Хмельницький національний університет

Факультет програмування

та комп’ютерних і телекомунікаційних систем

Кафедра телекомунікацій, медійних та інтелектуальних технологій

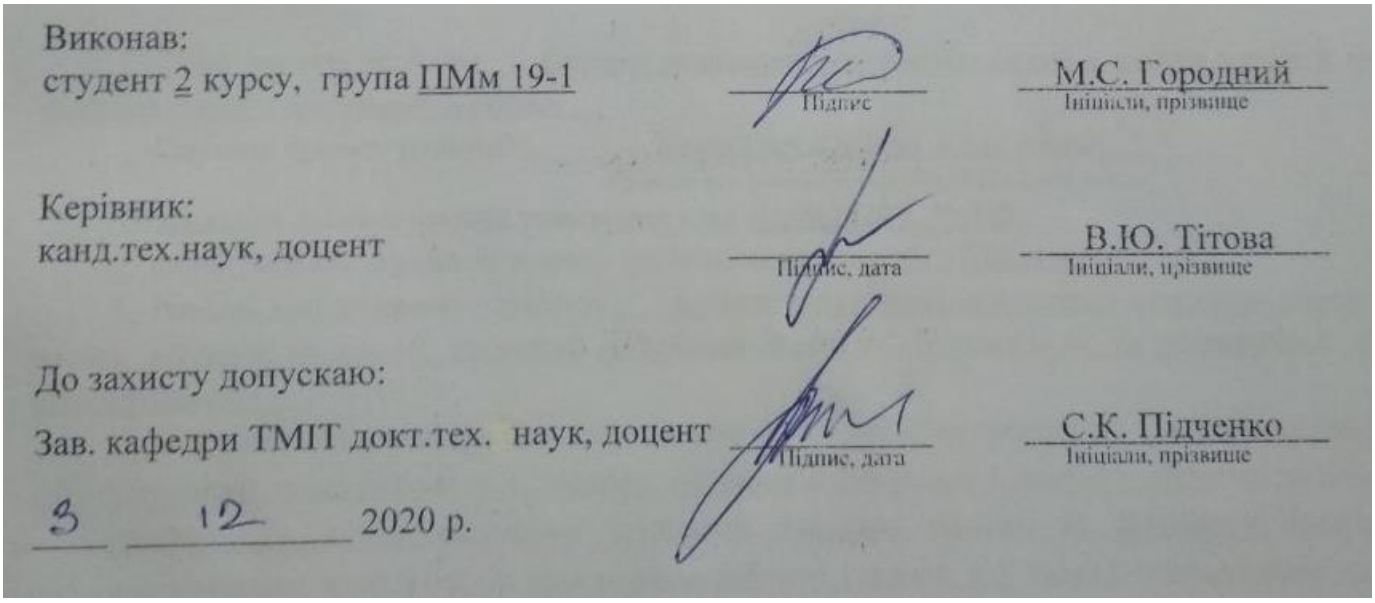
ДИПЛОМНА РОБОТА МАГІСТРА

СИСТЕМА МОНІТОРИНГУ ЗА ЖИТТЯМ ЛЮДЕЙ З ВАДАМИ ЗДОРОВ’Я НА ОСНОВІ ТЕХНОЛОГІЇ «РОЗУМНИЙ БУДИНОК»

Галузь знань 11 – Математика та статистика

Спеціальність 113 – Прикладна математика

Шифр ДРПМ.2019/102.01.07



Хмельницький, 2020

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Програмування та комп'ютерних і телекомунікаційних систем

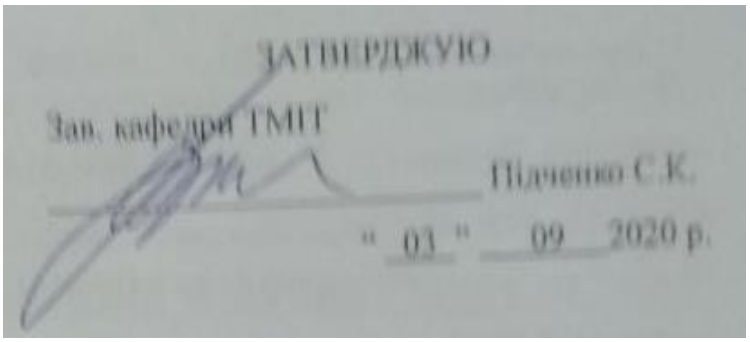
Кафедра телекомунікацій, медійних та інтелектуальних технологій

Освітній рівень магістр

Галузь знань 11 Математика та статистика

Спеціальність 113 Прикладна математика

Освітня програма освітньо-професійна програма підготовки магістра



ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ)

Городному Михайлу Сергійовичу

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Система моніторингу за життям людей з вадами здоров'я на основі технології «розумний будинок»

Керівник проекту (роботи) Тітова Віра Юріївна, к.т.н, доцент

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 01.09.2020 р. № 118

2. Строк подання студентом проекту (роботи) на кафедру 01.12.2020 р.

3. Вихідні дані до проекту (роботи) Архітектура системи моніторингу за життям людей з вадами здоров'я на основі технології «розумний будинок», її реалізація та рекомендації до стандартизації.

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Ознайомитись з існуючими технологіями та підходами, які використовуються в додатках «інтернет речей»; ознайомитись з архітектурними підходами «великих даних» та «інтернету речей»; запропонувати архітектуру, з мінімальним набором сервісів, яку можна використувати для розробки додатку для моніторингу на основі технології «інтернет речей»; навести приклад реалізації запропонованої архітектури.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1			
2			
3			
4			

7. Дата видачі завдання « 03 » вересня 2020 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів (розділів) дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Затвердження теми науковим керівником	01.09.2020 – 02.09.2020	Виконано
2	Аналіз джерел інформації про «інтернет речей»	03.09.2020 –08.09.2020	Виконано
3	Розробка 1 розділу написання ДРМ	09.09.2020 – 20.09.2020	Виконано
4	Аналіз існуючих архітектурних підходів до розробки додатків «інтернет речей»	21.09.2020 – 27.09.2020	Виконано
5	Розробка 2 розділу написання ДРМ	28.09.2020 – 7.10.2020	Виконано
6	Розробка архітектури та стандартизації для додатку	08.10.2020 – 13.10.2020	Виконано
7	Розробка 3 розділу написання ДРМ	14.10.2020 – 05.11.2020	Виконано
8	Розробка 4 розділу написання ДРМ	06.11.2020 – 08.11.2020	Виконано
9	Написання вступу, висновків, формування переліку джерел посилання та додатків	09.11.2020 –10.11.2020	Виконано
10	Подача роботи на: кафедру, антиплагіат, рецензування, нормоконтроль	12.11.2020 – 3.12.2020	Виконано
11	Захист дипломної роботи	12.12.2020	Виконано

Студент _____ М.С. Городний
Підпис

Керівник проекту (роботи) _____ В.Ю. Тітова
Підпис

АНОТАЦІЯ

Тема дипломної роботи: Система моніторингу за життям людей з вадами здоров’я на основі технології розумний будинок.

Автор роботи: Городний Михайло Сергійович.

Керівник роботи: Тітова Віра Юріївна.

Загальний обсяг роботи: 85 сторінок, 8 рисунків, 6 таблиць, 26 посилань.

РОЗУМНИЙ БУДИНОК, ВЕЛИКІ ДАНІ, СИСТЕМА МОНІТОРИНГУ, МІКРОСЕРВІСИ, АРХІТЕКТУРА.

Метою роботи є розробка архітектури бекенд додатку для «інтернету речей». Предмет дослідження: архітектура бекенд додатку на основі технології «інтернет речей». Наукова новизна: запропонована оригінальна архітектура для додатків «інтернет речей», що надає можливість моніторингу за життям і здоров’ям людей. Практичне значення: запропоновану систему можна застосовувати в сфері охорони здоров’я для моніторингу за пацієнтами.

ANNOTATION

Thesis topic: Monitoring system for the lives of people with disabilities based on smart home technology.

A master’s degree work of Mykhailo Horodnyi.

Mentor: Titova Vira.

Total volume of work: 85 pages, 8 figures, 6 tables, 26 references.

The aim of the work is to develop the architecture of a backend application for the "Internet of Things". Subject of research: backend application architecture based on Internet of Things technology. Scientific novelty: proposed original architecture for "Internet of Things" applications, which provides the ability to monitor people's lives and health. Practical significance: the proposed system can be used in the field of healthcare to monitor patients.

Зміст

ВСТУП

В сучасному світі є немало прикладів поєднання технологій і охорони здоров’я, наприклад всім відомий Apple Watch, який дозволяє слідкувати за серцем, за пройденою дистанцією і т.д. Також є системи «розумний будинок», які спрощують життя людей. Також є приклад поєднання системи «розумний будинок» та охорона здоров’я, це є так звана «розумна палата». Палата в лікарні, яка оснащена під «розумний будинок».

Але було б не погано, якщо такий моніторинг міг відбуватися спеціалістами віддалено, тобто мінімальна кількість необхідного могла б бути вмонтована в систему «розумний будинок», або бути окремою системою, яка основана на технології «розумний будинок». Це могло б дати розвантаження лікарень – не довелося б тримати в палатах людей, яким детальний догляд не потрібен і можна відправити їх додому під нагляд системи і моніторинг показників датчиків дистанційно.

Не менш важливим є те, що відбувається між об’єктом та суб’єктом моніторингу, тобто куди ці дані з датчиків відправляються, що з ними відбувається і як виводяться на екран людині, яка їх зчитує. В цьому є багато складних речей, такі як час затримки, великий трафік, складна логіка роботи, великий обсяг даних, які весь час надходять у систему і які треба зберігати. З цими проблемами зустрічаються при розробці додатків пов’язані з технологіями «великі дані» або «інтернет речей», які і являють собою «розумний будинок».

Є немало архітектурних рішень, які вирішують ті чи інші проблеми з перерахованих. Таким чином можна продовжити і запропонувати нові, вдосконалені існуючі та комбінації з декількох існуючих архітектурних рішень, які б спростили або вирішили існуючи проблеми та тим самим рухаючи технології і дослідження в цих напрямках і виводячи рівень життя та здоров’я людей на новий рівень.

Мета дослідження: розробка архітектури бекенд додатку для «інтернету речей».

Гіпотеза: розробити систему моніторингу, яка даватиме усі необхідні дані з мінімальною затримкою, мінімальним набором сервісів, баз даних, зв'язків та трафіком.

Задачі дослідження:

– Аналіз джерел інформації про «інтернет речей».

– Дослідження архітектурних рішень в технологіях «інтернет речей» та «великі дані».

– Розробка архітектури додатку на основі технології «інтернет речей».

Об’єкт дослідження: архітектура додатку на основі технології «розумний будинок».

Предмет дослідження: архітектура бекенд додатку на основі технології «інтернет речей».

Наукова новизна: запропонована оригінальна архітектура для додатків «інтернет речей», що надає можливість моніторингу за життям і здоров’ям людей.

Практичне значення: запропоновану систему можна застосовувати в сфері охорони здоров’я для моніторингу за пацієнтами. Також бекенд архітектуру можна застосовувати в різних сферах, для здійснення моніторингу за якимось обладнанням, як приклад.

Основна частина складається з 4 розділів:

Розділ 1. Теоретичні відомості про технології які використовуються в сферах «інтернет речей» та «розумний будинок». Тут дано опис основним технологіям, які використовуються для розробки бекенд додатків.

Розділ 2. Архітектурні рішення в додатках «інтернет речей» та «великі дані». Тут описані архітектурні рішення для технології «інтернет речей».

Розділ 3. Архітектура системи моніторингу на основі технології «розумний будинок». Запропонована архітектура системи моніторингу.

Розділ 4. Приклад реалізації описаної архітектури. Продемонстрований приклад реалізації запропонованої архітектури.

За темою роботи опубліковано статтю (Додаток А) – Городний М.С., Тітова В.Ю. Розробка архітектури додатку на основі технологій «розумний будинок» та «інтернет речей» // Актуальні проблеми комп’ютерних наук. Збірник наукових праць за матеріалами XII всеукраїнської науково-практичної конференції «Актуальні проблеми комп’ютерних наук АПКН-2020» – Хмельницький: ХНУ, 2020, Т.1. – С. 75 - 77.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО ТЕХНОЛОГІЇ ЯКІ ВИКОРИСТОВУЮТЬСЯ В СФЕРАХ «ІНТЕРНЕТ РЕЧЕЙ» ТА «РОЗУМНИЙ БУДИНОК»

1.1 Огляд технології «розумний будинок»

«Розумний будинок» являє собою автоматизовану систему управління різними компонентами домашньої інфраструктури. За допомогою спеціального обладнання, система може розпізнавати типові ситуації і реагувати на них, підключаючи ті чи інші компоненти. При цьому, «розумний дім» повністю контролює роботу кожного приладу і не допускає нераціонального їх використання. Таким чином, за рахунок «синергетичного ефекту», розумний будинок дозволяє забезпечити оптимальний режим використання всієї сукупності приладів в будинку. А це дозволяє створити максимально комфортні умови проживання людей при максимально економному споживанні ресурсів.

Вся система «розумний будинок» складається з трьох основних підсистем:

– точка управління: сучасні технології дозволяю забезпечити управління компонентами системи за допомогою самих різних пристроїв. Це може бути як простий вимикач, так і Touch-панель або iPad. Крім того, управління компонентами системи можна здійснювати за допомогою голосу або бавовни долонями. Дистанційне керування забезпечується за допомогою мобільного телефону, SMS та інших подібних рішень;

– центральний контролер: це, власне, головний мозок всієї системи. Саме сюди надходить вся інформація про роботу того чи іншого пристрою. Крім того, центральний контролер отримує і обробляє команди, одержувані від точок управління. Завдяки функціям центрального контролера став можливий ефективний контроль гармонійної роботи всіх приладів в будинку, починаючи від лампочки, до систем вентиляції або опалення;

– виконуючий пристрою: під цим терміном розуміється вся сукупність приладів і систем в будинку. Це можуть бути, як прості прилади, на зразок мікрохвильової печі або музичного центру, так і вельми складні інтелектуальні системи, на зразок системи опалення або системи відеоспостереження.

Передача інформації між підсистемами можлива як за допомогою дротового або бездротовим способом.

Як правило, дана система забезпечує взаємодію кількох систем, інтегрованих в єдину систему управління всіма приладами і комунікаціями будинку. При цьому, найбільш частими компонентами системи «розумний дім» є такі системи:

– електроживлення будинку: розумний будинок контролює наявність електроживлення всіх інших систем в будинку. У разі необхідності, він самостійно задіє джерела безперебійного живлення і, в разі необхідності, додаткові генеруючі потужності;

– освітлення: Оптимальне використання освітлювальних приладів, з урахуванням рівня природного освітлення дозволяє забезпечити максимальну економію електричної енергії. Крім того, система забезпечує оптимальні умови для комфортного проживання людей;

– температура, вологість і своєчасне надходження свіжого повітря. Датчики, що вимірюють вищевказані параметри дають команду систем опалення, вентиляції та кондиціонування в автоматичному режимі. А це означає, що «розумний будинок» завжди підтримує оптимальні параметри повітря в приміщенні;

– управління побутовою технікою: всі прилади, що працюють в будинку, можуть бути об’єднані в єдину мережу. Завдяки цьому, центральний контролер має можливість оптимально організувати роботу відеотехніки і кухонних приладів, систем підігріву ступенів і приводів автоматичних воріт;

– безпека: Сюди входять такі системи, як обмеження доступу в будинок небажаних осіб, контроль витоку газу, сигналізація і відеоспостереження та інші

системи, що дозволяють контролювати рівень безпеки в будинку. Крім того, розумний будинок може забезпечити віддалене інформування про будь-якому інциденті в приміщенні, під час відсутності людей і навіть зімітувати їх присутність;

Слід зауважити, що в кожному конкретному випадку не обов’язкова наявність всіх вищевказаних систем. Технологія «розумний дім» відрізняється гнучкістю, і може бути оптимізована під конкретні вимоги певного клієнта [1].

1.2 Огляд інтернету речей

Інтернет речей (IoT) - це мережа фізичних об'єктів або людей, які називаються "речами", вбудованими в програмне забезпечення, електроніку, мережу та датчики, що дозволяє цим об'єктам збирати та обмінюватися даними. Мета IoT - розширити можливості підключення до Інтернету від стандартних пристроїв, таких як комп’ютер, мобільний телефон, планшет, до відносно німих пристроїв, таких як тостер.

IoT робить практично все «розумним», покращуючи аспекти нашого життя завдяки можливості збору даних, алгоритму ШІ та мереж. Річ у IoT також може бути людиною з імплантатом монітора діабету, твариною з пристроями для відстеження тощо [2].

Робота IoT по-різному залежить від різних екосистем IoT (архітектури). Однак ключова концепція роботи там схожа. Весь робочий процес IoT починається з самих пристроїв, таких як смартфони, цифрові годинники, електронні прилади, які надійно спілкуються з платформою IoT. Платформи збирають та аналізують дані з усіх безлічі пристроїв і платформ і передають найцінніші дані разом із програмами на пристрої [3].

Програмне забезпечення IoT вирішує свої ключові сфери мереж та дій за допомогою платформ, вбудованих систем, партнерських систем та проміжного програмного забезпечення. Ці індивідуальні та головні програми відповідають за

збір даних, інтеграцію пристроїв, аналітику в режимі реального часу та розширення програм та процесів у мережі IoT. Вони використовують інтеграцію з важливими бізнес-системами (наприклад, системами замовлення, робототехнікою, плануванням тощо) при виконанні відповідних завдань.

– збір даних: це програмне забезпечення управляє зондуванням, вимірюваннями, фільтруванням світлових даних, захистом світлових даних та агрегуванням даних. Він використовує певні протоколи, щоб допомогти датчикам підключатися до мереж "машина-машина" в режимі реального часу. Потім він збирає дані з декількох пристроїв і розподіляє їх відповідно до налаштувань. Він також працює в зворотному порядку, розподіляючи дані по пристроях. Врешті-решт система передає всі зібрані дані на центральний сервер;

– інтеграція пристрою: програмне забезпечення, що підтримує інтеграцію, пов'язує (залежні взаємозв'язки) всі системні пристрої для створення корпусу системи IoT. Це забезпечує необхідну співпрацю та стабільну мережу між пристроями. Ці програми є визначальною програмною технологією мережі IoT, оскільки без них вона не є системою IoT. Вони керують різними програмами, протоколами та обмеженнями кожного пристрою, щоб забезпечити зв'язок.

– аналітика в реальному часі: ці програми беруть дані або дані з різних пристроїв і перетворюють їх на життєздатні дії або чіткі схеми для людського аналізу. Вони аналізують інформацію на основі різних налаштувань і конструкцій, щоб виконувати завдання, пов'язані з автоматизацією, або надавати дані, необхідні галузі.

– розширення заявки та процесу: ці програми розширюють діапазон існуючих систем та програмного забезпечення, щоб забезпечити ширшу, ефективнішу систему. Вони інтегрують заздалегідь визначені пристрої для конкретних цілей, таких як надання доступу певним мобільним пристроям або інженерним інструментам. Він підтримує підвищену продуктивність та точніший збір даних [12].

Системи IoT, що застосовуються в охороні здоров’я, покращують існуючі технології та загальну практику медицини. Вони розширюють охоплення професіоналів як в межах об’єкта, так і далеко за його межами. Вони збільшують як точність, так і розмір медичних даних завдяки різноманітному збору даних із великих наборів справжніх справ. Вони також покращують точність надання медичної допомоги завдяки більш досконалій інтеграції системи охорони здоров'я.

Значна частина сучасних медичних досліджень спирається на ресурси, у яких відсутня важлива реальна інформація. Він використовує контрольоване середовище, добровольців та, по суті, залишки для медичного обстеження. IoT відкриває двері для безлічі цінної інформації за допомогою польових даних, аналізу та тестування в реальному часі.

IoT може надавати відповідні дані, які перевершують стандартну аналітику, за допомогою інтегрованих інструментів, здатних проводити життєздатні дослідження. Це також інтегрується у фактичну практику, щоб надати більше ключової інформації. Це сприяє охороні здоров’я, забезпечуючи більш надійні та практичні дані та кращі потенційні клієнти; що дає кращі рішення та виявлення раніше невідомих проблем.

Це також дозволяє дослідникам уникати ризиків, збираючи дані без виготовлених сценаріїв та тестування на людях.

Поточні пристрої швидко покращуються в точності, потужності та доступності; однак вони все ще пропонують менше цих якостей, ніж система IoT, що ефективно інтегрує потрібну систему. IoT розкриває потенціал існуючих технологій і веде нас до нових та кращих рішень для медичних пристроїв.

IoT усуває прогалини між обладнанням та способом надання медичної допомоги, створюючи логічну систему, а не набір інструментів. Потім воно виявляє закономірності та відсутні елементи в охороні здоров’я, такі як очевидні необхідні вдосконалення чи величезні недоліки.

Однією із проблем медичної допомоги є розповсюдження точної та актуальної інформації серед пацієнтів. Охорона здоров'я також бореться з керівництвом, враховуючи складність наступних рекомендацій. Пристрої IoT не тільки покращують обладнання та професійну практику, але й покращують стан здоров’я у повсякденному житті людей.

Пристрої IoT забезпечують прямий цілодобовий доступ до пацієнта менш нав'язливо, ніж інші варіанти. Вони вивозять охорону здоров’я з закладів у будинок, офіс чи соціальний простір. Вони дають можливість людям стежити за власним здоров'ям і дозволяють постачальникам надавати кращу та детальнішу допомогу пацієнтам. Це призводить до зменшення кількості нещасних випадків через неправильне спілкування, покращення рівня задоволеності пацієнтів та покращення профілактичної допомоги.

Удосконалена автоматизація та аналітика IoT забезпечує більш потужні служби аварійної підтримки, які, як правило, страждають від своїх обмежених ресурсів і відключаються від базового обладнання. Це забезпечує спосіб більш повного аналізу надзвичайних ситуацій за милі. Це також надає більшій кількості постачальників послуг доступ до пацієнта до їх прибуття. IoT надає постачальникам важливу інформацію для надання необхідної допомоги після прибуття. Це також підвищує рівень надання допомоги пацієнтові, яку отримують фахівці з надзвичайних ситуацій. Це зменшує пов'язані з цим втрати та покращує екстрене медичне обслуговування [11].

Великі і стають більшими - у світі вже є більше пов’язаних речей, ніж люди.

Компанія-аналітик IDC прогнозує, що загалом до 2025 року буде 41,6 мільярда підключених пристроїв IoT, або "речей". Він також припускає, що промислове та автомобільне обладнання представляють найбільшу можливість пов'язаних "речей", але в найближчій перспективі також спостерігається сильне впровадження розумного будинку та носяться пристроїв.

Ще один технічний аналітик, Гартнер, прогнозує, що цього року на підприємства та автомобільний сектор припаде 5,8 мільярда пристроїв, що майже на чверть порівняно з 2019 роком. Утиліти стануть найвищим користувачем IoT завдяки постійному впровадженню розумних лічильників. Пристрої безпеки у вигляді виявлення зловмисників та веб-камер стануть другим за величиною використанням пристроїв IoT. Автоматизація будівель - як підключене освітлення - буде найбільш швидкозростаючим сектором, за яким слідують автомобільна промисловість (підключені автомобілі) та охорона здоров'я (моніторинг хронічних захворювань).

Промисловий Інтернет речей (IIoT), або четверта промислова революція, або Індустрія 4.0 - все це назви використання технології IoT в бізнес-середовищі. Концепція така ж, як і для побутових пристроїв IoT вдома, але в цьому випадку метою є використання комбінації датчиків, бездротових мереж, великих даних, AI та аналітики для вимірювання та оптимізації промислових процесів.

Якщо його впровадити по всьому ланцюгу поставок, а не лише по окремих компаніях, вплив може бути ще більшим за умови своєчасної доставки матеріалів та управління виробництвом від початку до кінця. Збільшення продуктивності робочої сили або економія витрат - дві потенційні цілі, але IIoT може також створити нові потоки доходів для бізнесу; замість того, щоб просто продавати автономний продукт - наприклад, як двигун - виробники можуть також продавати прогнозне обслуговування двигуна. [17]

1.3 Огляд мікросервісної архітектури

Мікросервісна архітектура – це архітектура, орієнтована на сервіси. В мікросервісній архітектурі існує велика кількість мікросервісів. Поєднуючи всі мікросервіси, створюється великий сервіс. В архітектурі мікросервісів всі служби взаємодіють між собою.

Архітектурний стиль мікросервісів - це підхід до розробки єдиного додатка як набору невеликих сервісів. Кожен мікросервіс запускає свій процес і взаємодіє з легкими механізмами. Ці послуги побудовані на базі бізнес можливостей і незалежно розроблена повністю автоматизована техніка розгортання.

Мікросервіси використовують службове відкриття, яке діє як керівництво для пошуку маршруту зв'язку між кожним з них. Потім мікросервіси взаємодіють між собою за допомогою HTTP запиту або веб сокетів.

Ці мікросервіси взаємодіють між собою за допомогою інтерфейсу прикладних програм (API). Крім того, мікросервіси розгортають статичний вміст у хмарній службі зберігання, яка може доставити їх безпосередньо клієнтам через мережі доставки вмісту (CDN) [4].

Шлюз API – це сервер, який є єдиною точкою входу в систему. Це схоже на фасадний малюнок з об’єктно-орієнтованого дизайну. Шлюз API інкапсулює внутрішню архітектуру системи та надає API, призначений для кожного клієнта. Він може мати інші обов'язки, такі як автентифікація, моніторинг, балансування навантаження, кешування, формування та управління запитами та обробка статичних відповідей.

Для більшості програм, що базуються на мікросервісах, має сенс впровадити шлюз API, який діє як одна точка входу в систему. Шлюз API відповідає за маршрутизацію запитів, композицію та трансформацію протоколів. Він надає кожному з клієнтів програми спеціальний API. Шлюз API також може маскувати збої в сервісах, повертаючи кешовані дані або дані за замовчуванням.

Шлюз API відповідає за маршрутизацію запитів, композицію та переклад протоколів. Усі запити від клієнтів спочатку проходять через шлюз API. Потім він направляє запити до відповідного мікросервісу. Шлюз API часто обробляє запит, включаючи кілька мікросервісів та агрегує результати. Він може перекладати запити між веб-протоколами, такими як HTTP та WebSocket, та веб-недружніми протоколами, які використовуються всередині.

Шлюз API також може надати кожному клієнту власний API. Як правило, для мобільних клієнтів доступний мінімізований API. Розглянемо, наприклад, сценарій деталізації товару. Шлюз API може надати кінцеву точку (/productdetails?Productid=xxx), яка дозволяє мобільному клієнту отримувати всі деталі продукту за допомогою одного запиту. Шлюз API обробляє запит, використовуючи різні послуги - інформацію про товар, рекомендації, огляди тощо - та поєднуючи результати.

Чудовим прикладом шлюзу API є шлюз API Netflix. Потокова послуга Netflix доступна на сотнях різних типів пристроїв, включаючи телевізори, телевізійні приставки, смартфони, ігрові системи, планшети тощо. Спочатку Netflix намагався надати універсальний API для своїх потокових служб. Однак вони виявили, що це не працює добре через різноманітний асортимент пристроїв та їх унікальні потреби. Сьогодні вони використовують API-шлюз, який надає API, адаптований для кожного пристрою за допомогою коду адаптера для конкретного пристрою. Зазвичай адаптер обробляє кожен запит, викликаючи в середньому шість-сім серверних служб. Шлюз API Netflix обробляє мільярди запитів на день [8].

Сервіс-орієнтована архітектура (SOA) – це стиль дизайну програмного забезпечення, при якому послуги надаються іншим компонентам за допомогою компонентів додатків через протокол зв'язку через мережу. Її принципи не залежать від постачальників та інших технологій. В архітектурі, орієнтованій на послуги, низка служб взаємодіють між собою одним із двох способів: шляхом передачі даних або через дві або більше служби, що координують діяльність. Це лише одне з існуючих визначень сервіс-орієнтованої архітектури.

У кожному з будівельних блоків сервіс-орієнтованої архітектури, є три ролі: постачальник послуг; брокер послуг, реєстр послуг, сховище послуг; та запитувач послуг / споживач.

Постачальник послуг працює спільно з реєстром послуг, обговорюючи причини та можливості пропонованих послуг, такі як безпека, доступність, плата та

багато іншого. Ця роль також визначає категорію послуг та наявність необхідних торгових угод.

Брокер послуг робить інформацію з сервісу доступною для тих, хто її запитує. Сфера діяльності брокера визначається тим, хто його впроваджує.

Запитувач послуг знаходить записи в реєстрі посередників, а потім прив’язує їх до постачальника послуг. Вони можуть або не можуть отримати доступ до декількох послуг; це залежить від можливостей запитувача послуг.

Що стосується впровадження сервіс-орієнтованої архітектури (SOA), існує широкий спектр технологій, які можна використовувати, залежно від того, яка ваша кінцева мета і що ви намагаєтесь досягти.

Як правило, сервіс-орієнтована архітектура реалізується з веб-службами, що робить "функціональні будівельні блоки доступними за стандартними протоколами Інтернету".

Прикладом стандарту веб-сервісу є SOAP , що розшифровується як Простий протокол доступу до об'єктів. У двох словах, SOAP - це специфікація протоколу обміну повідомленнями для обміну структурованою інформацією при реалізації веб-служб у комп’ютерних мережах. Хоча спочатку SOAP не приймали добре, з 2003 року він набув більшої популярності і стає все більш широко використовуваним і прийнятим. Інші варіанти реалізації сервіс-орієнтованої архітектури включають Jini, COBRA або REST.

Важливо зазначити, що архітектури можуть «працювати незалежно від конкретних технологій», що означає, що вони можуть бути реалізовані різними способами, включаючи обмін повідомленнями, наприклад, ActiveMQ; Apache Thrift; та SORCER [18].

1.4 Розгляд інших технологій, які використовуються при розробці додатків “інтернет речей” на мікросервісній архітектурі

1.4.1 Контейнери

Контейнер – це стандартна одиниця програмного забезпечення, яке пакує код та всі його залежності, завдяки чому програма працює швидко та надійно від одного обчислювального середовища до іншого. Імідж контейнера Docker - це легкий, автономний, виконуваний пакет програмного забезпечення, що включає все необхідне для запуску програми: код, час роботи, системні інструменти, системні бібліотеки та налаштування.

Іміджи контейнерів стають контейнерами під час виконання, а у випадку контейнерів Docker – іміджи стають контейнерами, коли вони працюють на Docker Engine. Контейнерне програмне забезпечення, доступне як для Linux, так і для Windows, завжди працюватиме однаково, незалежно від інфраструктури. Контейнери ізолюють програмне забезпечення від навколишнього середовища та забезпечують його рівномірну роботу, незважаючи на різницю, наприклад, між розробкою та інсталяцією.

Особливості Docker:

Стандарт: Docker створив галузевий стандарт для контейнерів, щоб їх можна було переносити де завгодно.

Легкий: контейнери мають спільне ядро ОС, тому не потрібна ОС для кожного додатка, що сприяє підвищенню ефективності роботи сервера та зменшує витрати на ліцензування сервера.

Безпека: додатки безпечніші в контейнерах, і Docker забезпечує найсильніші можливості ізоляції за замовчуванням у галузі.

Docker – це служба управління контейнерами. Ключові слова Docker - це розробка, доставка та запуск у будь-якому місці. Вся ідея Docker полягає в тому, щоб

розробники легко розробляли програми, передавали їх у контейнери, які потім можна було розгорнути де завгодно [5].

Kubernetes – інструмент управління контейнерами з відкритим кодом, належить Cloud Native Computing Foundation (CNCF). Він також відомий як вдосконалена версія Borg, яка була розроблена в Google для управління як тривалими процесами, так і пакетними завданнями, яка раніше оброблялася окремими системами. Поставляється з можливістю автоматизувати розгортання, масштабування програми та операції контейнерів програм у кластерах. Він здатний створити інфраструктуру, орієнтовану на контейнери [6].

1.4.2 Сокет

Веб-сокети визначаються як двосторонній зв'язок між серверами та клієнтами, що означає, що обидві сторони спілкуються та обмінюються даними одночасно. Ключовими моментами Web Sockets є справжня паралельність та оптимізація продуктивності, що призводить до більш гнучких та розширених веб-додатків [7].

1.4.3 Хмарні обчислення

Хмарні обчислення визначаються як зберігання та доступ до даних та обчислювальних послуг через Інтернет. Він не зберігає жодних даних на вашому персональному комп’ютері. Це доступність комп’ютерних послуг на замовлення, таких як сервери, сховище даних, мережа, бази даних тощо. Основна мета хмарних обчислень – надати доступ до центрів обробки даних багатьом користувачам. Користувачі також можуть отримати доступ до даних із віддаленого сервера.

Приклад: AWS, Azure, Google Cloud [9].

1.4.4 Брокери повідомлень

Брокер повідомлень – це програмне забезпечення, яке дозволяє програмам, системам та послугам взаємодіяти між собою та обмінюватися інформацією. Брокер повідомлень робить це, перекладаючи повідомлення між офіційними протоколами обміну повідомленнями. Це дозволяє взаємозалежним службам «спілкуватися» безпосередньо між собою, навіть якщо вони написані різними мовами або реалізовані на різних платформах.

Брокери повідомлень – це програмні модулі в рамках проміжного програмного забезпечення для обміну повідомленнями або орієнтованого на повідомлення проміжного програмного забезпечення (MOM). Цей тип проміжного програмного забезпечення надає розробникам стандартизовані засоби обробки потоку даних між компонентами програми, щоб вони могли зосередитися на його основній логікі. Він може служити розподіленим комунікаційним рівнем, який дозволяє програмам, що охоплюють кілька платформ, взаємодіяти внутрішньо.

Брокери повідомлень можуть перевіряти, зберігати, направляти та доставляти повідомлення до відповідних пунктів призначення. Вони служать посередниками між іншими програмами, дозволяючи відправникам надсилати повідомлення, не знаючи, де знаходяться приймачі, чи вони активні чи ні, чи скільки їх існує. Це полегшує роз'єднання процесів та послуг у системах.

Щоб забезпечити надійне зберігання повідомлень і гарантовану доставку, брокери повідомлень часто покладаються на підструктуру або компонент, який називається чергою повідомлень, який зберігає та впорядковує повідомлення, поки програми-споживачі не зможуть їх обробити. У черзі повідомлень повідомлення зберігаються в точному порядку, в якому вони були передані, і залишаються в черзі до підтвердження отримання.

Асинхронний обмін повідомленнями відноситься до типу взаємодії між програмами, яку роблять можливі брокери повідомлень. Це запобігає втраті цінних

даних і дозволяє системам продовжувати функціонувати навіть в умовах переривчастих проблем підключення або затримки, поширених у загальнодоступних мережах . Асинхронний обмін повідомленнями гарантує, що повідомлення будуть доставлені один раз (і лише один раз) у правильному порядку щодо інших повідомлень.

Брокери повідомлень можуть містити менеджерів черг для обробки взаємодій між кількома чергами повідомлень, а також служб, що забезпечують маршрутизацію даних, переклад повідомлень, збереження та функціональність управління станом клієнта.

REST API зазвичай використовуються для зв'язку між мікросервісами. Термін REST визначає набір принципів та обмежень, яких розробники можуть дотримуватися при створенні веб-сервісів. Будь-які служби, які їх дотримуються, зможуть спілкуватися через набір єдиних спільних операторів та запитів. Прикладний програмний інтерфейс (API) позначає базовий код, який, якщо він відповідає правилам REST, дозволяє службам спілкуватися між собою.

REST API використовують для зв’язку протокол передачі гіпертексту (HTTP). Оскільки HTTP є стандартним транспортним протоколом загальнодоступного Інтернету, API REST є широко відомим, часто використовуваним та широко взаємодіючим. Однак HTTP є протоколом запиту / відповіді, тому його найкраще використовувати в ситуаціях, коли потрібен синхронний запит / відповідь. Це означає, що служби, що надсилають запити через REST API, повинні бути спроектовані на очікування негайної відповіді. Якщо клієнт, який отримує відповідь, не працює, служба відправлення буде заблокована, поки вона чекає відповіді. В обидві служби слід вбудувати логіку відновлення після відмови та обробки помилок.

Брокери повідомлень дозволяють асинхронний зв’язок між сервісами, так що сервіс-відправнику не потрібно чекати відповіді сервісу-отримувачу. Це покращує стійкість до відмов і стійкість в системах, в яких вони працюють. Крім того, використання посередників повідомлень полегшує масштабування систем, оскільки

шаблон обміну повідомленнями pub / sub може легко підтримувати зміну кількості послуг. Брокери повідомлень також відстежують стан споживачів [10].

У Big Data використовується величезний обсяг даних. Що стосується даних, у нас є дві основні проблеми: перша проблема полягає в тому, як зібрати великий обсяг даних, а друга проблема полягає в аналізі зібраних даних. Щоб подолати ці виклики, вам потрібна система обміну повідомленнями.

Kafka призначена для розподілених систем з високою пропускною здатністю. Кафка, як правило, працює дуже добре як заміна більш традиційному брокеру повідомлень. У порівнянні з іншими системами обміну повідомленнями, Kafka має кращу пропускну здатність, вбудоване секціонування, реплікацію та властиву відмовостійкість, що робить її придатною для широкомасштабних додатків обробки повідомлень.

1.4.5 Кешування

Кешування покращує час відгуку програми, зберігаючи копії найбільш часто використовуваних даних в швидкоплинному і дуже швидкому сховищі. Рішення кешування в пам'яті, які утримують робочий набір у швидкій DRAM замість повільних обертових дисків, можуть бути надзвичайно ефективними для досягнення цих цілей. Хоча кешування зазвичай використовується для покращення затримки програми, високодоступний та еластичний кеш також може допомогти масштабувати програми. Розвантаження обов’язків з основної логіки програми на рівень кешування звільняє обчислювальні ресурси для обробки більшої кількості вхідних запитів.

Варіанти використання додатків для кешування:

– зберігання даних СУБД. Більшість традиційних баз даних розроблені для забезпечення надійної функціональності, а не швидкості в масштабі. Кеш бази даних часто використовується для зберігання копій таблиць пошуку та відповідей на дорогі

запити зі СУБД, щоб поліпшити продуктивність програми та зменшити навантаження на джерело даних;

– дані сеансу користувача. Кешування даних сеансу користувача є невід’ємною частиною побудови масштабованих та адаптивних програм. Оскільки кожна взаємодія користувача вимагає доступу до даних сеансу, збереження цих даних у кеші пришвидшує час відповіді користувачеві програми. Утримувати дані сеансу в кеші краще, ніж підтримувати липкість сеансів на рівні балансатора навантаження, оскільки кешування дозволяє обробляти запити будь-яким сервером додатків без втрати станів користувачів, тоді як підхід балансування навантаження ефективно примушує всі запити в сеансі для обробки одним сервером додатків;

– швидкий доступ до відповідей API. Сучасні додатки побудовані з використанням слабозв'язаних компонентів, які взаємодіють через API. Компоненти додатків використовують API для надсилання запитів на обслуговування від інших компонентів, будь то всередині (архітектура мікропослуг) або зовні (у випадку використання програмного забезпечення як послуги) самого додатка. Зберігання відповіді API у кеш-пам'яті, навіть короткочасне, покращує продуктивність програми, уникаючи цього міжпроцесорного спілкування;

Redis – це розширене сховище ключових значень із відкритим кодом та пристосоване рішення для створення високопродуктивних, масштабованих веб-додатків.

Надзвичайно швидкий – Redis дуже швидкий і може виконувати близько 110000 SETs в секунду, приблизно 81000 GETs в секунду.

Підтримує розширені типи даних – Redis спочатку підтримує більшість типів даних, які розробники вже знають, такі як список, набір, відсортований набір та хеші. Це полегшує вирішення різноманітних проблем, оскільки ми знаємо, з якою проблемою можна краще впоратися за допомогою якого типу даних.

Операції є атомарними – усі операції Redis є атомарними, що гарантує, що при одночасному доступі двох клієнтів сервер Redis отримає оновлене значення.

Багатофункціональний інструмент – Redis – це багатофункціональний інструмент, який можна використовувати в багатьох випадках, таких як кешування, черги обміну повідомленнями (Redis підтримує публікацію / підписку), будь-які короткочасні дані у вашому додатку, такі як веб сеанси додатків, кількість переглядів веб-сторінок тощо.

1.5 Опис архітектури інтернету речей

Приклад автоматичної системи моніторингу для пацієнтів літнього віку вимагає збору даних та аналізу в режимі реального часу, підключення до мережі для доступу до послуг інфраструктури та програми для підтримки користувальницького інтерфейсу та відображення. Отже, її архітектура повинна включати датчики тіла для збору даних про пацієнта, шлюзи для фільтрації та пересилання даних, мікроконтролери або мікропроцесори для аналізу та бездротової передачі даних у хмару, а також інструмент зв'язку для передачі даних у віддалене місце, як в службу порятунку або охорону здоров'я для моніторингу та відстеження.

Архітектура IoT для системи складається з трьох етапів: фізичного, комунікаційного та прикладного. Перший шар має багатосенсорну мережу, яка оцінює життєві показники пацієнта, такі як харчування, медичне споживання та фізичні навантаження. Також до фізичного рівня входить інша мережа моніторингу, яка складається із власних датчиків та виконавчих механізмів для підтримки якості повітря, температури та аналізу та визначення будь-яких небезпечних умов для пацієнта. Другий рівень включає в себе OT-пристрої, які збирають інформацію, зібрану датчиками, перетворюють її у значущі потоки даних і передають їх до внутрішнього призначення. Третій рівень – це місце, де дані приймаються, зберігаються та обробляються за допомогою хмарних механізмів аналізу даних та механізмів машинного навчання.

Представлена система моніторингу охорони здоров’я повинна забезпечувати доступність для різних користувачів. Наприклад, медичний працівник, сам пацієнт та будь-які члени сім'ї чи доглядачі. З огляду на це, однією з проблем використання IoT в рамках моніторингу охорони здоров’я є забезпечення безпеки даних та конфіденційності. Безпеки можна досягти за допомогою шифрування при передачі даних. Прикладом є використання мікропроцесора, який забезпечує безпечний метод комунікації через шифрований рівень сокету (SSL) [13].

IoT, безсумнівно, трансформує галузь охорони здоров’я, переосмислюючи простір взаємодії пристроїв та людей при наданні рішень у галузі охорони здоров’я. IoT має програми в галузі охорони здоров’я, які приносять користь пацієнтам, сім’ям, лікарям, лікарням та страховим компаніям.

IoT для пацієнтів – пристрої у формі носимих пристроїв, такі як фітнес-стрічки та інші бездротові пристрої, такі як манжети для контролю артеріального тиску та серцевого ритму, глюкометр тощо надають пацієнтам доступ до персоналізованої уваги. Ці пристрої можна налаштувати, щоб нагадувати про кількість калорій, перевірку фізичних вправ, зустрічі, коливання артеріального тиску та багато іншого.

IoT змінив життя людей, особливо літніх пацієнтів, завдяки постійному відстеженню стану здоров'я. Це має великий вплив на людей, які живуть поодинці, та їхні сім’ї. Про будь-які порушення або зміни у звичній діяльності людини механізм оповіщення надсилає сигнали членам сім'ї та зацікавленим медичним працівникам.

IoT для лікарів – використовуючи носимі пристрої та інше обладнання для домашнього моніторингу, вбудоване в IoT, лікарі можуть ефективніше відстежувати стан здоров’я пацієнтів. Вони можуть відстежувати дотримання пацієнтами планів лікування або необхідність негайної медичної допомоги. IoT дозволяє медичним працівникам бути пильнішими та активніше спілкуватися з пацієнтами. Дані, зібрані

з пристроїв IoT, можуть допомогти лікарям визначити найкращий процес лікування пацієнтів та досягти очікуваних результатів.

IoT для лікарень – окрім контролю за станом здоров'я пацієнтів, є багато інших областей, де IoT-пристрої дуже корисні в лікарнях. Пристрої IoT, позначені датчиками, використовуються для відстеження місцезнаходження медичного обладнання в реальному часі, такого як інвалідні візки, дефібрилятори, небулайзери, кисневі насоси та інше обладнання для моніторингу. Розміщення медичного персоналу в різних місцях також можна аналізувати в режимі реального часу.

Поширення інфекцій є основною проблемою для пацієнтів у лікарнях. Пристрої моніторингу гігієни з підтримкою IoT допомагають запобігти зараженню пацієнтів. Пристрої IoT також допомагають в управлінні активами, як контроль запасів аптек та моніторинг навколишнього середовища, наприклад, перевірка температури холодильника, контролю вологості та температури.

IoT для медичних страхових компаній – існує безліч можливостей для страховиків із підключеними до IoT інтелектуальними пристроями. Страхові компанії можуть використовувати дані, отримані за допомогою пристроїв моніторингу стану здоров'я, для здійснення своїх андеррайтингів та операцій зі збитками. Ці дані дозволять їм виявляти претензії щодо шахрайства та визначати перспективи андеррайтингу. Пристрої IoT забезпечують прозорість між страховиками та клієнтами у процесі андеррайтингу, ціноутворення, обробки претензій та оцінки ризиків. У світлі рішень, керованих даними IoT, у всіх операційних процесах, клієнти матимуть достатню видимість основних думок кожного прийнятого рішення та результатів процесу.

Страховики можуть запропонувати своїм клієнтам заохочення за використання та обмін даними про стан здоров'я, створені пристроями IoT. Вони можуть винагородити клієнтів за використання пристроїв IoT для відстеження їх рутинної діяльності та дотримання планів лікування та запобіжних заходів охорони здоров'я. Це допоможе страховикам значно зменшити претензії. Пристрої IoT також

можуть дозволити страховим компаніям перевіряти претензії за допомогою даних, що збираються цими пристроями.

1.6 Опис баз даних для додатків “інтернет речей”

Почнемо з розуміння фундаментальної різниці між статичною та потоковою базами даних. Статичні бази даних, також відомі як пакетні бази даних, керують даними в спокої. Дані, до яких користувачі повинні отримати доступ, містяться як збережені дані, керовані системою управління базами даних (СУБД). Користувачі роблять запити та отримують відповіді від СУБД, яка зазвичай, але не завжди, використовує SQL. Потокова база даних обробляє дані в русі. Дані постійно передаються через базу даних із безперервною низкою заданих запитів, як правило, мовою, специфічною для потокової бази даних. Висновок потокової бази даних може в кінцевому підсумку зберігатися в іншому місці, наприклад, у хмарі, і отримати доступ за допомогою стандартних механізмів запитів.

Потокові бази даних зазвичай розподіляються для обробки вимог до масштабу та завантаження великих обсягів даних. В даний час існує цілий ряд комерційних, власних баз даних та потокових баз даних із відкритим кодом, включаючи Google Cloud Dataflow, Microsoft StreamInsight, Azure Stream Analytics, IBM InfoSphere Streams та Amazon Kinesis. Системи з відкритим кодом значною мірою базуються на Apache і включають Apache Spark Streaming, що надається Databricks, Apache Flink від Data Artisans, Apache Kafka від Confluent та Apache Storm, який належить Twitter. Організації в основному використовують потокові бази даних для прийняття рішень у режимі реального часу та для задоволення вимог затримки майже миттєво.

Однак організації все ще можуть скористатися стандартними методами та схемами запитів, саме тому багато потокових баз даних також містять статичний компонент бази даних. Ці уніфіковані бази даних поєднують найкраще з обох світів

потокової та статичної баз даних, оскільки вони підтримують як можливості потокової бази даних у реальному часі, так і гнучкість процесу запитів та схеми статичної бази даних. Найкраща база даних для більшості програм IoT – це уніфікована база даних, яка поєднує як потокові, так і статичні можливості. З цієї причини більшість популярних баз даних постачальників включають обидва типи баз даних.

Бази даних SQL – це реляційні та мають статичні схеми, що описують порядок організації інформації. Це робить їх дуже керованими. Однак вони стикаються з проблемами масштабування ефективно. Бази даних NoSQL є нереляційними, не мають схем і, як правило, рекламуються як високо масштабовані та ефективніші, ніж бази даних SQL.

Найкраща база даних для більшості програм IoT – це уніфікована база даних, яка поєднує як потокові, так і статичні можливості.

Деякі технічні фахівці можуть подумати, що база даних NoSQL буде очевидним вибором, оскільки масштабованість є важливою для багатьох видів використання IoT. Але масштабованість та продуктивність - це лише два фактори, які технологам потрібно враховувати при виборі баз даних. Критичним фактором у багатьох сценаріях є простота інтеграції в існуючі системи, де SQL є більш ефективним. Багато інструментів і систем IoT припускають SQL. Особливо це стосується промислових середовищ, які базуються на старих протоколах повідомлень або платформах промислової автоматизації.

Вміння створювати схеми та керувати ними також є плюсом. Незважаючи на те, що технологи можуть визнати розробку схеми стримуючою, інформація повинна бути організованою. Якщо докласти зусиль для розробки схем заздалегідь, це економить значні зусилля для подальшої організації даних у середовищі, що не є схемою.

Організації також можуть виявити складним поєднання статичних та потокових баз даних, включаючи вибір між SQL та NoSQL. Теоретично статичною

базою даних або потоковою базою даних може бути SQL або NoSQL. На практиці бази даних спеціально встановлюються для того чи іншого. Технологи IoT, зацікавлені в певній уніфікованій базі даних, можуть знайти рішення щодо SQL та NoSQL, зумовлене дизайном бази даних.

Чи повинна організація вибирати базу даних SQL або NoSQL, залежить від більш широкого набору функціональних та технічних вимог, зокрема масштабованості, продуктивності та потреби інтеграції у застарілі системи [14].

Найпопулярніші бази даних:

– Oracle. Він справді відомий серед усіх розробників, простий у використанні, добре написані документи, дивовижні нові функції, такі як JSON від SQL тощо:

– MySQL. Підприємства можуть почати використовувати безкоштовний сервер спільноти і пізніше оновити комерційну версію:

– Microsoft SQL Server - це реляційна система управління базами даних, побудована для основної функції зберігання отримання даних відповідно до вимог інших програм:

– PostgreSQL (вимовляється як post-gress-QL) - це система управління реляційними базами даних з відкритим кодом, розроблена всесвітньою командою добровольців:

– MongoDB - це міжплатформна, орієнтована на документи база даних, яка забезпечує високу продуктивність, високу доступність та просту масштабованість:

MongoDB – це міжплатформна, орієнтована на документи база даних, яка забезпечує високу продуктивність, високу доступність та легку масштабованість. MongoDB працює над концепцією колекції та документа.

База даних - це фізичний контейнер для колекцій. Кожна база даних отримує власний набір файлів у файловій системі. Один сервер MongoDB зазвичай має кілька баз даних.

Колекція – це група документів MongoDB. Це еквівалент таблиці RDBMS. Колекція існує в межах однієї бази даних. Колекції не застосовують схему. Документи в колекції можуть мати різні поля. Як правило, усі документи в колекції мають подібне або пов'язане призначення.

Документ – це набір пар ключ-значення. Документи мають динамічну схему. Динамічна схема означає, що документи в одній колекції не повинні мати однаковий набір полів або структури, а загальні поля в документах колекції можуть містити різні типи даних.

– DB2 – це продукт баз даних від IBM. DB2 призначений для ефективного зберігання, аналізу та отримання даних.

– Redis. Має відкритий код, вдосконалене сховище ключ-значення та підходяще рішення для створення високопродуктивних масштабованих веб-додатків.

– Elasticsearch. Це розподілений у повному обсязі повнотекстовий механізм пошуку та аналітики з відкритим кодом.

– SQLite – це реляційна система управління базами даних, що міститься в бібліотеці C. На відміну від багатьох інших систем управління базами даних, SQLite не є механізмом баз даних клієнт-сервер. Швидше, це вбудовано в кінцеву програму. Найкраще для мобільних додатків.

– Microsoft Access. Це система управління базами даних від Microsoft, яка поєднує реляційний движок баз даних Jet з графічним інтерфейсом користувача та засобами розробки програмного забезпечення [15].

1.7 Опис мов програмування для додатків “інтернет речей”

Scala – кросовер з об'єктно-орієнтованого і функціонального програмування парадигм, Scala є швидким і надійним, а також популярний вибір мови для Big Data. Зазначимо, що два з найбільш популярних механізмів обробки великих даних в Apache Spark і Apache Kafka побудовані на Scala, розповідає все, що вам потрібно знати про потужність Scala.

Scala працює на JVM, що означає, що коди, написані в Scala, можуть бути легко використані в екосистемі Big Data на основі Java. Однак важливим фактором, який відрізняє Scala від Java , є те, що Scala набагато менш багатослівна в порівнянні. Ви можете написати 100 рядків Java-коду, заплутаного на вигляд, і у менш ніж у 15 рядків на Scala. Однак одним негативним аспектом Scala є її важкість навчання у порівнянні з такими мовами, як Go та Python , і це може відлякати початківців, які прагнуть використовувати її.

Python - відповідно до нещодавно проведеного опитування розробників Stack Overflow, Python було оголошено однією з найбільш швидкозростаючих мов програмування у 2018 році. Його загальний характер означає, що його можна використовувати в широкому спектрі випадків використання, а програмування великих даних є однією з основних областей застосування.

Багато бібліотек для аналізу та маніпулювання даними, які все частіше використовуються в рамках Big Data для очищення та маніпулювання великими шматками даних, такими як NumPy, SciPy, - все це на основі Python. Крім того найпопулярніші механізми машинного навчання та глибокого навчання, такі як scikit-learn, Tensorflow та багато іншого, також написані на Python і знаходять все більше застосування в екосистемі великих даних.

Одним недоліком використання Python і причиною того, що він ще не є першокласним громадянином, коли мова заходить про програмування великих даних, є те, що він повільний. Хоча дуже простий у використанні, фахівці Big Data

виявили, що системи, побудовані на таких мовах, як Java або Scala, швидші та надійніші у використанні, ніж системи, побудовані на Python.

Однак Python компенсує це обмеження іншими якостями. Оскільки Python - це в першу чергу мова сценаріїв, інтерактивне кодування та розробка аналітичних рішень для великих даних стає дуже простим. Python може без особливих зусиль інтегруватися з існуючими фреймворками великих даних, такими як Apache Hadoop та Apache Spark, що дозволяє виконувати прогнозовану аналітику в масштабі без будь-яких проблем.

R (мова програвмування) – для багатьох не буде несподіванкою те, що ті, хто любить статистику, люблять R. “Мова статистики”, як її в народі називають, використовується для побудови моделей даних, які можна використовувати для ефективного та точного аналізу даних.

Працюючи на великому сховищі пакетів R (CRAN, також Комплексна мережею архівів R), за допомогою R ви маєте майже кожен тип інструменту для виконання будь-якого завдання в обробці великих даних - від аналізу до візуалізації даних. R можна легко інтегрувати з Apache Hadoop та Apache Spark, та серед інших популярних фреймворків, для обробки та аналізу великих даних.

Одне з питань використання R як мови програмування для Big Data полягає в тому, що він не дуже загальний. Це означає, що код, написаний на R, не можна розгорнути у виробництві, і, як правило, його потрібно перекласти на якусь іншу мову програмування, таку як Python або Java. Тим не менш, якщо ваша мета полягає лише у створенні статистичних моделей для аналізу великих даних, R - це варіант, який ви обов'язково повинні розглянути.

Java – не менш важливим є те, що завжди є стара стара добра Java. Деякі традиційні фреймворки великих даних, такі як Apache Hadoop та всі інструменти в його екосистемі, засновані на Java і використовуються сьогодні на багатьох підприємствах. Не кажучи вже про той факт, що Java є найбільш стабільною та готовою до виробництва мовою серед усіх мов, які ми обговорювали до цього часу!

Використання Java для розробки додатків для великих даних дає вам можливість використовувати велику екосистему інструментів та бібліотек для взаємодії, моніторингу та багато іншого, більшість з яких вже випробувано та перевірено.

Одним з основних недоліків Java є її багатослівність. Той факт, що вам потрібно написати сотні рядків кодів на Java для завдання, яке може писати ледь 15-20 рядків коду на Python або Scala, може призвести до втрати багатьох початківців програмістів. Однак впровадження лямбда-функцій у Java 8 справді полегшує життя. Java також не підтримує ітераційну розробку, на відміну від нових мов, таких як Python, і це зосереджується на майбутніх випусках Java.

Незважаючи на недоліки, Java залишається сильним суперником, коли мова заходить про бажану мову для програмування великих даних, через свою історію та постійну залежність від традиційних інструментів та фреймворків великих даних.

Spring framework – це платформа Java з відкритим кодом, яка забезпечує всебічну підтримку інфраструктури для розробки надійних програм Java дуже легко і дуже швидко. Spring framework був спочатку написаний Родом Джонсоном і вперше випущений за ліцензією Apache 2.0 у червні 2003 року. Цей посібник написаний на основі Spring Framework версії 4.1.6, випущеної в березні 2015 року.

Нижче наведено перелік декількох великих переваг використання Spring Framework.

Основана на POJO – Spring дозволяє розробникам розробляти додатки корпоративного класу за допомогою POJO. Перевага використання лише POJO полягає в тому, що вам не потрібен продукт-контейнер EJB, такий як сервер додатків, але у вас є можливість використовувати лише надійний контейнер сервлетів, такий як Tomcat або якийсь комерційний продукт.

Модульність – Spring організован за модульним способом. Незважаючи на те, що кількість пакетів та класів значна, вам доведеться турбуватися лише про ті, які вам потрібні, а решту ігнорувати.

Інтеграція з існуючими фреймворками – Spring не винаходить колеса, натомість він по-справжньому використовує деякі існуючі технології, такі як кілька фреймворків ORM, фреймворки реєстрації, таймери JEE, Quartz та JDK та інші технології.

Тестування – тестування програми, написаної за допомогою Spring, є простим, оскільки в цей фреймворк переміщується код, що залежить від середовища. Крім того, за допомогою POJO JavaBeans стає легше використовувати ін’єкцію залежностей для ін’єкції тестових даних.

Web MVC – веб-фреймворк Spring – це добре розроблений веб-фреймворк MVC, який забезпечує чудову альтернативу веб-фреймворкам, таким як Struts чи інші надмірно розроблені або менш популярні веб-фреймворки.

Центральна обробка винятків – Spring надає зручний API для перекладу винятків, характерних для технологій (наприклад, JDBC, Hibernate або JDO), у послідовні, неперевірені винятки.

Легкий – легкі контейнери IoC, як правило, легкі, особливо у порівнянні з контейнерами EJB, наприклад. Це вигідно для розробки та розгортання додатків на комп'ютерах з обмеженою пам'яттю та ресурсами центрального процесора.

Управління транзакціями – Spring забезпечує послідовний інтерфейс управління транзакціями, який може масштабуватися до локальної транзакції (наприклад, з використанням однієї бази даних) та масштабуватися до глобальних транзакцій (наприклад, за допомогою JTA).

Spring потенційно може бути універсальним магазином для всіх ваших корпоративних програм. Однак Spring є модульним, що дозволяє вам вибрати, які модулі вам підходять, без необхідності вводити решту.

Go (Мова програмування) – не менш важливим є Go - одна з найбільш швидкозростаючих мов програмування за останній час. Розроблений групою інженерів Google, які були розчаровані C++ , ми вважаємо, що Go є хорошим вигуком у цьому списку - просто через той факт, що він використовує стільки

інструментів, що використовуються в інфраструктурі великих даних, включаючи Kubernetes, Docker та багато інших.

Go є швидким, простим у вивченні та досить простим для розробки додатків, не кажучи вже про їх розгортання. Що ще важливіше, оскільки бізнес розглядає створення систем аналізу даних, які можуть працювати в масштабі, системи на основі Go використовуються для інтеграції машинного навчання та паралельної обробки даних. Також можна відносно легко взаємодіяти з іншими мовами із системами на базі Go [16].

Тут ми розглянули основні технології з якими працюють при роботі з бекенд частиною. На основі цих технологій буде будуватися бекенд частина та будуть згадані в подальшому виконанні.

2 АРХІТЕКТУРНІ РІШЕННЯ В ДОДАТКАХ «ІНТЕРНЕТ РЕЧЕЙ» ТА «ВЕЛИКІ ДАНІ»

2.1 Шаблон конкуруючих споживачів

Дозволяє кільком одночасним споживачам обробляти повідомлення, отримані в одному каналі обміну повідомленнями. Це дозволяє системі одночасно обробляти кілька повідомлень для оптимізації пропускної здатності, поліпшення масштабованості та доступності та збалансування навантаження.

Контекст і проблема.

Очікується, що програма, яка працює в хмарі, обробляє велику кількість запитів. Замість того, щоб обробляти кожен запит синхронно, загальноприйнятою методикою є те, що програма передає їх через систему обміну повідомленнями до іншої служби (служби споживачів), яка обробляє їх асинхронно. Ця стратегія допомагає гарантувати, що бізнес-логіка програми не блокується під час обробки запитів.

Кількість запитів може суттєво змінюватися з часом з багатьох причин. Раптове збільшення активності користувачів або агреговані запити, що надходять від декількох сервісів, можуть спричинити непередбачуване навантаження. У години пік системі може знадобитися обробляти багато сотень запитів в секунду, тоді як в інший час кількість може бути дуже малою. Крім того, характер роботи, що виконується для обробки цих запитів, може бути дуже різним. Використання одного екземпляра споживчої послуги може призвести до того, що цей екземпляр заповниться запитами, або система обміну повідомленнями може бути перевантажена напливом повідомлень, що надходять із програми. Щоб впоратися з цим коливанням робочого навантаження, система може запустити кілька екземплярів послуги. Однак ці споживачі повинні бути скоординованими, щоб гарантувати, що кожне повідомлення доставляється лише одному споживачеві.

Рішення. Використовуйте чергу повідомлень для реалізації каналу зв'язку між додатком та екземплярами сервісів-споживачів. Додаток публікує запити у вигляді повідомлень до черги, а екземпляри служби обслуговування споживачів отримують повідомлення з черги та обробляють їх. Цей підхід дозволяє одному пулу примірників побутових послуг обробляти повідомлення з будь-якого примірника програми.

Це рішення має наступні переваги:

– забезпечує систему з рівнем навантаження, яка може обробляти широкі зміни в обсязі запитів, надісланих екземплярами додатків. Черга діє як буфер між екземплярами програми та сервісів-споживачів. Це може допомогти мінімізувати вплив на доступність та швидкість реагування як для програми, так і для екземплярів служби. Обробка повідомлення, яке вимагає певної тривалої обробки, не заважає одночасно обробляти інші повідомлення іншими екземплярами служби споживачів;

– підвищує надійність. Якщо виробник спілкується безпосередньо зі споживачем замість того, щоб використовувати цей шаблон, але не контролює споживача, існує велика ймовірність того, що повідомлення можуть бути втрачені або не оброблені, якщо споживач зазнає невдачі. За цим шаблоном повідомлення не надсилаються до певного екземпляра служби. Невдалий екземпляр служби не заблокує виробника, і повідомлення можуть обробляти будь-який робочий екземпляр служби;

– не вимагає складної координації між споживачами або між виробником та споживачами. Черга повідомлень гарантує, що кожне повідомлення доставляється принаймні один раз;

– це масштабовано. Система може динамічно збільшувати або зменшувати кількість випадків побутової послуги, коли обсяг повідомлень коливається;

– може покращити стійкість, якщо черга повідомлень забезпечує транзакційні операції зчитування. Якщо екземпляр служби обслуговування

споживачів зчитує та обробляє повідомлення як частину транзакційної операції, а екземпляр служби обслуговування споживачів не відповідає, цей шаблон може гарантувати, що повідомлення буде повернуто в чергу, щоб його забрати та обробити інший екземпляр служби обслуговування споживачів;

Враховуйте наступні моменти, вирішуючи, як реалізувати цей шаблон:

– впорядкування повідомлень. Порядок отримання повідомлень споживчих служб не гарантується і не обов'язково відображає порядок створення повідомлень. Спроектуйте систему так, щоб обробка повідомлень була не важливою, оскільки це допоможе усунути будь-яку залежність від порядку обробки повідомлень;

– проектування служб на стійкість. Якщо система призначена для виявлення та перезапуску невдалих екземплярів служби, може знадобитися реалізувати обробку, виконувану екземплярами служби, як ідемпотентні операції, щоб мінімізувати ефекти одного повідомлення, яке отримується та обробляється більше одного разу;

– виявлення «токсичних» повідомлень. Неправильно сформоване повідомлення або завдання, яке вимагає доступу до недоступних ресурсів, може спричинити збій екземпляра служби. Система повинна запобігати поверненню таких повідомлень до черги, а замість цього фіксувати та зберігати деталі цих повідомлень в іншому місці, щоб при необхідності їх можна було проаналізувати;

– обробка результатів. Екземпляр служби, який обробляє повідомлення, повністю відокремлений від логіки програми, яка генерує повідомлення, і вони можуть не мати можливості безпосереднього зв’язку. Якщо екземпляр служби генерує результати, які повинні бути передані назад логіці програми, ця інформація повинна зберігатися в місці, доступному для обох. Для того щоб логіка програми не отримувала неповні дані, система повинна вказати, коли обробка завершена;

– масштабування системи обміну повідомленнями. У широкомасштабному рішенні одна черга повідомлень може бути перевантажена кількістю повідомлень і стати вузьким місцем у системі. У цій ситуації розгляньте розділення системи обміну повідомленнями для надсилання повідомлень від конкретних виробників до певної черги або скористайтеся балансуванням навантаження для розподілу повідомлень по декількох чергах повідомлень;

– забезпечення надійності системи обміну повідомленнями. Потрібна надійна система обміну повідомленнями, щоб гарантувати, що після того, як програма додасть повідомлення в чергу, воно не буде втрачено. Це важливо для забезпечення того, щоб усі повідомлення доставлялися принаймні один раз.

Використовуйте цей шаблон, коли:

– навантаження програми розбито на завдання, які можуть виконуватися асинхронно;

– завдання незалежні і можуть виконуватися паралельно;

– обсяг роботи дуже мінливий, що вимагає масштабованого рішення;

– рішення повинно забезпечувати високу доступність і повинно бути стійким, якщо обробка завдання не вдається.

Цей шаблон може бути корисним, коли:

– нелегко розділити навантаження програми на окремі завдання, або існує велика ступінь залежності між завданнями;

– завдання повинні виконуватися синхронно, а логіка програми повинна чекати завершення завдання, перш ніж продовжувати;

– завдання повинні виконуватися в певній послідовності.

Деякі системи обміну повідомленнями підтримують сеанси, що дозволяють виробнику групувати повідомлення та гарантувати, що всі вони обробляються одним і тим же споживачем. Цей механізм може бути використаний з пріоритетними повідомленнями (якщо вони підтримуються) для реалізації форми упорядкування

повідомлень, яка послідовно доставляє повідомлення від виробника до одного споживача [20].

2.2 Архітектурний стиль, керований подіями (Event driven)

Архітектура, керована подіями, складається з виробників подій**,** які генерують потік подій, та споживачів подій, які слухають події.

Події доставляються майже в реальному часі, тому споживачі можуть негайно реагувати на події, коли вони відбуваються. Виробники відокремлені від споживачів – виробник не знає, кого споживачі слухають. Споживачі також відокремлені один від одного, і кожен споживач бачить усі події. Це відрізняється від шаблону конкуруючих споживачів, коли споживачі витягують повідомлення з черги, а повідомлення обробляється лише один раз (без помилок). У деяких системах, таких як IoT, події повинні сприйматися у дуже великих обсягах.

Архітектура, керована подіями, може використовувати модель pub / sub або модель потоку подій.

- pub / sub: інфраструктура обміну повідомленнями відстежує підписки. Коли подія опублікована, вона надсилає подію кожному абоненту. Після отримання події її неможливо повторно відтворити, а нові абоненти не бачать події;
- потік подій: події записуються в журнал. Події строго впорядковані (у межах розділу) та довговічні. Клієнти не підписуються на потік, натомість клієнт може читати з будь-якої частини потоку. Клієнт відповідає за просування своєї позиції в потоці. Це означає, що клієнт може приєднатися в будь-який час і може повторювати події;

Що стосується споживача, є кілька загальних варіацій:

- проста обробка подій. Подія негайно викликає дію у споживача. Наприклад, ви можете використовувати функції Azure з тригером шини

обслуговування, щоб функція виконувалась щоразу, коли повідомлення публікується в темі шини служби;

– складна обробка подій. Споживач обробляє ряд подій, шукаючи шаблони в даних подій, використовуючи такі технології, як Azure Stream Analytics або Apache Storm. Наприклад, ви можете агрегувати показання з вбудованого пристрою за часове вікно та генерувати повідомлення, якщо ковзна середня перетинає певний поріг;

– обробка потоку подій. Використовуйте платформу потокового передавання даних, таку як Azure IoT Hub або Apache Kafka, як конвеєр для передачі подій та передачі їх потоковим процесорам. Потокові процесори діють для обробки або перетворення потоку. Для різних підсистем програми може бути кілька потокових процесорів. Цей підхід добре підходить для навантажень IoT.

Джерело подій може бути зовнішнім для системи, наприклад, фізичні пристрої в рішенні IoT. У цьому випадку система повинна мати можливість приймати дані на рівні обсягу та пропускної здатності, необхідних джерелу даних.

Коли використовувати цю архітектуру:

– кілька підсистем повинні обробляти однакові події;

– обробка в реальному часі з мінімальним затримкою;

– складна обробка подій, така як узгодження шаблонів або агрегування за вікнами часу;

– великий обсяг та висока швидкість передачі даних, таких як IoT.

Переваги:

– виробники та споживачі не пов'язані між собою;

– немає інтеграції точка-точка. Додавати нових споживачів до системи легко;

– споживачі можуть негайно реагувати на події, коли вони прибувають;

– високо масштабований та розподілений;

– підсистеми мають незалежні уявлення про потік подій.

Виклики:

– гарантована доставка. У деяких системах, особливо в сценаріях IoT, дуже важливо гарантувати доставку подій;

– обробка подій в порядку або рівно один раз. Кожен тип споживача, як правило, працює в декількох екземплярах для забезпечення стійкості та масштабованості. Це може створити виклик, якщо події потрібно обробляти в порядку (у межах споживчого типу) або якщо логіка обробки не є ідемпотентною [21].

2.3 Архітектура великих даних

Архітектура великих даних призначена для обробки та аналізу даних, занадто великих або складних для традиційних систем даних. Поріг, при якому організації вступають у сферу великих даних, різниться залежно від можливостей користувачів та їх інструментів. Для одних це може означати сотні гігабайт даних, а для інших - сотні терабайт. У міру того, як інструменти для роботи з наборами великих даних прогресують, зростає і значення великих даних. Цей термін дедалі більше стосується значення, яке ви можете отримати із своїх наборів даних за допомогою розширеної аналітики, а не строго розміру даних, хоча в цих випадках вони, як правило, досить великі.

З роками ландшафт даних змінювався. Те, що ви можете зробити або очікуєте зробити з даними, змінилося. Вартість зберігання різко впала, тоді як засоби збору даних продовжують зростати. Деякі дані надходять швидкими темпами, постійно вимагаючи їх збирати та спостерігати. Інші дані надходять повільніше, але дуже великими шматками, часто у вигляді десятиліть історичних даних. Можливо, ви стикаєтесь із проблемою розширеної аналітики або з проблемою, яка вимагає машинного навчання. Ці проблеми вирішують архітектури великих даних.

Рішення для великих даних, як правило, передбачають один або декілька з таких типів робочого навантаження:

– Пакетна обробка джерел великих даних у стані спокою.

– Обробка великих даних у режимі реального часу в русі.

– Інтерактивне дослідження великих даних.

– Прогностична аналітика та машинне навчання.

Розглядайте архітектуру великих даних, коли вам потрібно:

– зберігати та обробляти дані в обсягах, занадто великих для традиційної бази даних.

– трансформувати неструктуровані дані для аналізу та звітності.

– збирати, обробляти та аналізувати необмежені потоки даних у режимі реального часу або з низькою затримкою.

Компоненти архітектури великих даних

На наступній схемі показані логічні компоненти, які вписуються в архітектуру великих даних (Рисунок 2.1). Індивідуальні рішення можуть містити не всі елементи цієї схеми.

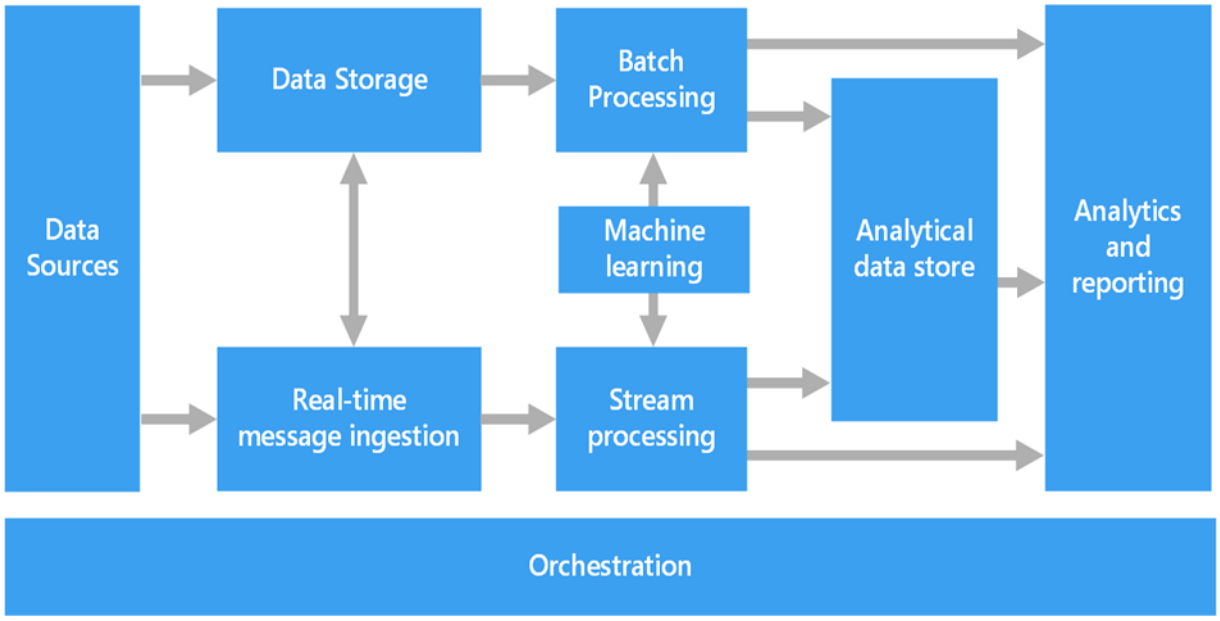


Рисунок 2.1 Загальна діаграма конвеєру даних

Більшість архітектурних рішень великих даних включають деякі або всі наступні компоненти:

a) джерела даних. Усі рішення для обробки великих даних починаються з одного або декількох джерел даних. Приклади включають:

1) зберігання даних додатків, таких як реляційні бази даних;

2) статичні файли, створені програмами, такими як файли журналів веб-сервера;

3) джерела даних у реальному часі, такі як пристрої IoT;

b) зберігання даних. Дані для операцій пакетної обробки, як правило, зберігаються у розподіленому сховищі файлів, що вміщує великі обсяги великих файлів у різних форматах. Цей тип сховища часто називають озером даних. Варіанти реалізації цього сховища включають Azure Data Lake Store або контейнери BLOB-об'єктів у сховищі Azure;

c) пакетна обробка. Оскільки набори даних настільки великі, часто рішення великих даних повинно обробляти файли даних за допомогою тривалих пакетних завдань для фільтрації, агрегування та підготовки даних до аналізу в іншому випадку. Зазвичай ці завдання передбачають читання вихідних файлів, їх обробку та запис вихідних даних у нові файли. Варіанти включають запуск завдань U-SQL в Azure Data Lake Analytics, використання Hive, Pig або спеціальних завдань Map / Reduce у кластері HDInsight Hadoop або використання програм Java, Scala або Python у кластері HDInsight Spark;

d) передача повідомлення в режимі реального часу. Якщо рішення включає джерела реального часу, архітектура повинна містити спосіб захоплення та зберігання повідомлень реального часу для обробки потоку. Це може бути просте сховище даних, де вхідні повідомлення потрапляють у папку для обробки. Однак багатьом рішенням потрібен магазин передачі повідомлень, який виступає в ролі буфера для повідомлень і підтримує обробку масштабу, надійну доставку та іншу семантику черги повідомлень. Цю частину потокової архітектури часто називають

буферизацією потоку. Серед варіантів - концентратори подій Azure, Azure IoT Hub та Kafka;

e) обробка потоку. Після захоплення повідомлень у режимі реального часу рішення має обробити їх, фільтруючи, агрегуючи та іншим чином готуючи дані до аналізу. Потім оброблені потокові дані записуються у вихідний стовбур. Azure Stream Analytics надає послугу керованої потокової обробки, засновану на постійно виконуваних SQL-запитах, які працюють з необмеженими потоками. Ви також можете використовувати технології потокового передавання Apache з відкритим кодом, такі як Storm та Spark Streaming, у кластері HDInsight;

f) зберігання аналітичних даних. Багато рішень для великих даних готують дані до аналізу, а потім подають оброблені дані у структурованому форматі, який можна отримати за допомогою аналітичних інструментів. Сховище аналітичних даних, що використовується для обслуговування цих запитів, може бути сховищем реляційних даних у стилі Кімбала, як це бачиться у більшості традиційних рішень бізнес-аналітики (BI). Крім того, дані можуть бути представлені за допомогою технології NoSQL із низькою затримкою, наприклад HBase, або інтерактивної бази даних Hive, яка забезпечує абстракцію метаданих над файлами даних у розподіленому сховищі даних. Azure Synapse Analytics надає керовану послугу для масштабного хмарного зберігання даних. HDInsight підтримує Interactive Hive, HBase та Spark SQL, які також можуть використовуватися для подання даних для аналізу;

g) аналіз та звітність. Метою більшості рішень для великих даних є надання уявлення про дані за допомогою аналізу та звітування. Щоб дати можливість користувачам аналізувати дані, архітектура може включати рівень моделювання даних, такий як багатовимірний куб OLAP або таблична модель даних у службах аналізу Azure. Він також може підтримувати BI самообслуговування, використовуючи технології моделювання та візуалізації в Microsoft Power BI або Microsoft Excel. Аналіз та звітування можуть також мати форму інтерактивного

дослідження даних науковцями даних або аналітиками даних. У цих сценаріях багато служб Azure підтримують аналітичні блокноти, такі як Jupyter, що дозволяє цим користувачам використовувати свої існуючі навички роботи з Python або R. Для широкомасштабного дослідження даних ви можете використовувати Microsoft R Server, як автономний, так і з Spark;

h) оркестровка. Більшість рішень для великих даних складаються з багаторазових операцій обробки даних, інкапсульованих у робочі процеси, які трансформують вихідні дані, переміщують дані між кількома джерелами та раковинами, завантажують оброблені дані в сховище аналітичних даних або надсилають результати прямо до звіту чи інформаційної панелі. Для автоматизації цих робочих процесів можна використовувати технологію оркестрування, таку як Azure Data Factory або Apache Oozie and Sqoop.

2.3.1 Лямбда-архітектура

При роботі з дуже великими наборами даних може знадобитися багато часу, щоб запустити такі запити, які потрібні клієнтам. Ці запити неможливо виконати в режимі реального часу, і часто потрібні такі алгоритми, як MapReduce, які працюють паралельно у всьому наборі даних. Потім результати зберігаються окремо від вихідних даних і використовуються для запитів.

Одним недоліком цього підходу є те, що він запроваджує латентність - якщо обробка займає кілька годин, запит може повернути результати, які мають кілька годин. В ідеалі ви хотіли б отримати деякі результати в режимі реального часу (можливо, з певною втратою точності) і поєднати ці результати з результатами пакетної аналітики.

Архітектуру лямбда, перший запропоновав Натан Марза. Він вирішив цю проблему, створюючи два шляхи для потоку даних. Усі дані, що надходять у систему, проходять ці два шляхи:

Пакетний шлях (холодний шлях) зберігає всі дані, що надходять в сирому вигляді і виконує пакетну обробку даних. Результат цієї обробки зберігається у вигляді пакетного подання.

Швидкий шлях (гарячий шлях) аналізує дані в режимі реального часу. Цей шар розроблений для низької затримки, за рахунок точності.

Пакетний шар подається в обслуговуючий шар, який індексує пакетний вигляд для ефективного запитування. Швидкісний рівень оновлює обслуговуючий рівень поступовими оновленнями на основі останніх даних.

Дані, що надходять у гарячий шлях, обмежуються вимогами до затримки, що накладаються шаром швидкості, так що їх можна обробити якомога швидше. Часто це вимагає компромісу певного рівня точності на користь даних, які готові якомога швидше. Наприклад, розглянемо сценарій IoT, коли велика кількість датчиків температури передає дані телеметрії. Швидкісний рівень може бути використаний для обробки ковзного часового вікна вхідних даних.

З іншого боку, дані, що надходять на холодний шлях, не підпадають під ті ж вимоги щодо низької затримки. Це дозволяє здійснювати обчислення з високою точністю для великих наборів даних, що може зайняти дуже багато часу.

Зрештою, гарячий і холодний шляхи сходяться в клієнтському додатку аналітики. Якщо клієнтові необхідно відображати своєчасні, але потенційно менш точні дані в режимі реального часу, він отримає свій результат із гарячого шляху. В іншому випадку він вибере результати з холодного шляху для відображення менш своєчасних, але більш точних даних. Іншими словами, гарячий шлях має дані за відносно невеликий проміжок часу, після чого результати можуть бути оновлені більш точними даними з холодного шляху.

Вихідні дані, що зберігаються на пакетному рівні, є незмінними. Вхідні дані завжди додаються до існуючих даних, а попередні дані ніколи не перезаписуються. Будь-які зміни у значенні певної бази даних зберігаються як новий запис події з позначкою часу. Це дозволяє проводити перерахунок у будь-який момент часу в

історії зібраних даних. Можливість перерахувати пакетний вигляд з вихідних даних є важливою, оскільки вона дозволяє створювати нові уявлення в міру розвитку системи.

2.3.2 Архітектура Kappa

Недоліком лямбда-архітектури є її складність. Логіка обробки з'являється у двох різних місцях - холодному та гарячому – за допомогою різних систем. Це призводить до подвійної обчислювальної логіки та складності управління архітектурою для обох шляхів.

Архітектура каппа була запропонована Джеєм Крепсом як альтернатива архітектурі лямбда. Вона має ті ж основні цілі, що і лямбда-архітектура, але з важливою відмінністю: всі дані протікають по одному шляху, використовуючи систему обробки потоків.

Є деякі подібності з пакетним шаром архітектури лямбда, оскільки дані про події є незмінними, і всі вони збираються. Дані потрапляють у вигляді потоку подій у розподілений та стійкий до несправностей єдине сховище. Ці події упорядковані, а поточний стан події змінюється лише додаванням нової події. Подібно до рівня швидкості архітектури лямбда, вся обробка подій виконується на вхідному потоці і зберігається як подання в режимі реального часу.

Якщо вам потрібно перерахувати весь набір даних (еквівалентно тому, що виконує пакетний шар у лямбда-сигналі), ви просто відтворюєте потік, як правило, використовуючи паралелізм, щоб своєчасно завершити обчислення.

2.3.3 Інтернет речей (IoT)

З практичної точки зору Інтернет речей (IoT) представляє будь-який пристрій, підключений до Інтернету. Це включає ваш ПК, мобільний телефон, розумний

годинник, розумний термостат, розумний холодильник, підключений автомобіль, імплантати для моніторингу серця та все інше, що підключається до Інтернету та надсилає або отримує дані. Кількість підключених пристроїв зростає з кожним днем, як і кількість даних, зібраних з них. Часто ці дані збираються у дуже обмежених, іноді з великою затримкою середовищах. В інших випадках дані надсилаються із середовищ із низькою затримкою тисячами або мільйонами пристроїв, що вимагає можливості швидкого введення даних та відповідної обробки. Тому для вирішення цих обмежень та унікальних вимог потрібно належне планування.

Архітектури, керовані подіями, є центральними для рішень IoT. На наступній схемі (Рисунок 2.2) показано можливу логічну архітектуру IoT. Діаграма базується на потокових компонентах архітектури.

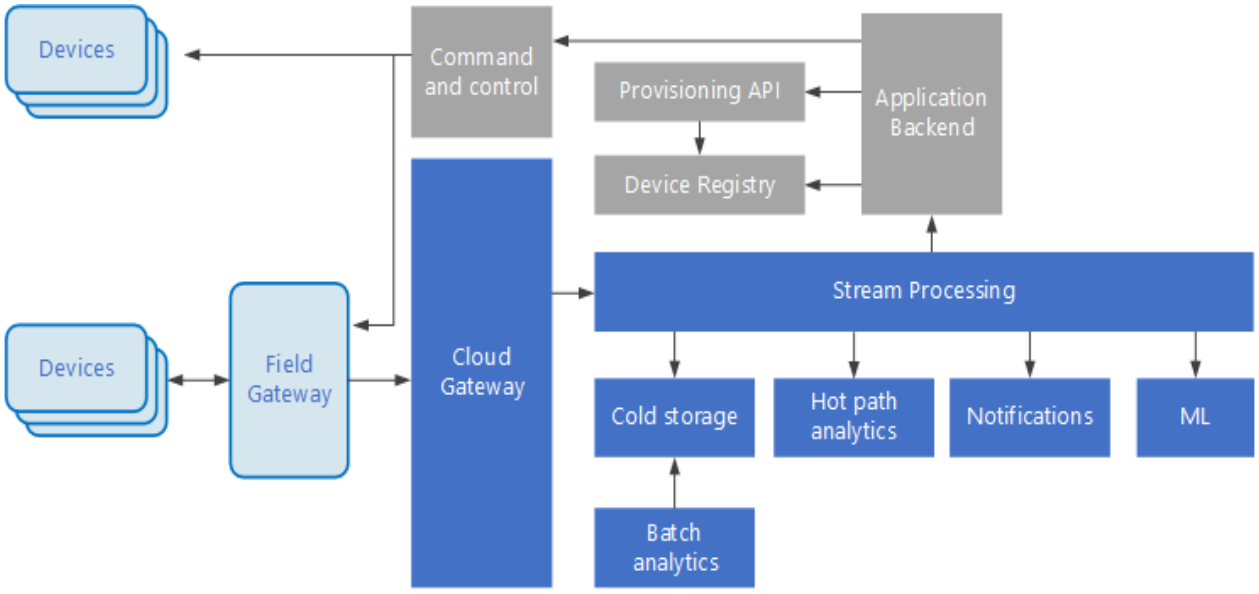


Рисунок 2.2 – архітектура інтернету речей

Cloud gateway ковтає події які подають пристрої, використовуючи надійну систему з низькою затримкою передачі повідомлень.

Пристрої можуть надсилати події безпосередньо на хмарний шлюз або через польовий шлюз. Польовий шлюз – це спеціалізований пристрій або програмне забезпечення, яке зазвичай розміщується разом із пристроями, яке приймає події та пересилає їх на хмарний шлюз. Польовий шлюз може також попередньо обробляти

необроблені події пристрою, виконуючи такі функції, як фільтрація, агрегування або перетворення протоколу.

Після поглинання події проходять через один або кілька потокових процесорів, які можуть направляти дані (наприклад, до сховища) або виконувати аналітику та іншу обробку.

Нижче наведено деякі поширені типи обробки (Цей список, безумовно, не є вичерпним.):

– запис даних про події в холодне сховище для архівування або пакетної аналітики;

– аналітика гарячих шляхів, що аналізує потік подій у (майже) реальному часі, для виявлення аномалій, розпізнавання закономірностей у прокручуваних вікнах часу або активації попереджень, коли в потоці виникає певна умова;

– обробка спеціальних типів нетелеметричних повідомлень від пристроїв, таких як сповіщення та сигнали тривоги;

– машинне навчання.

Поля, затінені сірим кольором, показують компоненти системи IoT, які безпосередньо не пов’язані з потоковим потоком подій, але включені сюди для повноти:

– реєстрація пристроїв являє собою базу даних з наданими пристроями, в тому числі ідентифікаторами пристроїв і зазвичай метаданих пристрою, такі як місце розташування;

– ініціалізації API є загальним зовнішнім інтерфейсом для ініціалізації і реєстрації нових пристроїв;

– деякі рішення Інтернету речей дозволяють надсилати на пристрої повідомлення про управління та управління [22].

2.4 Паттерни «інтернету речей»

2.4.1 Цикли виміру і управління

Цикл вимірювання та управління Інтернетом речей утримує пристрій IoT у допустимому діапазоні конфігурації заданих значень за допомогою процесу управління в режимі реального часу із замкнутим циклом. Пристрій може бути частиною більшої фізичної системи, керованої програмним забезпеченням, що містить один або кілька мережевих пристроїв.

Пристрій IoT, схильний до порушень від зовнішніх подій, вимагає замкненого процесу управління, щоб підтримувати його поблизу бажаної конфігурації заданого значення. Логіка управління вимірюванням та контуром управління спостерігає за пристроєм за допомогою метрик датчика та вживає коригуючих заходів за допомогою дій приводу .

Приклади циклів вимірювання та управління в дії включають:

– розумна мишоловка: спрацьовує подія закриття пастки, коли датчики виявляють мишу;

– датчики диму: спрацьовують спринклери при відчутті диму від декількох датчиків;

– силовий трансформатор: вимикає трансформатор під прогнозованою сильною грозою;

– моніторинг газопроводу: відкриває клапан, щоб компенсувати падіння тиску;

– домашній термостат: збільшує витрату газу нагрівача, відчуваючи, що температура в приміщенні падає нижче заданого значення;

– вітрогенератор: застосовуйте гальмо, щоб уповільнити ротор, коли він ось-ось досягне попереджувального порогу об / хв;

– сонячні панелі: врегульовує кут сонячної панелі, коли сонце рухається за горизонтом, щоб максимізувати генеровану енергію.

Вимірювальна та контрольна петля має масштаб у вигляді єдиної абстракції пристрою, що складається з датчиків, виконавчих механізмів та контролера. Ці цикли можуть інтегруватися з циклами аналізу та оптимізації, а також контролювати та управляти циклами, які працюють у значно більшому контексті [23].

2.4.2 Відстеження та керування циклами

Цикл моніторингу та управління Інтернетом речей (IoT) – це система нагляду, яка постійно контролює фізичну систему, керовану набором мережевих пристроїв IoT. Цикл моніторингу та управління гарантує, що система знаходиться в допустимому діапазоні бажаного значення заданого стану, і видає команди для управління системою.

У цьому підпункті подано огляд архітектури, характеристик та компонентів циклу контролю та управління ними.

Деякі приклади сценаріїв для моніторингу та управління циклами включають:

– розумний збір сміття: направляє вантажівку на маршрут, який найбільше потребує збору сміття;

– розумний кампус: повідомляє про евакуацію з кампусу при виявленні пожежі в декількох будинках;

– розподіл електроенергії: попередньо відключає електроенергію від декількох міських кварталів на основі прогнозу сильного вітру та дощу;

– моніторинг газопроводу: вимикає газоперекачувальну станцію після виявлення перепадів тиску на декількох сегментах віддаленого трубопроводу;

– розумні лічильники: відстежує споживання електроенергії та поєднує його з прогнозами погоди для автоматичного підвищення заданого значення

домашніх термостатів, як частина програми надання знижок економним споживачам енергії;

– вітряна електростанція: помітивши падіння коефіцієнта потужності на вітровій електростанції, запланує перевірку підозрюваних вітрових турбін;

– переробні галузі: моніторинг та контроль процесу крекінгу сирої нафти на нафтопереробному заводі. Контролює та керує виробництвом фарби та хімічної продукції;

– дискретне виробництво: відстежує та контролює клітинку інспекції віджетів та упаковки.

Цикл управління та моніторингу IoT – це система нагляду, яка забезпечує роботу системи в межах робочих порогів. Кілька пристроїв у системі повинні діяти спільно, щоб досягти і залишатися в межах допустимого діапазону бажаного стану. Цикл управління та моніторингу спостерігає та корелює тенденції гарячого сигналу телеметрії з декількох пристроїв для визначення поточного стану. Логіка поєднує ці тенденції з теплою історією часових рядів та сигналами корпоративної системи для обчислення нових знань. Потім цикл моніторингу та управління просуває інформацію через механізм правил, щоб генерувати команди виконавчого механізму або створювати тривоги за необхідності.

Цикли моніторингу та управління мають такі характеристики:

– може бути віддаленим або близьким до фізичних пристроїв IoT. Приміщення, віддалені за своєю природою, такі як нафто- і газопроводи, силові трансформатори, розумні дверні дзвінки, небезпечне середовище та засоби відстеження активів не можуть вмістити цю інфраструктуру. У цих середовищах моніторинг і управління циклами працюють із віддалених об'єктів, таких як загальнодоступні або приватні хмари. У переробних галузях промисловості, таких як переробка нафти та хімічне виробництво, моніторинг та управління циклами можуть бути розміщені ближче до пристроїв. Дискретне виробництво може також розгортати ці цикли локально, оскільки застої мережі можуть бути дорогими;

– залежна від циклів вимірювання та управління на основі пристроїв для основних процесів моніторингу та управління;

– може інтегруватися з іншими корпоративними системами, такими як планування корпоративних ресурсів (ERP), управління відносинами з клієнтами (CRM), управління життєвим циклом продукту (PLM) та системами підтримки для контекстуалізації операцій. Моніторинг та керування циклами не залежать від роботи цих систем;

– використовуйте потоки телеметрії датчиків та робіть внесок у останній відомий стан пристрою, гарячий кеш часових рядів, теплу історію часових рядів та сукупні зведення;

– виробляйте контрольні команди назад до пристроїв для умов, які потрібно виправити;

– обчислюйте залежні стани пристроїв та надайте канали подій для зовнішніх систем;

– в основному інтегрується з пристроями та корпоративними системами через мережеві протоколи HTTP, MQTT та AMQP;

– може мати час циклу в кілька секунд, залежно від сценарію IoT. Дисперсія затримки мережевих пакетів або тремтіння можуть виникати при використанні мережевих протоколів, нечутливих до часу, таких як MQTT, HTTP та AMQP [24].

Аналіз та оптимізація циклів

Цикл аналізу та оптимізації Інтернету речей (IoT) дозволяє генерувати та застосовувати ідеї оптимізації бізнесу для одного або декількох розгортань фізичних систем, керованих програмним забезпеченням, на основі всього бізнес-контексту підприємства. Аналізує та оптимізує телеметрію джерел циклу, як правило, здійснює моніторинг та управління цикловими процесами, вдосконалює його та поєднує з корпоративними джерелами даних для отримання статистичних даних.

Деякі приклади сценаріїв аналізу та оптимізації циклів включають:

– розумні простори: обчислює індекс безпеки кампусу та вживає відповідних заходів;

– передача електроенергії: корелює тенденції відключення електроенергії та подій, пов’язаних із пожежами, для попереджувального ремонту передачі та заміни приладів контролю;

– видобуток нафти та газу: обчислює тенденції видобутку нафти в басейні та порівнює їх із характеристиками ділянки;

– транспорт та логістика: обчислює тенденції вуглецевого сліду, порівнює їх із організаційними цілями та вживає коригуючих заходів;

– вітроелектростанція: обчислює коефіцієнт потужності всієї роботи вітроелектростанції та розробляє засоби для підвищення ефективності роботи кожного вітрогенератора;

– дискретне виробництво: збільшує темпи виробництва віджетів на багатьох заводах, щоб задовольнити ринковий попит.

У циклі аналізу та оптимізації дані з різних IoT, корпоративних, приватних та публічних джерел надходять у хмарні озера даних. Автономна аналітика використовує озера даних, щоб виявити приховані тенденції та ідеї оптимізації бізнесу. Дані про оптимізацію з офлайн-процесів аналітики повертаються до установок IoT через моніторинг та управління циклами, а також вимірювання та контроль циклів .

Важливими компонентами управління оптимізацією бізнесу є:

– озеро даних, великомасштабні зберігання оптимізовані для зниження витрат використання більш тривалих періодів часу. Зберігання HDFS в контексті обробки карт зменшення є прикладом такого озера даних. Озеро даних визначає структуру даних до часу їх обробки, тому добре підходить для зберігання як структурованих, так і неструктурованих даних;

– дані про холодні часові ряди, необроблена або оброблена телеметрія, що є важливим для аналітики в режимі офлайн і часто надходить із декількох систем IoT. Завдання Analytics додатково вдосконалюють та поєднують ці дані з корпоративними та зовнішніми наборами даних;

– дані підприємств, що виробляються корпоративними системами, такі як управління життєвим циклом продукції, ланцюжок поставок, фінанси, продаж, виробництво та розподіл, а також управління відносинами з клієнтами. Корпоративні дані у поєднанні із зовнішніми наборами даних, такими як погода, можуть контекстуалізувати телеметрію IoT у діловому масштабі для отримання сумісних статистичних даних;

– автономна аналітика для обробки великих даних у пакетному режимі. Ось кілька прикладів – іскрові завдання та обробка карт зменшення Hadoop. Відстежуйте та керуйте циклами, а також вимірюйте та контролюйте процеси циклу, а потім застосовуйте інформацію, отриману в результаті аналізу та оптимізації циклів, до пристроїв IoT [25].

В цьому розділі ми розглянули архітектурні підходи і вже володіємо потрібними знаннями для запропонування своїх ідей по темі роботи. Головне тут було зрозуміти основні проблеми, які зустрічаються в архітектурі та шляхи їх вирішення.

3 АРХІТЕКТУРА СИСТЕМИ МОНІТОРИНГУ НА ОСНОВІ ТЕХНОЛОГІЇ «РОЗУМНИЙ БУДИНОК»

Володіючи необхідною інформацією про «інтернет речей», «великі дані» та архітектурні підходи для додатків які базуються на цих технологіях можна переходити до практичної частини.

3.1 Загальний вигляд і опис архітектури

Будь який додаток починається з вхідних даних. В нашій архітектурі «інтернету речей» є потік даних або подій, які прямують до кінцевого користувача. Так як не вся інформація, яка прямує з джерел потрібна для кінцевого користувача, то треба визначити що саме має отримувати користувач та які данні матиме можливість запросити додатково. Для цього є слой обробки даних, в якому відбуваються основні дії. Ці всі слої можна побачити на рисунку 3.1.

Рисунок 3.1 – Загальний вигляд слоїв архітектури

Прямокутник з вертикальними полосами означає окрему частину додатку, а стрілочка означає направлення або запит даних.

Обробка даних це є бекенд додаток, який більш детально буде розглядатися в свою чергу. А вивід даних це фронтенд, на якому не буде зосереджено багато уваги, бо ця тема має дуже великі обсяги.

Слід зазначити, що в цій системі розглядається моніторинг, тому елементів керування і команд не передбачено. Але в разі необхідності система спроектована

так, щоб додавання додаткових елементів не викликало проблем і не заважало роботі існуючим сервісам. Також додавання нових девайсів в систему не є проблемою, тому що вони всі дані, які приходять зберігаються в базі. Додавання нових девайсів і, як наслідок, нових даних, які приходитимуть в систему буде нести за собою лише обробку цих даних (якщо це необхідно), і додавання можливості сервісу їх бачити (лише в тих сервісах, в яких це необхідно).

Також в подібних система треба володіти інформацією про об’єкт моніторингу. Ця інформація вноситься в базу даних системи окремо.

3.2 Джерела даних

Джерела даних це є самі девайси, тобто датчики, сенсори і тд. Вони збирають різного роду інформацію і відправляють її на field gateway (польовий шлюз). Там відбувається трансформація протоколів, фільтрація та агрегація даних. Це все те, що знаходиться фізично біля об’єкту моніторингу. Польові шлюзи відправляють отримані і оброблені дані в cloud gateway (хмарний шлюз), який ловить всі івенти, які приходять і передає їх далі на обробку. Схема цього зображена на рисунку 3.2.

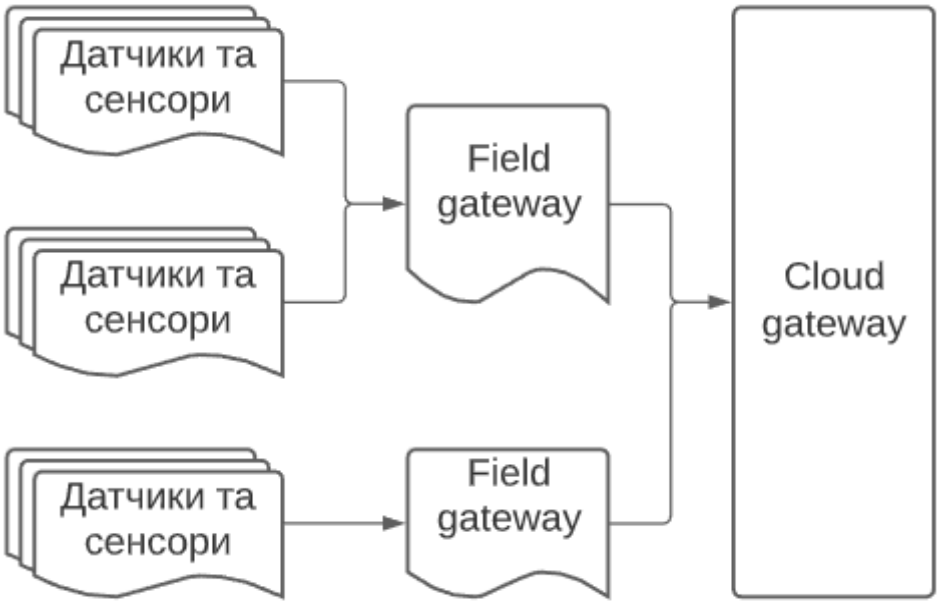


Рисунок 3.2 – Джерела даних

Обірваним прямокутником позначаються девайси, а простим прямокутником позначаються процеси. Можна вважати що прямокутник є одним або декількома сервісами, системою обміну повідомленнями або чергою.

Cloud gateway не є фізичним пристроєм біля об’єкту моніторингу, він є більше елементом обробки даних. Але було додано в частину з джерелами даних, бо для частини обробки даних саме звідти і приходять всі дані.

Датчики, необхідні для моніторингу:

– датчики температури приміщення;
– датчики вологості приміщення;
– пилові датчики;
– датчики для виміру пульсу, серцебиття, тиску, температури (наприклад Apple Watch).

Данні які формуються у Field gateway:

– timestamp – час, коли трапилась дана подія;
– eventName – назва події;
– patientId – унікальний ідентифікатор пацієнта;

– healthState – стан пульсу, серцебиття, тиску, температури (якщо підключений відповідний девайс);
– dustLevel – рівень пилу;
– humidity – рівень вологості;
– roomTemperature – температура в кімнаті.

Вище наведений перелік датчиків і даних, які вони формують не є повним, бо в залежності від особливостей об’єкту моніторингу вони будуть змінюватись. Основа, яка запропонована тут, може бути вдосконалена для потреб кожного випадку. Наприклад, для людей в комі можуть застосовуватися апарати штучного дихання і додаткові датчики виміру пульсу, серцебиття, тиску і т.д. В цій роботі розглядається основа з мінімальним набором датчиків, які можна застосовувати для моніторингу за людьми літнього віку, або за пацієнтами які потребують особливого догляду.

3.3 Вивід даних

Система має 2 додатки для Android та IOS, які спілкуються з бекенд додатком через API Gateway (шлюз API). Він є точкою входу в бекенд додаток та маршрутизує всі запити з фронтенду до відповідних сервісів. Дивиться рисунок 3.3.

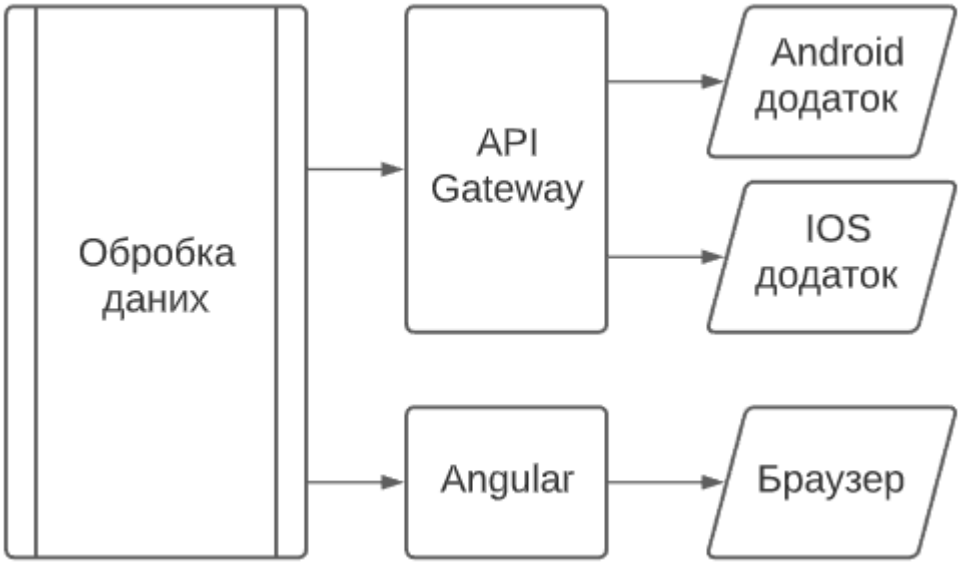


Рисунок 3.3 – Схема для фронтенд частини

Прямокутником з нахилом тут позначається вивід даних. Дані можуть виводитись в будь-який додаток.

Також можна побачити сервіс з назвою Angular. Це сервіс, в якому формуються HTML сторінка при запиті. Для цих цілей було вибрано саме Angular Framework бо він найкраще за все підходить для задач де є динамічні сторінки, які оновлюються в реальному часі. [26]

Angular сервіс не має потреби робити запити з API gateway, так як фактично є частиною бекенд додатку, коли як мобільні додатки є окремими додатками.

Іноді можна побачити що API gateway має кеш-дані щоб імітувати роботу системи, відповідаючи останніми стабільними даними. Але в нашому випадку він виконує лише роль роутинга, тому що додаток має динамічний характер і крім того, якщо щось із системою буде не так – краще про це знати одразу, бо це може загрожувати життю і здоров’ю людини. Проте ми маємо деякі кешовані дані, біля деяких сервісів, але це дані, які не змінюються в реальному часі, більш детально про це пізніше.

3.4 Обробка даних

Розглянемо потоки данних, які присутні в системі. Почнемо з обробника потоків. Обробник потоку це система обміну повідомленнями, яка дозволяє розподілювати дані в різну кількість потоків. Дивиться рисунок 3.4.

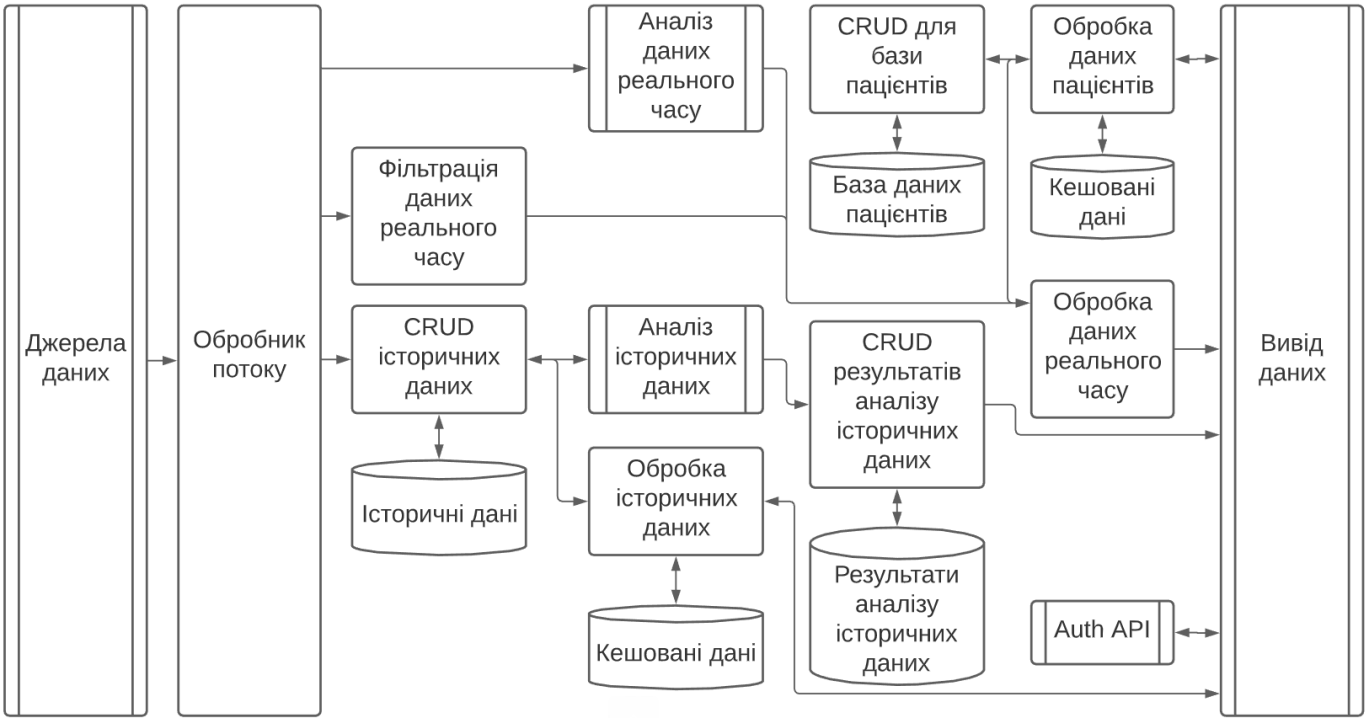


Рисунок 3.4 – Схема бекенд додатку

В нас є 3 потоки даних, 2 з них це потоки для обробки в реальному часі, а один для зберігання історичних даних та вивід їх на фронтенд по запиту. Їх кількість це кількість виводів з обробника потоку. Верхній потік це потік для аналізу даних в реальному часі. Другий це фільтрування і обробка даних в реальному часі. Ці два потоки одразу при обробці йдуть на вивід. Третій, самий нижній, записується в основну базу даних. Звідти також можна запросити дані при запиті.

CRUD (create, read, update, delete) – абревіатура, яка складається з перших літер назв основних операцій для взаємодії з будь-якою системою. Всі CRUD-сервіси виконують функцію взаємодії з базою даних.

Історичні дані – це дані, які надходили в систему протягом всього часу. В системі вони використовуються для аналізу, а також є можливість запросити їх зі сторони клієнта. Також на схемі можна побачити, що ці данні використовуються для аналізу. Це може бути формування статистики, прогнозування чи будь що інше, що може бути корисним.

Обробка історичних даних. Так як CRUD-сервіс виконує саме CRUD операції то обробкою і обрахунками цих даних має займатися окремий сервіс, слідуючи першому принципу SOLID – принцип єдиної відповідальності. Кожен сервіс має одну чітку задачу, яку він виконує. Якщо виникає потреба в другій задачі, то на це має виділятися окремий сервіс.

Аналіз історичних даних – це, як було зазначено вище, формування статистики, прогнозування на довгу перспективу тощо. Після завершення аналізу результати вносяться до бази даних результатів, які так само можна запросити з фронтенду.

Дані реального часу – це дані, які виводяться після того, як сформувалися з мінімальною затримкою. Аналіз даних реального часу – це невелике прогнозування на основі вхідних даних. Це можу бути як розрахунок ймовірності в найближчому часі серцевого нападу, втрати свідомості і т.д.

На схемі блоки з аналізом позначені як окремі частини додатку. Тема аналізу даних є окремою і дуже великою, тому не розглядається в рамках теми обробки даних.

Фільтрація даних реального часу – дані які формуються і відправляються не завжди корисні для моніторингу в реальному часі, тому їх необхідно відсіяти і відправити на обробку вже менший обсяг даних.

Обробка даних реального часу – на основі даних, які вже прийшли після фільтрації можуть бути вирахувані якісь інші дані. Наприклад з поля eventName можна взяти назву події і по назві вказати якусь додаткову інформацію, якщо це якась термінова подія. Також на основі поля patientId, з відповідного сервісу можна вилучити інформацію про пацієнта, таку як вік, ім’я. Також по запиту можна бути дістати історію хвороб.

Auth API – API для авторизації і регістрації користувачів системи. Має базу користувачів.

3.5 Стандартизація

Будь-яка система з часом буде розростатися, тому щоб при зростанні кількості сервісів, баз даних або будь-яких інших логічних та структурних елементів Все виглядало як одна цілісна система треба вводити стандарти.

Кожен сервіс має бути REST-API і написаним на Java з використанням Spring Boot фреймворку. На Java написані багато big data та enterprise систем і досі вона залишається найнадійнішою мовою. Використання Spring Boot зробить код сервісів більш зрозумілим, чітким і простим.

API Gateway має бути написаний на Spring Cloud, а обробник потоку має бути написаний на Spring Cloud Stream.

Якщо сервіс має кешовані дані, то використовувати для їх зберігання Redis. Він є найшвидшою базою на сьогоднішній день.

База даних зі статичною структурою, наприклад, база даних пацієнтів або зберігання історичних даних використовувати Microsoft SQL Server. Це є одна з найстабільніших баз даних. Буде краще її використовувати в більшості випадків, але якщо структура даних нечітка і змінна, то кращий вибір буде NoSQL бази.

Фронтенд має використовувати Angular Framework. Для додатків, які мають динамічний характер це найкращий фронтенд фреймворк.

Уся бізнес логіка має бути покрита юніт-тестами. Юніт тести це показник якості продукту, вони замінюють документацію коду і мають простий вигляд, що по ним можна краще зрозуміти як працює якийсь метод ніж по його коду.

Додаток має працювати на Docker та Kubernetes.

Головна частина цього розділу це запропонована архітектура, яка має децентралізовану систему баз даних, низьку зв’язність між сервісами та низьку затримку при виведенні даних в реальному часі. Також найважливішою тут є не стільки це все скільки стандартизація. Єдині правила для всього проекту (навіть якщо він має більше 100 сервісів) дають змогу краще працювати з ним і легко вирішувати більшість проблем.

4 ПРИКЛАД РЕАЛІЗАЦІЇ ОПИСАНОЇ АРХІТЕКТУРИ

Розглянемо один з можливих прикладів реалізації. Для початку визначимось з вхідними та вихідними даними, потім визначимось які з сервісів мають спілкуватися за допомогою веб-сокетів, а які за допомогою http-запитів і буде описана логіка роботи кожного з сервісів.

4.1 Вхідні та вихідні дані

Розглянемо приклад вхідних даних на рисунку 4.1.

```
{
    "timestamp": "2020-11-21T18:38:54.942Z",
    "eventName": "DataFlow",
    "patientId": "9999",
    "healthState": {
        "pulseRate": "60",
        "heartRate": "60",
        "temperature": "36.6"
    },
    "dustLevel": "Low",
    "humidity": "Low",
    "roomTemperature": "20"
}
```

Рисунок 4.1 – вхідні дані для додатку у форматі JSON

Це вигляд, в якому дані будуть надходити в систему.

- timestamp – час, коли трапилась дана подія;
- eventName – назва події;
- patientId – унікальний ідентифікатор пацієнта;
- healthState – стан пульсу, серцебиття, тиску, температури;

– dustLevel – рівень пилу;

– humidity – рівень вологості;

– roomTemperature – температура в кімнаті.

Серед вхідних даних є такі як EventName, DustLevel та Humidity. Визначемо які є варіанти для кожного з них.

Можливі варіанти виводу EventName:

– DataFlow – звичайний потік даних;

– Alarm – надзвичайна ситуація, така як немає пульсу або пульс дуже низький (аналіз та висновок робить система в реальному часі);

– Warning – ситуація близька до надзвичайної, наприклад пульс не дуже низький або високий, але близький до надзвичайної ситуації (аналіз та висновок робить система в реальному часі);

– NoData – дані не поступають.

Можливі варіанти виводу DustLevel та Humidity:

– Low – низький рівень;

– Medium – середній рівень;

– High – високий рівень.

Тепер розглянемо дані, які будуть виводитись кінцевим користувачам на рисунку 4.2

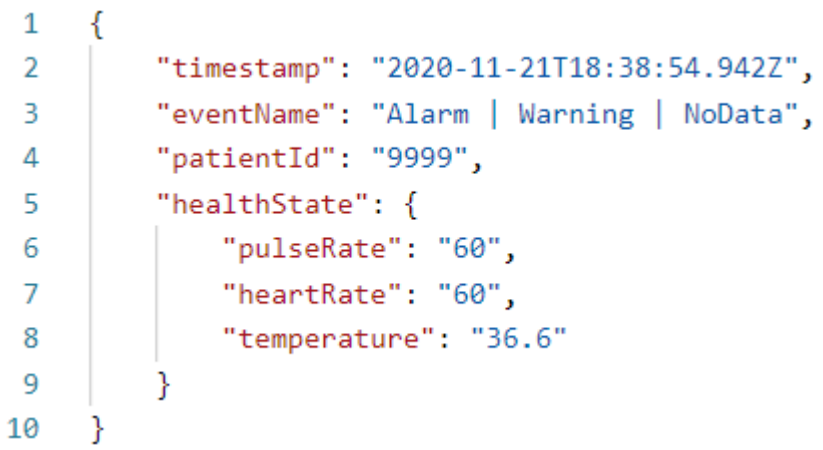


Рисунок 4.2 – приклад вихідних даних в реальному часі в форматі JSON

Поле eventName виводиться тільки коли є якась нетипова подія. Така як надзвичайна ситуація, немає даних або ситуація близька до надзвичайної. Це все буде визначатися в реальному часі, а також в аналізі історичних даних.

4.2 Комунікація між сервісами

Як було зазначено в 3 розділі схема роботи передбачає як мінімум 3 потоки даних. Два з них це потоки реального часу, а третій це потік для історичних даних.

З двох потоків реального часу є один аналітичний (виконується аналіз даних) і один потік виводу. Аналітичний потік виконує аналіз даних на коротку перспективу. Може вираховувати ймовірність наближення критичної ситуації і після чого він відправляє дані на обробку в сервіс обробки даних реального часу. До цього сервісу підключені 2 потоки реального часу. Там дані оцінюються і дається нове значення для поля eventName, а також по унікальному ідентифікатору дістаються дані пацієнта. Потік виводу проходить через фільтрацію. Там відсіюються непотрібні для виводу дані, такі як рівень вологості чи пилу. При потребі їх можна запросити додатково з історичних даних. Всі потоки реального часу з’єднані через веб-сокет

Потік історичних даних з’єднаний з CRUD сервісом для бази даних, в якій будуть зберігатися ці дані. З цього ж CRUD сервісу беруться дані для їх аналізу, на приклад, для формування статистики. Результати аналізу також зберігаються в базі доступ в яку здійснює CRUD сервіс. Результати аналізу також можна запросити з клієнта, вони не проходять ніякої обробки, бо вони вже в базі мають обрахований і кінцевий вигляд. Можлива буде необхідно фільтрація, але вона виконується при запиті до бази.

При потребі можна буде запросити історичні дані у тому вигляді, в якому вони прийшли. Для цього є окрема база даних, з якою спілкується CRUD сервіс. Він спілкується з сервісом – обробником даних. Цей сервіс виконує фактично туж саму роль що і сервіс для обробки даних в реальному часі, але тут можна робити більш складну логіку, тому що затримка в часі тут не стільки суттєва. CRUD сервіс для історичних даних приймає данні як веб-сокет, а відправляє їх за допомогою http запиту.

4.3 Опис таблиць в базах даних

Опишемо таблиці баз даних для пацієнтів та історичних даних. Таблиця з назвою “Patients” описана у таблиці 4.1.

Таблиця 4.1 – база даних пацієнтів

Назва поля	Тип даних	Not Null
patientId	int	true
firstName	varchar	true
lastName	varchar	true
age	int	true
currentDiagnosis	varchar	false

Тут ми бачимо основні дані пацієнта:

– patientId – унікальний ідентифікатор пацієнта;

– firstName – ім’я;
– lastName – прізвище;
– age – вік;
– currentDiagnosis – дійсний діагноз.

Також в стовпці «Not Null» є 2 варіанти, це «true» та «false». Якщо «true» значить значення для цього поля має бути обов’язковим, а якщо «false» то ні.

Тип даних показує в якому вигляді будуть зберігатися дані. Нижче наведено перелік:

– int – ціле число;
– varchar – текст;
– date – дата.

Це мінімальний набір даних, який треба знати про пацієнта. Тепер розглянемо таблицю з їх адресами, дивиться таблицю 4.2.

Таблиця 4.2 – адреса пацієнта

Назва поля	Тип даних	Not Null
addressId	int	true
patientId	int	true
country	varchar	true
city	varchar	true
street	varchar	true
home	int	true

Розглянемо назви полів:

– addressId – унікальний ідентифікатор адреси;
– patientId – унікальний ідентифікатор пацієнта;
– country – країна;
– city – місто;
– street – вулиця;

– home – будинок.

Це мінімально необхідна інформація, яка необхідна для того щоб визначати де знаходиться пацієнт з яким трапилась надзвичайна ситуація і викликати туди швидку допомогу.

Також в деяких випадках буде з’являтися необхідність перегляду історії захворювань. Переглянемо таблицю 4.3 з описом як зберігається історія захворювань.

Таблиця 4.3 – хвороби пацієнта

Назва поля	Тип даних	Not Null
patientId	int	true
diagnosis	varchar	true
date	date	false
wasCured	bit	true

Визначимо назви полів:

– patientId – унікальний ідентифікатор пацієнта;

– diagnosis – діагноз;

– date – дата;

– wasCured – чи одужав пацієнт.

Ця інформація може бути корисною, коли з’являється ситуація близька до надзвичайної.

Тепер опишемо історичні дані. В таблиці 4.4 присутні дані, які описують подію.

Таблиця 4.4 – Історичні дані, опис таблиці події

Назва поля	Тип даних	Not Null
timestamp	date	true
patientId	int	true
dustLevel	varchar	false
humidity	varchar	false
roomTemperature	int	false

Визначимо назви полів:

– timestamp – час виникнення події;
– patientId – унікальний ідентифікатор пацієнта;
– dustLevel – рівень пилу;
– humidity – рівень вологості;
– roomTemperature – температура кімнати.

З цією таблицею буде пов’язана ще одна, яка описує стан здоров’я пацієнта. Розглянемо її опис в таблиці 4.5.

Таблиця 4.5 – Історичні дані, опис таблиці стану здоров’я

Назва поля	Тип даних	Not Null
stateId	int	true
patientId	int	true
pulseRate	int	false
heartRate	int	false
temperature	int	false

Визначимо назви полів:

– stateId – унікальний ідентифікатор стану;
– patientId – унікальний ідентифікатор пацієнта;
– pulseRate – пульс;
– heartRate – серцебиття;
– temperature – температура.

Крім цих баз даних є ще база з аналізом історичних даних. Там має бути визначено скільки разів траплялися надзвичайні ситуації у різних пацієнтів, скільки разів близькі до надзвичайної ситуації і середні значення показників. Опис цих даних на таблиці 4.6.

Таблиця 4.6 – середні значення показників

Назва поля	Тип даних	Not Null
meanId	int	true
patientId	int	true
heartRate	float	true
pulseRate	float	true
temperature	float	true

Визначимо назви полів:

– meanId – унікальний ідентифікатор значень;

– patientId – унікальний ідентифікатор пацієнта;

– heartRate – серцебиття;

– pulseRate – пульс;

– temperature – температура.

Також система має кеш-сховища, які використовують Redis. Це NoSQL база даних типу key-value (ключ-значення). Кешуються тільки ті дані, які є незмінними, інакше є ймовірність що дані, які приходитимуть будуть неактуальними.

4.4 Опис сервісів

Для початку опишемо сервіси, які мають CRUD функції. Ці сервіси знаходяться біля бази даних і виконують взаємодію з нею. Вони мають з’єднання з однією базою даних.

Тепер розглянемо фільтрацію даних реального часу. Як було описано вище ми маємо відфільтрувати деякі дані, бо вони не є важливими для моніторингу в реальному часі, а також для економії трафіку. В нашому випадку фільтруватися будуть dustLevel, humidity і roomTemperature так як вони не є важливими.

Далі йде обробка даних реального часу. Там для поля eventName буде визначатися назва події, а також будуть братися дані пацієнта при потребі. Всі вони поєднанні веб-сокетом.

При обробці історичних даних будуть виконуватися як і ті самі дії що і при обробці даних в реальному часі, так і більш складні.

ВИСНОВКИ

За результатами виконаних теоретичних та практичних досліджень у дипломній роботі розроблено архітектуру системи моніторингу за життям людей з вадами здоров’я на основі технології «розумний будинок».

Було дано опис однієї з можливих реалізацій архітектури. На основі цього можна сказати, що архітектура є досить прозорою і зрозумілою, навіть при зростанні даних та ускладнення логіки і структури може залишатися такою ж прозорою, особливо якщо дотримуватися стандартизації

Завдяки такій системі є можливість людям з вадами здоров’я бути під наглядом навіть дома і не займати палати у лікарнях без необхідності частого втручання лікарів. В разі потреби дуже ретельного догляду можна обладнати такою системою палату і використовувати систему як «розумну палату». Також частина системи – бекенд додаток можна використовувати як основу для архітектури додатків для моніторингу. На приклад, це може бути моніторинг стану приладів або їх роботи.

Отриману систему можна ще вдосконалювати та доповнювати. В даній роботі не приділялось уваги до фізичного розташування датчиків, як вони мають працювати, яку інформацію збирати і т.д. Також не багато уваги приділялося фронтенд частині, це дуже важливо щоб вся необхідна інформація стисло могла розташуватися на маленьких екранах смартфонів, або моніторах. Варто приділити увагу до аналізу даних. При якісному аналізі можна отримати багато корисної наукової інформації. Також якщо в систему вбудувати аналіз на основі штучного інтелекту, то можна зробити систему більш самостійної і надати їй змогу, наприклад, викликати швидку.

Список використаних джерел

1. Система розумний будинок — що це і як працює? URL: https://kievnovbud.com.ua/ua/2017/08/sistema-rozumnij-budinok-shho-ce-i-yak-pracyuye/
2. Що таке Інтернет речей? Основи IoT URL: https://www.guru99.com/iot-tutorial.html
3. Інтернет речей. URL: https://www.javatpoint.com/iot-internet-of-things
4. Що, чому та як в мікросервісній архітектурі. URL: https://medium.com/hashmapinc/the-what-why-and-how-of-a-microservices-architecture-4179579423a9
5. Docker – огляд. URL: https://www.tutorialspoint.com/docker/docker_overview.htm
6. Kubernetes – Огляд. URL: https://www.tutorialspoint.com/kubernetes/kubernetes_overview.htm
7. WebSockets – Overview. URL: https://www.tutorialspoint.com/websockets/websockets_overview.htm
8. Створення мікросервісів: Використання шлюзу API. URL: https://www.nginx.com/blog/building-microservices-using-an-api-gateway/
9. Підручник з хмарних обчислень для початківців. URL: https://www.guru99.com/cloud-computing-for-beginners.html
10. Брокери повідомлень. URL: https://www.ibm.com/cloud/learn/message-brokers
11. Інтернет речей – програми охорони здоров’я. URL: https://www.tutorialspoint.com/internet_of_things/internet_of_things_healthcare_applications.htm
12. Інтернет речей - програмне забезпечення. URL: https://www.tutorialspoint.com/internet_of_things/internet_of_things_software.htm

13. Що таке архітектура IoT. URL: https://www.avsystem.com/blog/what-is-iot-architecture/

14. Як правильно вибрати архітектуру бази даних для IoT. URL: https://internetofthingsagenda.techtarget.com/tip/How-to-select-the-right-IoT-database-architecture#:~:text=than%20SQL%20databases.-,The%20best%20database%20for%20most%20IoT%20applications%20is%20a%20unified,essential%20for%20many%20IoT%20uses.

15. 10 найкращих баз даних, які ви повинні вивчити в 2020 році. URL: https://www.improgrammer.net/top-10-databases-should-learn-2015/

16. 5 найкращих мов програмування для ефективного використання великих даних. URL: https://hub.packtpub.com/top-5-programming-languages-big-data/

17. Що таке IoT? Все, що вам потрібно знати про Інтернет речей прямо зараз. URL: https://www.zdnet.com/article/what-is-the-internet-of-things-everything-you-need-to-know-about-the-iot-right-now/

18. Що таке сервіс-орієнтована архітектура. URL: https://medium.com/@SoftwareDevelopmentCommunity/what-is-service-oriented-architecture-fa894d11a7ec#:~:text=Service%2DOriented%20Architecture%20(SOA),of%20vendors%20and%20other%20technologies.

19. Чим інтернет речей може допомогти охороні здоров’я? URL: https://www.wipro.com/en-US/business-process/what-can-iot-do-for-healthcare-/#:~:text=IoT%20enables%20healthcare%20professionals%20to,and%20reach%20the%20expected%20outcomes.

20. Шаблон конкуруючих споживачів. URL: https://docs.microsoft.com/en-us/azure/architecture/patterns/competing-consumers

21. Архітектурний стиль, керований подіями. URL: https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/event-driven

22. Архітектури великих даних. URL: https://docs.microsoft.com/en-us/azure/architecture/data-guide/big-data/

23. Виміри і контролювання циклів. URL: https://docs.microsoft.com/en-us/azure/architecture/example-scenario/iot/measure-control-loop

24. Аналіз та оптимізація циклів. Відстеження та керування циклами. URL: https://docs.microsoft.com/en-us/azure/architecture/example-scenario/iot/monitor-manage-loop

25. Аналіз та оптимізація циклів. URL: https://docs.microsoft.com/en-us/azure/architecture/example-scenario/iot/analyze-optimize-loop

26. Найкращі фронтенд-фреймворки 2020 року. URL: https://www.simform.com/best-frontend-frameworks/

ДОДАТОК А

Стаття за темою диплому

Актуальні проблеми комп'ютерних наук

УДК 004.4

Городний М. С., Тітова В. Ю.

Хмельницкий національний університет

РОЗРОБКА АРХІТЕКТУРИ ДОДАТКУ НА ОСНОВІ ТЕХНОЛОГІЙ «РОЗУМНИЙ БУДИНОК» ТА «ІНТЕРНЕТ РЕЧЕЙ»

На основі аналізу наукових джерел можемо зазначити, що система "розумний будинок" напряму пов'язана з технологіями інтернет речей і великі дані. Також була розглянута одна з популярних сервіс-орієнтованих архітектур, а саме мікросервісна.

Based on the analysis of scientific sources, we can say that the "smart home" system is directly related to the technology of the Internet of Things and big data. One of the popular service-oriented architectures, namely microservice, was also considered.

«Розумний будинок» являє собою автоматизовану систему управління різними компонентами домашньої інфраструктури. За допомогою спеціального обладнання, система може розпізнавати типові ситуації і реагувати на них, підключаючи ті чи інші компоненти. При цьому, «розумний дім» повністю контролює роботу кожного приладу і не допускає нераціонального їх використання. Таким чином, за рахунок «синергетичного ефекту», розумний будинок дозволяє забезпечити оптимальний режим використання всієї сукупності приладів в будинку. А це дозволяє створити максимально комфортні умови проживання людей при максимально економному споживанні ресурсів.[1]

Підключення повсякденних речей, вбудованих в електроніку, програмне забезпечення та датчики, до Інтернету, що дозволяє збирати та обмінюватися даними без взаємодії людини, називається Інтернетом речей (IoT).

Термін "Речі" в Інтернеті речей відноситься до всього і всього у повсякденному житті, до якого можна отримати доступ або підключитися через Інтернет.

IoT - це вдосконалена система автоматизації та аналітики, яка займається штучним інтелектом, датчиками, мережею, електронними повідомленнями, обміном повідомленнями тощо, щоб забезпечити цілісні системи для продукту чи послуг.

Робота IoT по-різному залежить від різних екосистем IoT (архітектури). Однак ключова концепція роботи там схожа. Весь робочий процес IoT починається з самих пристроїв, таких як смартфони, цифрові годинники, електронні прилади, які надійно спілкуються з платформою IoT. Платформи збирають та аналізують дані з усіх безлічі пристроїв і платформ і передають найцінніші дані разом із програмами на пристрої. [2]

Хоча кожна система IoT відрізняється, основа для кожної архітектури Інтернету речей, а також загального потоку процесів обробки даних приблизно однакова. Перш за все, він складається з речей, які є об'єктами, підключеними до Інтернету, які за допомогою вбудованих датчиків і виконавчих механізмів здатні відчувати навколишнє середовище та збирати інформацію, яка потім передається шлюзам IoT. Наступний етап складається із систем збору даних IoT та шлюзів, які збирають величезну масу необроблених даних, перетворюють їх у цифрові потоки, фільтрують та попередньо обробляють, щоб вони були готові до аналізу. Третій шар представлений крайовими пристроями, відповідальними за подальшу обробку та посилений аналіз даних. На цьому рівні також можуть вступити технології візуалізації та машинного навчання. Після цього, дані передаються в центри обробки даних, які можуть бути або хмарними, або локально встановленими. Саме тут дані зберігаються, управляються та аналізуються для поглибленого аналізу.

Охорона здоров’я є однією з основних галузей, яка була лідером і попередником у впровадженні технологій Інтернету речей. Причиною цього є те, що системи IoT допомагають використовувати високоякісну допомогу для пацієнтів і поєднують її з довгостроковою, але масовою економією.

В рамках охорони здоров’я ключові програми IoT включають, але не обмежуються цим, підвищення безпеки та безпеки пацієнтів та персоналу, зменшення непотрібних витрат на охорону здоров’я та надання відповідної підтримки в потрібний час, використовуючи інтелектуальні медичні та аварійні системи, що забезпечують IoT. .

Зважаючи на величезні проблеми з населенням, що попереду, однією з найбільших проблем у галузі охорони здоров’я є догляд за літніми людьми та моніторинг таких захворювань, як діабет та хвороби серця. Таким чином, профілактика відіграє ключову роль у забезпеченні кращого здоров'я пацієнтів похилого віку. Тому не дивно, що Інтернет речей набуває популярності, особливо в моніторингу стану здоров’я, де надійність, безпека та точний контроль у режимі реального часу є обов’язковими.

Приклад автоматичної системи моніторингу для пацієнтів літнього віку вимагає збору даних та аналізу в режимі реального часу, підключення до мережі для доступу до послуг інфраструктури та програми для підтримки користувальницького інтерфейсу та відображення. Отже, її архітектура повинна включати датчики тіла для збору даних про пацієнта, шлюзи для фільтрації та пересилання даних, мікроконтролери або мікропроцесори для аналізу та бездротової передачі даних у хмару, а також інструмент зв'язку для передачі даних у віддалене місце, як-от екстрене постачальника послуг або охорони здоров'я для моніторингу та відстеження.[3]

Вже багато років ми будуємо системи та вдосконалюємось. За ці роки з’явилося кілька технологій, архітектурних зразків та найкращих практик. Мікросервіси - одна з тих архітектурних моделей, яка з’явилася у світі доменного

дизайну, постійної доставки, автоматизації платформ та інфраструктури, масштабованих систем, програмування поліглотів та наполегливості.

Що таке архітектура мікросервісів у двох словах? Роберт К. Мартін ввів принцип єдиної відповідальності в якому йдеться про те, щоб зібрати разом ті речі, які змінюються з тієї самої причини, і розділити ті речі, які змінюються з різних причин.

Архітектура мікропослуг застосовує той самий підхід і поширює його на нещільно пов'язані служби, які можна розробляти, застосовувати та підтримувати самостійно. Кожна з цих служб відповідає за дискретні завдання і може спілкуватися з іншими службами за допомогою простих API для вирішення більшої складної бізнес-проблеми.[4]

Перелік посилань

1. https://kievnovbud.com.ua/ua/2017/08/sistema-rozumnij-budinok-shho-ce-i-yak-pracyuye/
2. https://www.javatpoint.com/iot-internet-of-things
3. https://www.avsystem.com/blog/what-is-iot-architecture/
4. https://medium.com/hashmapinc/the-what-why-and-how-of-a-microservices-architecture-4179579423a9

Додаток Б

Презентація

Вступ

Основні пункти роботи

Опис дослідження

Мета дослідження: розробка архітектури бекенд додатку для «інтернету речей».

Гіпотеза: розробити систему моніторингу, яка даватиме усі необхідні дані з мінімальною затримкою, мінімальним набором сервісів, баз даних, зв'язків та трафіком.

Задачі дослідження:
- Аналіз джерел інформації про «інтернет речей».
- Дослідження архітектурних рішень в технологіях «інтернет речей» та «великі дані».
- Розробка архітектури додатку на основі технології «інтернет речей».

Об'єкт дослідження: архітектура додатку на основі технології «розумний будинок».

Предмет дослідження: архітектура бекенд додатку на основі технології «інтернет речей».

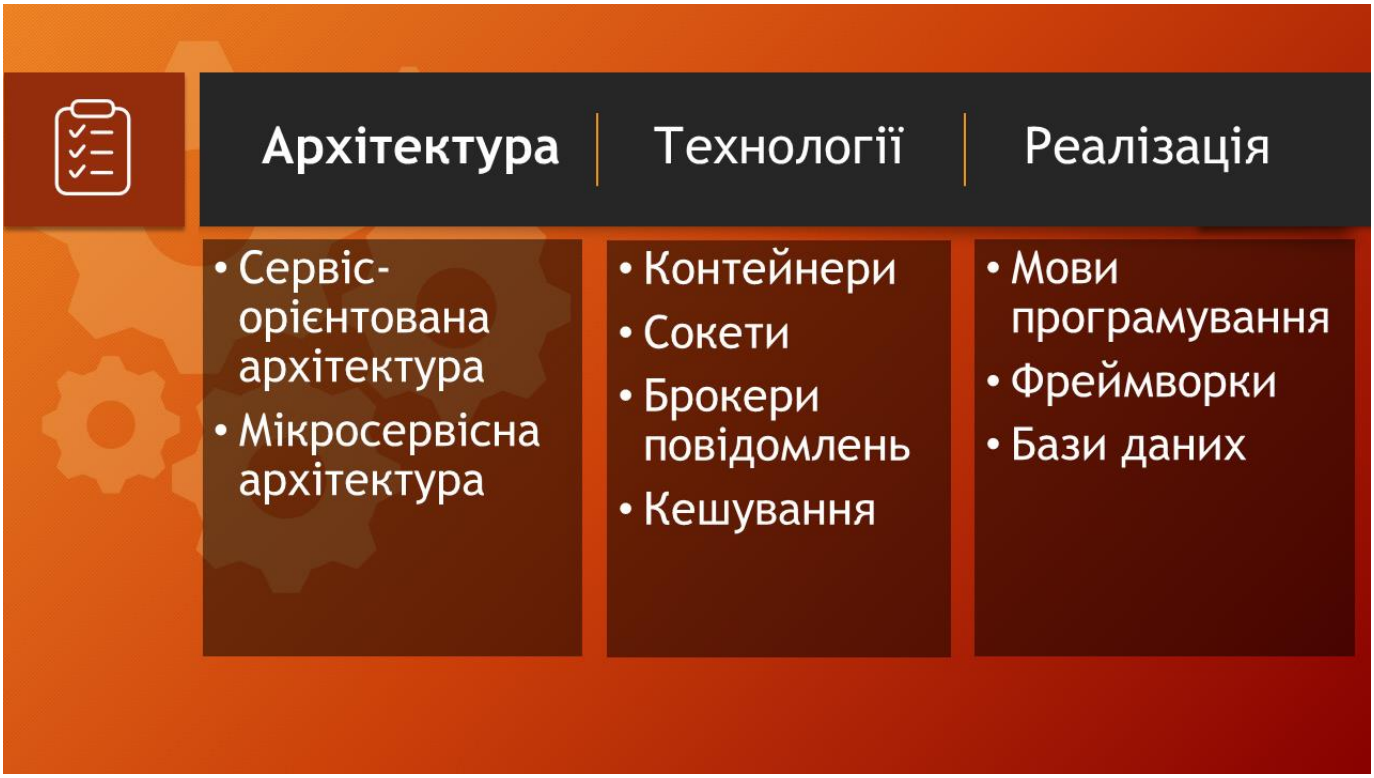
Наукова новизна: запропонована оригінальна архітектура для додатків «інтернет речей», що надає можливість моніторингу за життям і здоров'ям людей.

Виконана робота по розділам

1. Розглянуті бекенд технології
2. Розглянуті архітектурні підходи
3. Запропонована архітектура додатку для моніторингу
4. Запропонован приклад реалізації запропонованої архітектури

Розділ 1

Теоретичні відомості про технології які використовуються в сферах «інтернет речей» та «розумний будинок»


Архітектура
Технології
Реалізація
• Сервіс-орієнтована архітектура
• Мікросервісна архітектура
• Контейнери
• Сокети
• Брокери повідомлень
• Кешування
• Мови програмування
• Фреймворки
• Бази даних


Архітектура
Сервіс орієнтована архітектура:
• Повторне використання сервісу
• Простота обслуговування
• Незалежна платформа
• Доступність
• Надійність
• Масштабованість
Мікросервісна архітектура:
• Легко обслуговується та тестується
• Низька зв'язність
• Незалежно розгортається

Технології

- Контейнери (Docker, Kubernetes)
- Сокети
- Брокери повідомлень (Kafka, RabbitMQ, Spring Cloud Stream)
- Кешування (Redis)

Реалізація

Мови програмування: Java, TypeScript;
Фреймворки: Spring (Core, Boot, Data, MVC, Cloud);
Бази даних: Redis, MS SQL.

Розділ 2

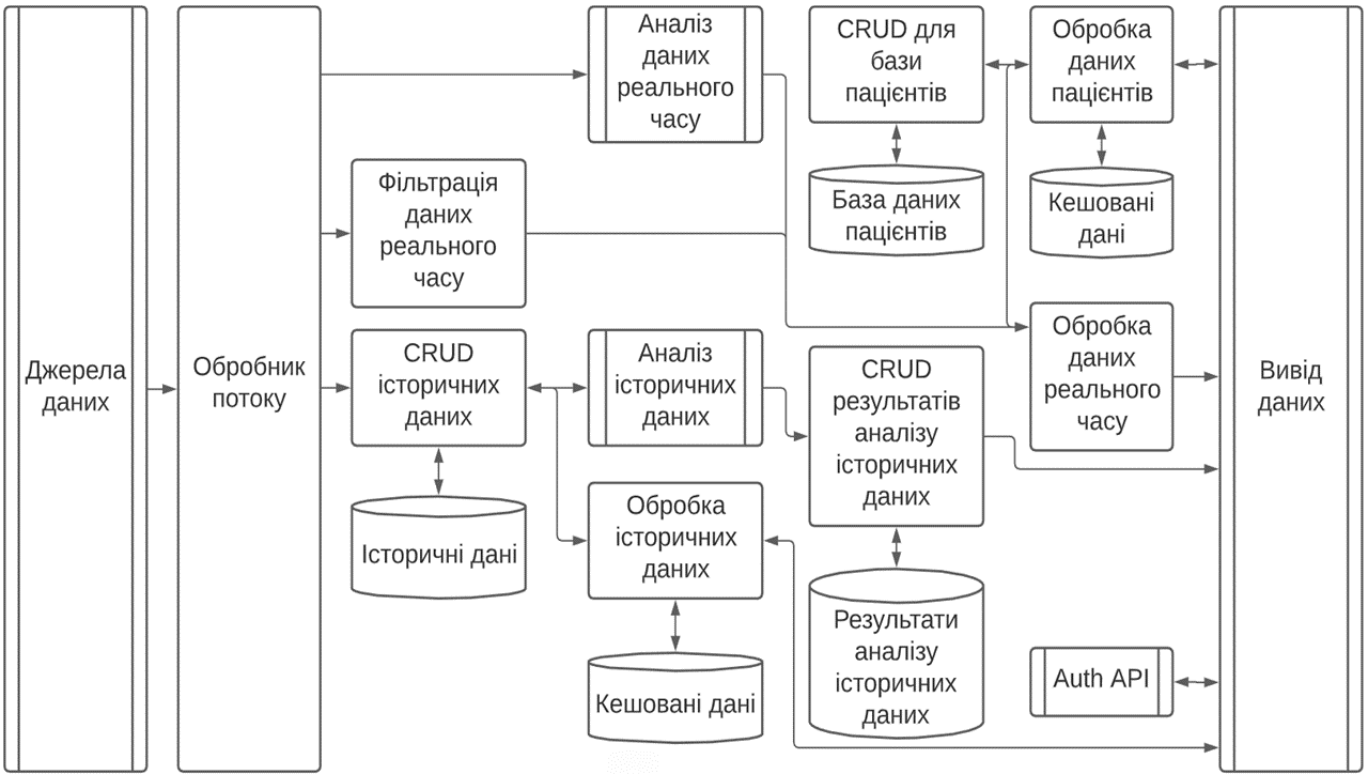
Архітектурні рішення в додатках «інтернет речей» та «великі дані»

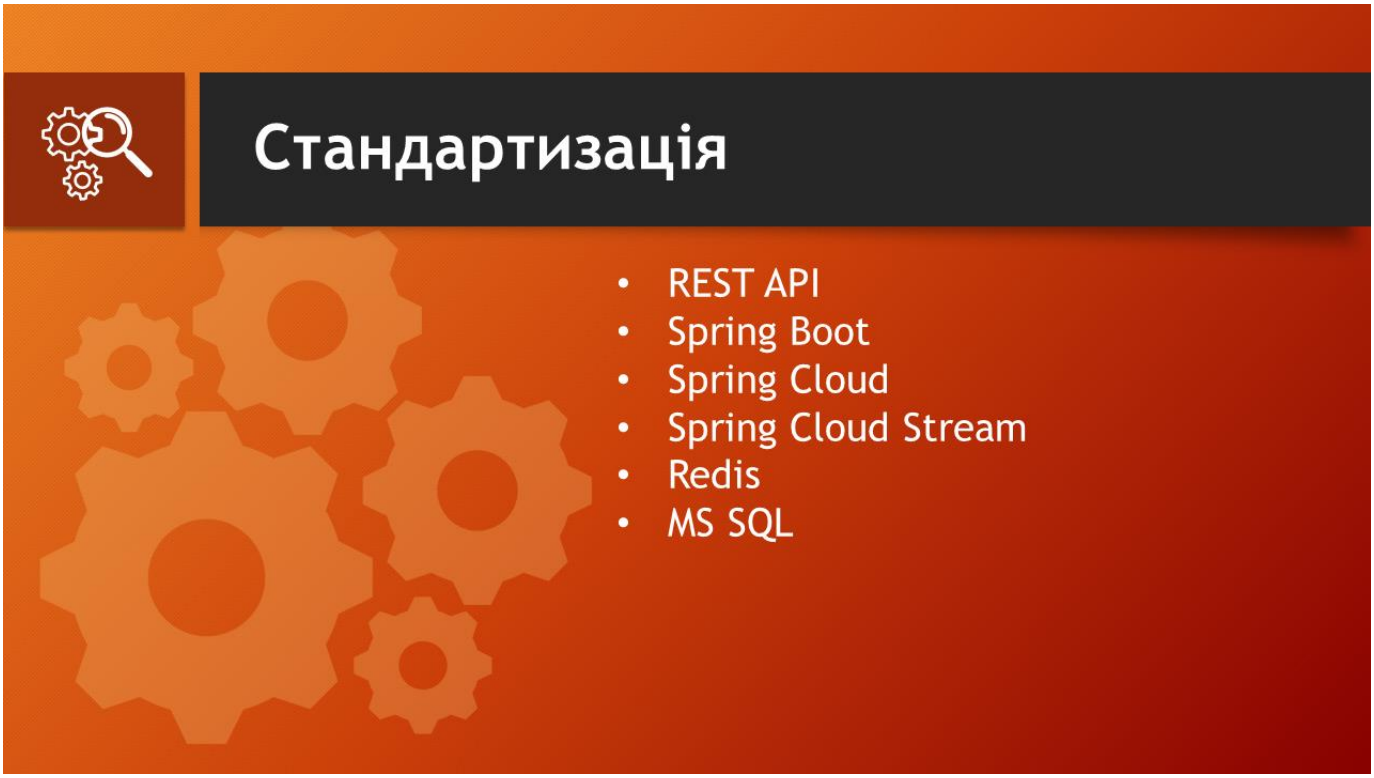
Архітектури

Event driven
Big data
IoT
Competing Consumers pattern

Розділ 3

Архітектура системи моніторингу на основі технології «розумний будинок»

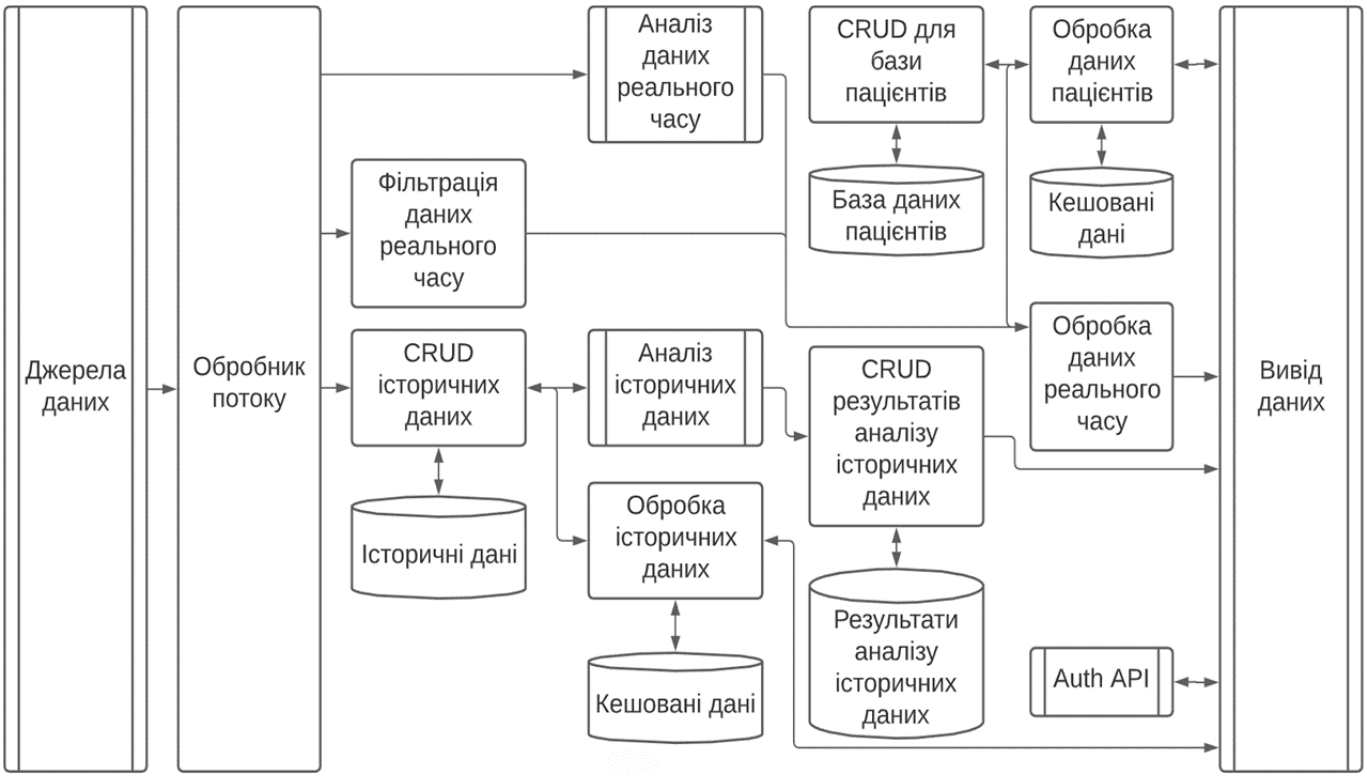



Стандартизація
REST API
Spring Boot
Spring Cloud
Spring Cloud Stream
Redis
MS SQL


Розділ 4
Приклад реалізації описаної архітектури

Вхідні дані

```
{
    "timestamp": "2020-11-21T18:38:54.942Z",
    "eventName": "DataFlow",
    "patientId": "9999",
    "healthState": {
        "pulseRate": "60",
        "heartRate": "60",
        "temperature": "36.6"
    },
    "dustLevel": "Low",
    "humidity": "Low",
    "roomTemperature": "20"
}
```



Вхідні дані

```
{
    "timestamp": "2020-11-21T18:38:54.942Z",
    "eventName": "Alarm | Warning | NoData",
    "patientId": "9999",
    "healthState": {
        "pulseRate": "60",
        "heartRate": "60",
        "temperature": "36.6"
    }
}
```

Висновок

Було дано опис однієї з можливих реалізацій архітектури. На основі цього можна сказати, що архітектура є досить прозорою і зрозумілою, навіть при зростанні даних та ускладнення логіки і структури може залишатися такою ж прозорою, особливо якщо дотримуватися стандартизації

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА ДИПЛОМНУ РОБОТУ

Дипломник Городній Михайло Сергійович

Тема Система моніторингу за життєвими людей і водяним здоров'ям на основі технології дерева будинок

Спеціальність 113 – Прикладна математика

Обсяг дипломної роботи:

Кількість листів креслень 7; кількість сторінок записки 85

1. Короткий зміст ДР та прийнятих рішень У роботі розроблено архітектуру сервісу побудови системи моніторингу за життєвими і здоров'ями людей.

2. Висновок про відповідність ДР поставленому завданню відповідає у повній мірі

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У вступі подано загальну характеристику поставленої задачі. Вказано предмет дослідження, актуальність. У першому розділі було описано основні технології, інструменти-засоби. У другому розділі проведено аналіз предметної області. У третьому розділі запропоновано архітектуру системи моніторингу за життєвими людей і водяним здоров'ям. У четвертому розділі продемонстровано приклад використання розробленої архітектури.

4. Позитивні сторони роботи Робота має комплексну наукову та практичну цінність. Наукова цінність полягає в розробці архітектури для сервісу інтернету речей, який дозволяє виконувати моніторинг за життєвими людей і водяним здоров'ям. Практична цінність полягає у актуальності задачі моніторингу за життєвими людей і водяним здоров'ям у сучасному світі та потребі у розробці такого сервісу.

5. Негативні сторони роботи в роботі не повністю наведено огляд технічних засобів в галузі, деякими що не тягнуть в кваліфікації та практичної цінності висновку.

6. Оцінка графічного оформлення та пояснювальної записки роботи відповідає вимогам

7. Відгук про роботу в цілому Робота заслуговує позитивної оцінки. Матеріал добре структурований та чітко викладений, автором, набагато досконалість та ефективність прийнятих у роботі рішень

8. Інші зауваження —//—

9. Оцінка дипломної роботи виховується позитивні та несуттєві сторони роботи відповідає кваліфікаційним вимогам, заслуговує оцінки „добре / В" 84,50 бл.

РЕЦЕНЗЕНТ (прізвище, ім'я, по-батькові, посада, місце роботи) Ключ Юрій Павлович, к.т.н., доцент кафедри комп'ютерних та кібернетичних систем і мереж

"24" 11 2020 р. _____
(підпис)

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ ТЕЛЕКОМУНІКАЦІЙ, МЕДІЙНИХ ТА ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Система моніторингу за життям людей з вадами здоров'я на основі технології «розумний будинок»

Автор: Городний Михайло Сергійович

Спеціальність: 113 – Прикладна математика

Освітня програма: Прикладна математика

Науковий керівник: Тітова Віра Юріївна, к.т.н., доцент

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних). Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи.	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) більшість запозичень розміщені в розділах аналізу існуючих аналогів та прототипів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи;
2) частина запозичень є фрагментарними та мають належним чином оформленні посилання;
3) окремі виявлені збіги є загальновідомою інформацією або загальновживаними фразами, які не потребують спеціальних посилань на першоджерела;
4) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування загальновживаних англомовних та україномовних термінів (наприклад http-запит, тощо), що не є модифікацією тексту.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 3.88%, що, з урахуванням наведених обґрунтувань, повністю відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ В.Ю. Тітова

Завідувач кафедри ТМІТ _____ С.К. Підченко

Дата: 07.12.2020

UNICHECK

Имя пользователя:
Kafedra TMIT KhNU

ID проверки:
1005372594

Дата проверки:
05.12.2020 01:17:00 EET

Тип проверки:
Doc vs Internet + Library

Дата отчета:
05.12.2020 01:20:21 EET

ID пользователя:
100005657

Название файла: Городний_ПМм-19-1

Количество страниц: 74 Количество слов: 14979 Количество символов: 111277 Размер файла: 712.48 KB ID файла: 1005665203

659 слов помечены как "исключенные" и не учитываются в подсчете слов

3.88% Совпадения

Наибольшее совпадение: 3.3% с Интернет-источником (https://kievnovbud.com.ua/ua/2017/08/sistema-rozumnij-budin..

3.69% Источники из Интернета	7	Страница 76
0.27% Источники из Библиотеки	15	Страница 76

0.69% Цитат

Цитаты	4	Страница 77

Не найдено ни одной ссылки

0% Исключений

Нет исключенных источников

Модификации

Обнаружены модификации текста. Подробная информация доступна в онлайн-отчете.

Замененные символы	2

Tue Nov 24 12:15:34 EET 2020, Стецюк Віктор Іванович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальное совпадение с одним документом 1.0%

Словари проверки: en_US, ru_RU, ua_UA. **Ошибок в документах: 7%**

ID: 81076 Название: Система моніторингу за життям людей з вадами здоров'я на основі технологій розумного будинку Добавлено в БД: 2020-11-24 Авторы: Городний Михайло Сергійович Руководители: Тітова Вєра Юріївна Консультанты: Опоненты:	Документ		Суммарное совпадение по Базе Данных	
	Символы	Лексемы	Символы	Лексемы
	103504	904	2212 (2%)	27 (3%)

Источник плагиата

ID	Описание	Наличие плагиата в документе	
		Символы	Лексемы

Завідувачу кафедри ТМІТ
д-р.техн.наук Підченку С.К.

Городний Михайло Сергійович
ПІБ здобувача вищої освіти

ФПКТС, 2 курсу, групи ПІМм-19-1

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіатоповіщений та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів(Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

12.11.2020
дата

підпис