НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Приладобудівний факультет
Кафедра інформаційно-вимірювальних технологій

«На правах рукопису»
УДК 621.3

«До захисту допущено»
Завідувач кафедри
Володимир ЄРЕМЕНКО
(підпис) (ініціали, прізвище)
“___”_____2020р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 152 «Метрологія та інформаційно вимірювальна техніка»
на тему: «Система збору експериментальних даних на основі мікросервісної архітектури»
Виконав: студент VI курсу, групи ПА–91мп
(шифр групи)

Токаренко Олександр Володимирович _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник
доцент кафедри ІВТ, к.т.н., доцент Богомазов С. А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

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

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

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) Приладобудівний факультет
(повна назва)

Кафедра інформаційно-вимірювальних технологій
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою «Інформаційні вимірювальні технології та системи»

Спеціальність 152 «Метрологія та інформаційно вимірювальна техніка»
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Володимир ЄРЕМЕНКО

«___»_____2020__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Токаренку Олександру Володимировичу
(прізвище, ім’я, по батькові)

1. Тема дисертації: Система збору експериментальних даних на основі мікросервісної архітектури

Науковий керівник дисертації Богомазов Сергій Анатолійович, к.т.н., доц.
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Термін подання студентом дисертації: 09.12.2020 р.

3. Об’єкт дослідження: Мережеві інформаційно-вимірювальні системи збору даних для Інтернету речей (IoT)

4. Предмет дослідження: Організація апаратно-програмного забезпечення інформаційно-вимірювальних систем IoT на базі мікросервісної архітектури

5. Перелік питань, які потрібно розробити:

5.1. Огляд особливостей використання мікросервісної архітектури в системах збору даних

5.2 Розробка апаратно-програмного забезпечення рівня збору даних на базі одноплатного комп'ютера Raspberry Pi.

5.3. Розробка серверної частини вимірювальної системи

5.4. Розробка клієнтської частини вимірювальної системи.

5.5. Розробка стартап-проекту Measuring-System.

6. Орієнтовний перелік ілюстративного матеріалу:

Макет демонстраційної системи, презентація, доповідь.

7. Орієнтовний перелік публікацій:

1) “Система збору експериментальних даних на основі Java-фреймворків”

2)” Розробка системи Інтернету речей на базі платформо-незалежних технологій”

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Стартап-проект	Бояринова К.О., к.е.н., доцент, викладач кафедри менеджменту		

9. Дата видачі завдання 15.09.2020 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Огляд існуючих рішень для систем збору даних на базі мікросервісної архітектури	до 29.11.2020 р.	
2.	Розробка апаратного-програмного забезпечення рівня збору даних	до 01.02.2020 р.	
3.	Розробка серверного програмного забезпечення	до 20.03.2020 р.	
4.	Розробка клієнтського програмного забезпечення інформаційно-вимірювальної системи	до 01.07.2020 р.	
5.	Розробка стартап-проекту	до 7.12.2020 р.	
6.	Оформлення пояснювальної записки та ілюстративного матеріалу.	до 7.12.2020 р.	
7.	Підготовка дисертаційної роботи, доповіді-презентації	08.12.2020 – 15.12.2020	

Студент _____ Олександр ТОКАРЕНКО
(підпис) (ім’я, прізвище)

Науковий керівник дисертації _____ Сергій БОГОМАЗОВ
(підпис) (ім’я, прізвище)

РЕФЕРАТ

Магістерська дисертація на тему: «Система збору експериментальних даних на основі мікросервісної архітектури», 104 сторінки, 2 додатки, 24 джерела.

Об’єкт дослідження: Мережеві інформаційно-вимірювальні системи збору даних для Інтернету речей (IoT).

Предмет дослідження: Організація програмного забезпечення інформаційно-вимірювальних систем IoT на базі мікросервісної архітектури.

Мета роботи: Розробка типових рішень організації системи збору даних для Інтернету речей на основі мікросервісної архітектури.

Методи дослідження та апаратура: Робота з технічною документацією, експериментальні дослідження. Датчик температури та вологості DHT22, одноплатний комп’ютер Raspberry Pi, фреймворк Spring WebFlux, база даних MongoDB, програмні засоби для організації мікросервісної архітектури Netflix OSS, фреймворк Angular.

Результати роботи та сучасність продукту: Розроблено мережеву інформаційно-вимірювальну систему збору даних на базі мікросервісної архітектури. Розроблено вузли збору даних на основі одноплатного комп’ютеру RaspberryPi та датчиків DHT22. На базі фреймворку Spring WebFlux, брокеру повідомень Kafka та нереляційної бази даних MongoDB розроблено серверне програмне забезпечення для обробки, збереження та надання веб-клієнту експериментальних даних. Мікросервісний підхід був реалізований на основі сучасного стеку технологій для розподілених веб-систем Netflix OSS, що надало широкі можливості для масштабування та модифікації системи.

Рекомендації щодо використання результатів роботи: Результати розробки можуть бути використані для створення систем Інтернету речей.

МІКРОСЕРВІС, ТЕМПЕРАТУРА, ВОЛОГІСТЬ, ВИМІРЮВАННЯ, АСИНХРОННІСТЬ, СЕНСОРНА СИСТЕМА, РЕАКТИВНІСТЬ, DHT22

ABSTRACT

Master's thesis: “Experimental data collection system based on microservice architecture”, 104 pages, 2 applications, 24 sources

Object of research: Demonstration system for collecting experimental data.

Subject of research: Microservice architecture in data collection systems.

Objective: Development of standard solutions for organizing a data collection system based on microservice architecture**.**

Research methods and equipment: Work with official product documentation, experimental research. Temperature and humidity sensor DHT22 (AM2302), Raspberry Pi single board computer, laptop, WebFlux technology, MongoDB database, technologies for organizing microservice architecture NETFLIX OSS, Angular framework).

Results of work and their novelty: Hardware and software have been developed for the Raspberry Pi microcomputer, to which a sensor for measuring temperature and humidity is connected to GPIO pins. As a result, a demonstration system for collecting experimental data was developed, which consists of a sensor, a single-board computer and a laptop. The server part on the basis of microservice architecture with the asynchronous approach and use of the nonrelational MongoDB database is developed.

Recommendations on the use of work results: The developed system allows measuring and storing temperature and humidity data to monitor this data by the customer or provide third-party service with up-to-date information in real time.

TEMPERATURE, HUMIDITY, MEASUREMENTS, ASYNCHRONITY, SENSOR SYSTEM, MICROSERVICES, REACTIVITY, DHT22

ЗМІСТ

ВСТУП

В сучасних системах Інтернету речей важливу роль відіграє здатність серверного програмного забезпечення до масштабування, спрощення процесу його розробки і модифікації.

Програмне забезпечення на основі монолітної архітектури будується як єдине ціле, в якому інтерфейс користувача і реалізація доступу до даних об'єднані в одну програму на одній платформі. Труднощі при використанні монолітної архітектури виникають при масштабуванні додатків. Кожного разу, коли розробляються, тестуються та впроваджуються нові функціональні можливості, необхідно змінювати весь моноліт, тому що присутня велика зв’язаність між модулями системи.

Мікросервісна архітектура розподіляє додаток на більш дрібні, повністю незалежні компоненти, що забезпечує їм більшу гнучкість і масштабованість. Цей тип архітектури передбачає велику кількість невеликих сервісів, кожен з яких виконує свої власні функції і може бути незалежно розгорнутий. Такі сервіси виконуються в окремих процесах та комунікують між собою через веб-запити або віддалені виклики процедур. При цьому виникає задача загальної організації системи, так як подальша розробка та можливість внесення змін будуть залежати саме від цього. Було проведено аналіз особливостей реалізації такого типу систем та розроблено демонстраційну систему збору експериментальних даних на базі мікросервісної архітектури на основі сучасних Java фреймворків та нереляційної бази даних.

На базі мікросервісного архітектурного стилю розроблена демонстраційна система збору експериментальних даних. Вона складається з наступних компонентів: сервіси, API Gateway, Message broker, MongoDB. **С**лужба API Gateway надає єдину точку входу для певних груп мікросервісів. Служба реалізована на основі за допомогою серверу-шлюзу Zuul. Це маршрутизатор і серверний балансувальник навантаження від

Netflix, що працює в розробленій системі сумісно з сервером регістрації мікросервісів Eureka.

За допомогою брокера повідомлень Kafka в системі реалізовано архітектурний шаблон Message broker**.** Kafka – це високопродуктивна розподілена система обміну повідомленнями. Черга повідомлень дозволяє позбутись зв'язаності між мікросервісами. При її використанні немає необхідності знати про особливості інших компонентів системи, все що потрібно – це передати повідомлення до брокера. Таким чином елементи системи можуть бути замінені без порушення загального функціонування системи з мінімальними зусиллями з боку розробників.

Важливою частиною систем збору та обробки експериментальних даних є збереження отриманої інформації в базі даних. Використано MongoDB – документоорієнтовану систему управління базами даних, яка не потребує опису схеми таблиць. Вважається одним з класичних прикладів NoSQL-систем, використовує JSON-подібні документи і схему бази даних. Однією з переваг такого типу баз даних є масштабованість, ефективне збереження великих об'ємів даних, швидкість виконання операцій та вбудована підтримка асинхронного виконання запитів.

Для реалізації вузлів збору даних було використано одноплатний комп'ютер RaspberryPi та датчики температури та вологості DHT22. Основною завданням програмного забезпечення даного компонента є передача експериментальних даних через API шлюз до мікросервісу Pie-Service.

Для реалізації системи на базі Java-технологій було обрано платформу WebFlux Spring. Платформа WebFlux є альтернативою для версії Spring MVC і реалізує реактивний підхід для створення веб-сервісів. Spring WebFlux реалізує асинхронний і неблокуючий веб-стек, який дозволяє обробляти велику кількість одночасних з'єднань.

Даний підхід дав можливість створити систему з низькою зв’язаністю елементів та широкими можливостями до масштабування та модифікації. Було організовано передачу вимірювальних даних до шлюзу реалізованої веб-системи та їх подальшу обробку, збереження та представлення. В результаті використання реактивного підходу зросли відмовостійкість системи та можливість витримувати великі навантаження.

1 ОСОБЛИВОСТІ ВИКОРИСТАННЯ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ В МЕРЕЖЕВИХ СИСТЕМАХ ЗБОРУ ДАНИХ

1.1 Аналіз основних архітектурних типів побудови серверного програмного забезпечення систем збору даних

Використання сучасних способів розробки і розгортання програмних продуктів є важливим фактором для ефективної розробки систем збору даних. Такі підходи, як сервіс-орієнтована архітектура (SOA) і мікросервіси забезпечують гнучкість при створенні і модифікації додатків, які не реалізуються традиційними монолітними підходами. Однак варто розуміти відмінності між ними, щоб обрати оптимальний підхід до створення майбутньої системи.

Монолітний додаток побудовано як єдине ціле, в якому інтерфейс користувача і доступу до даних об'єднані в одну програму на одній платформі. Корпоративні монолітні додатки складаються з трьох частин:

- база даних, що складається з багатьох таблиць, зазвичай в системі керування базами даних;
- клієнтський інтерфейс, що складається з HTML і JavaScript, запускається в браузері;
- серверні додатки, які працюють як посередник між призначеним для користувача інтерфейсом і базою даних, основним завданням яких є обробка HTTP запитів, виконання деякої специфічної для домену логіки, виконання CRUD операції над ресурсами в базі даних і заповнення HTML-шарів для відправки в браузер.

Труднощі при використанні монолітної архітектури виникають при масштабуванні додатків. Кожен раз, коли ви будуєте, тестуєте та впроваджуєте новий функціонал, ви повинні змінити весь моноліт, тому що присутня велика зв’язаність між модулями системи. Монолітна архітектура

найбільш ефективна в невеликих проектах з чітко визначеною областю, де ви навряд чи будете постійно змінювати або розвивати кодову базу (рис.1.1).

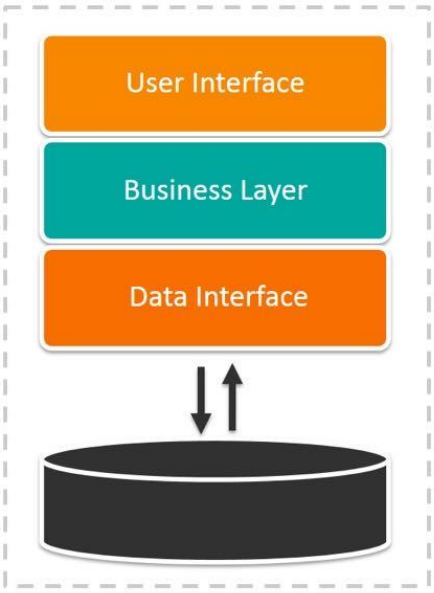

Рисунок 1.1 – Монолітна архітектура

SOA - це архітектурний підхід для визначення, зв'язування і інтеграції бізнес-сервісів багаторазового використання, які мають чіткі межі і незалежні від своїх власних функцій. Ці сервіси взаємодіють один з одним, щоб забезпечити просту передачу даних, яка може включати два або більше сервісів, які координують комунікацію. Складність кожного сервісу в SOA зазвичай дуже низька, і вони взаємодіють один з одним через набір API. Зазвичай SOA використовується в додатках банківського сектора і страхування, де є доступ до різних баз даних і необхідно часто отримувати бізнес-дані і технічні деталі для побудови графічного інтерфейсу. SOA дає можливість великої гнучкості при побудові складних архітектур, і якщо один сервіс вийде з ладу при розгортанні, то це не приведе до падіння всієї системи. Один очевидний недолік полягає в тому, що, не дивлячись на простоту окремих сервісів, архітектура може стати занадто складною, щоб відповідати зростаючим вимогам бізнесу. Тому, якщо немає необхідності розділяти окремі функції системи, то від цього підходу краще утриматися,

оскільки це вимагатиме великих ресурсів та надлишкової програмної реалізації (рис.1.2).

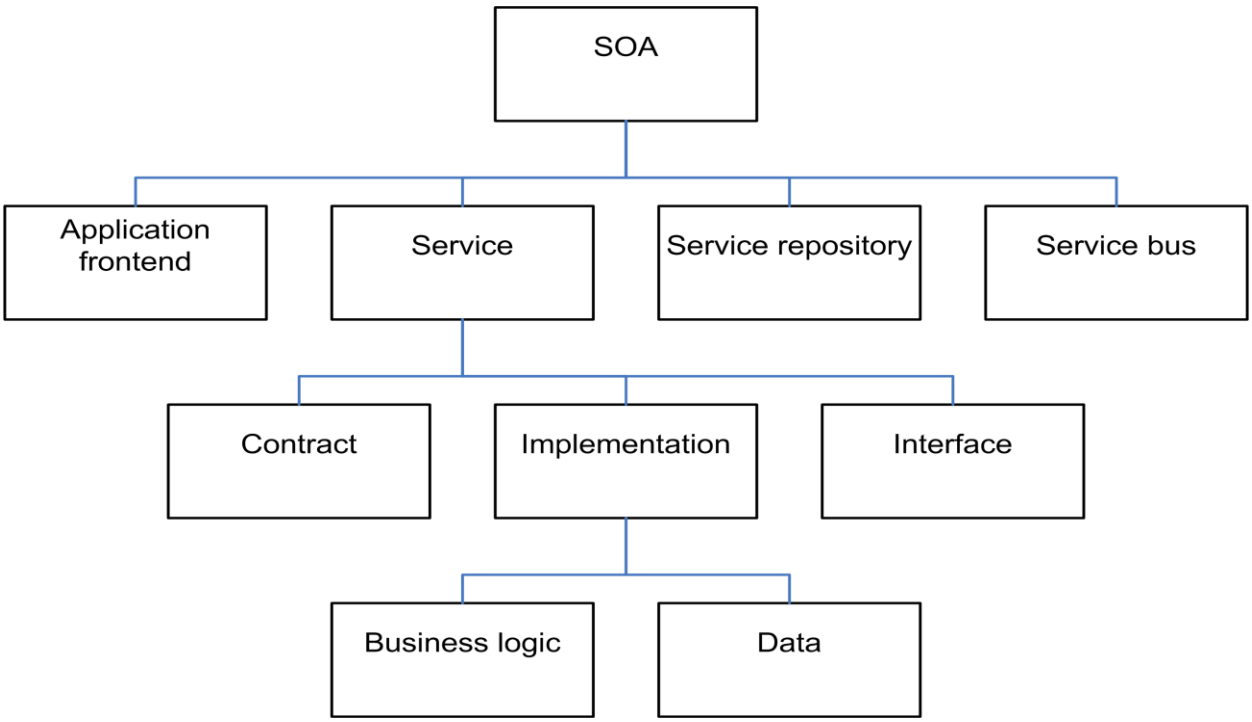

Рисунок 1.2 – SOA архітектура

Мікросервісна архітектура розділяє додаток на більш дрібні, повністю незалежні компоненти, що забезпечує їм більшу гнучкість і масштабованість. Це логічна еволюція SOA, яка відповідає сучасним бізнес-процесам і вимогам. Мікросервіси вирішують проблеми застарілих монолітних систем. Цей тип архітектури складається з великої кількості невеликих сервісів, кожен з яких виконує свої власні функції і може бути незалежно розгорнутий, такий сервіс простіший для розуміння та розробки, що в свою чергу надає можливість безперервного постачання і поліпшення продукту. Архітектура мікросервісов використовує бібліотеки, але їх основний спосіб розбиття додатку - шляхом ділення його на сервіси. Ми визначаємо бібліотеки як компоненти, які підключаються до програми і викликаються нею в тому ж процесі, в той час як сервіси - це компоненти, що виконуються в окремому процесі та комунікуючих між собою через веб-запити або remote procedure call (RPC) (рис.1.3) [1].

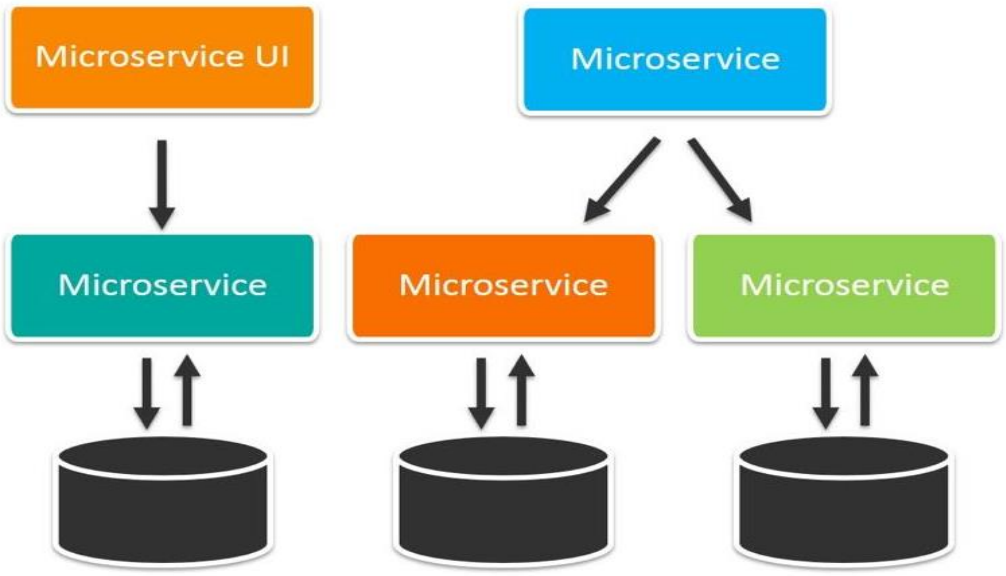

Рисунок 1.3 – Мікросервісна архітектура

Нижче наведений порівняльний аналіз кожного типу архітектури за основними критеріями: проектування, масштабованість, гнучкість та розробка (табл.1.1).

Таблиця 1.1 – Порівняльний аналіз архітектурних підходів

	Мікросервіси	SOA	Моноліт
Проектування	Мікросервіси побудовані у вигляді невеликих додатків, які взаємодіють один з одним	Розмір сервісів може змінюватись від невеликих додатків до дуже великих корпоративних сервісів, які включають більше бізнес функцій	Монолітні додатки розростаються до величезних розмірів, і в таких умовах складно зрозуміти всю суть додатку.
Масштабування	Мікросервіси існують як незалежні одиниці розгортання і можуть масштабуватись незалежно від інших сервісів.	Залежності між сервісами і компонентами, що використовуються повторно, можуть створювати проблеми масштабування	Масштабування монолітних додатків часто може бути проблемою.

Продовження таблиці 1.1

	Мікросервіси	SOA	Моноліт
Гнучкість	Невеликі незалежні модулі спрощують управління збиранням і розгортанням, тим самим забезпечуючи високу ступінь гнучкості програми.	Високий рівень спільного використання компонентів, що збільшує залежність і обмежує можливості управління.	Важко домогтися гнучкості при повторному розгортанні артефактів монолітних додатків.
Розробка	Дозволяє розробникам використовувати відповідне середовище розробки для поставленого завдання.	Багаторазові компоненти і стандартні методи допомагають розробникам в реалізації.	Реалізується з використанням фіксованого стека розробки, який може обмежувати доступність інших технологій.

Таким чином, вважається, що використання мікросервісів є кроком вперед. Головна перевага, в порівнянні з іншими архітектурами в тому, що мікросервісний підхід передбачає розбиття на кілька окремих сервісів, кожен з яких може бути розгорнутий незалежно від інших. Також варто розуміти, що у разі простої системи, де бізнес не передбачає подальшого розширення реалізованого додатку чи глобальне інтегрування нового функціоналу, мікросервісна архітектура буде надлишковою, так як більш проста платформа може бути простіше в реалізації та економніше[2]. Необхідно чітко розуміти ситуацію, в якій використання мікросервісів буде більш доцільним.

1.2 Порівняльна характеристика мікросервісної та монолітної архітектури

Монолітна архітектура вважається традиційним способом побудови додатків. Такий додаток будується як єдине і неподільне ціле. Зазвичай таке рішення містить клієнтський інтерфейс, серверний додаток і базу даних. Він

уніфікований, а всі функції керуються та обслуговуються в одному місці. Зазвичай монолітні додатки мають одну велику кодову базу і не мають модульності. Якщо розробники хочуть щось оновити або змінити, вони працюють з тією ж кодовою базою. Таким чином, зміни вносяться у весь стек одночасно.

Переваги монолітної архітектури. Важливим фактором монолітної архітектури є простота у розробці. Так як монолітний підхід є стандартним способом побудови додатків, будь-яка група розробників володіє необхідними знаннями та можливостями для розробки монолітного додатку. Також не можна не відзначити простоту розгортання такого типу додатку, таким чином не треба організовувати підняття окремих систем для того щоб отримати в повній мірі функціональний додаток, достатньо описати один файл в одному каталозі. На відміну від архітектури мікросервісів, монолітні додатки набагато простіше налаштовувати і тестувати. Оскільки монолітний додаток являє собою єдиний неподільний блок, ви можете виконати наскрізне тестування набагато швидше [3].

Недоліки монолітної архітектури. Однак не дивлячись на вже майже стандартизований на протязі років підхід до розробки веб-додатків, присутні надзвичайно великі мінуси, які в своїй перспективі можуть значно погіршити, та навіть унеможливити подальший розвиток додатку. Проблема масштабованості додатку проявляється в тому, що даний процес не може бути застосований до окремого компонента чи частини системи, а тільки до всією системи цілком. Наступною проблемою є бар’єр для впровадження нових технологій. Вкрай проблематично застосувати нову технологію в монолітному додатку, тому що тоді весь програмний продукт має бути переписано у відповідності новому стеку технологій. Також варто відмітити, що в процесі розширення, додаток стає занадто складним для розуміння і як результат складність внесення будь-яких змін. Складніше реалізувати зміни в великому і складному додатку з дуже сильною залежністю. Будь-яка зміна

коду впливає на всю систему, тому такі процеси повинні бути ретельно скоординовано. Це робить загальний процес розробки набагато довшим.

У той час як монолітний додаток являє собою єдине ціле, архітектура мікросервісів розділяє продукт на сукупність більш дрібних незалежних одиниць. Кожна з цих одиниць відіграє окрему роль у системі та надає відповідний функціонал (послугу). Таким чином, всі сервіси мають свою власну логіку і базу даних, а також виконують певні функції.

Мартін Фаулер визначив архітектуру мікросервісів як підхід до розробки набору невеликих сервісів, що працюють як єдиний додаток. Сервіси обмінюються даними через легкі механізми, такі як HTTP протокол, і кожен сервіс працює незалежно в своєму власному процесі [3]. Іншими словами, в архітектурі мікросервісів вся функціональність розділена на незалежні модулі, які взаємодіють один з одним через певні методи, так звані API (інтерфейси прикладного програмування). Кожен сервіс охоплює свою власну область і може оновлюватися, розгортатися і масштабуватися незалежно[4].

Переваги мікросервісної архітектури. В першу чергу варто відмітити високий рівень відмовостійкості, так як будь-яка помилка в додатку впливає тільки на конкретну сервіс, а не на всю систему. Таким чином, всі зміни та експерименти реалізовані з меншими ризиками і меншою кількістю помилок. Також до плюсів варто додати можливість великої варіативності при виборі стеку технологій. Інженерні команди не обмежені технологією, обраною з самого початку. Вони можуть вільно застосовувати різні рішення і структури для кожного мікросервіса окремо. Важливо відмітити простоту в управлінні і розумінні. Мікросервісний додаток, розбитий на більш дрібні і прості компоненти, як результат - простіший для розуміння і управління. Тобто, можна сконцентруватись на конкретному сервісі та функціоналі, який він надає. Всі раніше перераховані переваги забезпечуються завдяки незалежності компонентів. По-перше, всі служби можуть бути розгорнуті і

оновлені незалежно, що дає велику гнучкість. По-друге, помилка в одному мікросервісі впливає тільки на конкретний сервіс і не впливає на весь додаток. Крім того, набагато простіше додавати нові функції в мікросервісну систему, ніж в монолітну. Як результат - краща масштабованість. Кожен елемент може масштабуватися незалежно. Таким чином, весь процес є менш витратним і економним у порівнянні з монолітами, коли необхідно масштабувати весь додаток, навіть якщо це буде надлишковим. Крім того, у кожному моноліті є обмеження з точки зору масштабованості, тому чим більше користувачів, тим більше проблем з монолітом. Саме через це зараз швидкозростаючі компанії витрачають величезну кількість фінансів на перебудову монолітної архітектури на мікросервісну[5].

Недоліки мікросервісної архітектури. Одним з основних недоліків є розподіленість системи, архітектура мікросервісів являє собою складну систему, що складається з декількох модулів і баз даних, тому забезпечення всіх з'єднань потребую достатньо багато часу та знань. З цього випливає додаткова складність, оскільки архітектура мікросервісів передбачає розподілену систему, необхідно вибрати і налаштувати з'єднання між усіма модулями і базами даних. Крім того, оскільки такий додаток включає в себе незалежні сервіси, всі вони повинні розгортатися незалежно, що в свою чергу передбачає додаткові витрати на апаратну частину, тобто машини на яких будуть відбуватись ці процеси[6].

1.3 Збірка та розгортання мікросервісів

У будь-якому архітектурному рішенні існують компроміси. Зокрема, мікросервісна архітектура тягне за собою серйозну зміну процесу розгортання [7]. Доводиться обслуговувати цілу екосистему невеликих сервісів, а не єдиний, чітко визначене монолітний додаток (рис.1.4).

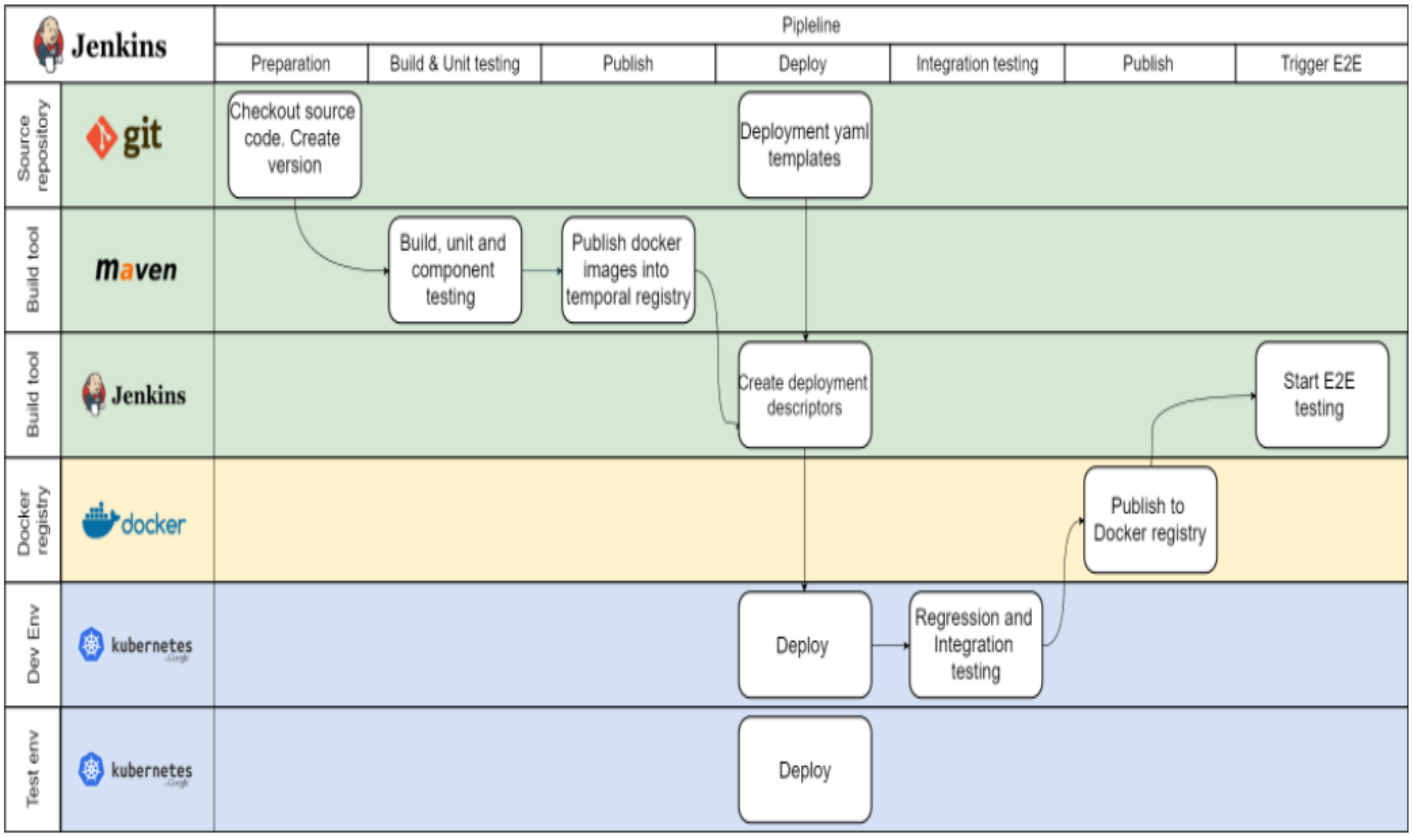

Рисунок 1.4 – Процес розгортання мікросервісів

Наслідком використання сервісів як компонентів є необхідність проектування додатків так, щоб вони могли працювати при відмові окремих сервісів. Будь-яке звернення до сервісу може не спрацювати через його недоступність. Клієнт повинен реагувати на це настільки терпимо, наскільки можливо. Це є недоліком мікросервісної архітектури в порівнянні з монолітом, так як це вносить складність в додаток. Як наслідок, при розробці мікросервісної архітектури слід постійно думати над тим, як недоступність сервісів повинна впливати на загальну роботу системи. Для симуляції збоїв потрібно штучно викликати відмови сервісів, для тестування відмовостійкості додатку і служб моніторингу.

Так як сервіси можуть відмовити в будь-який час, дуже важливо мати можливість швидко виявити неполадки і, якщо можливо, автоматично відновити працездатність сервісу.

Розміщення компонентів у сервісах додає можливість більш точного планування релізу системи. З монолітом будь-які зміни вимагають повторної збірки і розгортання всієї програми. У разі мікросервісної архітектури потрібно розгорнути тільки ті сервіси, які змінилися. Це дозволяє спростити і прискорити процес релізу. Недолік такого підходу полягає в тому, що зміни в одному сервісі можуть бути не погоджені з іншими сервісами. Для забезпечення узгодженості змін в сервісах, і працездатності системи в цілому, потрібно безперервно здійснювати збірку, тестування, перевірку інтеграції [5]. Для побудови мікросервісної інфраструктури потрібно інвестувати час в автоматизацію процесів розробки (виділення ресурсів, збірка, автоматичне тестування, перевірка інтеграції, розгортання). Це дозволить зробити процес однаковим і прозорим, так щоб розгортання системи з однієї складової (монолітні додатки) або з багатьох (мікросервіси) не відрізнялося за своєю складністю.

1.4 Опис архітектури REST API

REST - це архітектурний стиль взаємодії компонентів розподіленого веб-додатку. Принципи REST сформулював Рой Філдінг у своїй дисертації в 2000 році, але не існує офіційно прийнятого стандарту або специфікації REST (на відміну, наприклад, від протоколу SOAP) [8]. API-інтерфейс може вважатися RESTful тільки в тому випадку, якщо дотримані всі вимоги. При створенні демонстраційної системи збору експериментальних даних, були враховані дані вимоги, і тому веб-додаток є RESTful. Існує шість обов'язкових обмежень для проектування REST-додатків:

Модель клієнт-сервер. Система повинна бути поділена на клієнти і сервери, тобто в основі даного обмеження лежить принцип розмежування потреб. Це дозволяє спростити серверну частину програми, що в майбутньому спрощує її масштабованість. При цьому у клієнтського інтерфейсу підвищується можливість перенесення коду на інші платформи.

Відсутність стану. Взаємодія між клієнтом і сервером будується по наступній умові: в моменти часу між запитами ніяка інформація про клієнта не повинна зберігатися на сервері, а кожен запит від клієнта повинен бути складений таким чином, щоб на сервер надійшла вся необхідна інформація про транзакцію для її завершення. При цьому інформація про стан сесії зберігається у клієнта. У той час як відбувається обробка запитів від клієнта, вважається, що він знаходиться в перехідному стані. Дане обмеження дозволяє системі краще масштабуватися, так як відсутність необхідності зберігати інформацію про стани звільняє ресурси сервера, які можуть бути спрямовані на обробку одночасно більшої кількості клієнтів.

Кешування. Клієнти можуть виконувати кешування відповідей сервера. В цьому випадку відповіді сервера повинні бути позначені як ті , що можуть бути кешовані, таким чином зменшуючи кількість запитів до серверу. Грамотно побудований процес кешування здатний зменшити взаємодію між клієнтом і сервером, що збільшує продуктивність і масштабованість розподіленої веб-системи в цілому.

Однорідність інтерфейсу. Для ефективної взаємодії компонентів і кешування в мережі необхідний уніфікований інтерфейс, що дозволяє незалежно розвиватися окремим сервісам.

У свою чергу для уніфікованих інтерфейсів існує чотири умови.

1) Ідентифікація ресурсів. Будь-яка інформація може бути ресурсом - для цього вона повинна мати ім'я. Кожен ресурс повинен бути ідентифікований за допомогою ідентифікатора, що не змінюється при зміні стану ресурсу. Ідентифікатором в REST є URI.
2) Маніпуляція ресурсами через уявлення. Ресурси можуть бути представлені різними способами, наприклад: JSON, HTML, XML опис і так далі. Представлення є описом поточного стану ресурсу і використовується для виконання операцій над ресурсом. Взаємодії клієнтів з ресурсами відбувається за допомогою уявлень.

3) Повідомлення, що описують самі себе. Кожен запит (відповідь) зберігає в собі всю необхідну інформацію, щоб зрозуміти, як його потрібно обробити. Для обробки одного запиту не повинно бути додаткових повідомлень.
4) Гіпермедіа, як засіб зміни стану програми. Для навігації по API повинен бути використаний гіпертекст, що дозволяє клієнтам виявляти ресурси за допомогою гіперпосилань.

Багатошарова система. В архітектурі REST можливий поділ системи на ієрархію шарів, але з умовою: окремий компонент системи може бачити тільки компоненти найближчого рівня. Використання проміжних проксі-серверів дозволяє збільшити масштабованість, збалансувавши навантаження та розподіливши кешування. Також це дозволяє використовувати політику безпеки для забезпечення конфіденційності даних.

Код за необхідністю. Є необов'язковим вимогою. В архітектурі REST, дана умова дозволяє завантажувати програми (код) для виконання на стороні клієнта [9].

Для взаємодії з REST додатком використовуються HTTP методи та відповідна модель запитів на маніпуляцію з ресурсом (об’єктом) (рис.1.5).

HTTP method	Resource URI	Description
GET	/users	gets a list of users
GET	/users/1234	gets a user identified by '1234'
POST	/users	creates a new user
PUT	/users/1234	updates a user identified by ‘1234’
DELETE	/users	deletes all users
DELETE	/users/1234	deletes a user identified by ‘1234’

Рисунок 1.5 – HTTP методи для REST API запитів

Ідемпотентність. Також варто відмітити що не всі методи є ідемпотентними. З точки зору RESTful-сервісу, операція (або виклик сервісу)

ідемпотентна тоді, коли клієнти можуть робити один і той же виклик неодноразово при одному і тому ж результаті, працюючи як "сеттер" в мові програмування. Іншими словами, створення великої кількості ідентичних запитів має такий же ефект, як і один запит. Зауважте, що в той час, як ідемпотентні операції виробляють один і той же результат на сервері (побічні ефекти), відповідь сама по собі може не бути тим такою ж (наприклад, стан ресурсу може змінитися між запитами).

Методи PUT і DELETE за визначенням ідемпотентні. Проте, є один нюанс з методом DELETE. Проблема в тому, що успішний DELETE-запит повертає статус 200 (OK) або 204 (No Content), але для подальших запитів буде весь час повертати 404 (Not Found), якщо тільки сервіс не налаштований так, щоб "позначати" ресурс як видалений без його фактичного видалення. Як би там не було, коли сервіс насправді видаляє ресурс, наступний виклик не знайде цей ресурс і поверне 404. Стан на сервері після кожного виклику DELETE той самий, але відповіді різні. Методи GET, HEAD, OPTIONS і TRACE визначені як безпечні, що також робить їх ідемпотентними.

Безпека. Деякі HTTP-методи (наприклад: HEAD, GET, OPTIONS і TRACE) визначені як безпечні, це означає, що вони призначені тільки для отримання інформації та не повинні змінювати стан сервера. Іншими словами, вони не повинні мати побічних ефектів, за винятком нешкідливих ефектів, таких як: логування, кешування, показ банерної реклами або збільшення веб-лічильника. Створений довільний GET-запит, який не враховує контекст стану програми, слід вважати безпечним. Спрощено, безпечність означає, що виклик методу не має побічних ефектів. Отже, такі запити, клієнти можуть безпечно здійснювати неодноразово, не побоюючись змінити стан сервера. Це означає, що сервіси повинні дотримуватися визначення безпеки для GET, HEAD, OPTIONS і TRACE операцій. Не виконання цієї властивості може вводити в оману споживача сервісу, а також викликати проблеми для веб-кешування, пошукових систем та інших

автоматизованих агентів, які ненавмисно будуть змінювати стан сервера. За визначенням, безпечні операції ідемпотентні, так як вони призводять до одного і того ж результату на сервері. Безпечні методи реалізовані як операції тільки для читання. Однак безпечність не означає, що сервер повинен повертати той же самий результат кожного разу[10].

Висновки

В даному розділі було розглянуті архітектурні підходи до розробки веб-орієнтованих систем збору та обробки експериментальних даних, проведено аналіз та порівняння переваг і недоліків як сучасних підходів так і застарілих. Розглянуто недоліки та переваги використання монолітної та мікросервісної архітектури в сучасному процесі розробки. Проведено аналіз архітектури REST API.

2 РОЗРОБКА АПАРАТНО-ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ РІВНЯ ЗБОРУ ДАНИХ НА БАЗІ ОДНОПЛАТНОГО КОМП'ЮТЕРА RASPBERRY PI

2.1 Організація бездротового з'єднання функціональних елементів системи

Raspberry Pi - це мініатюрний одноплатний комп'ютер, який з легкістю поміститься на долоні дорослої людини. Незважаючи на свої скромні розміри, плата має високу продуктивність, що дозволяє їй вийти на один рівень зі стаціонарними ПК. Спочатку Raspberry Pi була розроблена, як навчальний посібник з інформатики. Але сама ідея виявилася настільки вдалою, що за кілька років міні-комп'ютер став популярний в дуже широких колах. З плином часу Raspberry Pi пережила кілька модифікацій, кожна з яких відрізнялася від попередника тим чи іншим параметром. Такий підхід дозволив регулювати вартість виробу в залежності від потреб користувача, що також позитивно позначилося на популярності пристрою. Вся лінійка Raspberry Pi застосовує процесори з ARM-архітектурою, яка зарекомендувала себе з кращого боку. На рис. 2.1 показаний зовнішній вигляд однієї з популярних плат Raspberry Pi B +.

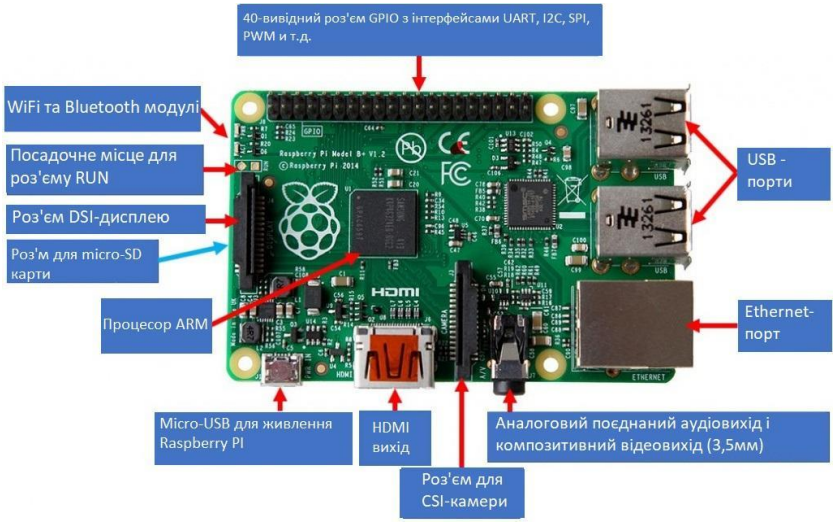

Рисунок 2.1 - Огляд складових елементів Raspberry Pi

На сьогоднішній день (період 2012-2019рр.) Існує 11 різновидів Raspberry Pi. Останні версії оснащені бездротовими WiFi і Bluetooth модулями, які розширюють межі застосування міні-пк в області Ethernet-технологій. Нижче наведено порівняльну таблицю (табл.1.1), в якій відображені особливості кожної модифікації із зазначенням деяких технічних даних.

Таблиця 2.1 - Особливості модифікацій Raspberry Pi

Модифікація	Процесор	Тактова частота	Кількість ядер	Об'єм ОЗУ	Кількість GPIO	Кількість USB	Підтримка Ethernet	Підтримка WiFi	Підтримка Bluetooth	Рік випуску
B	ARM1176JZ-F	700 МГц	1	512 МБ	26	2	√			2012
A	ARM1176JZ-F	700 МГц	1	256 МБ	26	1				2013
B+	ARM1176JZ-F	700 МГц	1	512 МБ	40	4	√			2014
A+	ARM1176JZ-F	700 МГц	1	256 МБ	40	1				2014
2B	ARM Cortex-A7	900 МГц	4	1 ГБ	40	4	√			2015
Zero	ARM1176JZ-F	1 ГГц	1	512 МБ	40	1				2015
3B	Cortex-A53 (ARM v8)	1,2 ГГц	4	1 ГБ	40	4	√	802.11n	4.1	2016
Zero W	ARM1176JZ-F	1 ГГц	1	512 МБ	40	1		802.11n	4.0	2017
3B+	Cortex-A53 (ARM v8)	1,4 ГГц	4	1 ГБ	40	4	√	802.11n	4.2	2018
3A+	Cortex-A53 (ARM v8)	1,4 ГГц	4	512 МБ	40	1		802.11n	4.2	2018
4B	Cortex-A72 (ARM v8)	1,5 ГГц	4	1, 2, 4 ГБ	40	4	√	802.11n	5.0	2019

Як видно з вищенаведеної таблиці, навіть наймолодша модель в лінійці має високі характеристики, з огляду на те, що це одноплатний комп'ютер розміром трохи більше кредитної картки.

На рис. 2.2 зображена остання модифікація Raspberry Pi 4B, запущена в продаж в червні 2019 р.

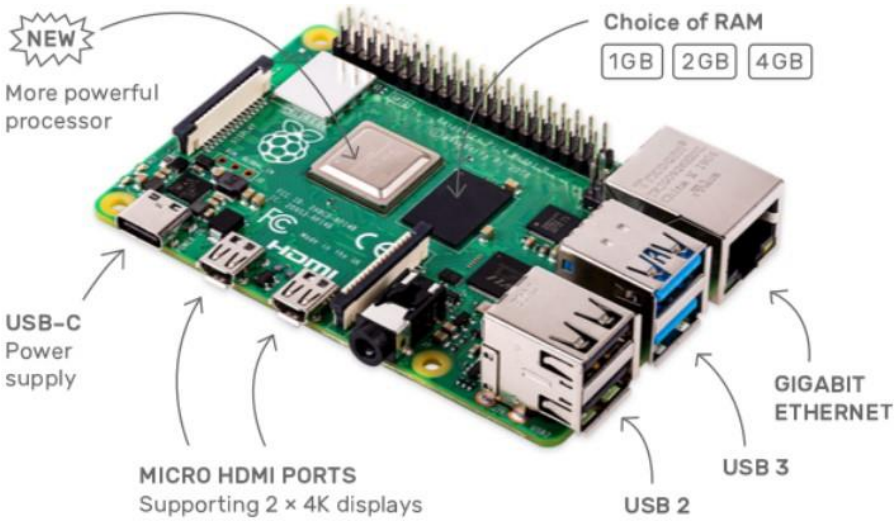

Рисунок 2.2 - Зовнішній вигляд Raspberry Pi 4B

Вона оснащена додатковим графічним процесором VideoCore VI (OpenGL ES 3.x), а також апаратним декодером 4Kp60 для відтворення HEVC відео. Два порти microHDMI з можливістю пропускати сигнал до 4K, дозволяють підключити одночасно два монітори.Основною відмінною рисою Raspberry Pi від звичайних комп'ютерів, є наявність програмованих портів введення-виведення GPIO. За допомогою них можна керувати різними пристроями і приймати телеметрію з різного роду датчиків.

Операційні системи. Хоч Raspberry Pi зовні може нагадати нам Arduino, він все-таки використовує кардинально інший спосіб функціонування. Дана плата, як і звичайний ПК, працює під управлінням однієї із спеціалізованих операційних систем. Залежно від області застосування або особистих симпатій, кожен може вибрати для себе свою. Нижче наведено перелік найбільш популярних «операційних систем» для Raspberry Pi з їх коротким описом.

Raspbian - дана операційна система в 2015 році була представлена як основна для Raspberry Pi. Вона по максимуму оптимізована для процесорів з ARM-архітектурою і досить активно продовжує розвиватися. Основою операційної системи є Debian GNU / Linux. Середовище робочого столу складається з LXDE (середа для UNIX та інших POSIX-сумісних систем типу

Linux і BSD), а також менеджера вікон Openbox (безкоштовний менеджер для X Window System). До складу дистрибутива входять: програма комп'ютерної алгебри Mathematica; модифікована версія Minecraft PI; урізана версія Chrome.

Debian - операційна система з відкритим вихідним кодом. До складу Debian входить більш 59000 пакетів вже скомпільованого ПО. Система використовує ядро Linux або FreeBSD. У стандартний дистрибутив включені: Стільниця GNOME з набором найбільш популярних програм, таких як Firefox, LibreOffice, Evolution, і інший набір для роботи з мультимедіа. Також є можливість установки образів з використовуваними середовищами робочих столів KDE, Xfce, LXDE, MATE і Cinnamon.

Ubuntu - система заснована на Debian GNU / Linux. За популярністю Ubuntu займає перше місце серед дистрибутивів Linux, призначених для web-серверів. До складу дистрибутива входять: програма для перегляду Інтернет; офісний пакет, програми для комунікації і т.д.

Fedora - ця операційна система заснована на дистрибутиві Linux від відомої фірми Red Hat. До складу дистрибутива входять LibreOffice, Mozilla Firefox, а також інше програмне забезпечення, яке можна додатково встановити через Цент Додатків GNOME.

Arch Linux - це вільно поширюваний дистрибутив GNU / Linux загального призначення. Особливістю даної системи є відсутність графічного інсталятора, що може неабияк потренувати навички затятих дослідників Linux.

Gentoo Linux - один з основних систем GNU / Linux з гнучкою технологією управління пакетами. У системі передбачена можливість максимальної оптимізації під конкретне апаратне рішення. алгоритм управління пакетами дає можливість легко реалізувати як робочу станцію, так і сервер.

RISC OS - операційна система спеціально розроблялася для процесорів з архітектурою ARM. Особливості ядра RISC OS дозволяють системі виробляти прискорений запуск за рахунок зберігання даних в ПЗУ. Такий підхід також допомагає захистити дані при різного роду збоїв і впливу шкідливого ПО.

OpenELEC - це програмний комплекс для організації домашнього кінотеатру під управлінням GNU / Linux.

OSMC - ще один комплекс для реалізації домашнього кінотеатру.

В мережі Інтернет, крім перерахованих операційних систем, можна знайти ще безліч модифікацій для самих різних призначень. Але так як Raspbian є основним середовищем для Raspberry Pi, то в подальшому будемо використовувати саме її[11].

Особливості підключення мікрокомп'ютера Raspberry PI. Raspberry Pi призначений для підключення до Інтернету. Його здатність до обміну інформацією через Інтернет є однією з його ключових особливостей і відкриває всілякі можливі способи використання, включаючи автоматизацію домашніх систем, веб-обслуговування, моніторинг мережі тощо. З'єднання може бути налаштовано за допомогою кабелю Ethernet (принаймні у випадку моделі B), або Pi може використовувати USB-модуль для забезпечення мережевого з'єднання. Також однією з переваг підключеного до мережі Raspberry Pi є те, що ви можете підключитися до нього віддалено з іншого комп'ютера. Це дуже корисно в ситуаціях, коли сам Raspberry Pi недоступний і до нього не підключені клавіатура, миша та монітор.

Rapberry Pi має дуже велику варіативність у підключенні та конфігуруванні. Основними шляхами є :

1) Підключення монітору та клавіатури і подальше конфігурування за допомогою цих пристроїв;

2) За допомогою Ethernet кабелю до комп'ютера або WiFi роутера;

3) Бездротове з’єднання через WiFi мережу за допомогою вбудованого модулю або підключеного WiFi-адаптера.

Дротове підключення. У випадку дротового підключення треба мати на увазі, що в деяких моделях не має RJ45 роз’єму для підключення Ethernet кабелю. В усіх інших випадках, потрібно підключити патч-кабель Ethernet до роз’єму RJ45, інший кінець до WiFi маршрутизатора як зображено на рис. 2.3.

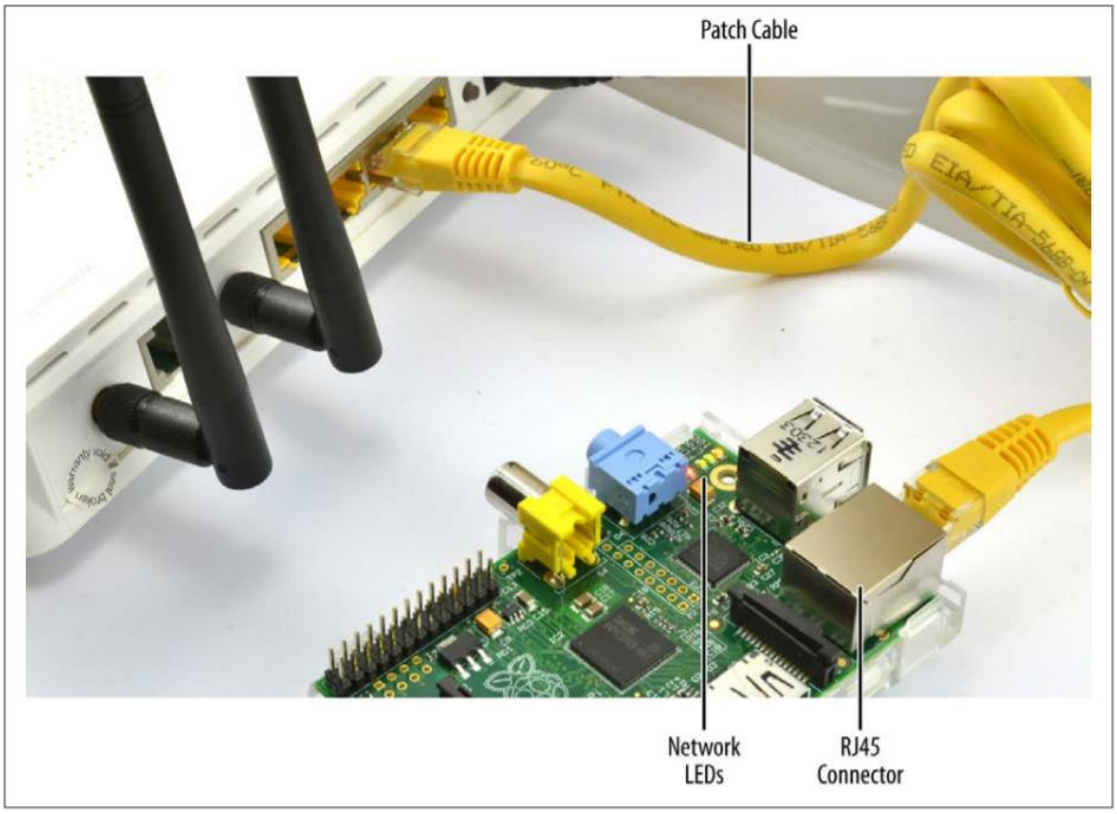

Рисунок 2.3 – Дротове підключення Raspberry PI

Світлодіоди мережі на Raspberry Pi повинні негайно почати блимати під час підключення Raspberry Pi до мережі.

Дистрибутиви Raspbian та Occidentalis попередньо налаштовані для підключення до будь-якої мережі за допомогою DHCP (Протокол динамічної конфігурації хосту). Їм буде автоматично присвоєна IP-адреса, якщо DHCP увімкнено у налаштуваннях мережі. Якщо мережеві світлодіоди на Raspberry Pi не світяться, під час підключення до домашнього маршрутизатора, треба

переконатися, що кабель дійсно робочий або змінити налаштування виходів роутера.

Бездротове підключення за допомогою технології Wi-Fi. Wi-Fi є торговою маркою Wi-Fi Alliance для бездротових мереж на базі стандарту IEEE 802.11.Спочатку назва Wi-Fi була задумана для залучення споживачів схожістю з Hi-Fi (від англ. High Fidelity, що в перекладі «висока точність»). У перших прес-релізах зустрічалася повна назва Wireless Fidelity, але в даний час від нього повністю відмовилися на користь скорочення. Тепер Wi-Fi ніяк не розшифровується. Будь-яке обладнання, яке відповідає стандарту IEEE 802.11, може пройти тестування в Wi-Fi Alliance; виробник цього обладнання при позитивних результатах випробувань отримує відповідний сертифікат, а також право нанести логотип Wi-Fi.

Wi-Fi залишається однією з найбільш перспективних технологій бездротового зв'язку. Вона стрімко розвивається і розробляє нові рішення для організації бездротового з’єднання, що дозволяють збільшити швидкість передачі даних. Навіть з розвитком LTE-мереж, Wi-Fi не залишається осторонь, а скоріше отримує додаткову гілку розвитку, розвантажуючи трафік в найбільш популярних ділянках мережі. Wi-Fi для застосування всередині приміщень в рамках встановленої законодавством потужності випромінювання не вимагає отримання дозволу на використання частот. Крім того, організація Wi-Fi-мережі в умовах будинку або невеликого офісу досить проста, завдяки чому, найчастіше, можна обійтися своїми силами. Проте, при проектуванні мережі з високими вимогами до якості зв'язку, щільності покриття і пропускної здатності, як правило, вдаються до допомоги фахівців. Розгортання Wi-Fi-мережі займає на порядок менше часу в порівнянні з прокладкою СКС до робочих місць. Саме через простоту налаштування, розгортання, відносну дешевизну і зручність, Wi-Fi по праву вважається однією з перспективних технологій. Вимоги до Wi-Fi-обладнання описані в наборі стандартів IEEE 802.11. З випуском кожного нового

стандарту, до 802.11 додавалася буква, наприклад, 802.11a / b / n і т.д. На сьогоднішній день налічується кілька десятків різновидів стандартів Wi-Fi. Не всі стандарти були спрямовані на збільшення швидкості передачі даних, деякі з них торкаються питань безпеки (наприклад, 802.11i), інші включали опис роботи роумінгу (802.11r) і т.д.

У таблиці нижче (табл.2.2) наведені стандарти бездротового зв'язку Wi-Fi, в яких проводилося збільшення швидкостей передачі даних:

Таблиця 2.2 - Стандарти бездротового зв'язку Wi-Fi

Стандарт	Діапазон	Рік випуску	Приблизна швидкість, Мбіт/с
802.11	2.4 ГГц	1997	1
802.11b	2.4 ГГц	1999	5 (11)
802.11a	5 ГГц	2001	54
802.11g	2.4 ГГц	2003	54
802.11n	2.4 / 5 ГГц	2009	600
802.11ac	5 ГГц	2014	7000
802.11ad	60 ГГц	2009	7000
802.11ax	2.4 / 5 ГГц	2019	11 000
802.11ay	60 ГГц	у розробці	20 000

При цьому слід зазначити, що не всі перераховані стандарти Wi-Fi служать для організації бездротових локальних мереж як звичні нам роутери, що працюють в діапазонах 2.4 і 5 ГГц (стандарти 802.11 a / b / g / n / ac). Такі стандарти як 802.11ad і 802.11ay спочатку планувалося випустити для передачі даних на невеликі відстані - від 1 до 10 метрів - і, в перспективі, використовувати їх для організації високошвидкісних інтерфейсів передачі даних, наприклад для підключення моніторів до ПК і передачі зображення в форматі 8K . Однак, в результаті розвитку 5G-мереж і переходом в діапазон

до 100 ГГц, пристрої з підтримкою 802.11ad стали застосовуватися для організації радіо-доступу поза приміщеннями (але для таких частот повинні бути забезпечені умови прямої видимості).

Таким чином, у Wi-Fi велике майбутнє, яке дозволить використовувати дану технологію в абсолютно різних додатках. Безсумнівно, дана технологія знайде своє місце як в 5G-мережах, IoT-рішеннях, так і в VR-додатках, що зображено на рис. 2.4.

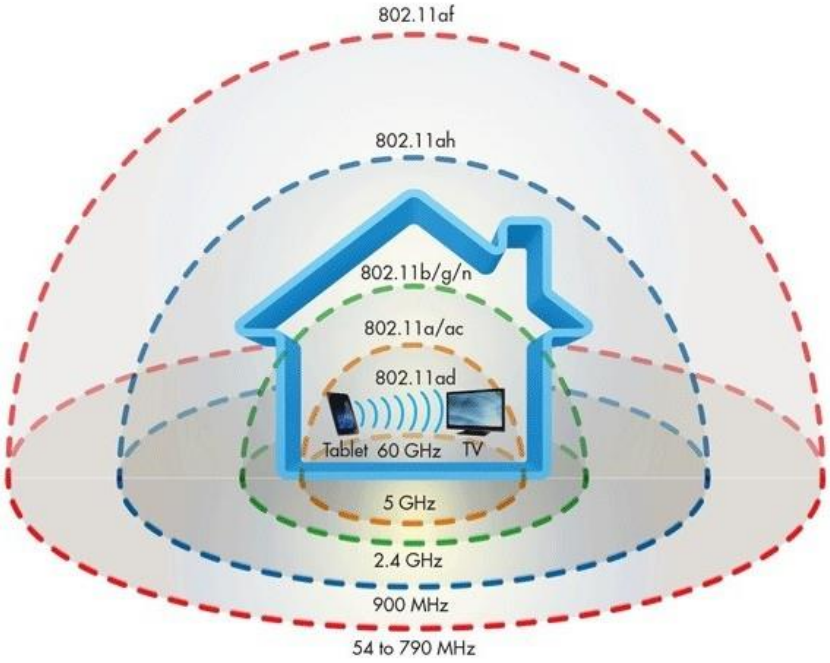

Рисунок 2.4 – Вживання різноманітних стандартів Wi-Fi

Одним з важливих моментів у використанні WiFi є безпека та захищеність трафіку.

Існує безліч небезпечних ризиків, пов'язаних з бездротовими протоколами і методами шифрування. Таким чином, для їх мінімізації використовується надійна структура різних протоколів бездротової безпеки. Ці протоколи бездротової безпеки забезпечують запобігання несанкціонованого доступу до комп'ютерів шляхом шифрування переданих даних в бездротовій мережі.

Нижче перераховані протоколи безпеки, відсортовані за ступенем безпеки (вгорі - найбільш безпечні).

1) WPA3
2) WPA2 Enterprise
3) WPA2 Personal
4) WPA + AES
5) WPA + TKIP
6) WEP
7) Open Network (без захисту)

Wired Equivalent Privacy (WEP). Перший протокол безпеки був названий Wired Equivalent Privacy або WEP. Цей протокол залишався стандартом безпеки з 1999 по 2004 рік. Хоча ця версія протоколу була створена для захисту, проте, вона мала досить посередній рівень безпеки і була складна в налаштуванні. У той час імпорт криптографічних технологій був обмежений, а це означало, що багато виробників могли використовувати тільки 64-бітове шифрування. Це дуже низько-бітове шифрування в порівнянні з 128-бітними або 256-бітними опціями, доступними сьогодні. В кінцевому рахунку, протокол WEP не стали розвивати далі.

WiFi Protected Access (WPA). Для поліпшення функцій WEP в 2003 році був створений протокол Wi-Fi Protected Access або WPA. Цей покращений протокол і раніше мав відносно низьку безпеку, але його легше було налаштувати. WPA, на відміну від WEP, використовує протокол Temporary Key Integrity Protocol (TKIP) для більш безпечного шифрування. Оскільки Wi-Fi Alliance зробили перехід з WEP на більш просунутий протокол WPA, вони повинні були зберегти деякі елементи WEP, щоб старі пристрої все ще були сумісні. На жаль, це означає, що такі вразливості як функція налаштування WiFi Protected, яку можна зламати відносно легко, все ще присутні в оновленій версії WPA.

WiFi Protected Access 2 (WPA2). Роком пізніше, в 2004 році, стала доступна нова версія протоколу Wi-Fi Protected Access 2. WPA2 володіє більш високим рівнем безпеки, а також він простіше налаштовується в

порівнянні з попередніми версіями. Основна відмінність в WPA2 полягає в тому, що він використовує покращений стандарт шифрування Advanced Encryption Standard (AES) замість TKIP. AES здатний захищати надсекретну урядову інформацію, тому це хороший варіант для забезпечення безпеки WiFi будинку або в компанії. Єдина помітна вразливість WPA2 полягає в тому, що як тільки хтось отримує доступ до мережі, він може атакувати інші пристрої, підключені до цієї мережі. Це може стати проблемою в тому випадку, якщо у компанії є внутрішня загроза, наприклад, нещасний співробітник, який здатний зламати інші пристрої в мережі компанії (або надати для цих цілей своє пристрою хакерам-професіоналам).

WiFi Protected Access 3 (WPA3). У міру виявлення вразливостей вносяться відповідні зміни і поліпшення. У 2018 році Wi-Fi Alliance представив новий протокол WPA3. Як очікувалось, ця нова версія буде мати «нові функції для спрощення безпеки Wi-Fi, забезпечення більш надійної аутентифікації і підвищення криптографічного стійкості для високочутливих даних». Нова версія WPA3 все ще впроваджується, тому обладнання, сертифіковане для підтримки WPA3, поки не є доступним для більшості людей.

Порівняння WPA та WPA2 наведено в таблиці нижче (табл.2.3):

Таблиця 2.3 – Порівняльна характеристика WPA та WPA2

	WPA	**WPA2**
Рік випуску	2003	2004
Метод шифрування	Temporal Key Integrity Protocol (TKIP)	Advanced Encryption Standard (AES)
Рівень безпеки	Вище ніж у WEP, пропонує базовий рівень безпеки	Вище ніж у WPA, пропонує підвищений рівень безпеки
Підтримка пристроїв	Може підтримувати більш застаріле ПО	Сумісний тільки з більш новим ПО
Довжина паролю	Допускається більш короткий пароль	Потребує більший пароль

Необхідні обчислювальні можливості	Мінімальні	Необхідно більше обчислювальних можливостей

Таким чином, зрозуміло, що протокол WPA2 є кращим варіантом, за умови, якщо пристрій підтримує даний протокол шифрування[12].

Підключення Raspberry PI до Wi-Fi мережі. Підключити Raspberry Pi до Wi-Fi можна за допомогою звичайного Wi-Fi-адаптера. Для цього необхідно підключити адаптер через USB роз'єм, та перевірити видимість даного пристрою в операційній системі, за допомогою команди "lsusb", як зображено на рис. 2.5.

```
pi@raspberrypi:~ $ lsusb
Bus 001 Device 004: ID 2357:010c TP-Link TL-WN722N v2
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp. SMSC9512/9514 Fast
Ethernet Adapter
Bus 001 Device 002: ID 0424:9512 Standard Microsystems Corp. SMC9512/9514 USB Hu
b
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Рисунок 2.5 – Перелік USB підключень в ОС Raspberry PI

Таким чином, дійсно спостерігаємо, що присутній Wi-Fi адаптер TP-Link TL-WN722N v2. Тобто, система дійсно ідентифікувала даний пристрій. Наступним кроком буде перевірка мережевих інтерфейсів за допомогою команди "ifconfig -a", що зображено на рис. 2.6.

```
pi@raspberrypi:~ $ ifconfig -a
eth0: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether b8:27:eb:22:67:5e  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 2  bytes 78 (78.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2  bytes 78 (78.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.104  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::d237:45ff:fe7a:1178  prefixlen 64  scopeid 0x20<link>
        ether d0:37:45:7a:11:78  txqueuelen 1000  (Ethernet)
        RX packets 809  bytes 229568 (224.1 KiB)
        RX errors 0  dropped 25  overruns 0  frame 0
        TX packets 132  bytes 21026 (20.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Рисунок 2.6 – Перелік мережевих інтерфейсів Raspberry Pi

Таким чином бачимо, що wlan0 дійсно присутній, тож тепер залишається лише налаштувати даний вид підключення. Для цього треба відредагувати файл мережевих інтерфейсів, що знаходиться в наступній директорії “/etc/network/interfaces”. Для редагування необхідно ввести команду “ sudo nano /etc/network/interfaces” та пересвідчитись у наявності наступних рядків у даному файлі конфігурації (рис.2.7):

```
allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

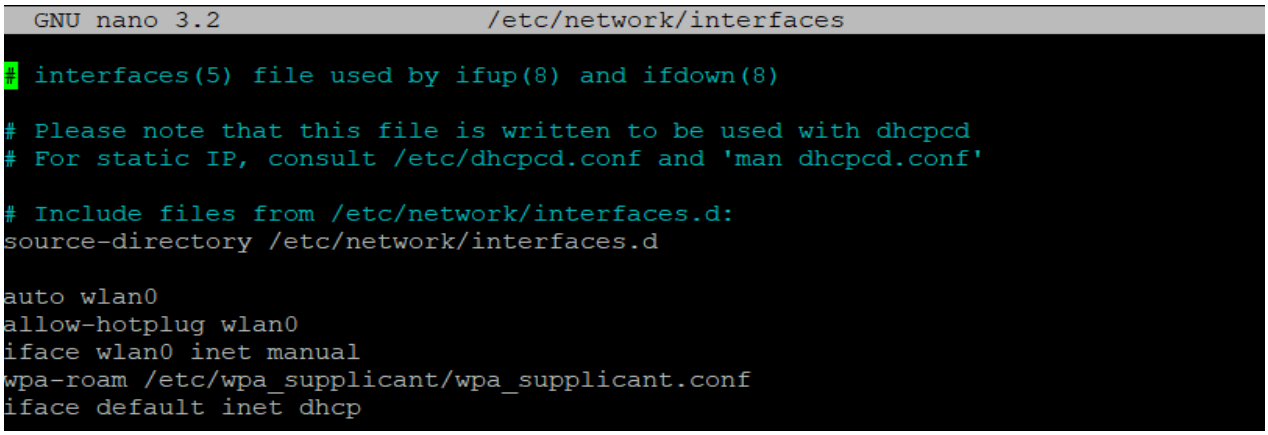

Рисунок 2.7 – Файл конфігурації мережевих інтерфейсів

Далі за допомогою команди “ sudo iwlist wlan0 scan | grep ESSID” необхідно перевірити які саме Wi-Fi мережі ідентифікує даний адаптер (рис.2.8):

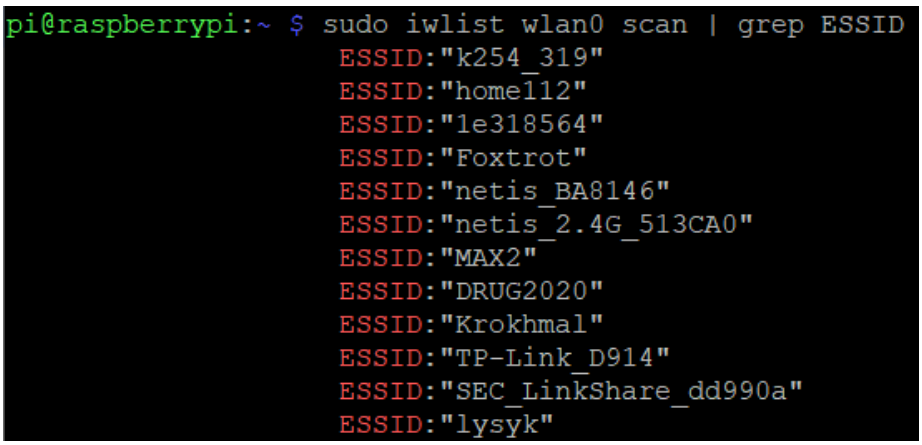

Рисунок 2.8 – Сканування доступних Wi-Fi мереж

Бачимо, що наша мережа тут присутня і сам адаптер правильно функціонує. Тепер необхідно внести інформацію про точку доступу в файл конфігурації “/etc/wpa_supplicant/wpa_supplicant.conf” за допомогою команди “ sudo nano /etc/wpa_supplicant/wpa_supplicant.conf” та вказати дані, що відповідають мережі, а саме (рис.2.9):

- ssid="Назва точки доступа";
- psk="пароль для підключення";
- key_mgmt=WPA-PSK (тип протоколу шифрування паролю);

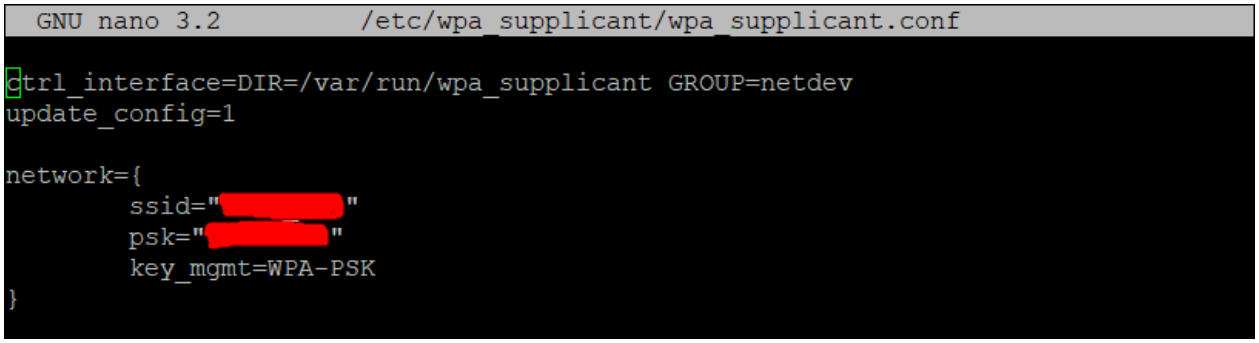

Рисунок 2.9 – Конфігурації підключення до мережі

Після цього необхідно виконати перезапуск, використавши команду “sudo reboot” і після перезавантаження перевірити вдалість підключення до мережі, виконавши команду “ iwconfig” , як зображено на рис. 2.10.

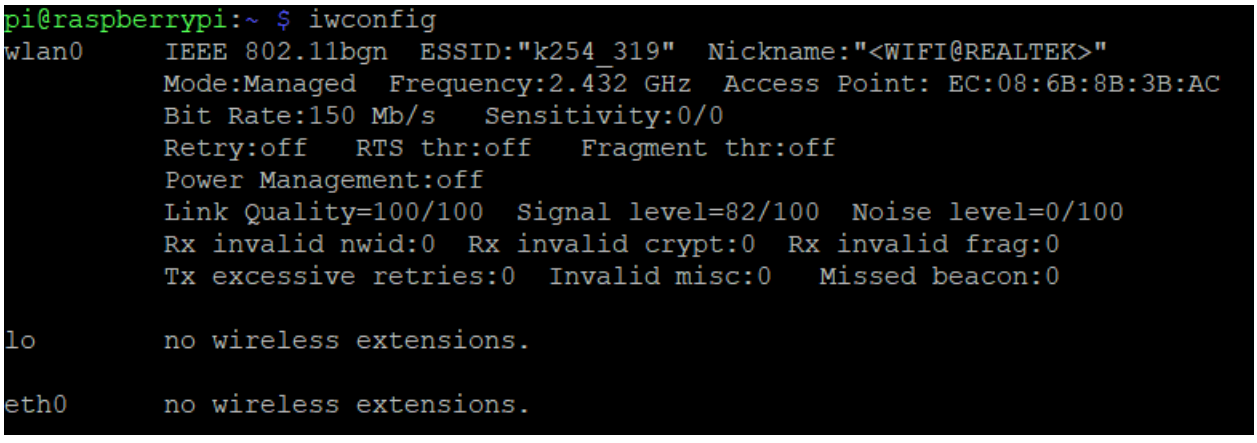

Рисунок 2.10 – Перевірка статусу підключення до бездротової мережі

Таким чином, бездротове підключення одноплатного комп'ютера Raspberry Pi вважається завершеним[13].

Керування мікрокомп’ютером Raspberry Pi. Raspberry Pi - міні-комп'ютер, з яким в більшості випадків взаємодіють віддалено. Це зручно, так як не доводиться кожен раз, коли потрібно щось підключити або змінити конфігурацію, підключати монітор, клавіатуру та мишу.

Існує декілька способів, за допомогою яких можливо з'єднатися з Raspberry Pi. Один з них - це використання Secure Shell.

SSH (від англ. "Secure Shell") - це протокол віддаленого адміністрування, розроблений для здійснення віддаленого управління операційними системами і тунелювання TCP-з'єднання. Використання цього протоколу допускає використання різних алгоритмів шифрування, що дозволяє безпечно працювати практично в будь-якому незахищеному середовищі: працювати з ПК через командну оболонку, передавати по шифрованому каналу будь-який тип даних (наприклад, відео-та аудіо-файли). Перший реліз протоколу відбувся в 1995 р, а вже в 1996 р була представлена вдосконалена його версія, яка і стала основою для подальшого розвитку продукту. Сьогодні для всіх мережевих ОС доступні SSH сервер і SSH клієнт, а сам протокол SSH є одним з найбільш популярних рішень для віддаленого управління системами і передачі важливої інформації.

Принцип роботи. SSH це протокол, який використовує клієнт-серверну модель для аутентифікації віддалених систем і забезпечення шифрування даних, обмін якими відбувається в рамках віддаленого доступу. За замовчуванням для роботи протоколу використовується TCP-22 порт: на ньому сервер (хост) очікує вхідне підключення і, після отримання команди і проведення аутентифікації, організовує запуск клієнта, відкриваючи обрану користувачем оболонку. При необхідності користувач може змінювати використовуваний порт. Для створення SSH підключення клієнт повинен ініціювати з'єднання з сервером, забезпечивши захищене з'єднання і підтвердивши свій ідентифікатор (перевіряються відповідність ідентифікатора з попередніми записами, що зберігаються в RSA-файлі, і

особисті дані користувача, необхідні для аутентифікації). Використання SSH підключення має ряд переваг:

- безпечна робота на віддаленому ПК з використанням командної оболонки;
- використання різних алгоритмів шифрування (симетричного, асиметричного і хешування);
- можливість безпечного використання будь-якого мережевого протоколу, що дозволяє передавати захищений обмін файлами будь-якого розміру.

Щоб забезпечити SSH доступ користувачеві необхідні SSH-клієнт і SSH-сервер. Кожна операційна система має свій набір програм, що забезпечують з'єднання. Так, для Linux це lsh (server і client), openssh (server і client). Для Mac OS часто використовується NiftyTelnet SSH. А в ОС Windows для реалізації з'єднання через SSH протокол найчастіше використовується додаток PuTTY https://www.putty.org. Для використання PuTTY необхідно завантажити та інсталювати додаток, після чого в графічному інтерфейсі можна здійснити настройку програми. Додаток має 4 вкладки [14]:

- Session. У цій вкладці здійснюється настройка підключення до сервера.
- Terminal. Тут можна коригувати налаштування роботи терміналу, через який і здійснюється вся робота.
- Connection. У цій вкладці можна задати параметри підключення, вибрати алгоритм шифрування і задати інші настройки з'єднання.
- Window. У цьому вікні користувач може вибрати зовнішній вигляд програми, змінити шрифт і колір тексту (рис. 2.11).

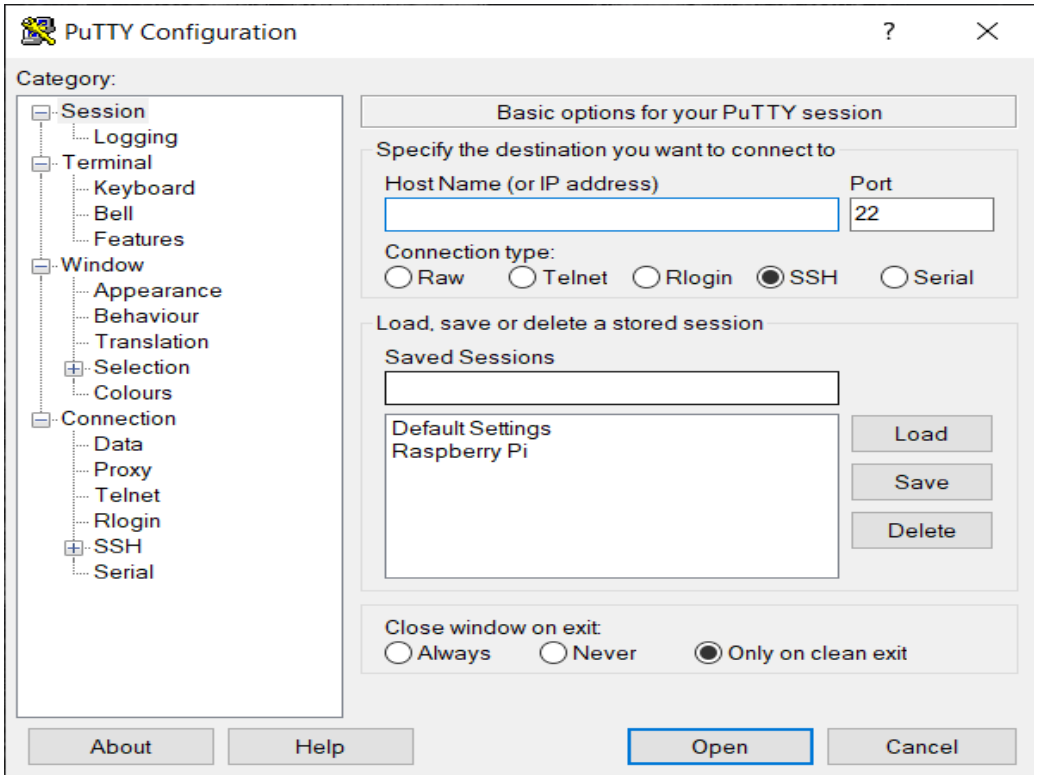

Рисунок 2.11 – Графічний інтерфейс додатку PuTTY

У PuTTY, як і в інших додатках для запуску SSH-протоколу, використовуються два основні методи аутентифікації: за паролем і по ключу. При використанні аутентифікації по паролю користувач використовує для встановлення з'єднання персональний логін і пароль. У разі використання ключа попередньо генеруються відкритий (на пристрої, до якого будуть підключатися) і закритий (на пристрої, з якого буде відбуватися підключення) ключі для кожного окремого користувача. Ці файли не передаються під час аутентифікації, система тільки перевіряє, щоб власник відкритого ключа володів і закритим. Цей метод використовується для автоматичного входу в віддалений ПК. У PuTTY для управління віддаленим ПК використовується термінал: і команди, і передача файлів здійснюються тільки через нього. SSH - один з найбезпечніших протоколів для реалізації віддаленого доступу до ПК. Сучасні алгоритми шифрування і широкий вибір інструментів для настройки протоколу роблять його найпопулярнішим варіантом для віддаленого адміністрування комп'ютерів та безпечної передачі даних.

Таким чином, для керування одноплатним комп'ютером необхідно додати відповідну IP адресу до якою ми збираємося підключитись. Для виявлення точної адреси необхідно скористатися командою “ifconfig”, як зображено на рис. 2.12.

```
pi@raspberrypi:~ $ ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 2  bytes 78 (78.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2  bytes 78 (78.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.104  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::d237:45ff:fe7a:1178  prefixlen 64  scopeid 0x20<link>
        ether d0:37:45:7a:11:78  txqueuelen 1000  (Ethernet)
        RX packets 4970  bytes 1504592 (1.4 MiB)
        RX errors 0  dropped 54  overruns 0  frame 0
        TX packets 407  bytes 66540 (64.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Рисунок 2.12 – Перевірка IP-адреси WiFi-адаптера

Бачимо, що використовується адреса 192.168.0.104, тому вказуємо її при створенні підключення та помічаємо тип підключення – “SSH”(рис.2.13):

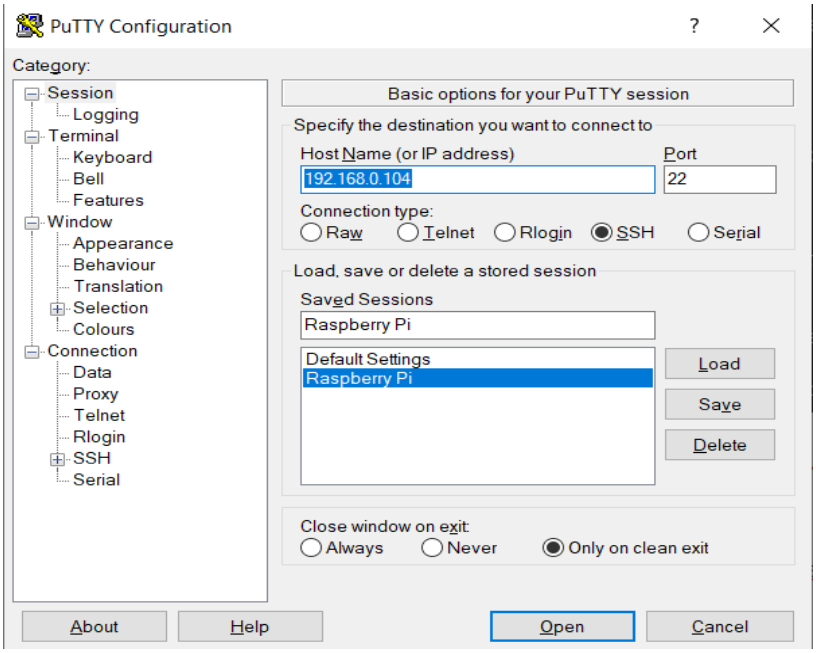

Рисунок 2.13 – Організація підключення за допомогою PuTTY

В консольному вікні, що відкрилося після натискання на кнопку “Open” необхідно буде ввести логін та пароль підключення і натиснути “Enter”. Логін – “pi” та стандартний пароль “raspberry” (рис.2.14):

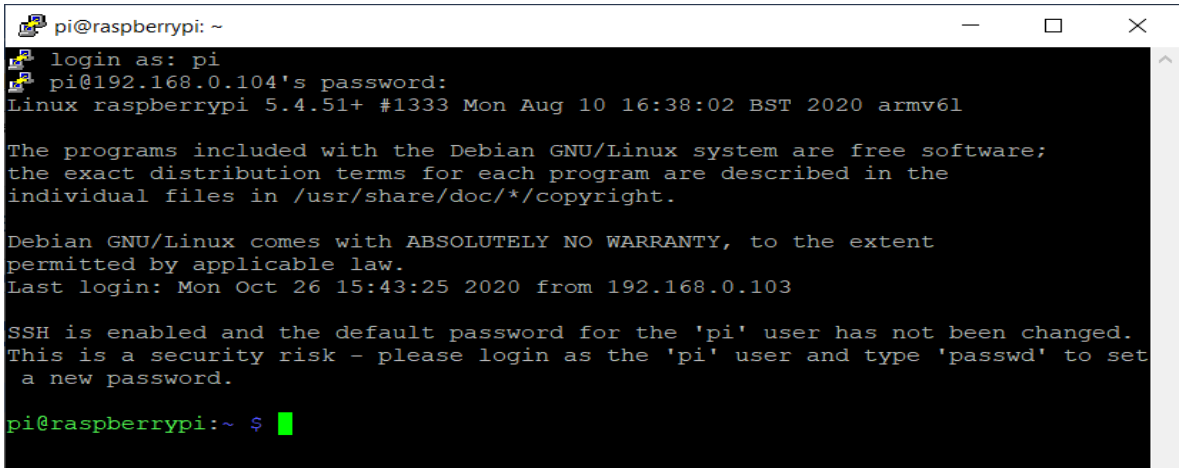

Рисунок 2.14 – Авторизація підключення

Як результат виконано вдале підключення до консолі одноплатного комп’ютера Raspberry Pi.

2.2 Організація підключення сенсора DHT22 до одноплатного комп’ютера Raspberry Pi за допомогою GPIO портів

Потужною особливістю Raspberry Pi є ряд контактів для портів GPIO (вхідні / вихідні) вздовж верхнього краю плати. 40-контактний ряд GPIO(рис.2.15) знаходиться на платах Raspberry Pi.

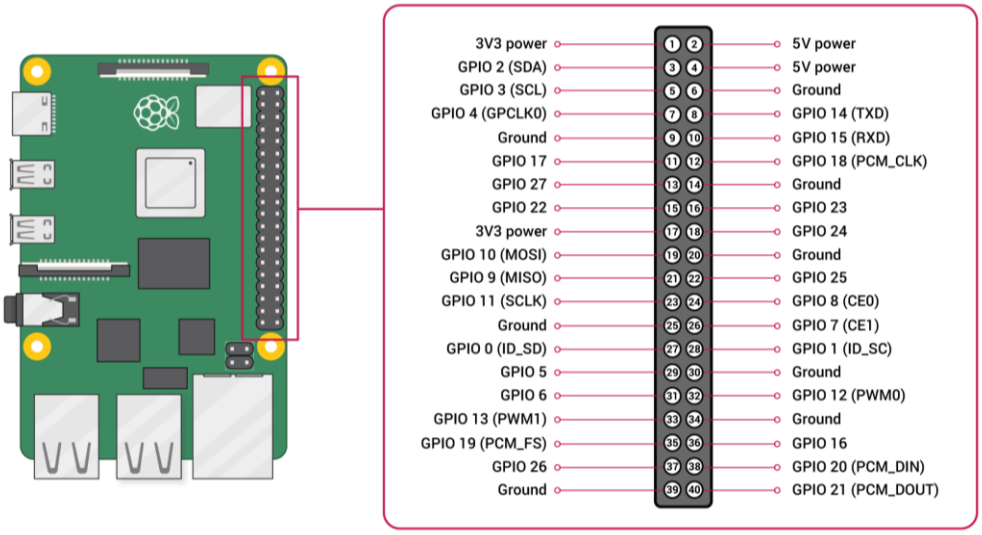

Рисунок 2.15 – Діаграма портів GPIO

В нашому випадку виконано підключення датчика вологості та температури повітря DHT22 (рис.1.16).

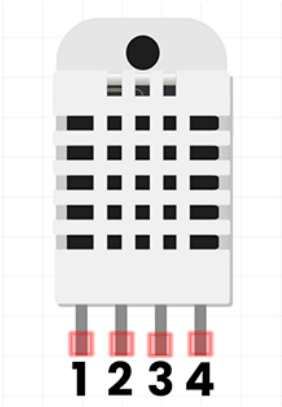

Рисунок 2.16 – Сенсор DHT22

Для більшої зручності контакти сенсора було пронумеровано. Відповідно до кожного з виходів маємо наступні піни GPIO.

Пін 1 – VCC(Джерело живлення).

Пін 2 – DATA(сигнал даних).

Пін 3 – NULL(не підключати).

Пін 4 – GND(земля).

Таким чином, згідно до раніше описаних інструкцій виконано підключення датчика. На рис.2.17 зображено демонстраційний макет.

Рисунок 2.17 – Підключення датчика в демонстраційному макеті

2.3 Розробка апаратно-програмної частини та реалізація передачі даних за допомогою мови Python з використанням HTTP протоколу

Проаналізувавши можливі способи програмної реалізації взаємодії з сенсором DHT22, було прийнято рішення використовувати для цього мову програмування Pyhon.

Таким чином, при запуску система виконує скрипт, основним завданням якого є запуск Python файлу для взаємодії з датчиком та відправки інформації на відповідний сервіс. Файл скрипту знаходиться в директорії “/home/pi/script/launcher.sh”. Реалізація скрипта:

```
#!/bin/sh
#launcher.sh

sudo python3 /home/pi/humidity.py
```

Останній рядок виконує запуск Python файлу, що знаходиться за вказаним шляхом.

Реалізація комунікації з датчиком знаходиться у файлі “humidity.py”, код наведений нижче:

```
import Adafruit_DHT
import requests
import time
import json

DHT_SENSOR = Adafruit_DHT.DHT22
DHT_PIN = 4

while True:
    humidity, temperature = Adafruit_DHT.read_retry(DHT_SENSOR,
DHT_PIN)

    if humidity is not None and temperature is not None:
        print("Temp={0:0.1f}*C
Humidity={1:0.1f}%".format(temperature, humidity))
        url = 'http://192.168.0.103:8083/pie/measurements'
        requests.post(url,   json={"temperature":   temperature,
"humidity": humidity, "location": "50.5124138,30.7913499"})
    else:
        print("Failed to retrieve data from humidity sensor")
```

Командою `import Adafruit_DHT` ми імпортуємо відповідну бібліотеку для взаємодії з сенсором DHT. Потім вказуємо тип сенсору `DHT_SENSOR = Adafruit_DHT.DHT22`. Це необхідно, так як дана бібліотека дає можливість працювати не тільки з датчиком DHT22 але і з DHT11.

Далі зазначено, з якого саме піна GPIO буде надходити інформація `DHT_PIN = 4` . Основна частина скрипта виконана у вигляді нескінченного циклу отримання та відправки даних. Отримання даних відбувається за допомогою виклику метода `Adafruit_DHT.read_retry(DHT_SENSOR, DHT_PIN)`, який в свою чергу повертає значення вологості та температури у неформатованому вигляді. Для уникнення роботи з пустими змінними виконується перевірки на наявність виміряних даних. Наступний рядок виводить результати вимірювання в консоль як зображено на рис.2.18:

```
print("Temp={0:0.1f}*C    Humidity={1:0.1f}%".format(temperature, humidity)).
```

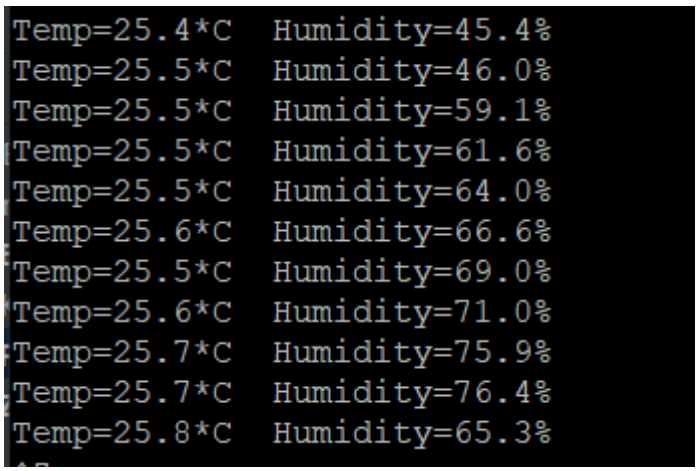

Рисунок 2.18 – Консольний вивід даних з сенсору

Для відправки даних на вхідний шлюз серверної частини системи використовується Python бібліотека “requests” та “json” за допомогою запиту по HTTP протоколу. Попередньо вказана url адреса

```
url = 'http://192.168.0.103:8083/pie/measurements'.
```

Виконується метод

```
requests.post(url, json={"temperature": temperature, "humidity": humidity, "location": "50.5124138,30.7913499"}),
```

в аргументах якого вказана адреса, на яку буде відбуватися відправка POST запиту та сформований за допомогою бібліотеки json об’єкт, який містить виміряні дані та умовну геолокацію датчика для коректного відображення даних з боку клієнту. Зазначена раніше url адреса є вхідним шлюзом (API Gateway) мікросервісної екосистеми. Шлюз направляє потік даних на відповідальний за обробку такого роду даних сервіс, який має відповідну організовану систему мапінгу запитів.

Висновки

Було проведено огляд особливостей використання одноплатного комп’ютера Raspberry Pi в системах збору експериментальних даних, викладено методику бездротового з’єднання функціональних елементів

демонстраційної системи збору даних, налаштовано підключення сенсора DHT22 до мікрокомп'ютера Raspberry Pi, розроблено апаратно-програмне забезпечення рівня збору даних.

3 РОЗРОБКА СЕРВЕРНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА БАЗІ ФРЕЙМВОРКУ SPRING WEBFLUX ТА НЕРЕЛЯЦІЙНОЇ БАЗИ ДАНИХ MONGODB

3.1 Організація демонстраційної системи збору експериментальних даних на базі мікросервісної архітектури

Під час розробки мікросервісної архітектури важливим питанням є організація структури мікросервісів та їх синхронізація в рамках власної екосистеми. Також виникає необхідність в системі моніторингу та впорядкованості існуючих окремих сервісів. Задача була вирішена на базі шаблону мікросервісних систем – Service Discovery. Було обрано найбільш актуальну реалізацію цього шаблону , а саме – Netflix Eureka сервер.

Eureka Server - це додаток, який містить інформацію про всі клієнтські та сервісні додатки. Кожен мікросервіс реєструється на сервері Eureka, і Eureka знає всі клієнтські додатки, що працюють на кожному порту і їх IP-адреси. Eureka Server також відомий як Discovery Server.

Можна сказати, що це сервер імен або реєстр сервісів, основним завданням якого є надання імені, реєстрація та моніторинг кожного мікросервісу. Таким чином, кожен сервіс реєструється в Eureka і відправляє ехо-запит серверу Eureka, щоб повідомити, що він активний. Для цього сервіс повинен бути позначений як @EnableEurekaClient, а сервер @EnableEurekaServer (рис.3.1) [15].

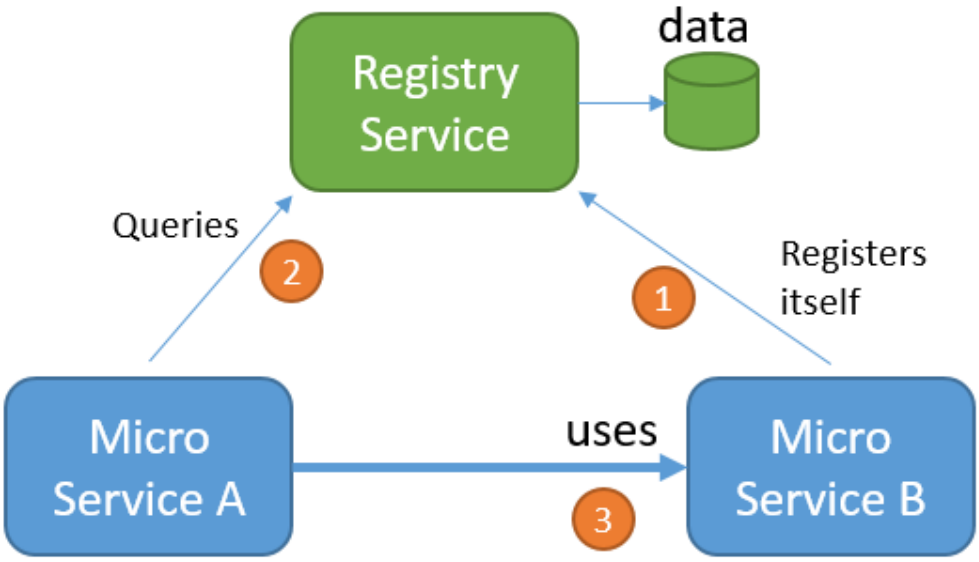

Рисунок 3.1 – Схема роботи Discovery Server

Для реєстрації сервісів було створено окремий додаток Eureka server, конфігурування виконане за допомогою анотацій, та відповідного yaml файлу, що має назву application.yaml. Код файлу конфігурації:

```
server:
  port: ${port:8761}
spring:
  application:
    name: eureka

eureka:
  client:
    register-with-eureka: false
    fetch-registry: false
```

В даному файлі вказано порт, на якому буде запускатись додаток та безпосереднє ім’я додатку. В другій половині файлу вказується актуальна конфігурація Eureka. Так як даний додаток виступає в ролі сервера, що реєструє всі інші сервіси, у нього немає потреби реєструвати самого себе на цьому сервері, тому вказано `eureka.client.register-with-eureka: false`.

Анотації для конфігурування вказані в основному класі додатку.

```
@SpringBootApplication
@EnableEurekaServer
public class EurekaApplication {

    public static void main(String[] args) {
        SpringApplication.run(EurekaApplication.class, args);
    }
}
```

Використана анотація @EnableEurekaServer відповідає за позначення додатку як сервера для реєстрації сервісів, та накладає певні обов'язки постійного моніторингу сервісів за допомогою ехо-запитів з певним інтервалом. Анотація `@SpringBootApplication` відповідає за автоконфігурування компонентів веб додатку і агрегує в собі десятки анотацій, таким чином спрощуючи конфігурування додатку та подальшу розробку функціональності.

3.2 Розробка вхідного шлюзу мікросервісної системи збору даних

При проектуванні і розробці великих або складних додатків на основі мікросервісів з декількома клієнтськими додатками рекомендується використовувати шлюз API. Це служба, що надає єдину точку входу для певних груп мікросервісів. Вона схожа на шаблон фасаду з об'єктно-орієнтованого проектування, але в цьому випадку фасад включається в розподілену систему. Шаблон шлюзу API також іноді називають "серверною частиною для клієнтської частини" (BFF), так як вона створюється з урахуванням потреб клієнтської програми. Таким чином, шлюз API розташовується між клієнтськими додатками і мікросервісами. Він виконує функцію зворотного проксі, передаючи запити від клієнтів до служб. Також він може надавати додаткові наскрізні функції, наприклад аутентифікацію, завершення SSL-підключення та хешування. На рис.3.2 показано, як користувацький шлюз API можна використовувати в спрощеній архітектурі на основі мікросервісів, яка включає декілька мікросервісів.

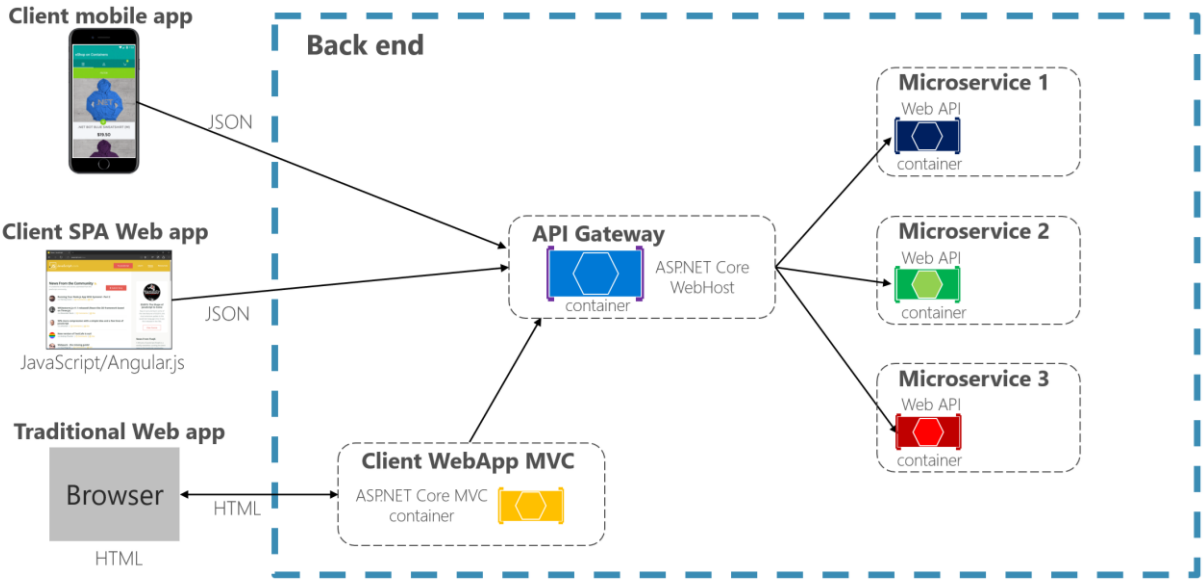

Рисунок 3.2 – Використання шлюзу API

Програма взаємодіє з однієї кінцевою точкою - шлюзу API, який налаштований для пересилання запитів в окремі мікросервіси.

В даній роботі патерн шлюзу API реалізується за допомогою технології від компанії Netflix – Zuul.

Zuul - це проксі, шлюз, проміжний рівень між користувачами і вашими сервісами. Це заснований на JVM маршрутизатор і серверний балансувальник навантаження від Netflix. У нас може бути кілька служб, що працюють на різних портах[16]. Zuul потрібен для прийому зовнішніх запитів і маршрутизації в необхідні послуги внутрішньої інфраструктури. Spring Cloud нативно інтегрований з ним (рис.3.3).

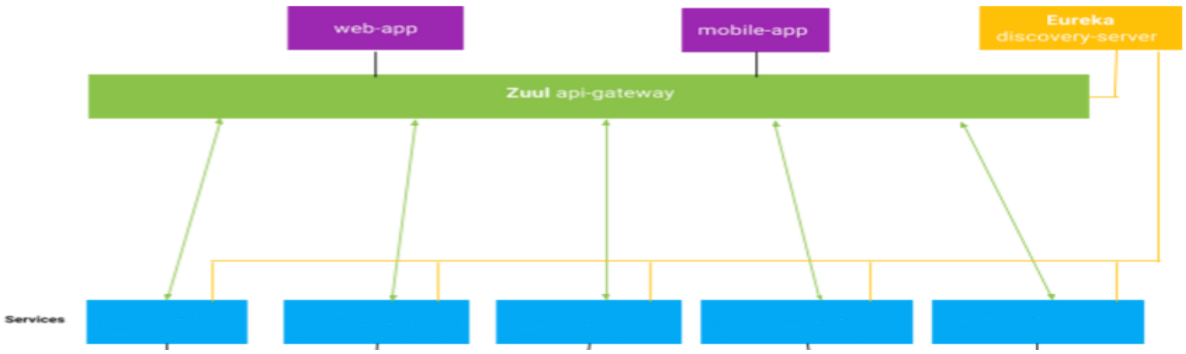

Рисунок 3.3 – Zuul як шлюз API

Використовується з анотацією @EnableZuulProxy в основному класі. Реалізація основного класу:

```
@EnableZuulProxy
@EnableEurekaClient
@SpringBootApplication
public class ZuulServiceApplication {

    public static void main(String[] args) {

                SpringApplication.run(ZuulServiceApplication
                .class, args);
    }

}
```

Також треба позначити цей сервіс як @EnableEurekaClient, щоб наш Eureka Server виявив його і зареєструвався. При старті сервісу він виконує ехо-запит запит на Eureka Server і отримує підтвердження від Eureka. Zuul автоматично вибере список серверів в Eureka. Він запускає попередні фільтри (pre-filters), потім передає запит за допомогою клієнта Netty, а потім повертає відповідь після запуску постфільтрів (post-filters).

Фільтри є основою функціональністю Zuul. Вони можуть виконуватися в різних частинах життєвого циклу "запит-відповідь", тому що вони відповідають за бізнес-логіку додатка і можуть виконувати найрізноманітніші завдання.

Zuul фільтри. Фільтри написані на Groovy, але Zuul підтримує будь-яку мову на основі JVM. Вихідний код кожного фільтра записується в зазначений набір каталогів на сервері Zuul, які автоматично оновлюються в разі будь-яких нововведень. Оновлені фільтри зчитуються, динамічно компілюються в працюючий сервер і викликаються Zuul для кожного наступного запиту.

Пул підключень. Zuul використовує свій власний пул підключень за допомогою клієнта Netty. Це зроблено для того, щоб зменшити зміну контексту між потоками і забезпечити працездатність. В результаті весь запит виконується в одному і тому ж потоці.

Повтор відправки запиту. Однією з ключових функцій, які використовуються Netflix для забезпечення відмовостійкості, є повторна спроба відправка запиту в таких випадках.

- Помилка тайм-ауту;
- Помилка у випадку статус коду(наприклад статус 503).

Але в наступних випадках повторний запит не буде виконано:

- Якщо втрачена частина тіла запиту;
- Якщо була почата відповідь клієнту.

Push Notifications. Починаючи з версії 2.0 Zuul підтримує відправку push-повідомлення - відправку повідомлень з сервера на клієнт. Push-з'єднання відрізняються від звичайних HTTP-запитів тим, що вони постійні і довговічні. Воно підтримує два протоколи, WebSockets і Server Sent Events (SSE)[17].

В демонстраційній системі збору експериментальних даних використані наступні налаштування у файлі application.yml:

```
spring:
  application:
    name: zuul-service
server:
  port: 8083
eureka:
  client:
    serviceUrl:
      defaultZone: ${EUREKA_URI:http://localhost:8761/eureka}
  instance:
    preferIpAddress: true
```

Також було налаштовано роутинг маршрутів на мікросервіси. Тобто, щоб була можливість потрапити на pie-service не через http://localhost:8080, а безпосередньо через Zuul у вигляді http://localhost:8766/pie. Таким чином сервіси нічого не знають один про одного і не знають нічого про Zuul. Це забезпечує слабку зв'язаність мікросервісів, що є однією з переваг використання даної архітектури. Нам взагалі не важливо де знаходиться наприклад pie-service. Нам навіть не треба знати на якому порту, його адресу та інше. Нам треба знати тільки його ім'я.

Для цього було додано наступні конфігурації до файлу application.yml:

```
zuul:
  routes:
    pie-service:
      path: /pie/**
      service-id: pie-service
    keeper-service:
      path: /keeper/**
          service-id: keeper-service
```

та вказаний наступний параметр: “zuul.ignored-services=*”, основна функція якого - зручне найменування URL адресів. Таким чином якщо перейти по адресу http://localhost:8083/pie, то маємо доступ до ресурсів(URL) pie-сервісу, так як Zuul буде перенаправляти запити саме на цей сервіс.

Як результат, запустивши вже розроблену частину додатку можна спостерігати (рис.3.4) новий екземпляр мікросервісу, який зареєструвався на Eureka сервері.

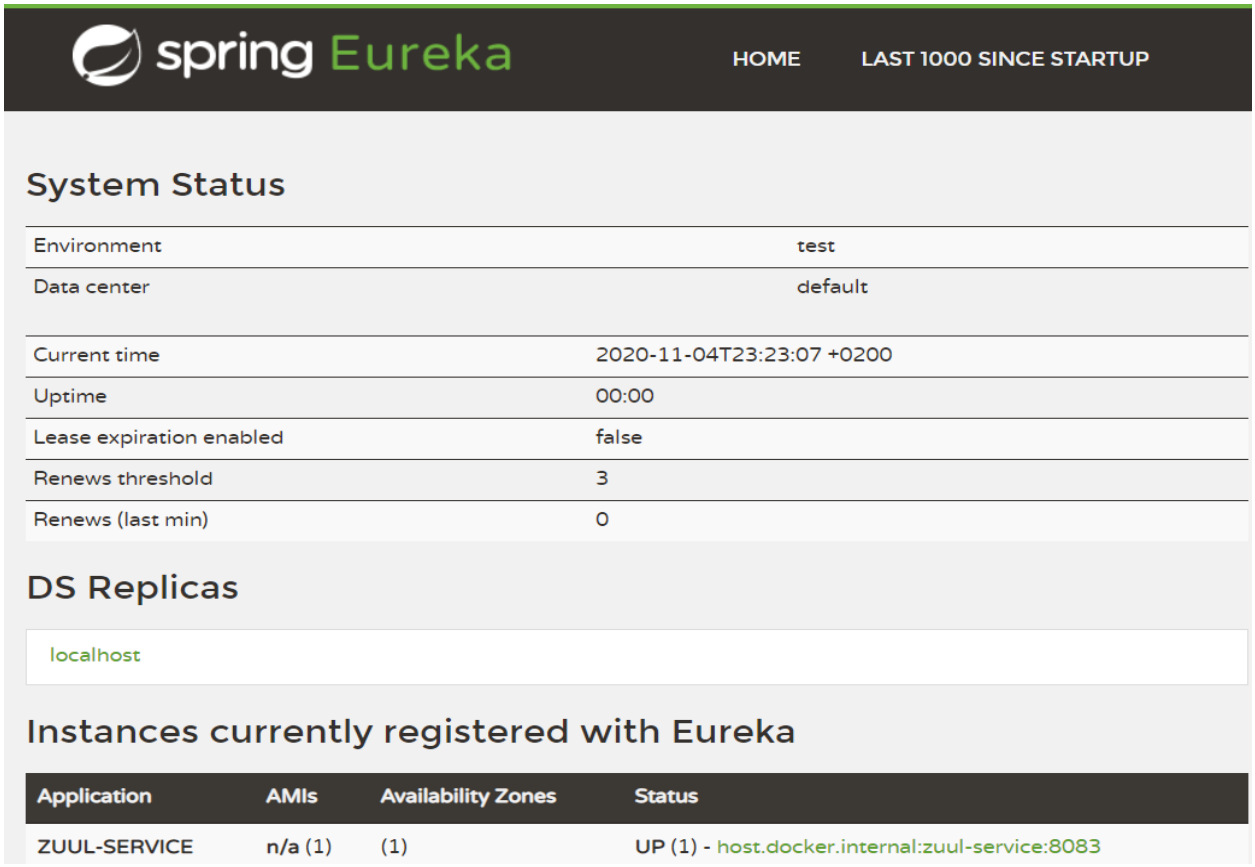

Рисунок 3.4 – Eureka server з зареєстрованим шлюзом API

3.3 Розробка мікросервісу збору експериментальних даних демонстраційної системи

Після реалізації апаратної частини було розроблено сервіс, що спеціалізується на обробці даних, що надходять з одноплатного комп'ютера Raspberry Pi. Сервіс має назву "Pie-Service", та відповідальний за прийом, обробку та подальше розповсюдження даних всередині мікросервісної екосистеми. Принцип роботи полягає в тому, що при надходженні запиту до сервісу, запит проходить так звану процедуру "handler mapping", основною задачею якої є пошук та співставлення спеціалізованого запиту з контролерами та методами в них, що здатні опрацювати цей запит. Даний процес реалізується автоматично, за допомогою Spring Framework. Реалізація веб контролера:

```
@RestController
public class PieController {

    @Autowired
    private SenderService senderService;

    @RequestMapping(method = RequestMethod.POST,value =
"/measurements")
    public void createMeasurement(@RequestBody Measurement
measurement) {
        measurement.setCreatedDate(LocalDateTime.now());
        senderService.sendMeasurement(measurement);
    }
}
```

Над класом веб-контролера необхідно вказати анотацію @RestController, вказуючи, що цей клас буде приймати участь в процесі співставлення запитів та методів їх обробки. За обробку запитів які надходять з Raspberry Pi відповідальний метод `createMeasurement(@RequestBody Measurement measurement)`, над яким також розміщена анотація @RequestMapping яка має в свою чергу декілька параметрів, а саме:

- Method – тип HTTP метода що буде надходити
- Value – частина url шляху на який буде відправлятись запит

За допомогою анотації @RequestBody вказуємо, що в тілі запиту буде знаходитись сутність Measurement, таким чином Spring автоматично перетворить JSON об’єкт в об’єкт типу Measurement, що зменшує кількість коду порівняно з випадком, якщо б цей процес потрібно було реалізовувати вручну за допомогою відповідних класів “маперів”. Основною задачею метода “createMeasurement” є отримання експериментальних даних, створення об’єкту, додавання дати та часу, коли було отримано дані та відправлення даних в мікросервісну екосистему, зокрема до брокеру повідомлень. Сутність Measurement має вигляд:

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Measurement {

    private String id;
    private Double temperature;
    private Double humidity;
    private String location;
    private LocalDateTime createdDate;
}
```

В даному випадку настільки компактний вигляд, класу сутності, забезпечений завдяки інструменту Project Lombok. Проект Lombok - це плагін компілятора, який додає в Java нові «ключові слова» і перетворює анотації в Java-код, зменшуючи зусилля на розробку і забезпечуючи додаткову функціональність. Нижче наведено короткий опис функціональності використаних в даному класі анотацій Project Lombok.

@Data – агрегуюча анотація, яка включає до себе @ToString, @EqualsAndHashCode, @Getter на всі поля, @Setter на всі не фіналізовані поля, та @RequiredArgsConstructor.

@Builder – надає імплементацію відомого патерна конструювання об’єкта Builder.

@NoArgsConstructor – створює реалізацію стандартного конструктора без аргументів.

@AllArgsConstructor – створює реалізацію конструктора з усіма полями класу.

Основний клас додатку:

```
@EnableEurekaClient
@SpringBootApplication
public class PieApplication {

    public static void main(String[] args) {
        SpringApplication.run(PieApplication.class, args);
    }
}
```

Клас `PieApplication` відповідальний за запуск всього додатку, над цим класом розміщені анотації додаткового конфігурування для Spring Framework.. Анотація `@EnableEurekaClient` відповідальна за включення режиму клієнту Eureka серверу, а саме вказує на те, що даний сервіс повинен бути зареєстрований на Eureka сервері та відсилати запит реєстрації на Eureka Server.

Аннотація `@SpringBootApplication` відповідає за автоконфігурування компонентів веб додатку і також агрегує в собі десятки анотацій, таким чином спрощуючи конфігурування додатку та подальшу розробку функціональності.

Крім конфігурування анотаціями, також був створений файл конфігурації application.yml.

```
spring:
    application:
        name: pie-service
Server:
    port: 8080
eureka:
    client:
        serviceUrl:
            defaultZone:
${EUREKA_URI:http://localhost:8761/eureka}
    instance:
        preferIpAddress: true
```

В ньому вказано ім’я додатку `spring: application: name: pie-service`, яке буде відображатись на Eureka Server, порт на якому додаток буде запущений, та безпосередні конфігурації Eureka кліенту, а саме шлях до серверу реєстрації та налаштування IP адреси даного сервісу.

Після запуску Eureka Server та Pie-Service достатньо ввести адресу Eureka серверу в браузері (рис.3.5).

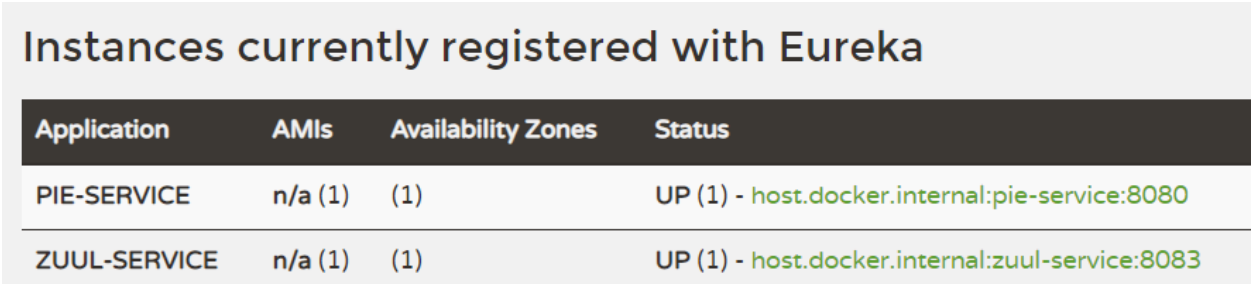

Рисунок 3.5 – Eureka Server

Як результат, можемо спостерігати роботу Eureka серверу та зареєстровані на ньому сервіси: Pie-Service, Zuul-Service (шлюз API).

Після підключення Raspberry Pi до живлення в консольному вікні Pie-Service спостерігаються данні які ми отримуємо за допомогою HTTP запитів від апаратної частини (рис.3.6) до шлюзу API.

```
com.alex.pie.service.SenderService       : Measurement :temperature[24.6]humidity[35.3]time[21:32:39]
com.alex.pie.service.SenderService       : Measurement :temperature[24.7]humidity[35.0]time[21:32:42]
com.alex.pie.service.SenderService       : Measurement :temperature[24.6]humidity[35.4]time[21:32:43]
com.alex.pie.service.SenderService       : Measurement :temperature[24.6]humidity[35.4]time[21:32:43]
com.alex.pie.service.SenderService       : Measurement :temperature[24.6]humidity[35.0]time[21:32:49]
```

Рисунок 3.6 – Консольний вивід Pie-Service

3.4 Розробка сервісу збереження експериментальних даних з використанням MongoDB та Spring WebFlux

Одним з важливих компонентів системи є процес збереження отриманих даних та можливість в подальшому отримання потоку збережених даних для надання інформації клієнту. Для збереження даних використано нереляційна базу даних MongoDB.

MongoDB - це високопродуктивна документо-орієнтована база даних без реляційних схем даних.

Структура. Містить в собі сукупність колекцій, які містять сукупність документів (об'єктів), які в свою чергу містять пари ключ-значення. Документ має динамічну схему, що означає:

1) Документи в одній і тій же колекції не повинні мати однакове кількість пар ключ-значення;
2) Їх типи можуть бути різними.

Структура документа для збереження експериментальних даних:

```
Measurement {

    String id;
    Double temperature;
    Double humidity;
    String location;
    LocalDateTime createdDate;
}
```

Філософія створення.

- Нові технології БД потрібні для сприяння горизонтальному масштабуванню даних, полегшення розробки і здатність зберігати більше інформації.
- Написання коду має бути легше, швидше і зручніше.
- Вміст документа (JSON / BSON) легко кодується, легкий в управлінні і дає відмінну продуктивність, групуючи відповідні дані разом.

Основні характеристики такої БД це:

1) Гнучкість. За рахунок зберігання даних в JSON документах.
2) Потужність. За рахунок збереження безлічі характеристик RDBMS, таких як вторинний ключ, динамічні запити, сортування, легка агрегація і інші. Це надає більшу функціональність, порівняно з тією, яку забезпечують реляційні бази.
3) Швидкість і масштабованість. Зберігаючи пов'язану інформацію в одному документі, запити можуть виконуватися набагато швидше.

Autosharding дозволяє швидке масштабування за рахунок лінійного підключення «машин».

4) Легкість у використанні. MongoDB легка в установці, налаштуванні і використанні[18].

Загальний процес роботи сервісу збереження та доступу до експериментальних даних виглядає наступним чином. Сервіс користується брокером повідомлень, а саме брокером Kafka. Він виступає в ролі Subscriber, таким чином він буде повідомлений про наявність нових елементів в брокері повідомлень, в даному випадку при появі нових результатів вимірювання температури та вологості середовища. Після отримання повідомлення сервіс зберігає дані в MongoDB та виконує відповідне логування в систему, інформуючи таким чином, що процес збереження був вдалим. Налаштування брокеру повідомлень проводилось наступним чином:

В даному випадку Pie-Service виступає в ролі Producer та має наступний сервіс для відправки даних на сервер брокера (Kafka).

```
@Slf4j
@Service
@EnableBinding(Source.class)
@AllArgsConstructor

public class SenderService {

    private final Source source;

    public void sendMeasurement(Measurement measurement) {
        boolean status = source.output().send( MessageBuilder
.withPayload(measurement)
.buld() );
        if (status)
            log.info(measurement + " was successful sent to the
KAFKA(broker)");
        else
            log.warn("Unsuccessful sending measurement to the
KAFKA(broker)");
    }
}
```

За допомогою анотації `@EnableBinding(Source.class)` позначаємо даний клас як джерело з якого надходить інформація. За допомогою методів `source.output().send(MessageBuilder.withPayload(measurement).build())` будуємо повідомлення, в якому буде знаходитись об’єкт результатів вимірювань та відправляємо його. Метод повертає boolean значення успішності відправлення, на базі якого відбувається логування в систему. В файлі налаштувань сервісу application.properties вказано відповідні конфігурації отримувача:

```
spring.cloud.stream.bindings.output.destination=keeper
spring.cloud.stream.bindings.output.group=keeper
spring.cloud.stream.kafka.binder.auto-add-partitions=true
```

В свою чергу Keeper-Service також має відповідний сервіс для прийняття повідомлень з брокеру:

```
@Slf4j
@Service
@EnableBinding(Sink.class)
@MessageEndpoint
public class CheckMeasurementService {

    @Autowired
    private MeasurementService measurementService;

    @StreamListener(Sink.INPUT)
    public void getAvailableMeasurementToSave(Measurement
measurement) {
        log.info("Got [Measurement] from KAFKA(broker): " +
measurement);
        measurementService.saveMeasurement(measurement);
    }
}
```

В даному випадку анотація `@EnableBinding(Sink.class)` помічає клас як отримувача повідомлень. `@StreamListener(Sink.INPUT)` позначає метод як слухача повідомлень. Таким чином метод `getAvailableMeasurementToSave(Measurement measurement)` має один аргумент, а саме об’єкт результатів вимірювання measurement. Таким чином відразу після отримання, за допомогою методу `measurementService.saveMeasurement(measurement)` відбувається

збереження виміряних даних в MongoDB, а факт успішного отримання та збереження об’єкту логується в систему за допомогою методів типу `log.info(…)`.

Для конфігурування сервісу як отримувача повідомлень було додано відповідні рядки до файлу конфігурацій application.properties:

```
spring.cloud.stream.bindings.input.destination=keeper
spring.cloud.stream.bindings.input.group=keeper

spring.cloud.stream.kafka.binder.auto-add-partitions=true
```

За процес збереження відповідає клас MeasurementService реалізацію якого наведено нижче:

```
@Service
@Slf4j
public class MeasurementService {

    @Autowired
    private MeasurementRepository measurementRepository;

    public void saveMeasurement(Measurement measurement) {
        measurementRepository.save(measurement).subscribe();
        log.info("Measurement: temperature=" +
measurement.getTemperature() +
                " | humidity= " + measurement.getHumidity() + "
was saved");
    }

    public Flux<Measurement> getAllMeasurements(){
        log.info("GET ALL MEASUREMENTS");
        return measurementRepository.findAll();
    }

   public Flux<Measurement> findAllByLocation(String location){
     return measurementRepository
          .findAllByLocationAndCreatedDateGreaterThan(location,
                    LocalDateTime.now().minusMinutes(1));
  }
}
```

Метод `saveMeasurement(Measurement measurement)` зберігає об’єкт результатів вимірюваня та виконує відповідне логування події в систему. Для отримання даних з БД використовується метод `findAllByLocation(String`

`location)`, параметром якого є рядок який містить дані локації, на якій було отримано дані та за допомогою методу `findAllByLocationAndCreatedDateGreaterThan(location, LocalDateTime.now().minusMinutes(1))` отримуються найбільш актуальні вимірювання, а саме за останню хвилину. В свою чергу даний сервіс користується методами інтерфейсу для взаємодії з БД - `MeasurementRepository` , код якого наведено нижче:

```
@Repository
public interface MeasurementRepository extends
ReactiveMongoRepository<Measurement, String> {
    @Tailable
    Flux<Measurement>
findAllByLocationAndCreatedDateGreaterThan(String location,
LocalDateTime dateTime);
}
```

Даний інтерфейс позначено анотацією @Repository для того щоб Spring розумів що даний компонент відповідальний за роботу з БД, а при створенні інтерфейсу вказано що він наслідує `ReactiveMongoRepository<Measurement, String>` , зазначений тип об’єкту з яким доведеться працювати та тип поля id, що є унікальним ідентифікатором конкретного об’єкту вимірювання. Анотація `@Tailable` дає можливість тримати з’єднання з БД відкритим за необхідності надання потоку даних в реальному часі. Такий підхід можливий завдяки використанню реактивного підходу при роботі з запитами між сервером та базою даних, що зазначено в основному класі Keeper-Service за допомогою анотації `@EnableReactiveMongoRepositories`. Код основного класу:

```
@EnableEurekaClient
@SpringBootApplication
@EnableReactiveMongoRepositories
public class KeeperApplication {

    public static void main(String[] args) {
        ConfigurableApplicationContext context =
SpringApplication.run(KeeperApplication.class, args);
```

```
        // Explicit creation of capped collection, because
spring haven't any options for specifying collection type
        ReactiveMongoTemplate mongoTemplate =
context.getBean(ReactiveMongoTemplate.class);

mongoTemplate.createCollection(Measurement.class.getSimpleName()
.toLowerCase(),
CollectionOptions.empty().capped().size(5242880).maxDocuments(50
0)).subscribe();

    }

}
```

Варто відмітити, що для підтримки відкритого з'єднання за потреби, важливою умовою MongoDB є те, що колекція повинна бути CAPPED типу. Таким чином наступні рядки коду відповідальні за створення колекцій саме такого типу :

```
ReactiveMongoTemplate mongoTemplate =
context.getBean(ReactiveMongoTemplate.class);

mongoTemplate.createCollection(Measurement.class.getSimpleName()
.toLowerCase(),
CollectionOptions.empty().capped().size(5242880).maxDocuments(50
0)).subscribe();
```

3.5 Контейнеризація мікросервісів та оточуючого їх середовища за допомогою технології Docker

Один із принципів мікросервісної архітектури полягає в тому, що сервіс повинен бути ізольованим і автономним, повністю інкапсулюючи оточення виконання. Для дотримання цього принципу всі компоненти, такі як операційна система, середовище виконання і код мікросервіса повинні бути автономними і ізольованими. Єдиний спосіб домогтися цього – реалізовувати підхід - один мікросервіс на одну віртуальну машину. Однак це призведе до недостатнього використання ресурсів віртуальної машини. Також у багатьох випадках додаткові накладних витрати можуть звести всі переваги мікросервісів нанівець. Технологія контейнеризації далеко не нова і не

новаторська технологія. Вона використовується вже досить тривалий час. Проте, дана технологія стала набирати значну популярність з поширенням хмарних технологій. Недоліки традиційних віртуальних машин стали каталізатором зростання популярності контейнерів. Постачальники інструментів для роботи з контейнерами, наприклад, Docker, в значній мірі спростили технологію контейнеризації, що сприяло широкому впровадженню даної технології.

Технологія контейнеризації надає приватне оточення в операційній системі. Дана технологія також називається віртуалізацією операційної системи [19]. В даному підході, ядро операційної системи надає ізольований віртуальний простір. Кожен з віртуальних просторів називається контейнером. Контейнери дозволяють процесам створювати ізольоване оточення в операційній системі, що містить ці контейнери.

Docker технологія. Docker є найпопулярнішою платформою, що використовує технологію контейнеризації [19]. Платформа Docker вирішує три основні проблеми розгортання сервісів:

- доставка коду на сервер;
- запуск коду;
- однаковість оточення.

Docker дозволяє ізолювати сервіси від інфраструктури, таким чином досягається можливість доставляти їх набагато швидше. Docker дозволяє керувати інфраструктурою за допомогою тих же принципів, які використовуються при керуванні додатками. При використанні інструментів Docker для доставки, тестування і розгортання, значно скорочується затримка між фіксацією нового коду в системі контролю версій і запуском його на сервері промислової експлуатації.

Docker надає можливість упаковувати і запускати сервіси в ізольованому оточенні, яке і називається контейнером. Дана ізольованість і дозволяє безпечно запускати кілька контейнерів на одному хості одночасно.

Контейнери легковагові, оскільки не вимагають роботи гіпервізора. Вони запускаються безпосередньо на ядрі хост машини. Таким чином досягається можливість запуску більшої кількості контейнерів, ніж віртуальних машин, на одному і тому ж фізичному обладнанні. Також, Docker контейнери можна запускати в самих віртуальних машинах.

Docker надає наступні інструменти і платформу для управління життєвим циклом контейнерів. За допомогою Docker відбувається упаковка додатки і їх компонентів в контейнер. В екосистемі Docker контейнер стає атомарним юнітом для поширення і тестування програми. Після розробки, додаток може бути розгорнуто на сервері промислової експлуатації вручну або за допомогою оркестровщика контейнерів (рис.3.7).

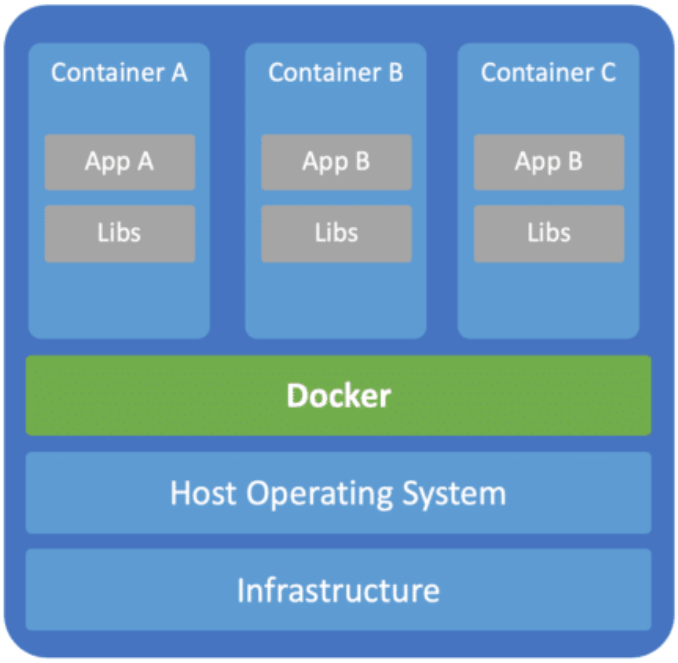

Рисунок 3.7 – Docker система

Дана техніка розгортання залишається однаковою незалежно від того де розгортається додаток для промислової експлуатації, будь то в локальному дата центрі, хмарному провайдері або в гібридному оточенні [19].

В даній роботі також було застосовано технологію контейнеризації. Таким чином в кореневій папці кожного мікросервіса було створено докер

файл, який описує, як саме та з чого повинно створитись докер-зображення. Всі докер файли мають досить схожу структуру, тому нижче показано лише один з них.

```
FROM azul/zulu-openjdk-alpine:11
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

В даному файлі ми вказуємо, яку версію Java потрібно використовувати для додатку, де знаходиться JAR файл, який можна запустити, команду та параметри для запуску файлу. Таким чином, створивши, за допомогою команди “docker build” у відповідній директорії докер-зображення, відправляємо його в докер репозиторій який має назву Docker HUB. В результаті необхідно просто звертатись до репозиторію, де будуть знаходитись зображення докер-контейнерів, які можна скачати та запустити на будь-якій машині з встановленим Docker. Відправка до репозиторію виконується за допомогою команди “docker push”. Після того, як всі компоненти системи відправлено до репозиторію, було використано Docker-compose технологію. Дана технологія дозволяє налаштувати розгортання системи шляхом компонування існуючих докер-зображень та налаштування мережі для їхнього спілкування. Код docker-compose.yml:

```
version: '3'

services:
  zookeeper:
    image: wurstmeister/zookeeper
    container_name: zookeper
    ports:
      - "2181:2181"

  kafka:
    image: wurstmeister/kafka
    container_name: kafka
    ports:
      - "9092:9092"
    environment:
      KAFKA_ADVERTISED_HOST_NAME: kafka
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
    depends_on:
```

```
    - zookeeper

mongo:
  image: mongo
  container_name: mongodb
  ports:
    - "27017:27017"

eureka:
  image: dgkyk/eureka:discoveryserver
  container_name: eureka
  ports:
  - "8761:8761"

pie:
  image: dgkyk/pie:service
  container_name: pie-service
  ports:
  - "8080:8080"
  depends_on:
    - eureka

keeper:
  image: dgkyk/keeper:service
  container_name: keeper-service
  ports:
    - "8081:8081"
  depends_on:
    - eureka

zuul:
  image: dgkyk/zuul:gateway
  container_name: zuul-service
  ports:
  - "8083:8083"
  depends_on:
    - eureka
```

Можна побачити, що для конфігурації відповідного сервісу достатньо вказати джерело, з якого буде братись докер-зображення `image: dgkyk/zuul:gateway` , ім’я яке буде мати контейнер з цим зображенням - `container_name: zuul-service` , порт на якому буде запущено додаток , та порт по якому він буде доступний для звернення зовні - `ports: - "8083:8083"`. Також є можливість вказати на залежності при запуску.

Більшість контейнерів залежать від Eureka контейнеру, запускаючись після того, як буде запущено Eureka server.

Для запуску усіх раніше описаних компонентів достатньо просто виконати команду “docker-compose up”. Результат виконання команди на рис.3.8.

```
Creating network "measurement-system_default" with the default driver
Creating eureka ...
Creating mongodb ...
Creating zookeper ...
Creating keeper-service ...
Creating zuul-service   ...
Creating kafka          ...
Creating pie-service    ...
'Compose: docker-compose.yml' has been deployed successfully.
```

Рисунок 3.8 – Docker-compose

В UI додатку Docker Desktop можна спостерігати статус запущених додатків (рис.3.9).

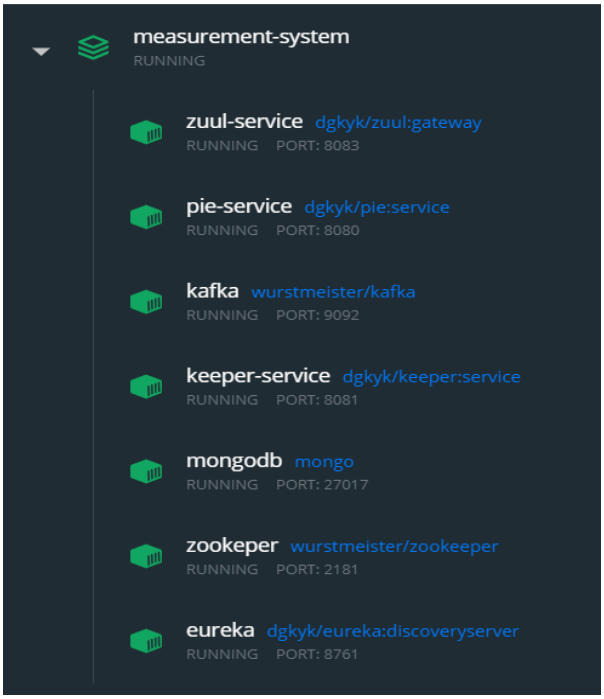

Рисунок 3.9 – Docker Desktop

Висновки

В даному розділі було розроблено мікросервісну архітектуру демонстраційної системи збору експериментальних даних, розроблено вхідний шлюз мікросервісної системи, реалізовано мікросервіси взаємодії з апаратною частиною системи, збереження експериментальних даних та проведено процес контейнеризації системних компонентів за допомогою технологій Docker.

4 РОЗРОБКА КЛІЄНТСЬКОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЕМОНСТРАЦІЙНОЇ СИСТЕМИ ЗБОРУ ЕКСПЕРИМЕНТАЛЬНИХ ДАНИХ НА БАЗІ ФРЕЙМВОРКУ ANGULAR

4.1 Огляд фреймворку Angular

Angular - це платформа та фреймворк для побудови односторінкових клієнтських додатків за допомогою HTML та TypeScript. Angular реалізовано на мові TypeScript. Angular забезпечує основну та додаткову функціональність за допомогою бібліотек TypeScript, які імпортуються у програми користувача. Архітектура Angular спирається на певні фундаментальні концепції. Основними будівельними елементами фреймворку є Angular компоненти, які організовані в NgModules. NgModules збирають відповідний код у функціональні набори. Додаток Angular складається з набору NgModules. Додаток завжди має принаймні кореневий модуль, який необхідно завантажувати, і, як правило, має набагато більше модулів функцій. Компоненти визначають відображення, які являють собою набори елементів екрану, які Angular може обирати та модифікувати відповідно логіки програми та даних. Компоненти використовують служби, які надають певні функціональні можливості та не пов'язані безпосередньо з відображеннями. Постачальників послуг можна вводити в компоненти як залежності, роблячи код модульним, багаторазовим та ефективним.

Модулі, компоненти та послуги - це класи, в яких використовуються декоратори. Ці декоратори визначають свій тип і надають метадані, які вказують Angular, як з ними взаємодіяти. Метадані класу компонентів пов'язують їх із шаблоном, який визначає відображення. Шаблон поєднує звичайний HTML з директивами Angular та розміткою прив'язки, які дозволяють Angular змінювати HTML перед тим, як транслювати його для відображення. Метадані для класу служби надають інформацію, необхідну

Angular, щоб зробити її доступною для компонентів за допомогою інжекції залежностей (DI).

Компоненти програми зазвичай визначають множину відображень, розташованих ієрархічно. Angular надає послугу маршрутизатора, який дозволяє визначити шляхи навігації серед відображень. Маршрутизатор забезпечує складні навігаційні можливості в браузері.

Модулі. Angular модулі NgModules відрізняються від модулів JavaScript та доповнюють їх. NgModule оголошує контекст компіляції для набору компонентів, призначеного для домену програми, робочого процесу або тісно пов'язаного набору можливостей. Для формування функціональних одиниць NgModule може пов'язувати свої компоненти з відповідним кодом, таким як служби. Кожна програма Angular має кореневий модуль, який називається AppModule, який забезпечує механізм завантаження та запускає програму. Додаток, як правило, містить багато функціональних модулів. Як і модулі JavaScript, NgModules можуть імпортувати функціональні можливості з інших NgModules, а також дозволяють експортувати та використовувати власні функціональні можливості іншими NgModules. Наприклад, щоб використовувати службу маршрутизатора у вашому додатку, імпортується модуль Router NgModule.

Розподілення коду за різними функціональним модулями допомагає керувати розробкою складних додатків та подальшим повторним використанням. Крім того, ця техніка дозволяє скористатися перевагами lazy-loading (ледачого завантаження), тобто завантаження модулів за вимогою, щоб мінімізувати кількість коду, який потрібно завантажувати під час запуску.

Компоненти. Кожен Angular-додаток має принаймні один компонент - кореневий компонент, який з'єднує ієрархію компонентів із об'єктною моделлю документа для сторінки (DOM). Кожен компонент визначає клас, який містить дані програми та логіку і пов’язаний із шаблоном HTML, який

визначає відображення, яке показуватиметься у цільовому середовищі. Декоратор @Component() визначає клас як компонент і надає шаблон і відповідні метадані, що стосуються конкретних компонентів.

Шаблони, директиви та прив'язка даних. Шаблон поєднує HTML з Angular розміткою, яка може змінювати елементи HTML до їх відображення. Директиви шаблонів забезпечують логіку програми, а прив'язка розмітки з'єднує дані додатка та DOM. Існує два типи прив'язки даних.

Event binding (прив’язка подій) дозволяє програмі реагувати на вхідні дані користувачів в цільовому середовищі, оновлюючи дані програми.

Property binding (прив'язка властивостей) дозволяє перетворити значення у вигляд HTML.

Перед тим, як генерується відображення, Angular аналізує директиви та визначає синтаксис прив'язки в шаблоні для модифікації елементів HTML та DOM відповідно до даних та логіки програми. Angular підтримує двосторонню прив'язку даних, тобто зміни в DOM також впливають на дані програми.

Шаблони можуть використовувати канали для поліпшення взаємодії з користувачем, перетворюючи значення для їх відображення. Наприклад, можна використати конвеєри для відображення дат та значень валют, які відповідають локалі користувача. Angular забезпечує заздалегідь визначені канали для загальних перетворень, а також можна визначити власні канали.

Сервіси та інжекція залежностей. Для даних або логіки, які не пов’язані з певним видом, і які ви хочете надати компонентам для спільного доступу створюється клас обслуговування. Визначенню класу обслуговування безпосередньо передує декоратор @Injectable(). Декоратор надає метадані, які дозволяють вставляти залежності у клас.

Інжекція залежностей (DI) дозволяє зберегти класи компонентів ненадлишковими та ефективними. Вони не отримують дані з сервера, не

перевіряють введені користувачем дані та не реєструються безпосередньо в консолі. Вони делегують такі завдання сервісам.

Маршрутизація. Angular Router NgModule надає послугу, яка дозволяє визначити шлях навігації між різними станами програми та переглядати ієрархії для додатка. Він створений за зразком правил навігації браузера. Введення URL-адреси в адресний рядок - це перехід браузера на відповідну сторінку. Клік по посиланню на сторінці - це перехід браузера на нову сторінку. Клік на кнопки браузера вперед і назад - це переміщення браузера вперед і назад по історії сторінок.

Маршрутизатор відображає URL-подібні шляхи до відображень замість сторінок. Коли користувач виконує деяку дію, наприклад, натискання на посилання, яке завантажує нову сторінку у браузері, маршрутизатор перехоплює дії браузера та показує або приховує ієрархію відображення. Якщо маршрутизатор визначає, що поточний стан програми вимагає певної функціональності, а модуль, який визначає її не завантажився, маршрутизатор може завантажувати модуль за вимогою.

Маршрутизатор інтерпретує URL-адресу посилання відповідно до правил навігації відображення програми та стану даних. Можна переходити до нових відображень, коли користувач натискає кнопку або вибирає перехід у вікні. Маршрутизатор реєструє дії в історії браузера, тому також працюють кнопки назад і вперед.

Щоб визначити правила навігації, необхідно пов’язати маршрути навігації зі своїми компонентами. Маршрут використовує URL-подібний синтаксис, який інтегрує дані програми, приблизно так само, як синтаксис шаблону інтегрує відображення з даними програми. Потім можна застосувати логіку програми, щоб вибрати, які відображення показувати чи приховувати, у відповідь на введення користувача та правила доступу [20].

4.2 Розробка клієнтського програмного забезпечення системи збору експериментальних даних на базі фреймворку Angular

На базі фреймворку Angular була розроблена клієнтська частина демонстраційної системи збору експериментальних даних. Для розміщення основного контенту був створений відповідний компонент під назвою HomeComponent, в якому знаходиться код TypeScript, CSS стилі даного компонента та HTML сторінка (рис.4.1).

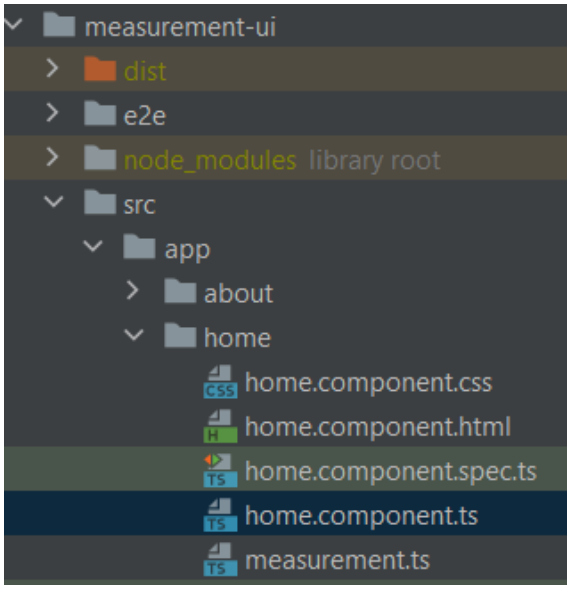

Рисунок 4.1 – Структура проекту

У файлі home.component.ts реалізований основний функціонал клієнтської частини. В методі обробки події ініціалізації `ngOnInit()` відбувається створення об’єктів для відображення графіків та їх налаштувань. Код даного методу наведений нижче:

```
ngOnInit(): void {
    this.temperatureChart = new
CanvasJS.Chart("temperatureChart",{
      exportEnabled: true,
      title:{
        text:"Temperature"
      },
      data: [{
        type: "spline",
```

```
        dataPoints : this.temperaturePoints,
      }]
    });
    this.humidityChart = new CanvasJS.Chart("humidityChart",{
      exportEnabled: true,
      title:{
        text:"Humidity"
      },
      data: [{
        type: "spline",
        dataPoints : this.humidityPoints,
      }]
    });
  }
```

Метод `clickedMarker()` відповідає за обробку події натиснення маркеру для датчика на карті:

```
clickedMarker(lat: number, lng: number) {
    console.log(`clicked the marker: ${lat} | ${lng}`)
    this.getMeasurements(`${lat},${lng}`).subscribe({
      next: data => {
        let jdata: Measurement = JSON.parse(data);
        console.log(jdata);
        this.updateTemperatureChart(jdata.temperature,
jdata.id);
        this.updateHumidityChart(jdata.humidity, jdata.id);
      },
      error: err => console.error(err)
    });
  }
```

При натисненні маркеру на карті відбувається відображення в консолі координат маркеру за допомогою методу `console.log(`clicked the marker: ${lat} | ${lng}`)`. Змінні lat та lng є полями класу Marker та є скороченнями від слів latitude та longitude (широта та довгота). Код для класу marker наведено нижче:

```
interface marker {
  lat: number;
  lng: number;
}
```

Метод, який відповідає за виконання запиту для отримання даних має назву getMeasurements. Код методу:

```
 getMeasurements(location: string): Observable<string> {
    return Observable.create(
      observer => {
        let source = new
EventSource("http://localhost:8081/stream/measurements?location=
" + location);
        source.onmessage = event => {
          this.zone.run(() => {
            observer.next(event.data)
          })
        }
        source.onerror = event => {
          this.zone.run(() => {
            observer.error(event)
          })
        }
      }
    )
  }
```

В даному методі створюється підписник на події, джерелом яких буде канал потоку даних який утворюється за допомогою запиту на адресу Keeper-Service, а саме `http://localhost:8081/stream/measurements?location=`. Таким чином отримуємо дані у вигляді JSON об’єктів.

Методи для оновлення графіків у зв’язку з надходженням нових даних мають назву `updateTemperatureChart` та `updateHumidityChart`. Реалізація наведена нижче:

```
updateTemperatureChart(temperature: number, measurementId:
string) {

    if (this.temperaturePoints.length > 20) {
      this.temperaturePoints.shift();
    }
    this.temperaturePoints.push({
      x: this.temperatureDpsLength,
      y: temperature
    });
    this.temperatureDpsLength++;
    this.temperatureChart.render();
```

```
}

updateHumidityChart(humidity: number, measurementId: string) {

  if (this.humidityPoints.length > 20) {
    this.humidityPoints.shift();
  }
  this.humidityPoints.push({
    x: this.humidityDpsLength,
    y: humidity
  });
  this.humidityDpsLength++;
  this.humidityChart.render();
}
```

Основним завданням цих методів є перевірка кількості записів що відображені. В разі досягнення числа 20, відбувається зсув графіка для демонстрації нових даних, які знаходяться в змінних x та y.

Код CSS стилів та зміст HTML сторінок наведено в лістингах додатку.

Загальний вигляд інтерфейсу користувача клієнтської частини наведено на рис.4.2.

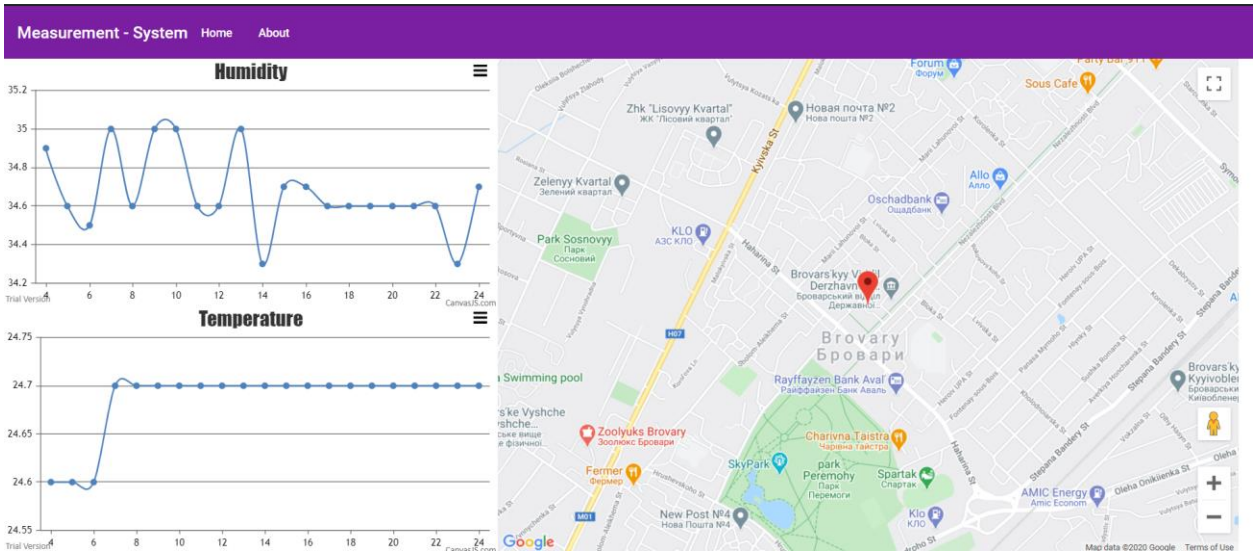

Рисунок 4.2 – Інтерфейс користувача клієнтської частини

Висновки

В даному розділі було проведено огляд сучасного фреймворку Angular для розробки клієнтської частини веб-додатку. На базі фреймворку Angular розроблено програмне забезпечення веб-клієнта. Для датчика з заданими координатами результати вимірювань, що отримуються як JSON об’єкти, відображаються в браузері у вигляді графіків.

5 РОЗРОБКА СТАРТАП ПРОЕКТУ «MEASURING-SYSTEM»

5.1 Опис ідеї проекту

Враховуючи апаратно-програмний потенціал виконаної роботи, в даному розділі розглянуто комерціалізації стартапу. Система збору експериментальних даних на основі мікросервісної архітектури має достатньо велику варіативність у застосуванні та можливість принести користь клієнту. Основні напрямки застосування, зміст ідеї та вигоди для користувача наведені в таблиці 5.1. Також з таблиці можна зрозуміти основні переваги та актуальність системи стартапа.

Таблиця 5.1 – Опис ідеї стартап проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Веб-додаток на базі мікросервісної архітектури з отримання експериментальних даних з сенсору мікрокомп'ютера.	Вимірювання, моніторинг та обробка даних.	Можливість спостереження за встановленими контрольними точками системи вимірювання
	Система безпеки.	Використання для сигналізації перевищення показників.
	Метеорологічні веб-додатки.	Можливість інтеграції отриманої інформації в свої веб-додатки.

Така програмна розробка як мікросервісна система має бути реалізована та побудована таким чином щоб мати можливість масштабування та інтеграції до власного бізнес проекту чи існуючого відповідно. Через численні переваги використаних передових технологій та актуальність вимірюваних даних, є доцільним певна комерційна діяльність в сфері надання послуг. Саме тому пропонується розробка стартап проекту, який має варіативність у застосуванні, можливість до масштабування та інтеграції в існуючі системи та використовує інноваційні технології.

Таким чином одним з розділів магістерської роботи є розробка стартап проекту. Для цього розглядаються рішення для реалізації, актуального на сьогоднішній день проекту який здатен створити достойну конкуренції існуючим аналогам у перспективності, організованості, планом для розвитку ідеї та залучення інвесторів, фінансовому аналізі, аналізі ризиків і можливостей, маркетинговому плануванні.

Таблиця 5.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів				W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	SSI	Solti	Фрілансс			
1.	Вартість розробки	5000	15000	11000	8000	-	-	+
2.	Час розробки	3 місяці	1 місяць	2 місяць	5 місяців	-	+	-
3.	Технології	Передові	Застарілі	Застарілі	Відносно сучасні	-	-	+
4.	Відомість	Відсутня	Велика	Середня	-	+	-	-

Отже виходячи з характеристик наведених в таблиці вище, можна зрозуміти що як мінімум в двох з чотрьох мій проект має переваги та здатен скласти конкуренції на ринку.

Нижче наведена морфологічна таблиця продукту:

Основні параметри	Проміжні рішення				
	1-ше	2-ге	3-тє	4-те	5-те
Мікрокомп’ютер	RaspberryPi	BnanaPi	Arduino	-	Інші
Датчик	DHT22	DHT11	MH-Z19	SHT1	
Архітектура	Моноліт	Мікросервіси	COA	-	Інші

додатку					
База даних	PostgreSQL	MySQL	MongoDB	Redis	Cassandra
Web-Framework	Spring	Grails	Play	Spark	Vaadin
Клієнтська частина	Angular	React	-	-	Інші

Таким чином було вирішено притримуватись напрямку який виділено більш темним кольором. Було обрано найбільш оптимальні складові вимірювальної системи.

5.2 Технологічний аудит проекту

Технологічний аудит — операція об'єктивної оцінки потенціалу інновації як об'єкта комерціалізації. Через те, що комерціалізація технологій — тривалий і дорогий процес, те, перш ніж витрачати чималі тимчасові й фінансові ресурси, необхідно оцінити реальність продажу ідеї або винаходи або їхнє успішне перетворення в ринковий продукт. Таку оцінку можуть провести як самі автори, так і автори із залученням сторонніх експертів.

Проведення аудита авторами інновації має як істотні позитивні сторони, так і не менш істотні негативні.

Позитивні сторони оцінки потенціалу інновації самими авторами:

- глибоке знання своєї ідеї (винаходу);
- широкі знання в даній області.

Негативні сторони:

- внаслідок того, що ідея (винахід) є "дитям", що довго виношувалося й пестувалося, на яке було витрачено багато кваліфікованої праці, часу й чинностей, авторам складно адекватно оцінювати свою ідею й порівнювати неї з аналогами;

- у чинність своєї професійної спеціалізації автори інновації поверхово оцінюють вартість комерціалізації й ринкові перспективи;
- автори переоцінюють можливості свого колективу по комерціалізації інновації.

Проведення аудита із залученням сторонніх експертів переважніше, тому що, хоча експерт і не знає досконально пропонований винахід (в оцінці технічної сторони інновації експертам допоможуть автори), але може неупереджено порівнювати його з аналогами, знаходити можливості різних практичних застосувань, оцінювати вартість його комерціалізації й ринкові перспективи. При проведенні незалежного аудита між авторами й експертами повинне укладатися договір про конфіденційність[21].

В цьому розділі розглянуто технічні особливості, спектр можливих технічних рішень. Таблиця 5.3 використана для аналізу здійсненності проекту.

Таблиця 5.3 Технологічна здійсненність ідеї проекту

№ п/п	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1.	Демонстраційна система збору експериментальних даних	Сучасні датчики	+	+
		Застарілі датчики	+	+
		Сучасні веб-фреймворки	+	+
		Додаток	+	+
		Мікросервісна архітектура	+	+
		Монолітна архітектура	+	+
Обрана технологія реалізації ідеї проекту: Система збору експериментальних даних на основі мікросервісної архітектури.				

Використання одноплатного комп’ютера з сенсором разом з веб-системами зустрічалось і раніше, але актуальность на ринку полягає у використанні максимально ефективних рішень. Саме з цією метою в даному проекті використовуються останні технології розробки програмного забезпечення, актуальна апаратно-верверна частина та протоколи передачі

даних. Ці технології та інструменти доступні та в повній мірі можуть бути використані в даному стартап проекті.

5.3 Аналіз ринкових можливостей запуску стартап-проекту

Визначення основних ризиків та можливостей, які можна очікувати від ринку відіграють дуже важливу роль, оскільки дозволяють спроектувати напрями розвитку проекту з урахуванням всіх нюансів ринкового положення, зацікавленості клієнтів та пропозицій конкурентів. Таким чином має сенс використання цих знань під час ринкового впровадження проекту(табл. 5.4).

Таблиця 5.4 Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	300
2	Загальний обсяг продаж, грн/ум.од	5000000
3	Динаміка ринку (якісна оцінка)	Стабільно зростаюча
4	Наявність обмежень для входу (вказати характер обмежень)	Відсутні
5	Специфічні вимоги до стандартизації та сертифікації	Відсутні
6	Середня норма рентабельності в галузі (або по ринку), %	15%

Розглянувши таблицю можна зрозуміти що даний ринок має безліч конкурентів, а сам по собі є зростаючим. Інноваційні компанії та компанії гіганти становлять основну загрозу.

Далі визначені потенційні групи клієнтів, їх особливості. Також сформовано приблизний перелік вимог до вимірювальних систем для кожної групи.

Таблиця 5.5 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до послуги
1	Моніторинг та вимірювання інформації для мас	Глобальні системи, персональні системи	Швидкодія, актуальність даних	Веб-додаток з великою пропускної здатністю
2	Протипожежний контроль	Системи	Надійність, точність	Безперебійна робота
4	Веб додатки для смартфонів	Звичайні люди, розробники операційних систем та додатків	Вартість	Доступність

Очевидно, що даний проект покриває дуже широкий спектр потреб в різних сферах, таким чином збільшуючи шанси на успіх. Такі відмінності як вартість та швидкодія є відносно простими в реалізації, на відміну від надійності, так як це потребую дадткових витрат на реалізацію та підтримку інших реплік компонентів системи.

В таблиці 5.6 наведені фактори, які можуть в значній мірі уповільнити розвиток стартап проекту. Для нейтралізації цих факторів було запропоновано відповідні дії, які спрямовані на покращення ситуації та налагодження процесу виробництва продутку. Таким чином, надійність стартапу знаходиться на високому рівні.

Таблиця 5.6 Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1.	Конкуренти	Можливість монопольної ситуації в сфері застосування	Притримування більш привабливої цінової політики
2.	Ринок праці	Труднощі при наборі штату	Організація бонусної системи та пільгових умов праці
3.	Застарілість технології	Поява більш інноваційних технологій	Відносно проста заміна компонентів системи на актульну реалізацію

4.	Не знання маркетингу	Невідомі шляхи зацікавлення клієнтів.	Залучення експертів та використання відомих ресурсів для маркетингу
5.	Організація виробництва	Неналагоджений процес виробництва програмного продукту та закупки апаратної части	Використання послуг логістики, та якісний відбір спеціалістів

В одночас з ризиками існують і фактори можливостей, що наведені в таблиці 5.7.

Таблиця 5.7. Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1.	Притік клієнтського спектру	Збільшення кількості клієнтів	Залучення додаткових спеціалістів до основної команди
2.	Потреба у високій пропускній здатності	Необхідність обробки дуже великого числа інформації.	Використання хмарних сервісів та технологій
3.	Постійно зростаюча актуальність у вимірюваних даних	Збільшення кількості клієнтів	Розширення сигменту користувачів
4.	Оновлення існуючих технологій	Покращення роботи системи	Оновлення до останньої версії технологій
5.	Привабливість системи на базі сучасних технологій	Збільшення кандидатів у штат	Підвищення професіональності команди розробників

Порівнюючи ризики з перерахованими можливостями, можна дійти висновку що ризики нейтралізовуються можливостями стартапу[21].

Одним з важливих моментів появи на ринку є розуміння своїх конкурентів, їх можливостей та швидкості адаптації. Також необхідно враховувати їх можливий вплив на ринок, що наведено в таблиці 5.8.

Таблиця 5.8. Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1.Чиста конкуренція	Жорстка боротьба за клієнта	Актуальні та бонусні пропозиції
2. Національний рівень	Локація і надання послуг конкурентів по всій	Оригінальність та швидкодія

	країні	
3. Міжгалузева конкурентність	Багато галузей використання даних технологій та систем	Маркетинг
4.Товарно-видова конкурентність	Системи з однаковим принципом роботи	Іноваційна реалізація
5. Нецінова конкуренція	Різна якість продукції	Вдосконалення якості продукції та технології виробництва
6. Марочна конкуренція	Відомості марки конкурентів	Набір додаткової брендинг-команди

У даній сфері присутня велика конкуренція. Ринок налічує багато компаній з довготривалими контрактами, конкуренти стаж роботи яких більш як 10 років у цій сфері. Через великі можливості середнього прибутку, дуже жорстка боротьба за клієнта. В таблиці були наведені можливі дії компанії для поліпшення конкурентного становища.

Після аналізу конкуренції нижче наведений більш детальний аналіз умов конкуренції в галузі (табл.5.9).

Таблиця 5.9 Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	SSI	Solti	TexasInstruments	Everest	Стаціонарні системи
Висновки:	Один з головних конкурентів.	Конкурент в майбутньому	Виробник мікросхем	Надає обладнання споживачам	Системи які використовуються біля користувача

Вказані компанії на даний момент дуже сильно закріплені на ринку. З таблиці видно можливості для конкуренто-спроможності стартапу.

В таблиці 5.10 нижче наведено обґрунтування факторів конкурентоспроможності.

Таблиця 5.10 Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Час розробки	Замовників цікавить найбільш оптимальний час розробки
2	Ціна	Ціна продукту нижче середньої вартості аналогів
3	Технології	Використання проектом найновіших технологій
4	Підтримка	Конкуренти надають такі послуги
5	Масштабованість	Можливість розширення системи є важливим фактором для клієнтів

Таблиця 5.11. Порівняльний аналіз сильних та слабких сторін

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з FiOTech						
			–3	–2	–1	0	+1	+2	+3
1	Ціна	18						+	
2	Час розробки	19				+			
3	Технології	17						+	
4	Підтримка	16	+						
5.	Гнучкість	15				+			

Фінальним етапом ринкового аналізу можливостей впровадження проекту є складання SWOT-аналізу (матриці аналізу сильних (Strength) та слабких (Weak) сторін, загроз (Troubles) та можливостей (Opportunities), пов'язаних з його здійсненням.

SWOT - аналіз стартап-проекту представлений в таблиці 5.12.

Таблиця 5.12. – SWOT- аналіз стартап-проекту

Сильні сторони: 1. Ціна 2. Гнучкість 4. Сучасність технологій	Слабкі сторони: 1.Час розробки 2. Слабкі можливості підтримки
Можливості: Притік клієнтського спектру Потреба у високій пропускній здатності Зростаюча актуальність вимірюваних даних Оновлення існуючих технологій Привабливість системи на базі сучасних технологій	Загрози: Конкуренти Ринок праці Не знання маркетингу Застарілість технологій Організація виробництва

Перелік ринкових загроз та ринкових можливостей складений базуючись на аналізі факторів загроз та факторів можливостей маркетингового середовища. Ринкові загрози та ринкові можливості є наслідками (прогнозованими результатами) впливу факторів, і, на відміну від них, ще не є реалізованими на ринку та мають певну ймовірність здійснення. Наприклад: домінація компаній гігантів – фактор загрози, на основі якого можна зробити прогноз щодо спаду попиту на веб-систему від нашої компанії, особливо в верхній ціновій категорії, це призведе до посилення значущості функціональних можливостей веб-системи при його продажі, щоб максимально розширити коло можливих клієнтів та зберегти розробку веб-систем на рентабельному рівні.

Таблиця 5.13. Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Стратегія нейтралізації ринкових загроз сильними сторонами стартапу	Середня	2 місяці
2	Стратегія компенсації слабких сторін стартапу наявними ринковими можливостями	Середня	6 місяців
3	Стратегія виходу з ринку	-	2 місяці

Отже, з зазначених альтернатив сильними сторонами стартапу є стратегія компенсації ринкових загроз**.**

5.4 Розроблення ринкової стратегії проекту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів.

Споживачами проекту обрано підприємства, що використовують, або можуть використовувати у своїй роботі вимірювальну систему з веб-додатками. Так як проект зосереджується не на одному сегменті а на декількох, обрано стратегію диференційованого маркетингу. Для роботи в обраних сегментах ринку було сформувано базову стратегію розвитку (табл. 5.14).

Таблиця 5.14. Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
	Промислові компанії	Так	Вище середнього	Велика	Просто
	Компанії розважального сегменту	Так	Середній	Середня	Просто
	Інжинирингові компанії	Так	Низький	Низька	Складно
Цільовими групами обрано: промислові компанії та компанії розважальної сегменту. Під час аналізу цільових груп потенційних споживачів було вирішено що проект буде працювати із промисловими компаніями та розважальними.					

Базуючись на проведеному аналізі потенційних груп споживачів, було обрано такі цільові групи як промислові компанії та компанії розважального сегменту. Дані цільові групи мають найвищій попит на продукцію типу вимірювальних веб-систем та мають відносно низький поріг входу на ринок. Недивлячись на високий рівень конкуренції, вибір вважаю виправданим.

Для роботи в обраному сегменті ринку необхідно сформувати базову стратегію розвитку(табл.5.15).

Таблиця 5.15 – Базова стратегія розвитку

Обрана альтернатива розвитку проекту	Стратегія спеціалізації (передбачає концентрацію на потребах одного цільового сегменту, без прагнення охопити увесь ринок. Мета тут полягає в задоволенні потреб вибраного цільового сегменту краще, ніж конкуренти. Така стратегія може спирається на лідерство по витратах у рамках сегменту веб-систем для розважальної сфери. Проте низька ринкова доля у разі невдалої реалізації стратегії може істотно підірвати конкурентоспроможність компанії.)
Стратегія охоплення ринку	Стратегія повного охоплення ринку (компанія прагне задовольнити потреби ринку в цілому. Ця стратегія може бути реалізована шляхом виготовлення сімейства універсальних веб-систем).
Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Покращення та здешевлення виробництва за рахунок масовості, підтримка веб-системи після продажу.
Базова стратегія розвитку	Стратегія концентрованого зростання (Дана стратегія передбачає зміну продукту і (або) ринку. У разі дотримання стратегії компанія поліпшує веб-додаток або починає розробку нового, не змінюючи цільовий напрям. Щодо ринку, компанія шукає можливості покращення свого становища на нинішньому ринку або ж переходу на новий).

В таблиці вище зазначено обрані стратегії розвитку. У вигляді базової стратегії було обрано стратегію концентрованого зростання, з постійним оновленням веб-додатків. Альтернативною стратегією є фокусування на конкретному сигменті, а саме розважальну.

Наступним кроком є вибір стратегії конкурентної поведінки (Таблиця 5.16).

Таблиця 5.16 – Вибір стратегії конкурентної поведінки

Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
Ні	Так	Ні	Стратегія наслідування лідеру

Стратегія лідерства за витратами передбачає, що компанія за рахунок факторів внутрішнього і/або зовнішнього середовища може забезпечити

більшу, ніж у конкурентів маржу між собівартістю товару і середньою ціною на ринку (чи ціною головного конкурента). Зокрема, ця стратегія передбачає, що за рахунок великих можливостей по об'ємах продажу товарів (укладених контрактів на постачання) та продуктивності, підприємство може досягти менших витрат.

Базуючись на вимогах споживачів обраного сегменту до постачальника і продукту, залежності від стратегії розвитку і стратегії конкурентної поведінки розроблено стратегію позиціювання, що передбачає формування ринкової позиції, за яким клієнти мають ідентифікувати проект

Таблиця 5.17. Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
	Низька ціна, універсальність, сучасність.	Стратегія концентрованого зростання	Сучасність рішень, швидкодія, ціна	Універсальність. Технології. Ціна.

Результатом даного підрозділу є система рішень щодо ринкової поведінки компанії, вона визначає в якому напрямі буде працювати компанія на ринку.

5.5 Розробка маркетингової програми стартап-проекту

Маркетинг стартапу – сукупність інструментів та заходів із ознайомлення цільової аудиторії з ідеєю продукту /послуги, просування ідеї та безпосередньо товару/послуги на ринок з метою підвищення зацікавленості та прихильності потенційних споживачів, залучення споживачів, інвесторів тощо[22].

Проблема в тому, що спочатку ідея відчувається автором тільки на рівні підсвідомості. Незважаючи на це йому, здається, що він розуміє свою ідею на всі 100%, але це зовсім не так. Через підсвідомість рівень розуміння ідеї не перевищує 1-2%. Ось тут і виникає головна проблема, адже тільки зрозумілу на 100% ідею можна донести інвесторам і потенційним споживачам. Якщо споживач або інвестор отримує лише 1-2% ідеї, він не може уявити її у себе «в голові» і не може її зрозуміти. Він зрозуміє ідею тільки в тому випадку, якщо отримає її на 100%[23,24].

На рисунку 5.1 наведені основні етапи маркетингу стартапу

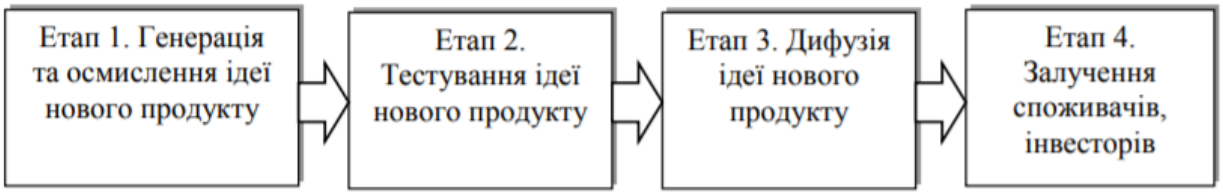

Рисунок 5.1 – Основні етапи маркетингу стартапу

Для створення ефективної маркетингової програми першим кроком є формування маркетингової концепції товару, який отримає споживач. Для цього у таблиці 5.18 підсумовано сумарні результати попереднього аналізу конкурентоспроможності товару.

Таблиця 5.18 – Визначення ключових переваг концепції потенційного товару

Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
Універсальний веб-додаток	Моніторинг виміряних даних в режимі реального часу	Сучасність використаних технологій, мікросервісна архітектура веб-додатку

Проаналізувавши ринок, виконавши детальний аналіз потреб клієнтів, та дослідивши конкурентний продукт, можна зробити висновок що для вирішення проблем моніторингу виміряних даних в режимі реального часу підходить розроблений універсальний веб-додаток з використанням сучасних

технологій та мікросервісної архітектури додатку за використанням мікрокомп'ютера.

Таблиця 5.19. Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
I. Товар за задумом	Система збору експериментальних даних на основі мікросервісної архітектури.		
II. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	1.Масштабованість	Нм	Тх
	2.Компонентність	М	Тх
	3.Швидкодія	Нм	Тх
	4.Підтримка масового використання	Нм	Тх
	5.Довговічність (немає строку давності)	Нм	Тх
	Якість: відповідає нормам ДСТУ 2844-94		
	Пакування: Мікроконтролер RaspberryPi з датчиком DHT22(AM2302), програмне забезпечення		
	Марка: “KPI Soft” зареєстрована ТМ. Під ТМ “KPI Soft” випускаються сучасні програмні та технічні рішення.		
III. Товар із підкріпленням	До продажу: Консультація з можливості інтегрування		
	Після продажу: Підтримка існуючих версій товару		

Розклавши модель товару на три рівні, продукт за задумом має п’ять технічних особливостей, одна з яких є матеріальною, а інші ні. Нормування якості товару проводиться згідно з ДСТУ 2844-94. До складу товару входить мікроконтролер RaspberryPi з датчикамом DHT22(AM2302) та програмне забезпечення. Також на третьому рівні наведені переваги для споживача.

Наступним кроком є визначення цінових меж, якими необхідно керуватися при встановленні ціни на потенційний товар, це передбачає аналіз цін товарів конкурентів, та доходів споживачів продукту (табл. 5.20).

Таблиця 5.20. Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1	60000 грн	20000 грн	150000 грн	9000/19900 грн
2	1800 грн	1500 грн	15000 грн	2000/2200 грн
3	10000 грн в місяць	10000 грн в місяць	200000 грн в місяць	5000/10000 грн в місяць

Ціна товарів замініків надто висока щоб враховувати при створенні цінової політики. Середня ціна на продукти аналоги складає 20000 грн, в той же час доходи потенційних компаній покупців починається від 150000 грн. Також розглянутий можливість виробництва систем для розважального сегменту, але отримані ціни не враховують вартості програмного забезпечення та відображають ціну обладнання. В третьому випадку наведені місячні витрати на підтримку веб-додатку.

Таблиця 5.21. Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Повторна закупівля	Організація обороту товару	1	Дистрибуція за допомогою посередника
2	Нова закупівля	Доставка товару у поштове відділення	0	Дистрибуція через Інтернет
3	Клієнти в невідомих або на ризикових ринках	Формування клієнтської бази; Організація попиту і стимулювання збуту;	1-3	Дистрибуція за допомогою посередника

За допомогою аналізу було виявлено два найбільш вдалих метода системи збуту. Відмінність нової та повторної закупівлі полягає в тому, що для повторної закупівлі очікується значне збільшення кількості продукції. Нова закупівля проводиться через інтернет, а повторна за допомогою посередника – транспортну компанію. Також було розглянуто функції збуту для клієнтів в невідомих або на ризикових ринках.

Останньою складової маркетингової програми є розробка концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів (табл. 22).

Таблиця 5.22. Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Організації, які користуються товарами та послугами конкурентів	Електронна пошта, телефон, конференції, веб-пошук.	Розсилка листів актуальних пропозицій, ділові зустрічі з метою обговорення потенційної співпраці.	Звернути увагу на переваги продукту та його сучасність.	Демонстрація потенційного зросту прибутків за рахунок використання нашої системи.
2	Організіції яким необхідна розробка нових веб-додатків	Електронна пошта, телефон, конференції, веб-пошук.	Розсилка листів актуальних пропозицій, ділові зустрічі з метою обговорення потенційної співпраці.	Демонстрація вигідності цінової політики в порівняні з конкурентами; Звернути увагу на переваги продукту та його сучасність.	Порівняльна діаграма.

Проаналізовані концепції, дадуть змогу пободови базису для розвитку комунікації між компанією та потенційними клієнтами. Основним напрямком вважається залучення клієнтської бази конкурентів. Так як на ринку спостерігається тендендія появи нових компаній, необхідно вчасно знаходити шляхи та вигідні умови для співпраці.

5.6 Ефективність стартап проекту

Оскільки стартап є інноваційним проектом, до нього можна застосувати існуючі показники оцінки ефективності такого проекту.

Для проведення розрахунків ефективності за проектом необхідно[21]:

1) визначити статті доходів, що визначаються бізнес-моделлю стартапу і прогнозованими обсягами продажів;
2) розбити витрати за проектом на постійні та змінні.

Дохід:

- Продаж мікроконтролерних систем.
- Продаж програмного забазпечення.
- Підтримка систем

До постійних витрат відносяться ті витрати, загальна сума яких за певний час не залежить від кількості виготовленої продукції.

- Оренда приміщень.
- Реклама
- Зарплата постійних працівників компанії (інженери, маркетологи, менеджери)

Змінні витрати представляють собою витрати, загальна сума яких за певний час залежить від обсягу виготовленої продукції.

- Закупка обладнання (мікроконтролери та давачі) необхідних для виготовлення комплексного обладнання;
- Витрати на електроенергію;
- Зарплата робочих;

При отриманні необхідного фінансування ми будем мати наступну фінансову ситуацію, при розробці веб-системи:

Постійні витрати:

- 7 000 грн/міс - Оренда приміщень.

- 1 000 грн/міс - Реклама
- 50 000 грн/міс - Зарплата постійних працівників компанії (інженери, маркетологи, менеджери)

Змінні витрати на виробництво одного комплексного обладнання:

- 2 000 грн - мікропроцессор;
- 12 000 грн - зарплата робочих;
- 1000 грн - давачі;
- Дохід:
- 25 000 грн - відпускна вартість веб системи + пдв 5 000 грн

Обсяги виробництва продукції на перші 5 місяців наведено в таблиці 5.23.

Таблиця 5.23 – Обсяги виробництва продукції

Показник	Значення по місяцях, тис. грн.				
	1	2	3	4	5
Загальна потреба в продукції, шт.	5	10	15	20	25
Ціна одного набіру макроконотролер+здавачі + ПЗ тис грн.	25	25	25	25	25
Річні обсяги випуску в вартісних показниках (тис. грн.)	125	250	375	450	625

З’ясуємо витрати, що потрібні для реалізації поточної діяльності за проектом (табл.5.24).

Таблиця 5.24 – Виробничі витрати

№ з/п	Стаття витрат	Сукупні витрати за період місяців, тис. грн.				
		1	2	3	4	5
1	Загальні витрати, оплата праці штату	50	50	50	50	50
1.1	Оренда та утримання приміщень, обладнання	20	20	20	20	20
1.2	Збут, реклама та просування	1	1	1	1	1
2	Матеріальні ресурси (сировина, комплектуючі)	52,5	94,5	147	210	262,5
3	Оплата праці штату	7,5	13,5	21	30	37,5
Разом:		131	179	239	311	371

Точка беззбитковості відображає обсяг виробництва інноваційної продукції, при досягненні якого виручка від реалізації покриває сумарні витрати на її виробництво. Розрахунок точки беззбитковості року проводиться за формулою:

$$Тб = \frac{C}{P-V} \quad (5.1)$$

де С – постійні витрати на весь обсяг продукції (ті, які не залежать від обсягу виробництва продукції – загальногосподарські витрати та витрати на оплату праці); Р – ціна одиниці продукції;

V – змінні витрати на одиницю продукції.

$$Тб=C/(P-V)=71/(25-13)=5,9 \text{ од.} \quad (5.2)$$

Висновки

Системи збору даних на основі мікросервісної архітектури набувають все більшої популярності, в результаті чого розширюються сфери їх застосування. Проблемою сучасних систем є висока вартість та вузька направленість використання даних систем, тому актуальним є розробка максимально дешевої та універсальної системи.

Використання одноплатного комп’ютера з сенсором разом з веб-системами зустрічалось і раніше, але актуальность на ринку полягає у використанні максимально ефективних рішень. Саме з цією метою в даному проекті використовуються останні технології розробки програмного забезпечення, актуальна апаратно-серверна частина та протоколи передачі даних. Для цього необхідно використовувати сучасні технології, які зможуть заюезпечити конкурентоспроміжність на сучасному ринку.

Даний стартап-проект може бути реалізований за допомогою однієї людини, але організувавши команду середнього рівня кваліфікації можна досягти результатів значно швидше, таким чином використати виграний час для посилення маркетингової компанії продутку.

Визначено загальні напрями використання та проаналізовано ринкові можливості щодо реалізації бізнес-проекту. Визначено, що сильні сторони стартапу мають переваги над слабкими. Проведено аналіз конкурентних можливостей. В результаті аналізу розроблено ринкову програму, яка враховує систему збуту, концепції товару, концепції маркетингових комунікацій та ціноутворення. За допомогою цього аналізу буде реалізовано проект, та його альтернативу ринкової поведінки.

Для стратегії розвитку стартапу було обрано стратегію концентрованого зростання. Це стратегія, яка передбачає зміну продукту. При дотриманні стратегії компанія покращує веб-додаток або починає розробку нового, не змінюючи призначення, а лиш покращуючи його характеристики та масштабованість. Щодо ринку, то компанія знаходиться в пошуках можливості покращення позицій на існуючому ринку або ж переходу на новий.

В результаті проведеного аналізу потенційних груп споживачів, було обрано наступні цільові групи : промислові компанії та компанії розважального сегменту. У цих цільових групах спостерігається найвищий попит на продукцію веб-додатків та мають невеликий поріг входу на ринок.

Стартап проект вважається перспективним, так як кількість переваг більша ніж ризиків. Було досліджено можливість комерціалізації проекту на ринку. Спостерігається значний попит на стрімко зростаючому ринку. Розроблений веб-додаток є перспективним для реалізації у виробництві, так як він є легко масштабованим, дешевим та універсальним в порівнянні з аналогами на ринку.

ВИСНОВКИ

В ході виконання дисертації було створено апаратно-програмне забезпечення демонстраційної системи збору експериментальних даних на основі сучасних Java фреймворків та мікросервісної архітектури.

Для реалізації апаратної частини демонстраційної системи було використано датчик температури та вологості DHT22 (AM2302). Було розроблено апаратно-програмне забезпечення рівня збору даних на базі одноплатного комп'ютера Raspberry Pi. Організовано бездротове з'єднання функціональних елементів системи. Програмне забезпечення для вузла збору та передачі даних розроблено на мові програмування Python з використанням протоколу HTTP.

Було розроблено серверне програмне забезпечення на базі фреймворку Spring WebFlux та нереляційної бази даних MongoDB. Для реалізації мікросервісної архітектури використано сучасний набір технологій для створення розподілених веб-систем Netflix OSS, що включає сервер реєстрації мікросервісів Eureka та маршрутизатор/серверний балансувальник Zuul. Проведено процес контейнеризації системи та оточуючого середовища за допомогою програмного забезпечення Docker.

Для розробки програмного забезпечення клієнтської частини системи було використано сучасний фреймворк Angular та мову програмування TypeScript.

Особливістю програмного забезпечення розробленої системи є можливості масштабування, розширення та пристосованість до внесення змін шляхом заміни окремих компонентів (сервісів) без втрати функціональності системи. Це дозволило створити систему, яка може бути легко модифікована та розгорнута за допомогою хмарних технологій.

Мікросервісна архітектура забезпечила низьку зв'язаність елементів системи та надала широкі можливості для її масштабування та модифікації.

Розроблено стартап проект, на базі якого зроблено висновок, що реалізована система є перспективною для впровадження у виробництво, оскільки вона дешева та легко масштабована в порівнянні з аналогами на ринку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Martin L. Abbott. The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise. – Addison-Wesley Professional, 2015. – 624 с.
2. Хабрахабр, Микросервисы (Microservices). – https://habrahabr.ru/post/249183/
3. Martin Fowler, Microservices. – https://martinfowler.com/articles/microservices.html
4. Vikram Murugesan, Microservices Deployment Cookbook. – Packt Publishing, 2017. – 378 с.
5. Sam Newman. Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, 2015. – 280 с.
6. Sourabh Sharma, Rajesh RV, David Gonzalez. Microservices: Building Scalable Software. – Packt Publishing Limited, 2017. – 919 с.
7. Rajesh RV, Spring Microservices. – Packt Publishing, 2016. – 436 с.
8. REST APIs - https://en.wikipedia.org/wiki/Representational_state_transfer
9. REST: простым языком - https://medium.com/@andr.ivas12/rest-%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%8B%D0%BC-%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%BC-90a0bca0bc78
10. Идемпотентность - http://www.restapitutorial.ru/lessons/idempotency.html
11. https://3d-diy.ru/wiki/arduino-platy/obzor-plat-raspberry-pi/
12. https://nag.ru/articles/reviews/104595/obzor-tehnologii-wi-fi.html
13. http://robocraft.ru/blog/electronics/3162.html
14. https://freehost.com.ua/faq/wiki/chto-takoe-ssh/
15. Eureka Server - https://www.tutorialspoint.com/spring_boot/spring_boot_eureka_server.htm#:~:text=Eureka%20Server%20is%20an%20application,also%20known%20as%20Discovery%20Server.

16. API Gateway - https://freecontent.manning.com/the-api-gateway-pattern/
17. Zuul API - https://medium.com/@kirill.sereda/spring-cloud-netflix-zuul-api-gateway-%D0%BF%D0%BE-%D1%80%D1%83%D1%81%D1%81%D0%BA%D0%B8-c1e819f042e1
18. MongoDB - https://devcolibri.com/краткое-знакомство-с-mongodb/
19. Docker, Docker Documentation – https://docs.docker.com/
20. Angular documentation – https://angular.io/guide/architecture
21. РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ © Гавриш Олег Анатолійович, Солнцев Сергій Олексійович, Дергачова Вікторія Вікторівна, Зозульов Олександр Вікторович, Юдіна Наталія Володимирівна, Царьова Тетяна Олександрівна, Бояринова Катерина Олександрівна, Кравченко Марина Олегівна, Жигалкевич Жанна Михайлівна, Київ НТУУ «КПІ ім. Ігоря Сікорського» 2016.
22. Орлова Н. Инвестиционная презентация стартапа. Стартап от А до Я. InnMind. – Выпуск № 16: веб-сайт. URL: https://innmind.com/articles/825
23. РОЗРОБКА СТАРТАП ПРОЕКТІВ - https://ela.kpi.ua/bitstream/123456789/29447/1/Rozrobka_startup-proektiv_Konsp.lekts.pdf
24. Технологічний аудит інноваційної продукції - http://market.avianua.com/?p=3307

ДОДАТОК А. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

А.1 Програмне забезпечення рівня збору даних

А.1.1 Основний Python скрипт humidity.py

```
import Adafruit_DHT
import requests
import time
import json

DHT_SENSOR = Adafruit_DHT.DHT22
DHT_PIN = 4

while True:
    humidity, temperature = Adafruit_DHT.read_retry(DHT_SENSOR,
DHT_PIN)

    if humidity is not None and temperature is not None:
        print("Temp={0:0.1f}*C
Humidity={1:0.1f}%".format(temperature, humidity))
        url = 'http://192.168.0.103:8083/pie/measurements'
        requests.post(url, json={"temperature": temperature,
"humidity": humidity, "location": "50.5124138,30.7913499"})
    else:
        print("Failed to retrieve data from humidity sensor")
```

А.1.2 Скрипт для запуска Python скрипта при включенні системи launcher.sh

```
#!/bin/sh
#launcher.sh

sudo python3 /home/pi/humidity.py
```

А.2 Програмне забезпечення серверної частини системи

А.2.1 Eureka server

А.2.1.1 Основний клас EurekaApplication.java

```
package com.alex.eureka;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.cloud.netflix.eureka.server.EnableEurekaServ
er;
```

```
@SpringBootApplication
@EnableEurekaServer
public class EurekaApplication {

    public static void main(String[] args) {
        SpringApplication.run(EurekaApplication.class, args);
    }

}
```

A.2.1.2 Файл конфігурації application.yaml

```
server:
  port: ${port:8761}
spring:
  application:
    name: eureka

eureka:
  client:
    register-with-eureka: false
    fetch-registry: false
```

A.2.1.3 Файл збірки build.gradle

```
plugins {
    id 'org.springframework.boot' version '2.3.3.RELEASE'
    id 'io.spring.dependency-management' version
'1.0.10.RELEASE'
    id 'java'
}

group = 'com.alex'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '11'

configurations {
    compileOnly {
        extendsFrom annotationProcessor
    }
}

repositories {
    mavenCentral()
}

ext {
    set('springCloudVersion', "Hoxton.SR7")
}

dependencies {
```

```
    implementation 'org.springframework.boot:spring-boot-
starter-webflux'
    implementation 'org.springframework.cloud:spring-cloud-
starter-netflix-eureka-server'
    compileOnly 'org.projectlombok:lombok'
    developmentOnly 'org.springframework.boot:spring-boot-
devtools'
    annotationProcessor 'org.projectlombok:lombok'
    testImplementation('org.springframework.boot:spring-boot-
starter-test') {
        exclude group: 'org.junit.vintage', module: 'junit-
vintage-engine'
    }
    testImplementation 'io.projectreactor:reactor-test'
}

dependencyManagement {
    imports {
        mavenBom "org.springframework.cloud:spring-cloud-
dependencies:${springCloudVersion}"
    }
}

test {
    useJUnitPlatform()
}
```

A.2.1.4 Файл докеру Dockerfile

```
FROM azul/zulu-openjdk-alpine:11
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

A.2.2 Zuul-Service

A.2.2.1 Основний клас ZuulServiceApplication.java

```
package com.alex.zuul;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;


@EnableZuulProxy
@EnableEurekaClient
@SpringBootApplication
public class ZuulServiceApplication {
```

```
    public static void main(String[] args) {
        SpringApplication.run(ZuulServiceApplication.class,
args);
    }

}
```

А.2.2.2 Файл конфігурацій application.yaml

```
spring:
  application:
    name: zuul-service
server:
  port: 8083
eureka:
  client:
    serviceUrl:
      defaultZone: ${EUREKA_URI:http://localhost:8761/eureka}
  instance:
    preferIpAddress: true

zuul:
  routes:
    pie-service:
      path: /pie/**
      service-id: pie-service
    keeper-service:
      path: /keeper/**
      service-id: keeper-service
```

А.2.2.3 Файл збірки build.gradle

```
plugins {
    id 'org.springframework.boot' version '2.3.4.RELEASE'
    id 'io.spring.dependency-management' version
'1.0.10.RELEASE'
    id 'java'
}

group = 'com.alex'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '11'

repositories {
    mavenCentral()
}

ext {
    set('springCloudVersion', "Hoxton.SR8")
}

dependencies {
```

```
    implementation 'org.springframework.boot:spring-boot-
starter-webflux'
    implementation 'org.springframework.cloud:spring-cloud-
starter-netflix-zuul'
    implementation 'org.springframework.cloud:spring-cloud-
starter-netflix-eureka-client'
    implementation 'org.springframework.boot:spring-boot-
starter-web'
    compileOnly 'org.projectlombok:lombok'
    developmentOnly 'org.springframework.boot:spring-boot-
devtools'
    testImplementation('org.springframework.boot:spring-boot-
starter-test') {
        exclude group: 'org.junit.vintage', module: 'junit-
vintage-engine'
    }
    testImplementation 'io.projectreactor:reactor-test'
}

dependencyManagement {
    imports {
        mavenBom "org.springframework.cloud:spring-cloud-
dependencies:${springCloudVersion}"
    }
}

test {
    useJUnitPlatform()
}
```

A.2.2.4 Файл докеру Dockerfile

```
FROM azul/zulu-openjdk-alpine:11
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

A.2.3 Pie-Service

A.2.3.1 Основний файл PieApplication.java

```
package com.alex.pie;

import com.alex.pie.config.RibbonConfiguration;
import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.cloud.client.loadbalancer.LoadBalanced;
import
org.springframework.cloud.netflix.eureka.EnableEurekaClient;
```

```
import org.springframework.cloud.netflix.ribbon.RibbonClient;
import org.springframework.context.annotation.Bean;
import
org.springframework.web.reactive.function.client.WebClient;

@EnableEurekaClient
@SpringBootApplication
@RibbonClient(name = "keeper-service", configuration =
RibbonConfiguration.class)
public class PieApplication {

    public static void main(String[] args) {
        SpringApplication.run(PieApplication.class, args);
    }

    @Bean
    @LoadBalanced
    public WebClient.Builder webClientBuilder() {
        return WebClient.builder();
    }

}
```

A.2.3.2 Основна сутність Measurement

```
package com.alex.pie.domain;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.time.LocalDateTime;


@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Measurement {

    private String id;
    private Double temperature;
    private Double humidity;
    private String location;
    private LocalDateTime createdDate;
}
```

A.2.3.3 Файл контролера PieController

```
package com.alex.pie.controller;
```

```
import com.alex.pie.domain.Measurement;
import com.alex.pie.service.SenderService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
import
org.springframework.web.reactive.function.client.WebClient;
import reactor.core.publisher.Flux;

import java.time.LocalDateTime;

@Slf4j
@RestController
public class PieController {

    @Autowired
    private WebClient.Builder webClient;
    @Autowired
    private SenderService senderService;

    @RequestMapping(method = RequestMethod.POST,value =
"/measurements")
    public void createMeasurement(@RequestBody Measurement
measurement) {
        measurement.setCreatedDate(LocalDateTime.now());
        senderService.sendMeasurement(measurement);
    }

    @RequestMapping(method = RequestMethod.GET, value =
"/measurements")
    public Flux<Measurement> getAllMeasurements() {
        return webClient.build()
                .get()
                .uri("http://KEEPER-SERVICE/measurements")
                .retrieve()
                .bodyToFlux(Measurement.class);
    }
}
```

A.2.3.4 Файл сервісу SenderService

```
package com.alex.pie.service;


import com.alex.pie.domain.Measurement;
import lombok.AllArgsConstructor;
import lombok.extern.slf4j.Slf4j;
```

```
import
org.springframework.cloud.stream.annotation.EnableBinding;
import org.springframework.cloud.stream.messaging.Source;
import org.springframework.messaging.support.MessageBuilder;
import org.springframework.stereotype.Service;


@Slf4j
@Service
@EnableBinding(Source.class)
@AllArgsConstructor
public class SenderService {

    private final Source source;

    public void sendMeasurement(Measurement measurement) {
        boolean status =
source.output().send(MessageBuilder.withPayload(measurement).bui
ld());
        if (status)
            log.info(measurement + " was successful sent to the
KAFKA(broker)");
        else
            log.warn("Unsuccessful sending measurement to the
KAFKA(broker)");
    }

}
```

A.2.3.5 Файл конфігурації application.properties

```
logging.file.name=logs/application.log

spring.cloud.stream.bindings.output.destination=keeper
spring.cloud.stream.bindings.output.group=keeper

spring.cloud.stream.kafka.binder.auto-add-partitions=true
```

A.2.3.6 Файл конфігурації application.yaml

```
Server:
    port: 8080

keeper-service:
    ribbon:
        eureka:
            enabled: true
        ServerListRefreshInterval: 1000

eureka:
    client:
        serviceUrl:
```

```
            defaultZone: ${EUREKA_URI:http://eureka:8761/eureka}
    instance:
        preferIpAddress: true


spring:
    application:
        name: pie-service
kafka
    cloud:
        stream:
            kafka:
                binder:
                    brokers: kafka
```

A.2.3.7 Файл збірки build.gradle

```
plugins {
    id 'org.springframework.boot' version '2.3.4.RELEASE'
    id 'io.spring.dependency-management' version
'1.0.10.RELEASE'
    id 'java'
}

group = 'com.alex'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '11'

configurations {
    compileOnly {
        extendsFrom annotationProcessor
    }
}

repositories {
    mavenCentral()
}

ext {
    set('springCloudVersion', "Hoxton.SR8")
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-
starter-webflux'
    implementation 'org.springframework.cloud:spring-cloud-
starter-netflix-eureka-client'
    //
https://mvnrepository.com/artifact/org.springframework.cloud/spr
ing-cloud-starter-stream-kafka
    compile group: 'org.springframework.cloud', name: 'spring-
cloud-starter-stream-kafka', version: '3.0.8.RELEASE'
```

```
    compileOnly 'org.projectlombok:lombok'
    developmentOnly 'org.springframework.boot:spring-boot-
devtools'
    annotationProcessor 'org.projectlombok:lombok'
    testImplementation('org.springframework.boot:spring-boot-
starter-test') {
        exclude group: 'org.junit.vintage', module: 'junit-
vintage-engine'
    }
    testImplementation 'io.projectreactor:reactor-test'
    compile group: 'org.springframework.data', name: 'spring-
data-jpa', version: '2.3.4.RELEASE'

}

dependencyManagement {
    imports {
        mavenBom "org.springframework.cloud:spring-cloud-
dependencies:${springCloudVersion}"
    }
}

test {
    useJUnitPlatform()
}
```

A.2.3.8 Файл докер Dockerfile

```
FROM azul/zulu-openjdk-alpine:11
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

A.2.4 Keeper-Service

A.2.4.1 Основний клас KeeperApplication.java

```
package com.alex.keeper;

import com.alex.keeper.domain.Measurement;
import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import
org.springframework.context.ConfigurableApplicationContext;
import org.springframework.data.mongodb.core.CollectionOptions;
import
org.springframework.data.mongodb.core.ReactiveMongoTemplate;
```

```
import
org.springframework.data.mongodb.repository.config.EnableReactiv
eMongoRepositories;

@EnableEurekaClient
@SpringBootApplication
@EnableReactiveMongoRepositories
public class KeeperApplication {

    public static void main(String[] args) {
        ConfigurableApplicationContext context =
SpringApplication.run(KeeperApplication.class, args);

        // Explicit creation of capped collection, because
spring haven't any options for specifying collection type
        ReactiveMongoTemplate mongoTemplate =
context.getBean(ReactiveMongoTemplate.class);

mongoTemplate.createCollection(Measurement.class.getSimpleName()
.toLowerCase(),
CollectionOptions.empty().capped().size(5242880).maxDocuments(50
0)).subscribe();

    }

}
```

A.2.4.2 Файл сутності Measurement

```
package com.alex.keeper.domain;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

import javax.validation.constraints.NotNull;
import java.time.LocalDateTime;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Document
public class Measurement {

    @Id
    private String id;
    private Double temperature;
```

```
    private Double humidity;
    private String location;
    @NotNull
    private LocalDateTime createdDate;
}
```

А.2.4.3 Файл контроллера MeasurementController

```
package com.alex.keeper.controller;

import com.alex.keeper.domain.Measurement;
import com.alex.keeper.service.MeasurementService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.*;
import reactor.core.publisher.Flux;

@CrossOrigin
@RestController
public class MeasurementController {

    @Autowired
    private MeasurementService measurementService;

    @RequestMapping(method = RequestMethod.GET, value =
"/measurements")
    public Flux<Measurement> getAllMeasurements() {
        return measurementService.getAllMeasurements();
    }

    @RequestMapping(method = RequestMethod.POST, value =
"/measurements")
    public void saveNewMeasure(@RequestBody Measurement
measurement) {
        measurementService.saveMeasurement(measurement);
    }

    @RequestMapping(method = RequestMethod.GET,
            value = "stream/measurements",
            produces = MediaType.TEXT_EVENT_STREAM_VALUE)
    public Flux<Measurement>
getAllMeasurementsStream(@RequestParam String location) {
        return measurementService.findAllByLocation(location);
    }
}
```

А.2.4.4 Файл сервісу MeasurementService

```
package com.alex.keeper.service;

import com.alex.keeper.domain.Measurement;
import com.alex.keeper.repository.MeasurementRepository;
```

```
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import reactor.core.publisher.Flux;

import java.time.LocalDateTime;

@Service
@Slf4j
public class MeasurementService {

    @Autowired
    private MeasurementRepository measurementRepository;

    public void saveMeasurement(Measurement measurement) {
        measurementRepository.save(measurement).subscribe();
        log.info("Measurement: temperature=" +
measurement.getTemperature() +
                " | humidity= " + measurement.getHumidity() + "
was saved");
    }

    public Flux<Measurement> getAllMeasurements(){
        log.info("GET ALL MEASUREMENTS");
        return measurementRepository.findAll();
    }

    public Flux<Measurement> findAllByLocation(String location)
{
        return
measurementRepository.findAllByLocationAndCreatedDateGreaterThan
(location,
                LocalDateTime.now().minusMinutes(1));
    }
}
```

A.2.4.5 Файл сервісу CheckMeasurementService

```
package com.alex.keeper.service;

import com.alex.keeper.domain.Measurement;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.cloud.stream.annotation.EnableBinding;
import
org.springframework.cloud.stream.annotation.StreamListener;
import org.springframework.cloud.stream.messaging.Sink;
import
org.springframework.integration.annotation.MessageEndpoint;
import org.springframework.stereotype.Service;
```

```
@Slf4j
@Service
@EnableBinding(Sink.class)
@MessageEndpoint
public class CheckMeasurementService {

    @Autowired
    private MeasurementService measurementService;

    @StreamListener(Sink.INPUT)
    public void getAvailableMeasurementToSave(Measurement
measurement) {
        log.info("Got [Measurement] from KAFKA(broker): " +
measurement);
        measurementService.saveMeasurement(measurement);
    }

}
```

A.2.4.6 Файл конфігурації application.properties

```
logging.file.name=logs/application.log

spring.cloud.stream.bindings.input.destination=keeper
spring.cloud.stream.bindings.input.group=keeper

spring.cloud.stream.kafka.binder.auto-add-partitions=true
```

A.2.4.7 Файл конфігурації application.yaml

```
spring:
  application:
    name: keeper-service
  data:
    mongodb:
      uri: mongodb://mongodb:27017/measurementdb
  cloud:
    stream:
      kafka:
        binder:
          brokers: kafka


Server:
  port: 8081

eureka:
  client:
    serviceUrl:
      defaultZone: ${EUREKA_URI:http://eureka:8761/eureka}
  instance:
    preferIpAddress: true
```

А.2.4.8 Файл збірки build.gradle

```
plugins {
    id 'org.springframework.boot' version '2.3.3.RELEASE'
    id 'io.spring.dependency-management' version
'1.0.10.RELEASE'
    id 'java'
}

group = 'com.alex'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '11'

configurations {
    compileOnly {
        extendsFrom annotationProcessor
    }
}

repositories {
    mavenCentral()
}

ext {
    set('springCloudVersion', "Hoxton.SR7")
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-
starter-actuator'
    implementation 'org.springframework.boot:spring-boot-
starter-webflux'
    implementation 'org.springframework.cloud:spring-cloud-
starter-netflix-eureka-client'
    implementation 'org.springframework.cloud:spring-cloud-
starter-netflix-ribbon'
    compile group: 'org.springframework.cloud', name: 'spring-
cloud-starter-stream-kafka', version: '3.0.8.RELEASE'
    implementation 'org.mongodb:mongodb-driver-
reactivestreams:4.1.1'
    compile group: 'org.springframework.boot', name: 'spring-
boot-starter-data-mongodb-reactive', version: '2.3.4.RELEASE'
    compileOnly 'org.projectlombok:lombok'
    developmentOnly 'org.springframework.boot:spring-boot-
devtools'
    annotationProcessor 'org.projectlombok:lombok'
    testImplementation('org.springframework.boot:spring-boot-
starter-test') {
        exclude group: 'org.junit.vintage', module: 'junit-
vintage-engine'
```

```
    }
    testImplementation 'io.projectreactor:reactor-test'
}

dependencyManagement {
    imports {
        mavenBom "org.springframework.cloud:spring-cloud-
dependencies:${springCloudVersion}"
    }
}

test {
    useJUnitPlatform()
}
```

A.2.4.9 Файл докеру Dockerfile

```
FROM azul/zulu-openjdk-alpine:11
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

A.2.5 Клієнтська частина Measurement-UI

A.2.5.1 Головна сторінка – index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>MeasurementUi</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-
scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <link
href="https://fonts.googleapis.com/css?family=Roboto:300,400,500
&display=swap" rel="stylesheet">
  <link
href="https://fonts.googleapis.com/icon?family=Material+Icons"
rel="stylesheet">
  <script type="text/javascript" src="canvasjs.min.js"></script>
</head>
<body class="mat-typography">
  <app-root></app-root>
</body>
</html>
```

A.2.5.2 Основний файл TypeScript – main.ts

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-
browser-dynamic';
```

```
import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
```

A.2.5.3 Таблиця стилів – styles.css

```
/* You can add global styles to this file, and also import other
style files */

html, body { height: 100%; }
body { margin: 0; font-family: Roboto, "Helvetica Neue", sans-
serif; }
```

A.2.5.4 apicall.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import {Measurement} from "./home/measurement";
import { from, Observable, throwError } from 'rxjs';
import { map, catchError } from 'rxjs/operators';
@Injectable({
  providedIn: 'root'
})
export class ApicallService {
  constructor(private httpClient: HttpClient) {}
  getUsers() {
    return
this.httpClient.get(`http://localhost:8083/keeper/measurements`)
.
    pipe(
      map((data: Measurement[]) => {
        return data;
      }), catchError( error => {
        return throwError( 'Something went wrong!' );
      })
    )
  }
}
```

A.2.5.5 app.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
```

```
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'measurement-ui';
}
```

A.2.5.6 app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import {HttpClientModule} from "@angular/common/http";
import { HomeComponent } from './home/home.component';
import { AboutComponent } from './about/about.component';
import {MatIconModule} from "@angular/material/icon";
import {MatToolbarModule} from "@angular/material/toolbar";
import {MatCardModule} from "@angular/material/card";
import {MatProgressSpinnerModule} from
"@angular/material/progress-spinner";
import {MatButtonModule} from "@angular/material/button";
import {BrowserAnimationsModule} from "@angular/platform-
browser/animations";
import { AgmCoreModule } from '@agm/core';

@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    AboutComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule,
    BrowserAnimationsModule,
    MatToolbarModule,
    MatIconModule,
    MatButtonModule,
    MatCardModule,
    MatProgressSpinnerModule,
    AgmCoreModule.forRoot({
      apiKey: 'AIzaSyDnGBPTHYv2MIjbVV1nt_RnZybKa4MgiH4'
    })
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

A.2.5.7 app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import {HomeComponent} from "./home/home.component";
import {AboutComponent} from "./about/about.component";

const routes: Routes = [
  { path: '', redirectTo: 'home', pathMatch: 'full'},
  { path: 'home', component: HomeComponent },
  { path: 'about', component: AboutComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

A.2.5.8 home.component.css

```
agm-map {
  height: 600px;
  width: 900px;
}
main {

}
```

A.2.5.9 home.component.html

```
<!--<h1>{{ measurement }}</h1>-->
<div id="main" style="width: 100%; height: 100%; display: flex;
flex-direction: row">
  <div style="height: 600px; width: 600px;">

    <div id="humidityChart" style="height: 300px; width:
600px;"></div>
    <div id="temperatureChart" style="height: 300px; width:
600px;"></div>
  </div>
  <!-- this creates a google map on the page with the given
lat/lng from -->
  <!-- the component as the initial center of the map: -->
  <div id="google_map">
    <agm-map [latitude]="lat" [longitude]="lng" [zoom]="zoom">
```

```
      <agm-marker
        *ngFor="let m of markers; let i = index"
        (markerClick)="clickedMarker(m.lat, m.lng)"
        [latitude]="m.lat"
        [longitude]="m.lng">
      </agm-marker>
    </agm-map>
  </div>
</div>
```

A.2.5.10 home.component.ts

```
import { Component, NgZone, OnDestroy, OnInit } from
'@angular/core';
import {Measurement} from "./measurement";
import {HttpClient, HttpClientModule} from
"@angular/common/http";
import {ApicallService} from "../apicall.service";
import { Observable, Subscription } from 'rxjs';
import * as $ from 'jquery';
import * as CanvasJS from '../../canvasjs.min';

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
export class HomeComponent implements OnInit {

  measurement: Measurement[];

  zoom = 15;
  lat = 50.5124138;
  lng = 30.7913499;

  constructor(public http: HttpClient, private apiService:
ApicallService, private zone: NgZone) { }

  temperaturePoints = [];
  humidityPoints = [];
  temperatureChart;
  humidityChart;
  temperatureDpsLength = 0;
  humidityDpsLength = 0;

  ngOnInit(): void {
    this.temperatureChart = new
CanvasJS.Chart("temperatureChart",{
      exportEnabled: true,
      title:{
        text:"Temperature"
      },
```

```
      data: [{
        type: "spline",
        dataPoints : this.temperaturePoints,
      }]
    });
    this.humidityChart = new CanvasJS.Chart("humidityChart",{
      exportEnabled: true,
      title:{
        text:"Humidity"
      },
      data: [{
        type: "spline",
        dataPoints : this.humidityPoints,
      }]
    });
  }

  clickedMarker(lat: number, lng: number) {
    console.log(`clicked the marker: ${lat} | ${lng}`)
    this.getMeasurements(`${lat},${lng}`).subscribe({
      next: data => {
        let jdata: Measurement = JSON.parse(data);
        console.log(jdata);
        this.updateTemperatureChart(jdata.temperature,
jdata.id);
        this.updateHumidityChart(jdata.humidity, jdata.id);
      },
      error: err => console.error(err)
    });
  }

  updateTemperatureChart(temperature: number, measurementId:
string) {

    if (this.temperaturePoints.length > 20) {
      this.temperaturePoints.shift();
    }
    this.temperaturePoints.push({
      x: this.temperatureDpsLength,
      y: temperature
    });
    this.temperatureDpsLength++;
    this.temperatureChart.render();
  }

  updateHumidityChart(humidity: number, measurementId: string) {

    if (this.humidityPoints.length > 20) {
      this.humidityPoints.shift();
    }
    this.humidityPoints.push({
      x: this.humidityDpsLength,
```

```
      y: humidity
    });
    this.humidityDpsLength++;
    this.humidityChart.render();
  }

  getMeasurements(location: string): Observable<string> {
    return Observable.create(
      observer => {
        let source = new
EventSource("http://localhost:8081/stream/measurements?location=
" + location);
        source.onmessage = event => {
          this.zone.run(() => {
            observer.next(event.data)
          })
        }
        source.onerror = event => {
          this.zone.run(() => {
            observer.error(event)
          })
        }
      }
    )
  }

  markers: marker[] = [
    {
      lat: 50.5124138,
      lng: 30.7913499,
    }
  ]

}

interface marker {
  lat: number;
  lng: number;
}
```

A.2.5.11 measurement.ts

```
export class Measurement{
  id: string;
  temperature: number;
  humidity: number;
  location: string;
  createdAt: any;
}
```

A.2.5.12 about.component.html

```
<div style="text-align: center; margin-top:200px">
  <h1>Author: Oleksandr Tokarenko</h1>
  <h1>Project: Measurement-System</h1>
  <h1>Group: ПА-91мп</h1>
</div>
```

A.2.5.13 about.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-about',
  templateUrl: './about.component.html',
  styleUrls: ['./about.component.css']
})
export class AboutComponent implements OnInit {

  constructor() { }

  ngOnInit(): void {
  }

}
```

A.2.6 Docker-Compose

```
version: '3'

services:
  zookeeper:
    image: wurstmeister/zookeeper
    container_name: zookeper
    ports:
      - "2181:2181"

  kafka:
    image: wurstmeister/kafka
    container_name: kafka
    ports:
      - "9092:9092"
    environment:
      KAFKA_ADVERTISED_HOST_NAME: kafka
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
    depends_on:
      - zookeeper

  mongo:
    image: mongo
    container_name: mongodb
    ports:
      - "27017:27017"
```

```
eureka:
  image: dgkyk/eureka:discoveryserver
  container_name: eureka
  ports:
  - "8761:8761"

pie:
  image: dgkyk/pie:service
  container_name: pie-service
  ports:
  - "8080:8080"
  depends_on:
    - eureka

keeper:
  image: dgkyk/keeper:service
  container_name: keeper-service
  ports:
    - "8081:8081"
  depends_on:
    - eureka

zuul:
  image: dgkyk/zuul:gateway
  container_name: zuul-service
  ports:
  - "8083:8083"
  depends_on:
    - eureka
```

ДОДАТОК Б. СПИСОК ПУБЛІКАЦІЙ

Б.1 XIX Міжнародна науково-технічна конференція “ПРИЛАДОБУДУВАННЯ: стан і перспективи” -2020

XIX Міжнародна науково-технічна конференція “ПРИЛАДОБУДУВАННЯ: стан і перспективи”, 13-14 травня 2020 року, КПІ ім. Ігоря Сікорського, Київ, Україна

УДК 621.3.087.44

РОЗРОБКА СИСТЕМИ ІНТЕРНЕТУ РЕЧЕЙ НА ОСНОВІ ПЛАТФОРМО-НЕЗАЛЕЖНИХ ТЕХНОЛОГІЙ

Токаренко О. В., Богомазов С. А.
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна
E-mail: tokarenko36@gmail.com

Протягом довгого часу домінуюче місце в розробці серверних додатків для Інтернету речей (Internet of Things, IoT) займала мова програмування Java [1]. Але не дивлячись на велику популярність даної мови, вона має ряд недоліків. В цілому засновані на Java веб-фреймворки не дозволяють швидке внесення змін в роботу. Ще один фактор, що сповільнює веб-розробку – недостатня гнучкість мови Java. В цьому випадку її статична типізація є недоліком, а не перевагою. Прискорити процес веб-розробки, прототипування, написання сценаріїв дозволяє використання JVM-сумісних програмних засобів, що підтримують динамічну типізацію, зокрема мови Groovy та фреймворку Grails.

Grails – це фреймворк для швидкої веб-розробки, в основі якого лежить динамічна мова Groovy. В першу чергу Grails допомагає заощадити час при роботі над автоматичним створенням структури проекту. Даний фреймворк активно використовує концепцію програмування за угодами, що застосовується в аналогічному фреймворці Rails для мови Ruby. Якщо код відповідає угодам, то фреймворк виконує багато завдань, що вирішуються на рівні шаблонного коду [2]. Groovy – це об'єктно-орієнтована динамічно типізована мова, яка працює на віртуальній машині Java (JVM), що забезпечує платформо-незалежність технології. Її динамічні можливості доповнюють статично типізовану мову Java. Використання динамічної типізації дозволяє скоротити код, тому що зникає оголошення «очевидних» типів, забезпечується більш швидкий зворотній зв'язок і гнучкість при присвоєнні об'єктів різних типів єдиній змінній, з якою виконуються операції. Groovy заснований на ідеях Smalltalk, Ruby і Python і підтримує функціональні літерали, вбудовані колекції, регулярні вирази. В Groovy значно спрощується робота з XML завдяки вбудованій підтримці XML-обробки.

На базі фреймворку Grails з використанням мови Groovy була розроблена серверна частина демонстраційної системи Інтернету речей. Програмне забезпечення серверної частини системи розроблено для запуску у віртуальній машині Java. Такий підхід дозволив отримати платформо-незалежне програмне рішення, що не залежить від конкретних архітектур. Демонстраційна система складається з інтелектуальних датчиків, CoAP сервера і сервера додатків з СУБД PostgreSQL. Дані про температуру та вологість надходять з інтелектуальних датчиків в одноплатний комп’ютер Raspberry Pi. За допомогою протоколу CoAP (Constrained Application Protocol) виконуються запити на отримання даних.

CoAP – це веб-протокол, оптимізований для мереж з обмеженими ресурсами, типових для IoT і M2M (Machine-to-Machine) додатків [3]. Протокол базується на REST архітектурі, в якій ресурси доступні і ідентифіковані за допомогою уніфікованих ідентифікаторів ресурсів (URI). З ресурсами можна взаємодіяти за допомогою тих самих методів, які використовуються в HTTP: GET, PUT, POST і DELETE. Протокол складається з підмножини функціональних можливостей HTTP, які були перероблені з урахуванням обмежених обчислювальних можливостей і необхідності споживання мінімальної енергії. Транзакція CoAP має в 10 разів менше за розміром повідомлення, ніж HTTP. Це є наслідком значного стиснення заголовку, який реалізовано в протоколі CoAP.

В якості хмарної платформи було використано Amazon EC2. За допомогою налаштування конфігурації iptables реалізовано можливість обміну даними з веб-сервером як за протоколом HTTP, так і за протоколом CoAP. Для обміну даними за протоколом CoAP використовувалась бібліотека Californium, для роботи через UART – бібліотека jssc, що дозволило абстрагуватись від низькорівневих деталей. Для дослідження роботи сервера використовувався додаток Copper для браузеру Firefox та програма для аналізу мережевих пакетів Wireshark.

Запит до CoAP-сервера типу GET coap://name:port/.well-known/core повертає відповідь в форматі xml, що містить перелік URI усіх ресурсів. Запит до CoAP-сервера типу observe для ресурсу має вигляд coap://192.168.2.1:5683/SensorNode/ObjectTemperature. В результаті сервер періодично посилає повідомлення з експериментальними даними при кожному оновленні CoAP ресурсу. У випадку, коли метод не підтримується ресурсом, відправляється повідомлення з кодом 4.05- method not allowed.

Застосування протоколу CoAP дозволило зменшити вимоги до пропускної здатності комунікаційного каналу, що дозволяє використовувати навіть низькошвидкісне модемне з'єднання. Структура системи є горизонтально масштабованою, тобто дозволяє розширити конфігурацію на будь-якому рівні. Суттєвою перевагою розробленої структури є можливість перенесення складних обчислювальних операцій на рівень "хмарних" сервісів.

Використання фреймворку Grails та мови Groovy дало змогу значно скоротити кількість коду, надати гнучкості і стабільності роботи при внесенні змін при збереженні платформонезалежності програмного забезпечення.

Ключові слова: Інтернет речей, обробка даних, протокол CoAP, веб-додаток.

Література

[1] О. В. Токаренко, та С. А. Богомазов, "Система збору експериментальних даних на основі Java-фреймворків", на *XV Всеукр. наук.-практ. конф. студ., асп. та мол. вч. Ефективність інженерних рішень у приладобудуванні*, Київ, 2019, с.532-534.

[2] Б. Эванс, и М. Вербург, *Java. Новое поколение разработки.* СПб, Россия: Питер, 2014.

[3] Z. Shelby, K.Hartke, and C. Bormann, The Constrained Application Protocol (CoAP). RFC 7252, 2014. [Online]. Available: http://www.rfc-editor.org/info/rfc7252. Accessed on: June 1, 2018.

Б.2 Конференція ЕІРП “Ефективність Інженерних Рішень у Приладобудуванні” – 2019

УДК 681.1

О.В. Токаренко, студент гр. ПА-91мп, к.т.н., доц. Богомазов С.А.
КПІ ім. Ігоря Сікорського

СИСТЕМА ЗБОРУ ЕКСПЕРИМЕНТАЛЬНИХ ДАНИХ НА ОСНОВІ JAVA-ФРЕЙМВОРКІВ

Анотація. У статті проведений опис та аналіз доцільності використання Java-фреймворків в програмних системах збору та обробки експериментальних даних. Визначені основні переваги використання даного підходу та перспективи його застосування. Наведено приклад реалізації системи на основі фреймворків Spring та Hibernate сумісно з мережевим протоколом CoAP. Проведений аналіз буде корисним для оптимізації програмного забезпечення систем які виконують збір та обробку експериментальних даних.

Ключові слова: Java, фреймворк, збір та обробка даних, база даних.

ПОСТАНОВКА ПРОБЛЕМИ

В системах автоматизованих вимірювань, контролю, технічного моніторингу важливу роль відіграють процеси збору, зберігання та аналізу експериментальних даних від різноманітних джерел, а саме проводових і безпроводових вимірювальних і сенсорних засобів. При створенні таких систем виникає необхідність розробки програмного забезпечення для зберігання, отримання та обробки експериментальних даних. Для цього проведено аналіз існуючих рішень для автоматизації даних процесів за допомогою сучасних Java-фреймворків.

ОСНОВНІ ПОЛОЖЕННЯ

Бібліотеки-фреймворки надають розробнику деяку постійну частину програмного забезпечення, до якої необхідно додати код користувача. Користувацька частина вбудовується в програмне забезпечення фреймворку. Фреймворк визначає загальну архітектуру програмного продукту та реалізує взаємодію між його компонентами.

На рис.1 зображені основні шари програмного забезпечення типового Web-орієнтованого програмного додатку. У випадку Web-орієнтованих програмних систем збору та обробки даних, що реалізовані за допомогою мови програмування Java, є доцільним застосування сучасних фреймворків в проміжних шарах такої системи, а саме:

1. При передачі вимірювальних даних шляхом запитів до серверу.
2. При зберіганні та обробці вимірювальної інформації.

Саме на цих рівнях програмного забезпечення заміщення процесу ручної розробки програмного забезпечення розробкою з використанням фреймворків буде найбільш ефективним. На допомогу приходять сучасні бібліотеки-фреймворки, такі як Spring Framework, Hibernate, React. На даний момент це одні з найвідоміших представників інструментів для автоматизації створення та покращення роботи програмних систем що виконують збір, передачу, обробку та аналіз вимірювальних даних.

Spring Framework дозволяє суттєво спростити реалізацію Web-орієнтованих систем шляхом автоматизації базових процесів [1]. В його основі лежить реалізація принципу зменшення кількості залежностей між частинами програмної системи шляхом використання технології інжекції залежностей

(Dependency Injection). Згідно з принципом інверсії керування (Inversion of Control, IoC), ядро фреймворку самостійно створює об'єкти конкретних класів та передає посилання на них (інжектує) в залежні від відповідних абстракцій об'єкти. Тим самим зберігається залежність лише від абстракцій, а не від конкретних класів, що значно спрощує модифікацію системи.

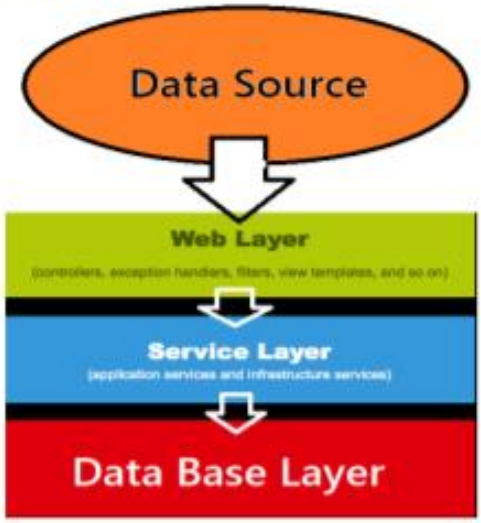

Рисунок 1. Шари типового Web-додатку для збору та обробки даних

За допомогою анотацій або конфігураційних файлів фреймворк Spring дозволяє співставити URI відправленого на сервер запиту відповідному контролеру для обробки та формування відповіді. Цей фреймворк дозволяє пришвидшити створення Web-орієнтованих програмних систем з архітектурою MVC (Model-View-Controller) та зменшує вимоги до кваліфікації розробників програмного забезпечення. Якщо необхідна модифікація системи для певних користувацьких завдань, Spring дозволяє легко вносити корективи в конфігурацію системи.

Важливою частиною системи збору та обробки експериментальних даних є збереження отриманої інформації в базі даних. Найбільш поширеним рішенням, яке дозволяє зменшити час на проектування та розробку шару зберігання даних є використання ORM-фреймворків. ORM – це Object Relationship Mapping, тобто співставлення об'єктних відношень з реляційними таблицями баз даних.

Одним з представників таких фреймворків є Hibernate [2]. Основним завданням цього фреймворку є автоматичне перетворення об'єктів, що використовуються в об'єктно-орієнтованій програмі в записи бази даних. Це дозволяє скоротити час розробки програмного забезпечення та пришвидшити процес зберігання та отримання інформації. Порівняно з ручною реалізацією за допомогою драйвера JDBC операцій CRUD (Create, Read, Update, Delete), даний фреймворк дозволяє перейти від низькорівневого програмування до високорівневого, використовуючи вже готові рішення для CRUD операцій. Для

цього розробнику достатньо лише сконфігурувати цей процес шляхом використання конфігураційних XML-файлів або анотацій. Анотації використовуються безпосередньо в класах, які описують структуру об'єктів, що створюються на базі інформації, отриманої з зовнішніх систем. Клієнти при цьому який можуть бути реалізовані як на основі персональних комп'ютерів або мобільних пристроїв, так і на основі інтелектуальних мережевих сенсорів, що надсилають потік вимірювальних даних. Для оптимізації роботи фреймворк Hibernate використовує кешування, скорочуючи часові витрати на обробку запитів.

На базі фреймворків Spring та Hibernate розроблено програмне забезпечення серверної частини демонстраційної Інтернет-системи збору експериментальних даних. За допомогою датчика DHT11, що підключений до одноплатного комп'ютера RaspberryPi, вимірюється температура та вологість в приміщенні для моніторингу його кліматичних параметрів. Для передачі результатів вимірювань через мережу Інтернет до серверної частини системи використано прикладний мережевий протокол CoAP (Constrained Application Protocol), що призначений для обміну інформацією для пристроїв з обмеженими ресурсами [3]. Це дозволило забезпечити передачу повідомлень малого розміру та зменшити енерговитрати автономних частин системи. Протокол CoAP відповідає структурі REST (Representational State Transfer) та забезпечує логічну сумісність запитів з запитами протоколу HTTP при значно меншій кількості службової інформації [4]. Технологія REST-сервісів підтримується сучасними версіями фреймворку Spring. Серверна частина системи за допомогою CoAP-проксі перетворює запити CoAP в запити HTTP, які обробляються серверним програмним забезпеченням.

ВИСНОВОК

Таким чином, використання сучасних Java-фреймворків дозволило оптимізувати процес розробки програмного забезпечення системи збору даних та полегшити внесення змін до програмного забезпечення системи. При цьому кожний елемент системи може бути модифікований незалежно від інших частин. Використання спеціалізованого прикладного протоколу CoAP дозволило зменшити мережевий трафік та збільшити час робочого циклу автономних мережевих вимірювальних пристроїв.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Spring Framework . – Режим доступу: https://spring.io – 21.11.2019 р.
[2] Hibernate ORM. – Режим доступу: https://hibernate.org/ – 21.11.2019 р.
[3] Гойхман В. Аналитический обзор протоколов Интернета вещей / В. Гойхман, А. Савельева // Технологии и средства связи. – 2016. – № 4. – С. 32–37.
[4] ITU-T/ The Constrained Application Protocol (CoAP) / RFC 7252 – Proposed Standard. 2014.

Наук. керівник – к.т.н., доц. Богомазов С.А.

Б.3 XIII Науково-практична конференція студентів, аспірантів та молодих вчених «Погляд у майбутнє приладобудування»-2020

УДК 621.3.087

О.В. Токаренко, студент гр. ПА-91мп, к.т.н., доц. Богомазов С.А.
КПІ ім. Ігоря Сікорського

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ МЕРЕЖЕВОЇ СИСТЕМИ ЗБОРУ ДАНИХ НА ОСНОВІ ПРОТОКОЛУ COAP

Анотація. В даній статті розглянуто особливості використання спеціалізованого Web-протоколу CoAP(Constrained Application Protocol) для реалізації розподіленої системи збору даних та її програмного забезпечення. Застосування протоколу CoAP зменшує вимоги до пропускної здатності мереж, що дозволяє використовувати низькошвидкісні модемні з'єднання. Особливістю розробленої системи є можливість перенесення ресурсоємних операцій на рівень "хмарних" сервісів.

Ключові слова: система збору даних, протокол CoAP, Інтернет речей.

ВСТУП

Інтернет речей (Internet of Things, IoT) – це концепція обчислювальної мережі фізичних об'єктів («розумних речей»), оснащених вбудованими технологіями для мережевої взаємодії один з одним або з зовнішнім середовищем. Ці «речі» постійно генерують величезну кількість даних, мають обмежений енергоресурс, невеликий обсяг пам'яті і невисоку потужність, тому в роботі з ними важливо забезпечувати низькі енерговитрати, використовувати передачу повідомлень малого обсягу.

Актуальним напрямом вирішення цієї проблеми є мікроконтролерна реалізація спеціалізованих Web-технологій, спрямованих на передачу вимірювальної інформації через Інтернет в умовах обмежених обчислювальних ресурсів вбудованих систем, зокрема спеціалізованого Web-протоколу передачі даних CoAP (Constrained Application Protocol).

Метою роботи є розробка розподіленої системи збору даних та її програмного забезпечення для передачі, зберігання та обробки вимірювальної інформації на базі спеціалізованого протоколу CoAP та JavaEE Web-додатку з використанням хмарних технологій.

МАТЕРІАЛИ І РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

CoAP – це програмний протокол, призначений для використання у простих електронних пристроях, що дозволяє їм обмінюватися інформацією в інтерактивному режимі, використовуючи мережу Інтернет. CoAP є протоколом прикладного рівня, який призначений для використання в пристроях з обмеженими ресурсами. Часто використовується для невеликих датчиків малої потужності, комутаторів та інших пристроїв, до яких потрібен віддалений доступ через мережу [1].

Протокол CoAP побудовано аналогічно до протоколу HTTP, але на відміну від останнього він базується на бінарному а не текстовому форматі повідомлень. CoAP створено на основі HTTP протоколу щоб спростити інтеграцію з мережею Інтернет, в той же час цей протокол простий у реалізації та має дуже малі накладні витрати.

Структура протоколу розроблена у відповідності з REST-архітектурою. CoAP може працювати на більшості пристроїв, які підтримують протокол UDP або аналог цього протоколу. CoAP базується на обміні компактними

повідомленнями, що за замовчуванням передаються за допомогою UDP дейтаграм. CoAP також може працювати з використанням DTLS (Datagram Transport Layer Security), SMS, TCP або SCTP.

CoAP використовує механізми подібні до HTTP – GET, PUT, POST, DELETE та інш. Таким чином використовується Web-стиль програмування додатків, але в набагато полегшеній реалізації, що важливо для пристроїв з обмеженими ресурсами. CoAP повідомлення кодуються в простому бінарному форматі. Заголовок CoAP повідомлення складає лише 4 байти, що вигідно відрізняє його від інших протоколів. Це дозволяє на порядок зменшити розмір службової інформації порівняно з протоколом HTTP [2].

Розроблена демонстраційна система збору даних на основі протоколу CoAP складається з сенсорних вузлів, CoAP-сервера і сервера додатків JavaEE з СУБД PostgreSQL. Для реалізації сенсорних вузлів було використано мікроконтролер MSP430G2553, що відрізняється ультранизьким енергоспоживанням. В сенсорному вузлі реалізовано три вимірювальних канали для наступних параметрів: температури, зовнішньої аналогової величини (постійна напруга від 0В до 2,5В) та напруги живлення мікроконтролера. Для збору і передачі даних від вимірювальних каналів в ролі CoAP-сервера використовувався одноплатний комп'ютер Raspberry Pi. Програмне забезпечення розроблено для запуску у віртуальній машині Java. Такий підхід дозволив отримати програмне рішення, що не залежить від архітектури. Для обміну даними за протоколом CoAP використано бібліотеку Californium, для передачі даних через UART – бібліотеку jssc.

Запит типу GET coap://"доменне ім'я:порт"/.well-known/core повертає xml-відповідь, яка містить перелік URI усіх ресурсів, що відповідає принципам REST архітектури [3]. Приклад переліку ресурсів наведено на рис.1.

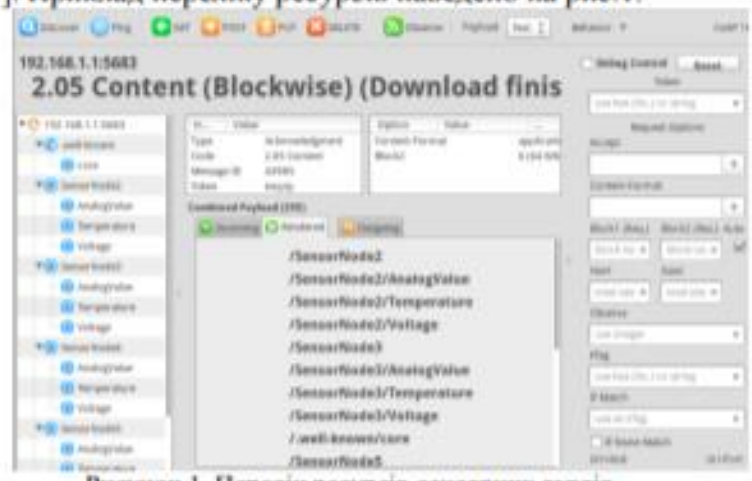

Рисунок 1. Перелік ресурсів сенсорних вузлів

В якості cloud платформи було використано Amazon Web Services – Elastic Cloud 2. За допомогою налаштування конфігурації iptables реалізовано можливість обміну даними як за протоколом HTTP з Web-сервером, так і за протоколом CoAP. Для зв'язку CoAP-сервера та Web-сервера було використано з'єднання PPTP (Point-to-Point Tunneling Protocol). При ввімкненні живлення

одноплатний комп`ютер Raspberry Pi автоматично встановлює підключення до Інтернету через CDMA модем, створює тунельоване з'єднання та виконує запуск CoAP-сервера. Структура системи наведена на рис.2.

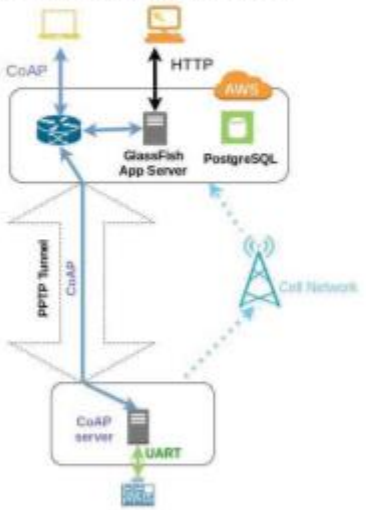

Рисунок 2. Структура мережевої системи збору даних

Пакетна структура розробленого програмного забезпечення містить п'ять пакетів: *.entity – містить класи сутності, які відповідають предметній області; *.dao – містить класи, що слугують для збереження або отримання даних з БД; *.logic – реалізує основну логіку додатку, а саме збір експериментальних даних за протоколом CoAP та їх збереження; *.util – містить допоміжні класи; *.web – містить класи, які відповідають за обробку запиту та формування відповіді на запит.

ВИСНОВКИ

Було розроблено структуру та організовано взаємодію компонентів розподіленої системи збору експериментальних даних на основі протоколу CoAP. Застосування цього протоколу зменшує вимоги до пропускної здатності комунікаційного каналу. Це дозволяє використовувати низькошвидкісні модемні з'єднання. Особливістю розробленої системи є перенесення ресурсоємних операцій на рівень "хмарних" сервісів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Z. Shelby, K.Hartke, and C. Bormann, The Constrained Application Protocol (CoAP). RFC 7252, 2014. [Online]. Available: http://www.rfc-editor.org/info/rfc7252. Accessed on: June 1, 2018.

[2] В. Гойхман, и А. Савельева, «Протокол Интернета вещей CoAP», Технологии и средства связи, № 4, с.20-24, 2017.

[3]Z. Shelby, Constrained RESTful Environments (CoRE) Link Format. RFC 6690, 2012. [Online]. Available: http://www.rfc-editor.org/info/rfc6690. Accessed on: June 1, 2019

Б.4 XVI Науково-практична конференція студентів, аспірантів та молодих вчених «Ефективність та автоматизація інженерних рішень у приладобудуванні» -2020.

УДК 681.1

О.В. Токаренко, студент гр. ПА-91мп, к.т.н., доц. Богомазов С.А.
КПІ ім. Ігоря Сікорського

СИСТЕМА ЗБОРУ ЕКСПЕРИМЕНТАЛЬНИХ ДАНИХ НА ОСНОВІ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ

Анотація. У статті наведено аналіз особливостей використання мікросервісної архітектури в програмному забезпеченні мережевих систем збору даних. Визначені основні переваги використання даної архітектури та перспективи її застосування. Систему реалізовано на основі фреймворку Spring WebFlux та нереляційної бази даних MongoDB. Проведений аналіз буде корисним для оптимізації програмного забезпечення систем збору та обробки експериментальних даних.

Ключові слова: мікросервіс, Java, фреймворк Spring, збір та обробка даних, база даних.

ПОСТАНОВКА ПРОБЛЕМИ

В сучасних системах Інтернету речей важливу роль відіграє здатність серверного програмного забезпечення до масштабування, спрощення процесу його розробки і модифікації.

Програмне забезпечення на основі монолітної архітектури будується як єдине ціле, в якому інтерфейс користувача і реалізація доступу до даних об'єднані в одну програму на одній платформі. Труднощі при використанні монолітної архітектури виникають при масштабуванні додатків. Кожен раз, коли розробляються, тестуються та впроваджуються нові функціональні можливості необхідно змінювати весь моноліт, тому що присутня велика зв’язаність між модулями системи.

Мікросервісна архітектура розподіляє додаток на більш дрібні, повністю незалежні компоненти, що забезпечує їм більшу гнучкість і масштабованість. Цей тип архітектури передбачає велику кількість невеликих сервісів, кожен з яких виконує свої власні функції і може бути незалежно розгорнутий. Такі сервіси виконуються в окремих процесах та комунікують між собою через веб-запити або віддалені виклики процедур. При цьому виникає задача загальної організації системи, так як подальша розробка та можливість внесення змін будуть залежати саме від цього. Було проведено аналіз особливостей реалізації такого типу систем та розроблено демонстраційну систему збору експериментальних даних на базі мікросервісної архітектури на основі сучасних Java фреймворків та нереляційної бази даних.

ОСНОВНІ ПОЛОЖЕННЯ

На базі мікросервісного архітектурного стилю [1] розроблена демонстраційна система збору експериментальних даних. Вона складається з

наступних компонентів: сервіси, API Gateway, Message broker, MongoDB. На рис.1 наведено структурну схему мікросервісної системи збору даних.

API Gateway. Це служба, що надає єдину точку входу для певних груп мікросервісів. Служба реалізована на основі за допомогою серверу-шлюзу Zuul. Це маршрутизатор і серверний балансувальник навантаження від Netflix, що працює в розробленій системі сумісно з сервером регістрації мікросервісів Eureka.

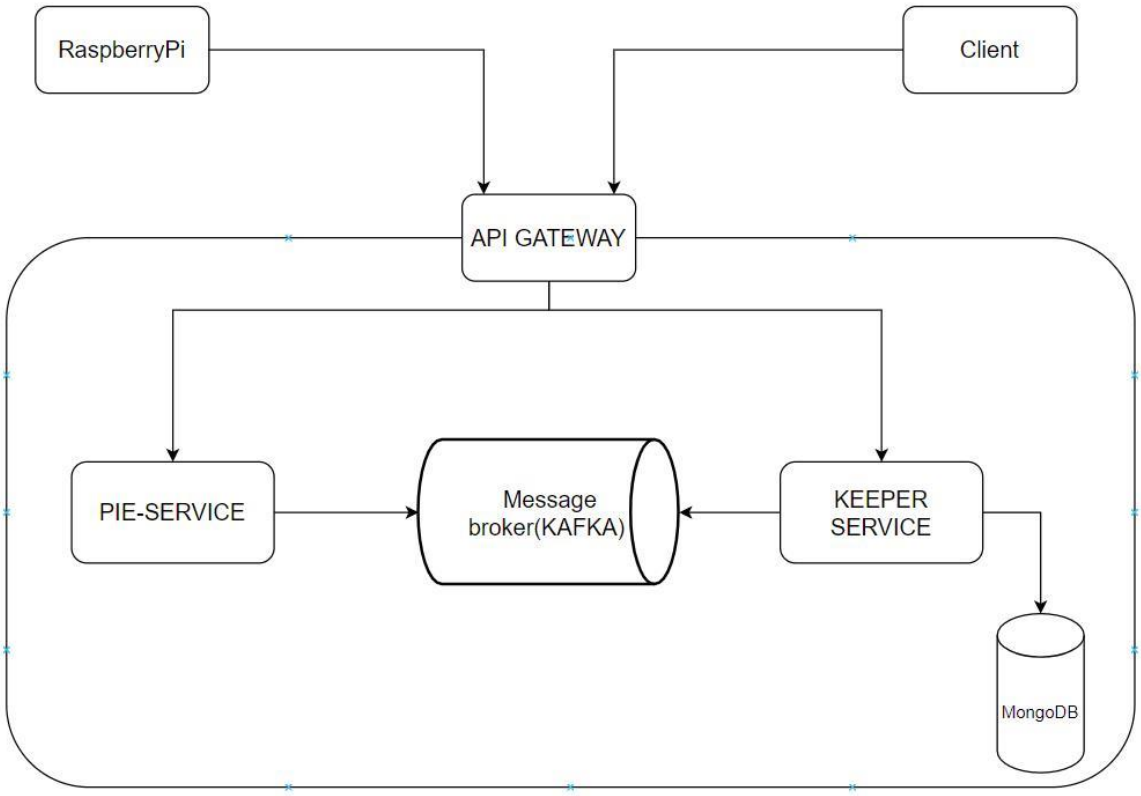

Рисунок 1. Структурна схема системи збору даних на базі мікросервісної архітектури

Message broker (Kafka). За допомогою брокера повідомлень Kafka в системі реалізовано архітектурний шаблон Message broker**.** Kafka – це високопродуктивна розподілена система обміну повідомленнями [2]. Черга повідомлень дозволяє позбутись зв'язаності між мікросервісами. При її використанні немає необхідності знати про особливості інших компонентів системи, все що потрібно – це передати повідомлення до брокера. Таким чином елементи системи можуть бути замінені без порушення загального функціонування системи з мінімальними зусиллями з боку розробників.

MongoDB. Важливою частиною систем збору та обробки експериментальних даних є збереження отриманої інформації в базі даних. Використано MongoDB – документоорієнтовану систему управління базами даних, яка не потребує опису схеми таблиць. Вважається одним з класичних прикладів NoSQL-систем, використовує JSON-подібні документи і схему бази даних. Однією з переваг такого типу баз даних є масштабованість, ефективне збереження великих об'ємів даних, швидкість виконання операцій та вбудована підтримка асинхронного виконання запитів [3].

RaspberryPi. Для реалізації вузлів збору даних було використано одноплатний комп'ютер RaspberryPi та датчики температури та вологості DHT22. Основним завданням програмного забезпечення даного компонента є передача експериментальних даних через API шлюз до мікросервісу Pie-Service.

Для реалізації системи на базі Java-технологій було обрано платформу WebFlux Spring. Платформа WebFlux є альтернативою для версії Spring MVC і реалізує реактивний підхід для створення веб-сервісів [4]. Spring WebFlux реалізує асинхронний і неблокуючий веб-стек, який дозволяє обробляти велику кількість одночасних з'єднань (рис.2).

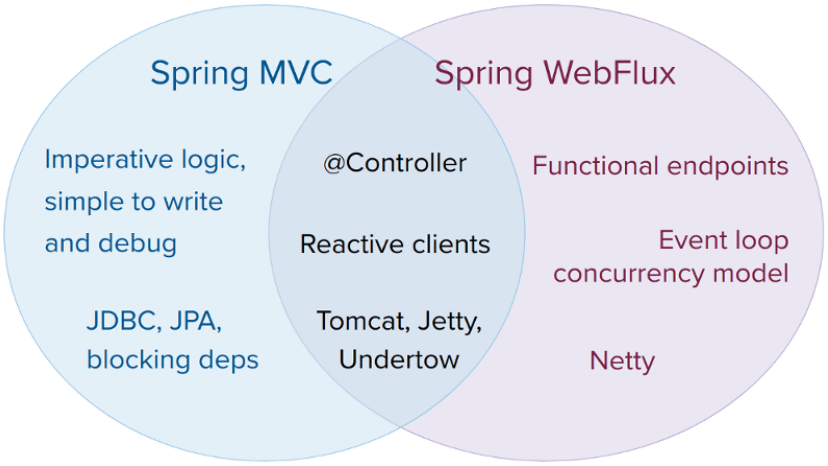

Рисунок 2. Особливості фреймворків Spring MVC та Spring WebFlux

В розробленій системі отримані від сенсорів вимірювальних пристроїв дані надсилаються до шлюзу веб-додатку, який перенаправляє їх до мікросервісу Pie-Service. Pie-Service додає відліки дати та часу та відсилає дані в чергу брокера повідомлень Kafka. Мікросервіс Keeper-Service є підписником на ці повідомлення. При появі нового повідомлення Keeper-Service отримує його, виконує обробку та зберігає до бази даних MongoDB.

Клієнт за запитом отримує дані для вибраного за відповідними координатами геолокації вузла збору даних. При зверненні клієнт відкриває з'єднання з базою даних. Так як MongoDB підтримує з'єднання типу tailable cursor, з'єднання може залишатися відкритим. Це дозволяє отримувати експериментальні дані в режимі, наближеному до реального часу.

ВИСНОВОК

Таким чином, була реалізована демонстраційна вимірювальна система збору експериментальних даних на базі мікросервісної архітектури. Даний підхід дав можливість створити систему з низькою зв'язаністю елементів та широкими можливостями до масштабування та модифікації. Було організовано передачу вимірювальних даних до шлюзу реалізованої веб-системи та їх подальшу обробку, збереження та представлення. В результаті використання реактивного підходу зросли відмовостійкість системи та можливість витримувати великі навантаження.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Software architecture. [Online]. Available: https://en.wikipedia.org/wiki/ Software_architecture. Accessed on: November 1, 2020.
[2] Message Broker. [Online]. Available:https://en.wikipedia.org/ wiki/ Message_broker. Accessed on: November 1, 2020.
[3] MongoDB. [Online]. Available:https://ru.wikipedia.org/wiki/MongoDB. Accessed on: November 1, 2020.
[4] Spring WebFlux. [Online]. Available:https://docs.spring.io/spring-framework /docs/current/reference/html/web-reactive.html. Accessed on: November 1, 2020.

Наук. керівник – к.т.н., доц. Богомазов С.А.