at) VALUES($1, $2, $3, $4, $5, $6)',
                  [params.device_id, 0, 0, params.temperature, false, new Date(Date.now()
- 2 * new Date().getTimezoneOffset() * 60000).toISOString()]
                )
              ]).then(() => resolve(true));
              return;
            }

            let gapMet = false;
            // turn device on if current time meets settled gaps
            deviceLog.time_gaps.forEach((time_gap) => {
              if (getDateFromTime(time_gap[0]) <= Date.now() && Date.now() <=
getDateFromTime(time_gap[1])) {
                gapMet = true;
                if (deviceLog.power === 0) {
                  resolve(true);
                  return;
                }
              }
            });

            // turn device off if current time does not meet settled gaps
            if (!gapMet && deviceLog.power !== 0) {
              db.none(
                'INSERT INTO logs(device_id, current, power, temperature, is_on,
created_at) VALUES($1, $2, $3, $4, $5, $6)',
                [params.device_id, 0, 0, params.temperature, false, new Date(Date.now() -
2 * new Date().getTimezoneOffset() * 60000).toISOString()]
              ).then(() => resolve(true));
              return;
            }

            // check settings metrics

            // calculate current total power
            let sum = logs.reduce((acc, el) => {
              return acc + el.power;
            }, 0);
```

```
            // turn device off if power threshold is exceeded
            if (sum > SETTINGS.POWER && deviceLog.power !== 0) {
              db.none(
                'INSERT INTO logs(device_id, current, power, temperature, is_on,
created_at) VALUES($1, $2, $3, $4, $5, $6)',
                [params.device_id, 0, 0, params.temperature, false, new Date(Date.now() -
2 * new Date().getTimezoneOffset() * 60000).toISOString()]
              ).then(() => resolve(true));
              return;
            }

            // turn device off if temperature is higher than threshold
            if (params.temperature > SETTINGS.TEMPERATURE) {
              Promise.all([
                db.none(
                  `UPDATE devices SET is_on = $1, is_valid = $2, last_active = $3 WHERE
id = $4`,
                  [false, false, new Date(Date.now() - 2 * new Date().getTimezoneOffset()
* 60000).toISOString(), params.device_id]
                ),
                db.none(
                  'INSERT INTO logs(device_id, current, power, temperature, is_on,
created_at) VALUES($1, $2, $3, $4, $5, $6)',
                  [params.device_id, 0, 0, params.temperature, false, new Date(Date.now()
- 2 * new Date().getTimezoneOffset() * 60000).toISOString()]
                )
              ]).then(() => resolve(true));
              return;
            }

            resolve(false);
          });
        });
    });
  });
};

function getDateFromTime(time) {
  const [hours, minutes, secs] = time.split(':');
  let date = new Date();
  date.setHours(hours);
  date.setMinutes(minutes);
  date.setSeconds(secs);
  return date.getTime();
}
```

device.html

```
<html>
  <head>
    <title>SmartDispatcher</title>
    <link rel="stylesheet" href="../styles/index.css">
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.1.4/Chart.min.js"></script>
    <script src="https://unpkg.com/vue"></script>
    <script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
    <script src="../javascripts/index.js"></script>
  </head>
  <body>
    <div id="device" v-cloak>
      <h3>Device info</h3>
      <table class="with-borders">
        <tr>
```

```
          <td><strong>Id</strong></td>
          <td><strong>Address</strong></td>
          <td><strong>Uuid</strong></td>
          <td><strong>Current Power</strong></td>
          <td><strong>Last Active</strong></td>
          <td><strong>Priority</strong></td>
          <td><strong>Time gaps</strong></td>
          <td><strong>Optimal current</strong></td>
          <td><strong>Is valid</strong></td>
          <td><strong>Is on</strong></td>
          <td><strong>Switch</strong></td>
        </tr>
        <tr>
          <td>{{device.id}}</td>
          <td>{{device.address}}</td>
          <td>{{device.uuid}}</td>
          <td>{{device.power && device.power.toFixed(4)}}</td>
          <td>{{device.last_active}}</td>
          <td>{{device.priority}}</td>
          <td>{{device.time_gaps}}</td>
          <td>{{device.optimal_current}}</td>
          <td>{{device.is_valid}}</td>
          <td>{{device.is_on}}</td>
          <td @click="switchDevice(device.id)" class="switch">Switch</td>
        </tr>
      </table>
      <canvas id="chart" height="100"></canvas>
      <h3>Device log</h3>
      <table class="with-borders">
        <tr>
          <td><strong>Id</strong></td>
          <td><strong>Power</strong></td>
          <td><strong>Current</strong></td>
          <td><strong>Is on</strong></td>
          <td><strong>Temperature</strong></td>
          <td><strong>Logged At</strong></td>
        </tr>
        <tr v-for="log in logs" :key="log.index">
          <td>{{log.index}}</td>
          <td>{{log.power.toFixed(4)}}</td>
          <td>{{log.current.toFixed(4)}}</td>
          <td>{{log.is_on}}</td>
          <td>{{log.temperature}}</td>
          <td>{{log.created_at}}</td>
        </tr>
      </table>
    </div>
  </body>
</html>
```

device.js

```
$(function () {
  if ($('#device').length > 0) {
    var deviceApp = new Vue({
      el: '#device',
      data: {
        logs: [],
        device: { id: window.location.pathname.split('/').pop() }
      },
      mounted() {
        var self = this;
        self.loadLogs();
        setTimeout(function() {
          generateChart(self.logs);
        }, 500);
        setInterval(self.loadLogs, 1000);
      },
```

```
      methods: {
        loadLogs() {
          var self = this;
          $.get('/devices/' + self.device.id).then(function(data) {
            self.device = data.splice(0, 1)[0];
            self.logs = data.map(function(log, index) {
              log.index = index;
              return log;
            });
          });
        },
        switchDevice() {
          var self = this;
          $.ajax({
            url: `/devices/${self.device.id}`,
            type: 'PATCH'
          }).then(function(data) {});
        }
      }
    });
  }
});

function generateChart(data) {
  var LOG_DELTA = 2.5; //seconds between logs
  var dates = data.map(function(d) {
    return Date.parse(d.created_at);
  });
  var SIZE = (new Date().getDate() + 1 - new Date(Math.min(...dates)).getDate());
  var startDate = getStartOfDay(new Date(Math.min(...dates)));
  var endDate = getEndOfDay(new Date());
  var chartLabels = [];
  var chartData = [];
  var groups = [];
  groups.length = SIZE;
  var dayDuration = 1000 * 3600 * 24;

  var logs = data.map(d => ({ date: Date.parse(d.created_at), power: d.power }));
  for(var i = 0; i < SIZE; i++) {
    groups[i] = [];
    logs.forEach(function(log) {
      if (log.date >= startDate + dayDuration * i && log.date <= startDate + dayDuration
* (i + 1)) {
        groups[i].push(log.power);
      }
    });
    chartLabels.push(new Date(startDate + dayDuration * i - new
Date().getTimezoneOffset() * 60000).toISOString().slice(0, 10));
  }

  chartData = groups.map(function(gr) {
    return gr.reduce(function(a, b) {
      return a + b * LOG_DELTA;
    }, 0).toFixed(2);
  });

  var ctx = document.getElementById('chart').getContext('2d');
  var myChart = new Chart(ctx, {
    type: 'line',
    data: {
      labels: chartLabels,
      datasets: [{
        label: 'consumption',
        data: chartData,
        backgroundColor: "rgba(153,255,51,0.4)"
      }]
    }
  });
}
```

package.json

```
{
  "name": "SmartDispatcher",
  "version": "1.0.0",
  "description": "SmartDispatcher process for BT LE communication",
  "repository": "https://github.com/dvoitekh/smart_dispatcher",
  "main": "index.js",
  "scripts": {
    "start": "babel-node --presets es2015 index.js"
  },
  "author": "Dmitry Voitekh",
  "license": "ISC",
  "dependencies": {
    "noble": "^1.8.1",
    "redis": "^2.7.1",
    "socket.io-client": "^1.7.3"
  }
}
```

index.js

```
import noble from 'noble';
import redis from 'redis';
import { sendCommandToPeripheral, discoverCharacteristic } from './utils/api_methods';

const SERVICE_UUID = '00035b0358e607dd021a08123a000300';
const CHARACTERISTIC_UUID = '00035b0358e607dd021a08123a000301'; //indicate, write mode
const HOST_URL = 'http://localhost:4000';
const SOCKET_KEY = 'deviceLog';
let DEVICES = [];
let socket = require('socket.io-client')(HOST_URL);
let client = redis.createClient();

client.keys('*smart_dispatcher*', (error, keys) => {
  keys.forEach((key) => {
    client.get(key, (err, response) => {
      DEVICES.push({ uuid: key.split(':').pop(), id: JSON.parse(response).id });
    });
  });
});

noble.on('stateChange', (state) => {
  if (state === 'poweredOn') {
    noble.startScanning();
  } else {
    noble.stopScanning();
  }
});

noble.on('discover', (peripheral) => {
  if (DEVICES.find(d => d.uuid === peripheral.id )) {

    peripheral.on('disconnect', () => {
      process.exit(0);
    });
```

```
    peripheral.connect((error) => {
      if (!!error) {
        console.log('error during connection', error);
      } else {
        console.log('connected to device with uuid', peripheral.id);
      }

      discoverCharacteristic(peripheral, SERVICE_UUID, CHARACTERISTIC_UUID,
(characteristic) => {
        characteristic.on('data', (data, isNotification) => {
          if (data.toString('utf8') !== '=') {
            socket.emit(SOCKET_KEY, {
              device_id: DEVICES.find(d => d.uuid === peripheral.uuid).id,
              current: + data.toString('utf8'),
              temperature: 22.1
            });
          }
        });

        socket.on(SOCKET_KEY, (data) => {
          if (data === undefined) { return; }
          let response = JSON.parse(data);
          console.log(response);
          if (response.switch && peripheral.uuid === response.uuid) {
            sendCommandToPeripheral(characteristic);
          }
        });
      });
    });
  }
});
```

api_methods.js

```
export function syncronizePeripheral(characteristic) {
  characteristic.write(new Buffer([0x01]), false, (error) => {
    if (error) {
      console.log('write error:', error);
    }
  });

  characteristic.subscribe((error) => {
    if (error) {
      console.log('subscription error:', error);
    }
  });
}

export function sendCommandToPeripheral(characteristic) {
  characteristic.write(Buffer.from('=>R1\r\n'), false, (error) => {});
}

export function discoverCharacteristic(peripheral, service_uuid, characteristic_uuid,
callback) {
  peripheral.discoverServices([service_uuid], (error, services) => {
    services[0].discoverCharacteristics([characteristic_uuid], (error, characteristics)
=> {
      syncronizePeripheral(characteristics[0]);
      callback(characteristics[0]);
    });
  });
}
```