Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Системотехніки
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Інформаційні технології проєктування клієнт-серверних систем обслуговування
(тема)

Виконав:
студент 2 курсу, групи СПРм-22-2
Давидов Д.О.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-наукова

Освітня програма Системне проєктування
(повна назва освітньої програми)

Керівник проф. СТ Гребеннік І.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ____________ Гребеннік І.В.
(підпис) (прізвище, ініціали)

2024 р.

Я, як студент ХНУРЕ, розумію і підтримую політику закладу із академічної доброчесності. Я не надавав і не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Давидов Д. О.

Кваліфікаційна робота не містить відомостей заборонених до відкритого опублікування.

Кваліфікаційна робота виконана у відповідності до стандартів, що діють в Україні.

Попередній захист проведено _________________________________

Керівник кваліфікаційної роботи проф. Гребеннік І. В.

Харківський національний університет радіоелектроніки

Факультет Комп’ютерних наук

Кафедра Системотехніки

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп’ютерні науки
(код і повна назва)

Тип програми освітньо-наукова

Освітня програма Системне проєктування
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. Кафедри ____________________________

«____»____________________ 20 ________ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Давидову Денису Олександровичу
(прізвище, ім’я, по батькові)

1. Тема роботи Інформаційні технології проєктування клієнт-серверних систем обслуговування
затверджена наказом університету від 01 квітня 2024 р. № 259 Ст___
2. Термін подання студентом роботи до екзаменаційної комісії 16 06 2022 р.
3. Вихідні дані до роботи проаналізувати сучасні інформаційні технології з проєктування клієнт-серверних систем обслуговування, порівняти їх один з одним, виробити рекомендації щодо їх застосування та на практиці розробити дану систему за виробленими рекомендаціями, потім дати оцінку цим рекомендаціям та зробити висновок щодо цього з описом також проблем та рекомендацій щодо вироблених рекомендацій
4. Перелік питань, що потрібно опрацювати в роботі провести аналіз сучасних інформаційних систем; обрати потрібні шляхом аналізу та порівняння; виробити рекомендації з проєктування клієнт-серверних систем обслуговування; розробити систему за цими рекомендаціями; провести аналіз предметної області; сформулювати та оформити вимоги до інформаційної системи; провести фізичне моделювання БД; реалізувати фізичну модель БД; створити серверну частину системи та розробити клієнтський інтерфейс для взаємодії з користувачем; провести аналіз створеної системи; описати висновки, проблеми і рекомендації.

5. Перелік графічного матеріалу із зазначенням кресленників, схем, плакатів, комп'ютерних ілюстрацій 26 слайдів та додаток В зі слайдами презентації.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання на атестаційну роботу	18.03.2024	виконано
2	Аналіз завдання, пошук літератури та джерел за темою атестаційної роботи	20.03.2024	виконано
3	Розробка постановки ходу дослідження	14.04.2024	виконано
4	Вироблення рекомендацій щодо інформаційних технологій	18.04.2024	виконано
5	Проєктування системи за виробленими рекомендаціями	27.04.2024	виконано
6	Розробка формули комбінованої метрики	26.05.2024	виконано
7	Проведення аналізу дослідження	30.05.2024	виконано
8	Опис результатів дослідження, проблем та рекомендацій	06.06.2024	виконано
9	Оформлення пояснювальної записки та підготовка кодової бази для звіту	10.06.2024	виконано
10	Перевірка на плагіат	17.06.2024	виконано
11	Представлення на рецензування	18.06.2024	виконано
12	Підготовка презентаційних слайдів	18.06.2024	виконано
13	Занесення диплому в електронний архів	18.06.2024	виконано
14	Отримання допуску у зав. кафедри	18.06.2024	виконано

Дата видачі завдання 18 березень 2024 р.

Студент ______________________________
(підпис)

Керівник роботи ______________________________ проф. СТ Гребеннік І.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до магістерської кваліфікаційної роботи: 149 с., 4 табл., 26 рис., 1 додаток, 26 джерел інформації.

ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ, КЛІЄНТ-СЕРВЕРНА СИСТЕМА, АРХІТЕКТУРА СПЕЦИФІКАЦІЯ, СИСТЕМА УПРАВЛІННЯ БАЗАМИ ДАНИХ, MYSQL, ФРЕЙМВОРК, HTML, CSS, REACT, JAVA, HIBERNATE, SPRING, REST, ІНТЕРФЕЙС, МІКРОСЕРВІС, МОНОЛІТ, КЕШ

Об'єктом дослідження є сучасні інформаційні технології проєктування клієнт-серверних систем обслуговування, їх недоліки і переваги, умови та обмеження їх використання.

Метою роботи є визначення ефективних інформаційних технологій для подальшого їх застосування у проєктуванні клієнт-серверної системи обслуговування.

Предметом досліджень магістерської роботи є інформаційні технології і програмні методи створення клієнтської і серверної частин систем обслуговування.

У ході роботи було проведено аналіз предметної області, визначено об’єкт та предмет дослідження, сформульовано мету та завдання, також надане обґрунтування актуальності даного дослідження, проведено аналіз сучасних інформаційних технологій, які зараз дуже широко використовуються в проєктуванні клієнт-серверних систем.

Підсумком дослідження є опис переваг та недоліків існуючих інформаційних технологій, їх застосування та призначення у проєктуванні таких систем, розглянуто типові архітектури, надано їм характеристики, спроєктовано клієнт-серверну систему обслуговування з обранням інформаційних технології та архітектури за виробленими рекомендаціями, проаналізовано результати.

ABSTRACT

Master’s Thesis: 149 pages, 4 tables, 26 figures, 1 appendices, 26 title.

INFORMATION TECHNOLOGY, CLIENT-SERVER SYSTEM, ARCHITECTURE SPECIFICATION, DATABASE MANAGEMENT SYSTEM, MYSQL, FRAMEWORK, HTML, CSS, REACT, JAVA, HIBERNATE, SPRING, REST, ACCESS INTERFACE, MICROSERVICE, MONOLITH, CACHE,

The object of research is modern information technologies for designing client-server service systems, their disadvantages and advantages, conditions and limitations of their use.

The purpose of the work is to identify effective information technologies for their further application in the design of client-server service systems.

The subject of the master's thesis is information technology and software methods for creating the client and server parts of the system.

In the course of the work, the subject area was analyzed, the object and subject of the study were determined, the goal and objectives were formulated, and the relevance of this study was justified, as well as an analysis of modern information technologies that are now widely used in the design of client-server systems.

The result of the study is a description of the advantages and disadvantages of existing information technologies, their application and purpose in the design of such systems, a review of typical architectures, their characteristics, the design of a client-server service system with the choice of information technology and architecture according to the developed recommendations, and analysis of the results.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧКИ ……………………………...…… 7
ВСТУП ………………………………………………………………...…... 8
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ……………………………….....10
1.1 Опис об’єкта дослідження ……………………………................. 10
1.2 Загальний опис сучасних архітектур ………………………………… 11
1.3 Аналіз монолітної архітектури ……………………………………… 12
1.4 Аналіз мікросервісної архітектури ………………………………… 14
1.5 Опис вимог до системи ……………………………………………… 17
1.6 Дослідження клієнт-серверної архітектури………………………… 18
1.7 Дослідження оптимізації процесу роботи з даними..………………… 20
1.8 Постановка задач до проєктування системи………………………… 22
2 ОПИС ПРИЙНЯТИХ ПРОЄКТНИХ РІШЕНЬ …………...…………….. 24
2.1 Обґрунтування вибору мікросервісної архітектури………..…………... 24
2.2 Рекомендації по використанню інформаційних технологій і архітектур… 26
2.3 Обґрунтування вибору стеку технологій для розробки...………….............. 40
2.4 Опис ходу розробки системи за виробленими рекомендаціями……......... 42
3 ПРОЄКТУВАННЯ АРХІТЕКТУРИ КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ…… 43
3.1 Опис схеми бази даних……………………………………….......... 43
3.2 Опис сервісів за предметною областю ………………………………... 44
3.3 Проєктування системи за виробленими рекомендаціями ………………… 51
3.4 Моніторинг та аналіз результатів проєктування і роботи системи ……….. 59
3.5 Проблеми та рекомендації до використання …………………………….. 67
ВИСНОВКИ …………………………………………………………… 69
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ……………………………………... 70
ДОДАТОК А ………………………………………………………………. 73
ДОДАТОК Б ………………………………………………………………. 98

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧКИ

БД – база даних;

ООП – об’єктно-орієнтоване програмування;

СУБД – система управління базами даних;

ІС – інформаційна система;

API – (англ. Application Programming Interface) програмний інтерфейс; додатку;

CSS – (англ. CascadingStyleSheets) каскадні таблиці стилів;

EER – (англ. ExtendedEntityRelationship) модель це розширена модель «сутність-зв’язок»;

JAVA – скомпільована мова програмування;

HTTP – (англ. Hyper Text Transfer Protocol) протокол передачі гіпертексту;

REST – (англ. Representational State Transfer) архітектурний стиль;

HTML – (англ. Hypertext MarkupLanguage) мова гіпертекстової розмітки;

URL – uniform resource locator;

JS – (англ. JavaScript) мова програмування;

MSA – (англ. MicroService Architecture) мікросервісна архітектура;

AWS – Amazon Web Services;

DAO – Data Access Object (об’єкт доступу до даних);

SQL – (англ. StructuredQueryLanguage) структурована мова запитів.

ВСТУП

В сучасному світі постійно з’являються нові поняття та технології у всіх сферах діяльності людини. Ці зміни являються прямим наслідком вирішення поставлених задач та проблем, які вже існують та потребують негайного вирішення, аби йти вперед. За останні декілька десятиліть сфера інформаційних технологій значно змінилася та з’явилося багато різноманітних напрямків для реалізації нових ідей, які стали дійсно реальними у реалізації. Це і створення першого комп’ютера, протоколів передачі даних, згодом інтернету та багато чого іншого пов’язаного з розвитком у цьому напрямі.

Поява цифрових даних вимагала пошуків у реалізації нових пристроїв, які мали б змогу зберігати ці дані у великих об’ємах, а також щоб доступ до цих даних також був швидкий і з ними можна було б проводити потрібні маніпуляції та розрахунки. Як результат, згодом почали з’являтися нові мови низькорівневого програмування, а вже з часом більш високорівневі.

Наприклад, вважається, що мова асемблера є низькорівневою мовою, що на відміну від високорівневих мов, що повністю або частково абстрагуються від команд процесора, або точніше деталей його реалізації, низькорівневі мови ближче до специфіки роботи програми або самого процесора. Вони є більш складніші для розуміння людиною та потребують дещо більшого розуміння, бо часто програма написана на асемблері під один тип процесорів не буде використовуватися та бути придатною для роботи з іншими типами. Але перевагою є те, що такі програми компактні та швидкі.

Високорівневі мови програмування потребують також досить багато знань та розуміння роботи програм в цілому і навіть на апаратному рівні, але звичайно не на такому ж рівні як низькорівневі мови. Можливим це стало тому, що почали з’являтися більш абстрактні реалізації однієї і тієї ж самої задачі, бо не потрібно думати про реалізацію команд для процесора, що значно пришвидшило розробку програм в цілому. І даний підхід розвитку високорівневих мов програмування почав розвиватися і почали з’являтися нові мови програмування, які дозволяють

значно швидше та головне простіше писати програмний код, який буде також більш зрозумілий для читання та розуміння. Як результат, почали з'являтися принципи та правила написання коду, але все одно залишалося багато проблем та задач для вирішення. Однією з проблем була велика кількість коду, яку хотілося зменшити для збільшення об'єму пам'яті та пришвидшити компіляцію та роботу програми. І тому було розроблено функціональне програмування, що дозволяє використовувати одну й ту ж саму функцію там де потрібно не переписуючи при цьому увесь код, а лише викликав цю функцію за її ім'ям. Також з'явилося ООП – це об'єктно-орієнтоване програмування, що також дозволило зменшити кількість дубльованого коду та підвищити його читабельність. Програми становилися все більш складними та дозволяли вирішувати вже значно складніші задачі, але залишалася проблема у архітектурі програм для вирішення таких задач. Тобто яким саме чином потрібно писати код. І проблемою, яка була раніше і залишається зараз – це підтримка старого коду і додавання нових функцій, намагаючись при цьому не переробլювати усю програму з нуля, бо це дуже дорого як з точки зору часу, так і коштовності.

Однією з найпопулярніших програмних архітектур – є моноліт. Це архітектура, яка виступає одним єдиним модулем, в якому зібрані всі функціональні частини програми разом, що дозволяє простіше розгортати та керувати додатком. В ній мається один кодовий репозиторій, пряма комунікація між компонентами та звичайно маються і проблеми. Дана архітектура чудово підходить для малих та середніх за розміром проектів, але зі збільшенням кодової бази та функціональності код стає більш складним сам по собі, а також у додаванні нового та підтримки старого.

Існують звичайно і багато інших сучасних архітектур таких як: мікросервісна архітектура, серверна архітектура з використання контейнерів та інші. І результат дослідження – це аналіз наявних та знаходження найоптимальніших інформаційних технологій і як задача – розробка клієнт-серверної системи обслуговування з обранням архітектури та інформаційних технологій проєктування.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис об’єкта дослідження

У сучасному цифровому світі клієнт-серверні системи обслуговування стають все більш важливими для забезпечення ефективності та продуктивності бізнес-процесів. Дане дослідження присвячено аналізу сучасних інформаційних технологій, використовуваних у проєктуванні таких систем, з урахуванням їхніх недоліків, переваг, а також умов та обмежень їх використання.

Метою даної роботи є систематизація та аналіз сучасних підходів в інформаційних технологіях для проєктування клієнт-серверних систем обслуговування, виявлення їхніх переваг та недоліків, а також встановлення умов та обмежень, які впливають на їхнє використання.

Основні завдання дослідження:

1. Провести огляд сучасних інформаційних технологій, що використовуються у клієнт-серверних системах обслуговування.

2. Виділити основні переваги та недоліки таких систем.

3. Проаналізувати умови та обмеження, які впливають на їхнє ефективне використання.

4. Запропонувати рекомендації щодо оптимального вибору та використання клієнт-серверних систем обслуговування в певних умовах.

В результаті дослідження спрямоване на розкриття сутності та значення сучасних інформаційних технологій у проєктуванні клієнт-серверних систем обслуговування. Аналіз недоліків та переваг, а також умов та обмежень використання таких систем допоможе підвищити ефективність їхнього впровадження та функціонування в різних сферах діяльності. На основі вироблених рекомендацій спроєктовано клієнт-серверу систему обслуговування.

1.2 Загальний опис сучасних архітектур

У сучасному цифровому світі інформаційні технології стають необхідною складовою для функціонування різноманітних сфер життя, від бізнесу та освіти до медицини та науки. Швидкий темп розвитку технологій вимагає постійного оновлення знань і використання новітніх рішень для оптимізації процесів та підвищення продуктивності.

Інформаційні технології охоплюють широкий спектр інструментів та систем, що дозволяють збирати, зберігати, обробляти та передавати дані. Основні галузі інформаційних технологій включають в себе комп'ютерну науку, мережеві технології, бази даних, інформаційну безпеку, штучний інтелект, обробку сигналів та багато інших.

Однією з ключових архітектурних концепцій є клієнт-серверна модель, яка забезпечує розділення обов'язків між клієнтськими та серверними пристроями. Ця модель дозволяє ефективно керувати обміном даних та ресурсами між різними компонентами системи. Клієнт-серверні системи обслуговування широко використовуються у веб-додатках, мобільних додатках, хмарних обчисленнях та інших сферах.

Поряд з клієнт-серверною архітектурою, існують інші моделі, такі як розподілена архітектура, що вказує на рівень самостійності та взаємозв'язку пристроїв у мережі, без прямого керування з боку центрального вузла. Також варто згадати про мікросервісну архітектуру, яка розділяє додаток на невеликі автономні модулі, що дозволяє гнучко керувати та масштабувати додаток.

У контексті цієї роботи, дослідження сучасних інформаційних технологій та архітектур для проєктування клієнт-серверних систем обслуговування є важливим кроком у напрямку оптимізації бізнес-процесів та забезпечення високої ефективності та надійності цих систем. Що у свою чергу каже про те, що дані системи повинні бути легко та швидко масштабованими під потрібне навантаження та і в цілому бути відмовостійкими.

1.3 Аналіз монолітної архітектури

Монолітна архітектура представляє собою традиційний підхід до розробки програмного забезпечення, де весь функціонал програми об'єднаний в один великий блок, що працює як єдиний юніт (рисунок 1.1).

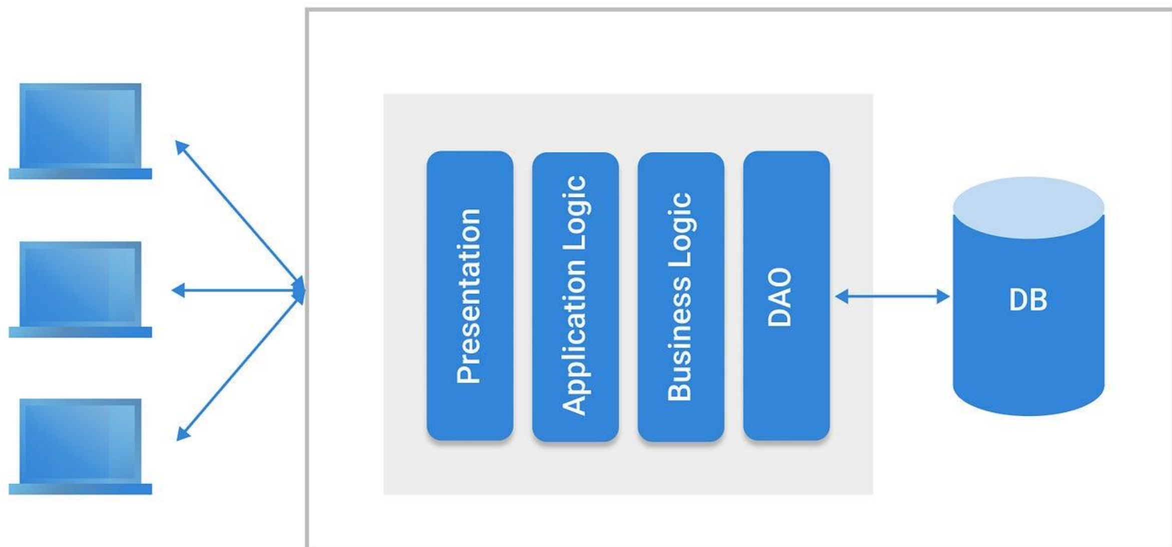

Рисунок 1.1 – Монолітна архітектура

Цей підхід характеризується тим, що весь код програми, включаючи логіку додатку та його інтерфейс, розміщений в одному збірнику або виконавчому файлі.

Основними перевагами монолітної архітектури є простота розробки та розгортання програмного забезпечення. Завдяки тому, що всі компоненти додатку знаходяться в одному місці, розробка стає більш прямолінійною, а впровадження - менш складним процесом. Крім того, у монолітних додатках немає необхідності у використанні спеціальних механізмів для взаємодії між різними компонентами, що спрощує процес комунікації між ними. Однак існують і недоліки монолітної архітектури. Наприклад, складності

масштабування та підтримки. У випадку потреби у розширенні функціоналу чи обробці більшого обсягу даних, необхідно модифікувати весь моноліт, що може призвести до збільшення часу та витрат на розробку та підтримку системи. Крім того, велика кількість коду в одному місці ускладнює розуміння та підтримку програмного забезпечення. Також важливо враховувати, що у разі виникнення помилки або несправності в одній частині додатку, це може призвести до зупинки всього додатку. У зв'язку з цим, в останні роки спостерігається тенденція до переходу від монолітних архітектур до більш розподілених та масштабованих підходів, таких як мікросервісна архітектура. Тим не менш, монолітні архітектури залишаються популярними у деяких випадках, зокрема, у малих проектах або там, де немає потреби у великому масштабуванні або частих змінах. Варто також відзначити, що використання монолітної архітектури може бути доречним у випадках, коли стабільність та надійність системи мають вищий пріоритет, ніж її гнучкість чи швидкість розгортання нових функцій.

Незважаючи на деякі недоліки, монолітна архітектура залишається популярним вибором у деяких випадках завдяки своїй простоті та зручності. Вона особливо ефективна в невеликих проектах або у випадках, коли швидкість розробки має вищий пріоритет, ніж масштабованість. Також важливою перевагою монолітної архітектури є можливість легкої локалізації помилок та їх виправлення у випадку виникнення проблем. Оскільки весь код знаходиться в одному місці, виявлення та виправлення помилок може бути здійснено швидко та ефективно. Крім того, монолітні системи можуть мати менший вплив на загальні витрати на розробку та підтримку порівняно з більш складними архітектурними підходами.

Важливо також враховувати, що монолітні системи можуть бути легше керованими з точки зору моніторингу та налагодження завдяки їхній централізованій структурі. Крім того, вони забезпечують більш простий механізм для контролю версій та розгортання нових випусків програмного забезпечення. У деяких випадках монолітна архітектура може бути ефективним вибором для швидкого відгуку на потреби ринку та втілення нових ідей. Також

варто відзначити, що у деяких випадках монолітна архітектура може бути менш витратною з точки зору інфраструктури, оскільки не потрібно підтримувати складні механізми для взаємодії різних сервісів чи модулів.

1.4 Аналіз мікросервісної архітектури

Мікросервісна архітектура є одним з передових підходів у сучасній розробці програмного забезпечення, яка дозволяє створювати складні системи, розділяючи їх на невеликі, самостійні та автономні компоненти. Цей підхід відбиває сучасні вимоги до гнучкості, масштабованості та швидкодії у розробці та експлуатації програмного забезпечення. Аналіз мікросервісної архітектури виявляє її переваги, недоліки, а також вимоги та обмеження у використанні (рисунок 1.2).

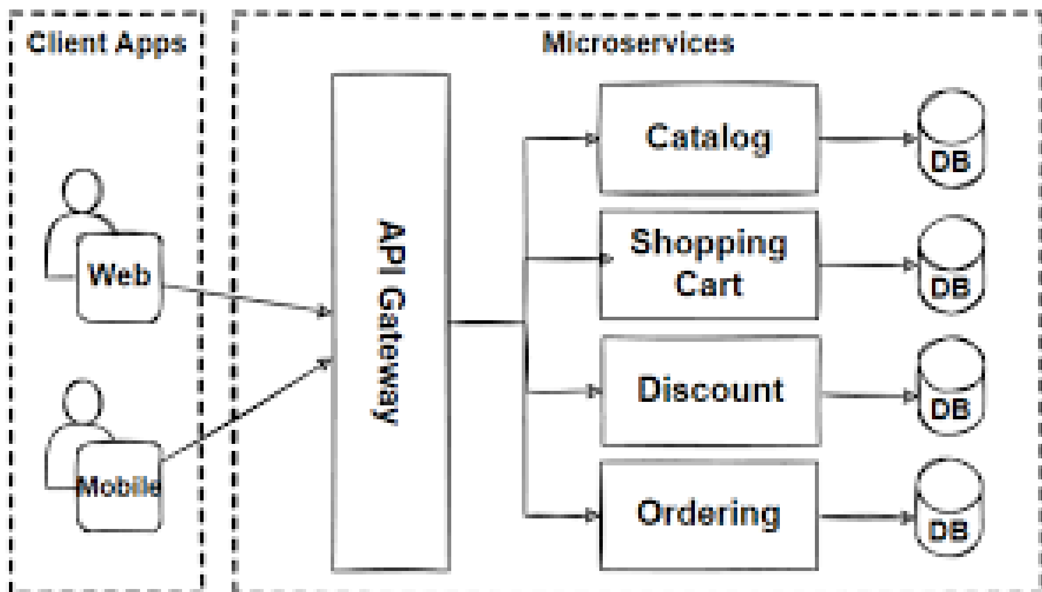

Рисунок 1.2 – Мікросервісна архітектура

Мікросервісна архітектура має наступні переваги:

1. Гнучкість та скорочення часу розробки: Розділення системи на окремі сервіси дозволяє розробникам працювати над ними паралельно, що значно прискорює процес розробки.

2. Масштабованість: Кожен мікросервіс може бути масштабований окремо, що дозволяє гнучко реагувати на зміну навантаження та вимог до системи.

3. Легка підтримка та оновлення: Оскільки кожен сервіс є окремим, оновлення можуть бути впроваджені без перерв у роботі інших компонентів системи.

4. Розділення відповідальностей: Кожен сервіс відповідає за виконання конкретного функціоналу, що спрощує розуміння та підтримку системи.

5. Технологічна гетерогенність: Розробники можуть використовувати різні технології та мови програмування для реалізації різних сервісів, що дозволяє вибирати оптимальні інструменти для кожного завдання [1].

Недоліки мікросервісної архітектури:

1. Складність управління: З введенням багатьох сервісів збільшується складність управління та моніторингу всією системою.

2. Мережева складність: Взаємодія між сервісами відбувається через мережу, що може призвести до збільшення затримок та проблем із безпекою.

3. Системна складність: Спрощення розробки окремих сервісів може призвести до складності інтеграції та тестування всієї системи в цілому.

4. Потреба у збереженні цілісності даних: Виникає необхідність у впровадженні механізмів синхронізації та збереження цілісності даних між різними сервісами.

Звичайно існують вимоги та обмеження у використанні мікросервісної архітектури:

1. Культурна зміна: Впровадження мікросервісної архітектури часто потребує зміни корпоративної культури та методології розробки.

2. Необхідність у високопродуктивній інфраструктурі: Розгортання багатьох сервісів може вимагати значних інвестицій у високопродуктивне обладнання та інфраструктуру.

3. Безпека: Збільшення кількості точок входу та взаємодії може підвищити ризик злому та кібератак.

4. Моніторинг та логування: Необхідно розробити ефективні механізми для моніторингу та логування роботи кожного сервісу [2].

Мікросервісна архітектура, безумовно, є потужним інструментом у сучасній розробці програмного забезпечення. Проте, перед впровадженням необхідно ретельно зважити на всі переваги та недоліки цього підходу, а також врахувати конкретні потреби та характеристики проекту. Наприклад, для великих та складних систем, які вимагають великої гнучкості та масштабованості, мікросервісна архітектура може бути оптимальним вибором. Однак у випадках, коли простота та швидкість розробки є критичними факторами, може бути доцільним залишитися при монолітній архітектурі. Крім того, важливо розглянути можливість комбінування обох підходів, де деякі частини системи реалізовані за мікросервісним принципом, а інші залишаються монолітними. Такий гібридний підхід дозволяє поєднати переваги обох архітектурних моделей та оптимально використовувати їх у конкретному проекті.

Треба звернути увагу, що успіх впровадження мікросервісної архітектури залежить не лише від технічних аспектів, але й від організаційної культури та процесів у компанії. Необхідно створити відповідні команди, процеси розробки та управління, які були б адаптовані до нового підходу. Також слід врахувати, що впровадження мікросервісної архітектури може вимагати значних змін у культурі розробки та комунікації в організації. Тому перед переходом до мікросервісів важливо ретельно проаналізувати всі аспекти та підготувати організацію до цього кроку.

В цілому, аналіз мікросервісної архітектури підкреслює її значення та потенціал у сучасній розробці програмного забезпечення. Однак успішне

впровадження вимагає не лише технічних знань та навичок, але й врахування організаційних, культурних та процесних аспектів. Лише з врахуванням усіх цих факторів можна забезпечити успішне та ефективне впровадження мікросервісної архітектури виходячи саме з відповідних потреб та поставлених задач на усіх етапах життєвого циклу проєкту.

1.5 Опис вимог до системи

Основні вимоги до клієнт-серверної системи обслуговування включають ряд ключових аспектів, які визначають її ефективність, надійність та зручність в експлуатації. Ось деякі з найважливіших вимог:

a) Надійність є однією з ключових характеристик клієнт-серверних систем. Це включає в себе не лише стійкість до відмов, але й можливість відновлення після них. Система повинна мати механізми резервування та відновлення, щоб забезпечити найвищу доступність сервісів для користувачів.

b) Масштабованість - це здатність системи зростати разом з ростом обсягів даних та трафіку. Це може бути досягнуто за допомогою горизонтального та вертикального масштабування, а також застосуванням розподіленої архітектури та хмарних рішень.

c) Ефективність: Під ефективністю розуміється не лише швидкодія та продуктивність системи, але й оптимальне використання ресурсів. Це включає в себе оптимізацію запитів, зменшення затримок та мінімізацію споживання пам'яті та пропускної здатності мережі.

d) Безпека є критично важливою для будь-якої системи, особливо якщо вона операційна в мережі Інтернет. Це включає в себе захист від атак, шифрування даних, аутентифікацію та авторизацію користувачів, а також захист від витоку інформації.

e) Спрощення розробки: Зручність розробки системи дозволяє швидше вносити зміни, проводити пошук потенційних проблем та підтримувати систему у робочому стані. Використання простих та зрозумілих інтерфейсів, а також підтримка популярних фреймворків та бібліотек, може значно полегшити роботу розробників.

f) Моніторинг та аналіз: Наявність ефективних інструментів моніторингу та аналізу дозволяє відстежувати роботу системи в реальному часі, виявляти проблеми та шукати шляхи їх вирішення.

g) Документація та підтримка: Якісна документація та ефективна підтримка допомагають розробникам та користувачам краще розуміти систему, швидше розв'язувати проблеми та максимально використовувати її можливості.

h) Гнучкість та розширюваність: Система повинна бути гнучкою та легко розширюватися для внесення нових функцій, підтримки нових платформ та технологій, а також адаптації до змін у вимогах та ринкових умовах.

Ці вимоги визначають основні аспекти, які повинні бути враховані при проектуванні та розробці клієнт-серверних систем обслуговування. Керуючись ними, можна створити продукт, який буде ефективно виконувати свої завдання та задовольняти потреби користувачів.

1.6 Дослідження клієнт-серверної архітектури

При дослідженні клієнт-серверної архітектури варто розглянути використання мікросервісної архітектури як альтернативного підходу. Мікросервісна архітектура розбиває серверний компонент на невеликі незалежні мікросервіси, кожен з яких відповідає за виконання конкретної функціональності. Під час дослідження можна оцінити, як така архітектура впливає на ефективність, надійність та безпеку системи. Також важливо

розглянути аспекти, пов'язані з моніторингом, керуванням та розгортанням мікросервісів, а також їхньою взаємодією та залежностями.

Для підвищення продуктивності та швидкодії системи рекомендується використовувати кешування. Кешування дозволяє зберігати результати попередньо обчислених або отриманих запитів та повторно їх використовувати для подальших запитів з аналогічними параметрами. Це дозволяє зменшити час відповіді та навантаження на сервер, особливо для повторюваних запитів.

Додатково, для забезпечення ефективної взаємодії між мікросервісами, можна використовувати систему обміну повідомленнями або шину подій. Це дозволить різним сервісам спілкуватися між собою, передавати дані та сповіщати про події без прямого зв'язку. Такий підхід допомагає знизити залежність між сервісами та робить систему більш гнучкою та масштабованою.

У мікросервісній архітектурі кожен мікросервіс відповідає за виконання конкретної функціональності, і між ними може бути встановлено зв'язок за допомогою мережевих запитів. Ці мікросервіси можуть бути розгорнуті як на сервері, так і на клієнтському пристрої.

Наприклад, клієнтська частина системи може бути побудована з використанням окремих мікросервісів, кожен з яких відповідає за певну функціональність, таку як автентифікація, рендеринг інтерфейсу, взаємодія з сервером тощо. Кожен з цих мікросервісів може бути незалежно розроблений, тестований і масштабований.

Такий підхід дозволяє розробляти складні системи, розділяючи їх на більш маневрені та легкі для управління частини. Він також сприяє швидкому впровадженню змін, оновленню та розвитку системи. Однак потребує обґрунтованого підходу до архітектурного дизайну та вимагає ефективного керування залежностями між мікросервісами.

Урахування цих аспектів допоможе приймати обґрунтовані рішення щодо використання клієнт-серверної архітектури та мікросервісного підходу при створенні програмних систем. Такий підхід дозволить побудувати ефективну, масштабовану та надійну систему обслуговування, яка задовольнятиме потреби

користувачів та і при збільшенні навантаження буде залишатися стабільною і в той самий час ніщо не буде заважити масштабувати систему, але при умові, що всі залежності між сервісами налаштовані правильно.

1.7 Дослідження оптимізації процесу роботи з даними

Дослідження оптимізації процесу роботи з даними є ключовим аспектом для підвищення продуктивності, швидкості та ефективності програмних систем. Цей процес включає в себе аналіз та впровадження стратегій та технологій, які дозволяють зберігати, обробляти, передавати та використовувати дані з максимальною ефективністю та ефективністю.

Першим кроком у дослідженні оптимізації процесу роботи з даними є аналіз поточного стану системи зберігання та обробки даних. Це включає в себе огляд структури баз даних, типів даних, обсягів даних, частоти та обсягів запитів, які відправляються до баз даних, а також швидкості та продуктивності операцій з даними.

Після цього важливо виявити можливі точки оптимізації. Це може включати удосконалення структури баз даних, використання індексів для прискорення запитів, оптимізацію алгоритмів обробки даних, кешування часто використовуваних даних, паралелізацію обчислень та використання спеціалізованих технологій для роботи з великими обсягами даних, таких як Big Data та інші. Однією з найпопулярніших технологій для роботи з великими обсягами даних є Google BigQuery. Даний веб-сервіс має досить велику кількість клієнтів завдяки його швидкодії та процесів оптимізації роботи з даними, але аби його використовувати потрібно мати дійсно великі за обсягом дані.

Для успішної оптимізації процесу роботи з даними також необхідно враховувати вимоги до безпеки, доступності та надійності даних. Це може

включати резервне копіювання даних, відновлення після відмов, шифрування даних та контроль доступу.

Один з найважливіших етапів у ході проєктування системи відіграє саме розуміння яку базу даних потрібно використовувати. В контексті оптимізації процесу роботи з даними важливо розглянути переваги використання SQL баз даних. Ось кілька причин, чому вони можуть бути кращим вибором:

– Структурованість даних: SQL бази даних дозволяють організувати дані у вигляді таблиць з визначеними відносинами між ними. Це сприяє швидкому та ефективному пошуку, фільтрації та обробці даних.

– Мова запитів: SQL надає потужну мову запитів для взаємодії з базою даних. З її допомогою можна легко виконувати складні операції вибору, сортування, фільтрації та об'єднання даних.

– Індексація: SQL бази даних підтримують можливість створення індексів для прискорення пошуку та фільтрації даних. Це дозволяє оптимізувати продуктивність запитів, особливо для великих обсягів даних.

– Транзакційна підтримка: SQL бази даних надають механізми транзакцій для забезпечення консистентності даних. Це важливо для захисту даних від втрати або пошкодження в результаті непередбачуваних подій.

– Підтримка зовнішніх ключів та обмежень цілісності: SQL бази даних дозволяють встановлювати зв'язки між таблицями за допомогою зовнішніх ключів, а також встановлювати різноманітні обмеження цілісності даних. Це допомагає уникнути помилок та забезпечити консистентність даних [3].

Отже, використання SQL баз даних може сприяти покращенню продуктивності, надійності та ефективності процесу роботи з даними, особливо в умовах оптимізації систем. І обирати правильний тип баз даних потрібно з самого початку, бо змінювати його у ході розробки буде дуже накладно з усіх точок зору. Тому для того, аби зберегти час команди у майбутньому, потрібно виділити його у самому початку шляху та виділити дійсно важливі аспекти, які і

будуть вирішальними у виборі потрібної, а головне відповідної висунутим вимогам бази даних.

Загальним результатом дослідження оптимізації процесу роботи з даними має бути покращення продуктивності, зменшення часу відповіді на запити, зменшення витрат ресурсів на обробку даних та підвищення загальної ефективності програмної системи

1.8 Постановка задач до проєктування системи

Виходячи з результатів дослідження можна сформулювати постановку задач для проєктування клієнт-серверної системи обслуговування:

1. Вибір архітектури системи: Визначити тип клієнт-серверної системи, який найбільше відповідає потребам проєкту, враховуючи потужність, масштабованість та ефективність системи.

2. Встановлення вимог до функціональності: Описати всі функціональні вимоги до системи, зокрема, операції з обробки даних, взаємодію з користувачами та інші ключові можливості.

3. Визначення нефункціональних вимог: Встановити вимоги до продуктивності, безпеки, доступності та інших нефункціональних аспектів системи, які мають бути забезпечені.

4. Встановити у відповідності до обраної архітектури, які додаткові технології та сервіси можна застосувати або інтегрувати, аби покращити певні аспекти у роботі готової системи.

5. Порівняти між собою інформаційні технології та архітектури за обраними характеристиками і виробити рекомендації, щодо використання.

6. На основі вироблених рекомендацій спроєктувати клієнт-серверну систему обслуговування та провести порівняльний аналіз.

7. Використовуючи результати дослідження описати, які проблеми були виявлені, можливі шляхи вирішення цих проблем, також описати рекомендації з використання, способи покращення, або як можна використати дане дослідження як підґрунтя для наступних.

8. Описати інформаційні технології, що вдалося використати, які ні і, які були б корисні у даному дослідженні, та як би їх можна було використати.

Важливим критерієм повинна бути простота викладення, а головне доцільність проведених порівнянь, обрані критерії у ході порівняння повинні відповідати тематиці даного дослідження та чітко відображати суть порівняння. Дослідження повинні мати описані результати з приведенням найкращого або оптимального вибору відносно обраних показників. Доцільним є використання математичного підходу, тобто використання вже наявних формул, або розробка нових, які б краще підходили до проведення аналізу та отримання результатів дослідження.

2 ОПИС ПРИЙНЯТИХ ПРОЄКТНИХ РІШЕНЬ

2.1 Обґрунтування вибору мікросервісної архітектури

У порівнянні з іншими архітектурами, такими як монолітна архітектура та сервісно-орієнтована архітектура (SOA), мікросервісна архітектура має свої особливості та переваги:

a) Гнучкість та масштабованість: Мікросервісна архітектура дозволяє гнучко масштабувати окремі компоненти системи, враховуючи їхні поточні потреби та навантаження. У порівнянні з монолітною архітектурою, де всі компоненти розгортаються разом, мікросервіси дозволяють оптимізувати ресурси та швидко реагувати на зміни обсягу роботи.

b) Розділення відповідальності: У мікросервісній архітектурі кожен сервіс відповідає за конкретну функціональність або бізнес-логіку, що полегшує управління проектом та збільшує його масштабованість. У SOA, хоча також є розділення компонентів, вони частіше мають більший обсяг функціональності та залежать один від одного.

c) Технологічна різноманітність: У мікросервісній архітектурі кожен сервіс може бути розроблений з використанням різних технологій та мов програмування в межах одного проекту. У порівнянні з SOA, де існує більша єдність технологій та стандартів, мікросервіси дозволяють командам використовувати та впроваджувати нові технології більш гнучко.

d) Стійкість до збоїв: У мікросервісній архітектурі окремі сервіси можуть продовжувати працювати незалежно від збоїв в інших сервісах, що забезпечує більшу стійкість системи до випадкових помилок або відмов.

e) Легка масштабованість команди розробників: У мікросервісній архітектурі команди можуть працювати над окремими сервісами незалежно один від одного, що полегшує комунікацію, співпрацю та розвиток проєкту. У

порівнянні з SOA, де команди можуть бути більш залежними одна від одної через більший обсяг спільної функціональності [4].

Приклад архітектури SOA представлено на рисунку 2.1

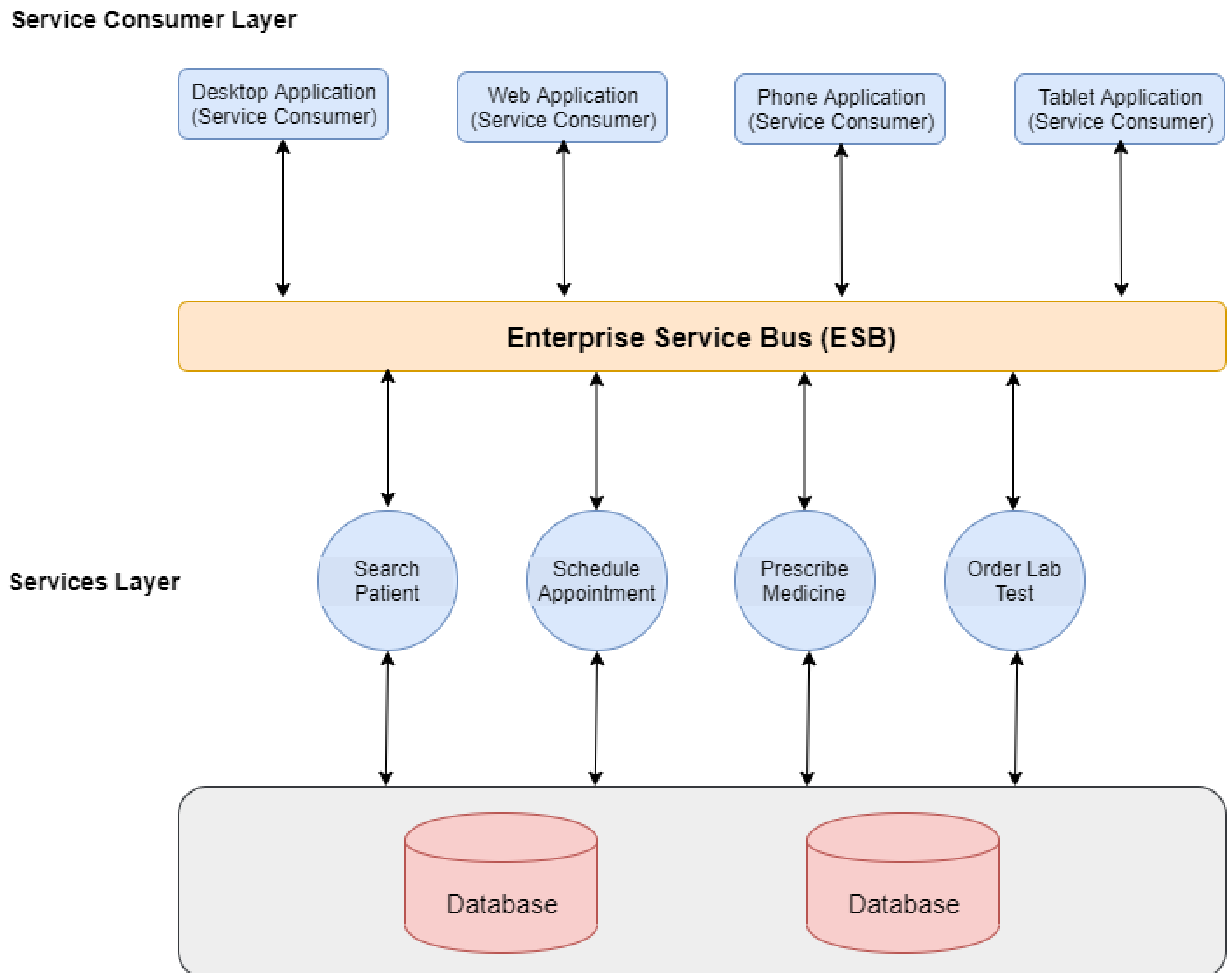

Рисунок 2.1 – SOA архітектура

Враховуючи ці фактори, мікросервісна архітектура може бути вигідним варіантом для проєктів, які потребують гнучкості, масштабованості та стійкості до збоїв, особливо в умовах наявності середовища з постійними змінами розробки та використання різноманітних технологій. Проте потрібно мати на увазі, що не завжди є змога дотримуватися правил мікросервісної архітектури, бо часто неможливо розділити один сервіс на декілька, або це просто недоцільно і тому в деяких випадках все ж таки маємо більше SOA архітектуру, а не мікросервісну. Але це потрібно розглядати з точки зору доцільності та вимог

конкретної задачі, а не намагатися слідувати одному напрямку. Зрештою, SOA архітектура також має велику кількість переваг, але вибір архітектури повинен бути у відповідності до поставлених задач і інколи можна комбінувати декілька архітектур разом і таких чином отримувати переваги від їх використання.

2.2 Рекомендації по використанню інформаційних технологій і архітектур

Беручи до уваги сучасні тренди в інформаційних технологіях та результати проведених досліджень, можна виробити наступні рекомендації щодо використання інформаційних технологій і архітектур:

– Java: Для реалізації серверної логіки рекомендується використовувати мову програмування Java. Java відома своєю надійністю, широким спектром бібліотек та великою кількістю фахівців у цій галузі.

– Spring Framework: Для побудови серверної архітектури рекомендується використовувати Spring Framework. Spring забезпечує швидку розробку, інверсію керування та керування залежностями, що дозволяє покращити продуктивність та підтримку коду.

– Guava Cache: Для оптимізації роботи з даними та зменшення навантаження на базу даних рекомендується використовувати Guava Cache. Це дозволить зберігати та отримувати дані швидше, що покращить продуктивність додатку.

– Apache Kafka: Для забезпечення асинхронного обміну повідомленнями між мікросервісами та іншими системами рекомендується використовувати Apache Kafka. Він забезпечує надійну та масштабовану систему обміну повідомленнями, що допоможе покращити стійкість та ефективність системи.

– SQL база даних: Для зберігання структурованих даних рекомендується використовувати SQL базу даних, наприклад, PostgreSQL або MySQL. SQL бази

даних забезпечують надійність, консистентність та можливість виконання складних запитів.

– Мікросервісна архітектура: Рекомендується використовувати мікросервісну архітектуру для побудови системи. Це дозволить розділити функціональність на невеликі та незалежні компоненти, що сприятиме гнучкості, масштабованості та простоті управління системою.

Одним з ефективних інструментів є Guava Cache. І рекомендовано саме його, бо це інструмент для кешування даних в пам'яті, який надає ряд переваг:

a) Ефективне кешування: Guava Cache забезпечує ефективне збереження та доступ до даних в оперативній пам'яті. Він працює на рівні програми, що дозволяє швидко отримувати доступ до даних без звертання до диску.

b) Автоматичне видалення старих даних: Guava Cache автоматично видаляє старі дані з кешу згідно з налаштованими правилами використання пам'яті або часовими обмеженнями. Це дозволяє уникнути переповнення пам'яті та забезпечити актуальність даних.

c) Підтримка асинхронного завантаження: Guava Cache підтримує асинхронне завантаження даних у кеш, що дозволяє уникнути блокування потоків в той час, коли дані завантажуються з диску або мережі.

d) Можливість налаштування параметрів кешування: Guava Cache надає різноманітні параметри налаштування, такі як розмір кешу, термін життя елементів, стратегії видалення та інші. Це дозволяє гнучко налаштовувати кеш під конкретні потреби додатку.

e) Простота використання: Guava Cache простий у використанні і інтеграції в вашу програму. Він надає зрозумілий API для додавання, отримання та видалення елементів з кешу.

Загалом, Guava Cache є потужним інструментом для оптимізації роботи з даними в пам'яті, що дозволяє підвищити продуктивність та ефективність.

Для обміну повідомленнями між мікросервісами використовується Apache Kafka яка є платформою потокової передачі даних тобто event streaming platform.

Потокове передавання подій - це цифровий еквівалент центральної нервової системи людського організму. З технічної точки зору, потокова передача подій - це практика збору даних у реальному часі з джерел подій, таких як бази даних, датчики, мобільні пристрої, хмарні сервіси та програмні додатки, у вигляді потоків подій; довготривале зберігання цих потоків подій для подальшого пошуку; маніпулювання, обробка та реагування на потоки подій у реальному часі, а також ретроспективно; і маршрутизація потоків подій до різних технологій призначення за потребою. Таким чином, потокова передача подій забезпечує безперервний потік та інтерпретацію даних, щоб потрібна інформація була в потрібному місці і в потрібний час [5].

Застосовувати потокові трансляції подій можна в найрізноманітніших сферах, в безлічі галузей і організацій. Серед численних прикладів можна назвати такі:

– Обробка платежів і фінансових транзакцій в режимі реального часу, наприклад, на фондових біржах, в банках і страхових компаніях.

– Відстеження та моніторинг легкових і вантажних автомобілів, автопарків і вантажів у режимі реального часу, наприклад, у логістиці та автомобільній промисловості.

– Безперервний збір та аналіз сенсорних даних з пристроїв Інтернету речей або іншого обладнання, наприклад, на заводах і вітряних електростанціях.

– Збирати та миттєво реагувати на взаємодію та замовлення клієнтів, наприклад, у роздрібній торгівлі, готельному та туристичному бізнесі, а також у мобільних додатках.

– Моніторинг пацієнтів, які перебувають на стаціонарному лікуванні, та прогнозування змін у їхньому стані для забезпечення своєчасного лікування в екстрених випадках.

– Об'єднувати, зберігати та надавати дані, отримані різними підрозділами компанії.

– Слугувати основою для платформ даних, архітектур, керованих подіями, та мікросервісів [6].

Також потрібно вказати, чому саме було обрано Apache Kafka. Kafka поєднує в собі три ключові можливості, щоб реалізувати свої сценарії використання для потокової передачі подій за допомогою одного перевіреного рішення:

1. Публікувати (писати) та підписуватися (читати) на потоки подій, включаючи безперервний імпорт/експорт даних з інших систем.

2. Зберігати потоки подій надійно та довговічно так довго, як потрібно.

3. Обробляти потоки подій по мірі їх виникнення або ретроспективно.

І вся ця функціональність забезпечується розподіленим, масштабованим, еластичним, відмовостійким і безпечним способом. Kafka може бути розгорнута на «голому» обладнанні, віртуальних машинах і контейнерах, локально або в хмарі. Також самостійно керувати середовищем Kafka або скористатися керованими послугами, що пропонуються різними постачальниками.

Це потужна та масштабована платформа для обміну повідомленнями, яка має ряд переваг:

1. Висока масштабованість: Kafka розроблена з урахуванням високої масштабованості. Вона може легко обробляти великі обсяги даних та мільйони повідомлень на секунду, що робить її ідеальним вибором для великих систем.

2. Надійність: Kafka забезпечує надійну доставку повідомлень. Вона зберігає дані на диску і може відновлювати їх в разі втрати зв'язку або аварії вузла.

3. Низька затримка: Kafka може обробляти повідомлення з низькою затримкою, що робить її ідеальним вибором для вимогливих до часу додатків, таких як фінансові системи або системи реального часу.

4. Гарантована доставка повідомлень: Kafka забезпечує гарантовану доставку повідомлень, що дозволяє впевнено обмінюватися даними між різними частинами системи.

5. Гнучкість і розширюваність: Kafka має гнучку архітектуру, яка дозволяє легко розширювати та змінювати її відповідно до потреб вашої системи. Вона також ідеально підходить для використання в мікросервісних архітектурах.

6. Широкий функціонал: У Kafka є багато корисних функцій, таких як розділення повідомлень на теми, можливість реплікації даних для забезпечення надійності та багато іншого (рисунок 2.2) [7].

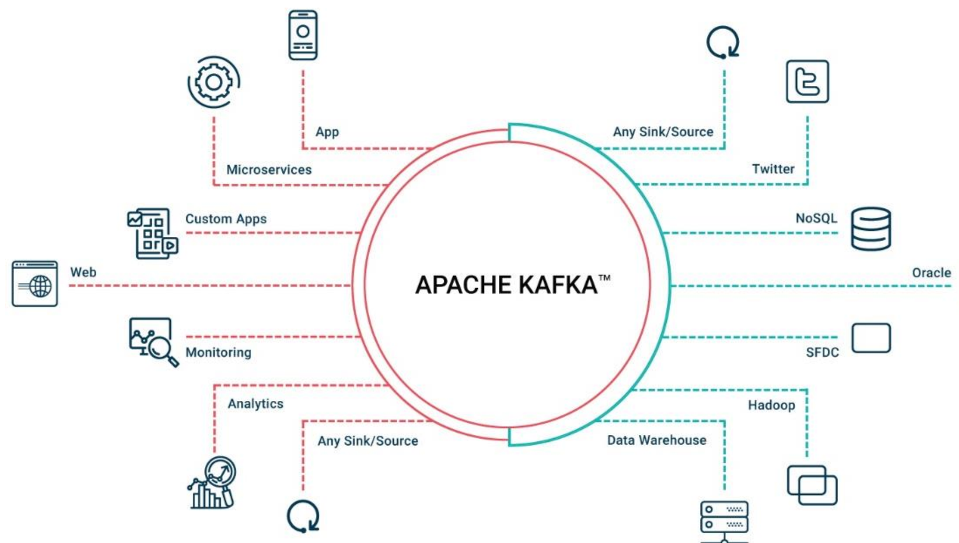

Рисунок 2.2 – Структура Kafka

У цілому, Apache Kafka є потужним інструментом для побудови надійних, масштабованих та ефективних систем обміну повідомленнями, який може задовольнити потреби різноманітних додатків і бізнес-вимог, а використання цих технологій та архітектур допоможе створити ефективну та надійну систему.

Сервіси та їх залежності можуть вийти з ладу з багатьох причин. Збій може зробити сервіс недоступним і призвести до відмови у виконанні запитів. Локальний збій має обмежений вплив, залежно від бізнес-функції сервісу. Однак, як ми вже обговорювали, коли цей збій викликає каскадний збій інших сервісів в архітектурі, вплив зростає в геометричній прогресії і може призвести до зупинки всієї системи. У типових мікросервісних архітектурах зв'язок між сервісами відбувається синхронно, наприклад, за допомогою HTTP-запитів. Така

архітектура схильна до розвитку складної мережі синхронних запитів. Ви можете пригадати приклад, який ми згадували, описуючи розподілене трасування в Главі 4, як Uber трасує 1000 мікросервісів і свою складну мережу. У мережі є великі вузли, від яких залежать кілька сервісів; якщо один з них стає недоступним, це, швидше за все, вплине на всю архітектуру. Але може бути і навпаки: менш критичні та більш ізольовані сервіси можуть каскадувати проблеми на інші, більш важливі сервіси і поглиблювати проблему [8].

Беручи до уваги той факт, що завжди будуть наявні якісь проблеми у системі, такі як мережеві збої, нехватка підключень до бази даних із-за того, що навантаження зросло різко і потрібно оброблювати велику кількість запитів, що призводить до блокування або запитів до бази даних, також система може просто перестати працювати через надто високу інтенсивність тих самих запитів. Зазвичай, у будь-якій системі бувають години піку та спаду і важливо пережити та витримати стрімко зростаючу кількість одночасних запитів до системи. Це цього можна досягти різними методами, але спочатку треба зрозуміти, а що саме призводить системи до її блокування.

Наразі детально розглянемо, як каскадний збій може вплинути на робочий процес електронної комерції подібно до тих, які ми обговорювали до цього часу в архітектурі мікросервісів (рисунок 2.3).

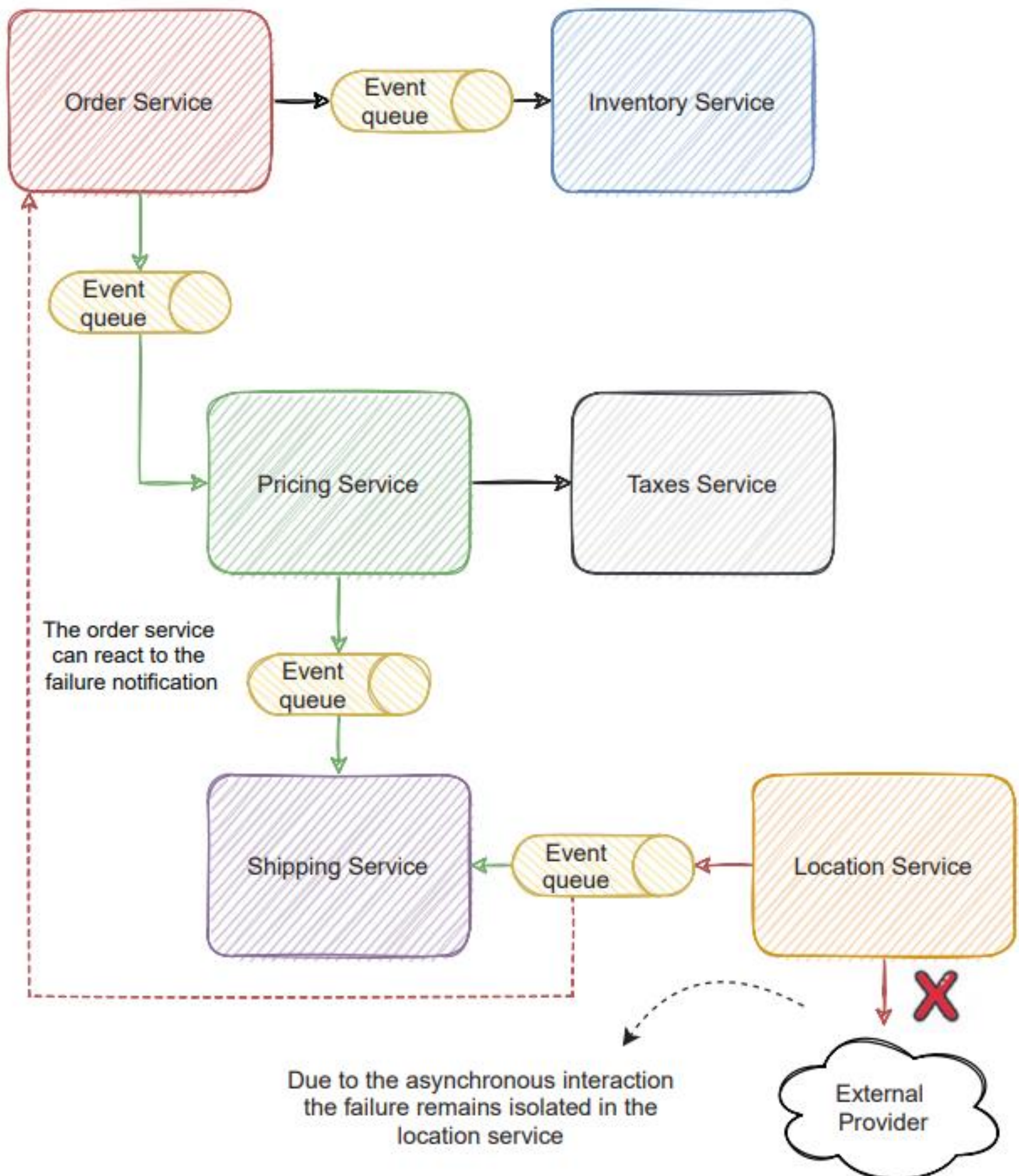

Рисунок 2.3 – Збій у зовнішньому провайдері каскадує на інші вихідні сервіси

Потік виконання замовлень запитує у служби ціноутворення розрахунок комісій за кожного отриманого замовлення. Служба ціноутворення залежить від служби доставки, яка обробляє інформацію про доставку та адресу, яка залежить від служби визначення місцезнаходження, який управляє більш загальною інформацією про географічне розташування та країну. Служба визначення

місцезнаходження також залежить від зовнішнього постачальника для отримання інформації про країну. У наведеному прикладі службі визначення місцезнаходження не вдалося отримати дані від зовнішнього постачальника. Внаслідок не зможе отримати інформацію, вона також не зможе успішно відповісти на запит служби доставки. Збій каскадом передається на інші попередні сервіси, що в підсумку призведе до збою виконання замовлення. У цьому випадку відносно ізольована проблема, не дуже важлива з точки зору бізнесу, в підсумку впливає на основний потік виконання замовлень [9].

У цьому випадку зовнішній провайдер не зміг впоратися з проблемою. Однак така ситуація може виникнути з будь-якою з інших служб, які ми розглядали раніше. Наприклад, база даних також може вийти з ладу, і хоча це трапляється відносно рідко, пікове навантаження може створити проблеми через обмежені фізичні ресурси. У таких випадках часто виникає певне явище. Звичним підходом до вирішення проблеми невдалих запитів є їх повторне виконання. Коли служба, яка стикається з надмірним навантаженням, починає відхиляти деякі запити, служби, що звертаються до неї, також збільшують свою пропускну здатність завдяки стратегії повторних спроб. У конкретній ситуації служба стала недоступною через проблеми з підключенням. Як тільки команда вирішила ці проблеми і служба почала відновлюватися, додаткова пропускна здатність повторних спроб кожної залежної служби повністю зайняла ресурси служби і знову призвела до її аварійного завершення через надмірне навантаження. Ця проблема підкреслила необхідність застосування відповідних стратегій, таких як повторні спроби зробити ще один запит, але з деякою затримкою в часі з відступом та вимикачі.

Частину цих проблем вирішує Eureka Service Discovery та правильно налаштований API Gateway.

Eureka Service Discovery – це патерн, за допомогою якого клієнтські програми можуть динамічно виявляти місцезнаходження сервісів під час виконання. Основна ідея полягає в тому, щоб мати центральний реєстр усіх

сервісів та їх поточного стану. До цього реєстру клієнти можуть звертатися, щоб визначити місцезнаходження конкретного сервісу [10].

API Gateway – це зворотній проксі, який знаходиться перед вашими мікросервісами і діє як єдина точка входу для вхідних запитів API. Він відповідає за маршрутизацію запитів, склад і переклад протоколів, серед іншого.

Зв'язок між Service Discovery і API-шлюзом полягає в тому, що API-шлюз використовує інформацію з реєстру Service Discovery для маршрутизації запитів до відповідного мікросервісу. Шлюз API запитує реєстр, щоб визначити місцезнаходження цільового мікросервісу, а потім перенаправляє вхідний запит до цього сервісу. Таким чином, API-шлюз діє як міст між клієнтськими додатками та мікросервісами, забезпечуючи єдину точку входу для вхідних запитів та абстрагуючись від складності мікросервісів [11].

Схема архітектури системи з використанням описаних інформаційних технологій, а саме Eureka Service Discovery та API Gateway на рисунку 2.4.

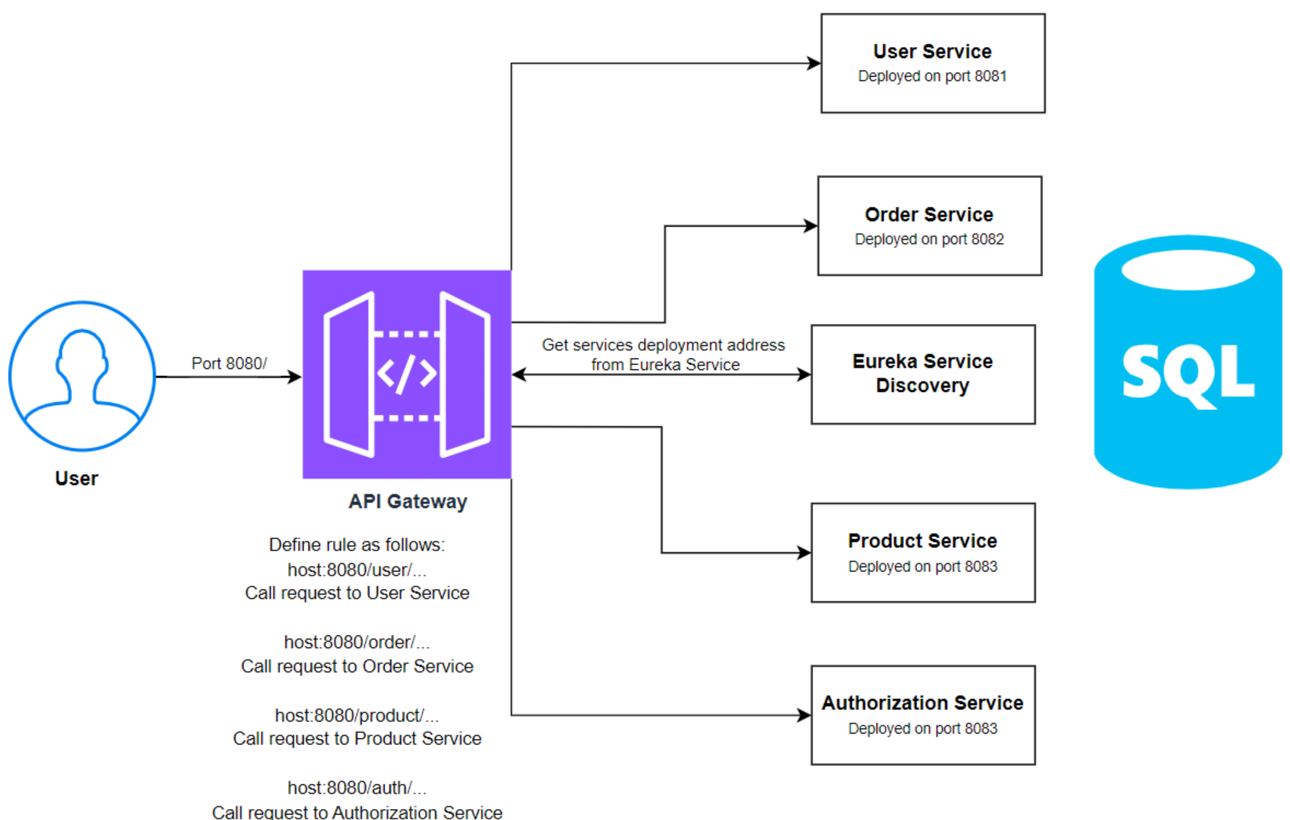

Рисунок 2.4 – Схема архітектури системи

Рекомендації до серверної частини наразі описані та обґрунтовані. Тепер треба описати, що використовувати для реалізації клієнтської частини. Наразі існують досить багато різних інформаційних технологій і які також будуть добре сумісні з рекомендаціями до серверної частини. Проте одним з найпопулярніших та досить розвинутим фреймворком є React.

React – це відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту вебсторінки, з якими стикаються в розробці односторінкових застосунків. Розробляється Meta (раніше Facebook) і спільнотою індивідуальних розробників.

React дозволяє розробникам створювати великі вебзастосунки, які використовують дані, котрі змінюються з часом, без перезавантаження сторінки. Його мета полягає в тому, щоб бути швидким, простим, масштабованим. React обробляє тільки користувацький інтерфейс у застосунках. Це відповідає видові у шаблоні модель-вид-контролер (MVC), і може бути використане у поєднанні з іншими JavaScript бібліотеками або в великих фреймворках MVC, таких як AngularJS. Він також може бути використаний з React на основі надбудов, щоб піклуватися про частини без користувацького інтерфейсу побудови вебзастосунків. Як бібліотеку інтерфейсу користувача React найчастіше використовують разом з іншими бібліотеками, такими як Redux. [12].

Основні принципи або особливості React:

1. Компонентний підхід – дозволяє використовувати повторно вже створені компоненти, що дозволяє зменшити кількість написаного коду та в результаті спрощує розробку та підтримку коду.

2. Віртуальний DOM – React для підвищення продуктивності та мінімізації реальних змін у DOM-дереві браузера використовує віртуальний DOM.

3. JSX – розширення для написання HTML-подібного коду, але всередині JavaScript для кращої взаємодії між компонентами.

4. Потік даних односпрямований – це підхід, у якому дані передаються тільки зверху вниз. Доцільність такого підходу у тому, що складність взаємодії між компонентами значно зменшується [13].

На рисунку 2.5 відображені основні аспекти React.

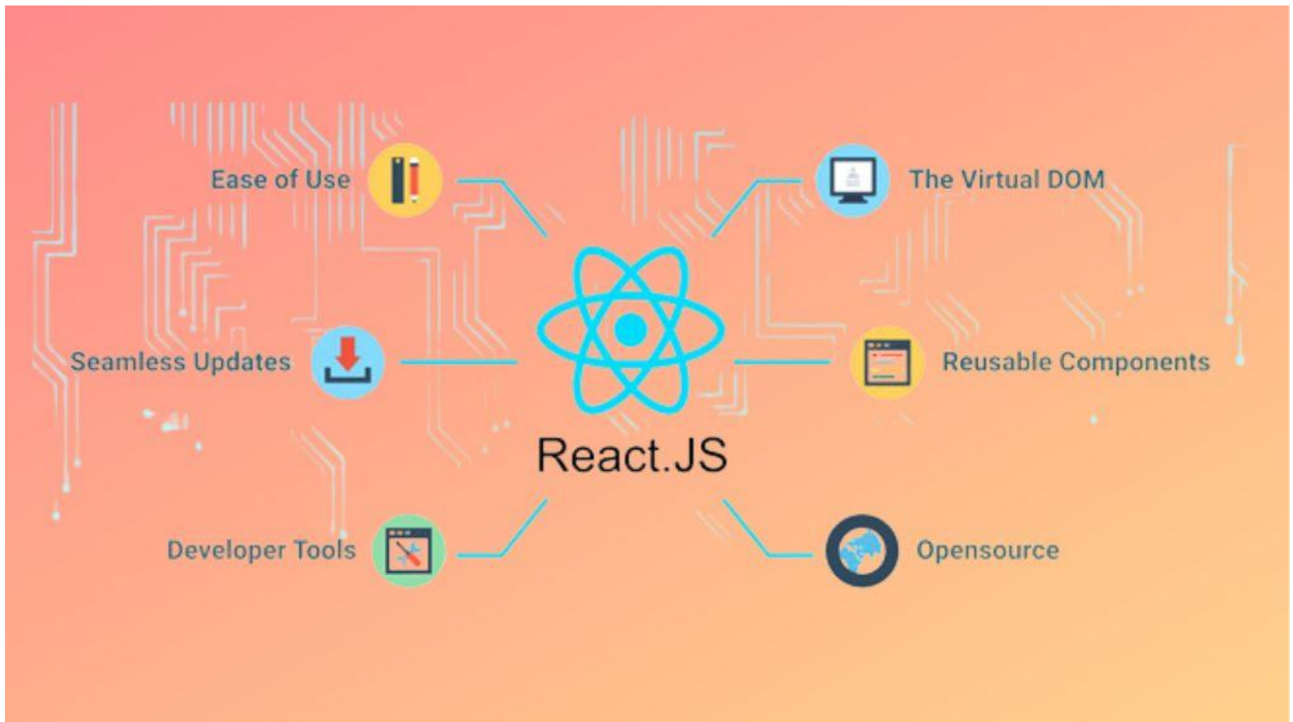

Рисунок 2.5 – Особливості React

Також React має велику кількість сторонніх бібліотек, які з легкістю можна підключити у якості залежностей до проекту та використовувати для намічених цілей. У ході розробки потрібно було керувати станом додатку і однією з найпопулярніших бібліотек є Redux.

Redux – це бібліотека для управління станом додатку, яка часто використовується разом з React. Вона забезпечує централізоване зберігання стану додатку та визначає єдиний спосіб оновлення цього стану [14].

Основні принципи Redux:

1. Єдине джерело правди: Весь стан додатку зберігається у єдиному об'єкті, який називається "store".

2. Стан лише для читання: Єдиний спосіб змінити стан — це створити дію (action), яка описує, що має змінитися.

3. Чисті функції (reducer): Reducer — це чиста функція, яка приймає попередній стан та дію і повертає новий стан. Reducers не повинні змінювати стан безпосередньо, а лише створювати новий об'єкт стану [15].

Проте також необхідним було і застосування Redux Thunk.

Redux Thunk – являє собою проміжне програмне забезпечення (middleware) для Redux, яке дозволяє писати асинхронні логіки у ваших діях (actions). Без Thunk, дії в Redux повинні бути чистими об'єктами. Thunk розширює можливості Redux, дозволяючи дії бути функціями, які можуть виконувати асинхронний код [16].

Переваги Redux Thunk:

– Асинхронні дії: Thunk дозволяє вам виконувати асинхронні операції, такі як запити до API, перед тим як диспетчеризувати справжні дії, що змінюють стан.

– Більше контролю: Використовуючи Thunk, ви можете диспетчеризувати кілька дій, робити умови та інші логічні операції перед оновленням стану.

– Простота інтеграції: Thunk легко інтегрується в існуючий проект Redux, не вимагаючи значних змін у коді [17].

Достатньо важливим етапом у розробці будь-якої системи є забезпечення захисту від різного роду атак на систему для збереження даних користувачів та цілісності самої системи. Технології, використані під час розробки, вже забезпечують доволі високий рівень безпеки. Наприклад, Spring Security – це фреймворк, який спеціалізується на аутентифікації та авторизації Java-додатків. MySQL також має вбудовану систему безпеки, яка налаштовується автоматично при встановленні бази даних, і часто цього вистачає для забезпечення початкової безпеки даних.

Однак було вирішено підвищити безпеку системи, застосовуючи JSON Web Tokens (JWT). Це один із найпопулярніших методів аутентифікації для Java веб-додатків. Використання JWT має кілька переваг:

- Зручність: немає потреби передавати логін і пароль при кожному запиті.

- Зменшення кількості запитів до бази даних, оскільки токен містить основну інформацію про користувача.

- Легка програмна реалізація: вже існують бібліотеки для генерації та декодування токенів.

JWT представляє собою рядок, який створюється під час реєстрації користувача в системі і використовується для авторизованих запитів на сервер. Згенерований токен зберігається у сесії користувача, причому кожен користувач має свій унікальний токен з обмеженим терміном дії. Коли термін дії токену закінчується, користувач отримує повідомлення і його перенаправляють на сторінку авторизації. При створенні токену в нього включається інформація про користувача, яка буде зчитуватись і перевірятись при кожному запиті.

Загальна схема роботи JSON Web Tokens починаючи з запиту і до обробки всередині системи, показана на рисунку 2.6.

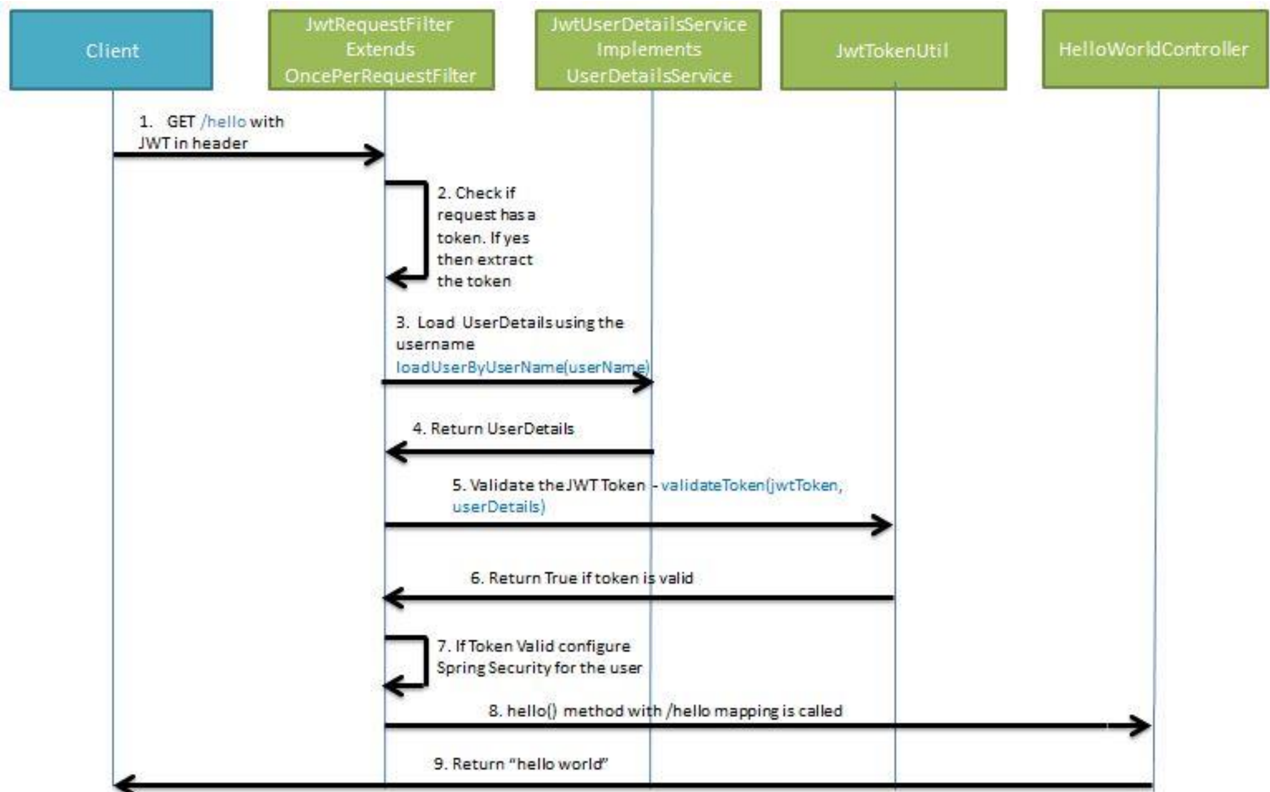

Рисунок 2.6 – Загальна схема роботи JSON WEB Tokens

JSON Web Tokens (JWT) – це спосіб передачі даних між двома або більше сторонами у форматі JSON. Зазвичай структура JWT складається з трьох основних частин:

– header – це заголовок токену;
– payload – це корисне навантаження;
– signature – це шифрований підпис токену.

Заголовок є службовою частиною токену, яка допомагає додатку визначити, як обробляти отриманий токен. Він може містити інформацію про тип токену та алгоритм, використаний для створення підпису.

Корисне навантаження може включати будь-яку інформацію, яка допомагає додатку ідентифікувати користувача, а також будь-які додаткові дані.

Підпис створюється з заголовку та корисного навантаження за допомогою алгоритму base64, після чого додається до токену через крапку.

Усі потрібні інформаційні технології для частин клієнт-серверної системи обслуговування описані. Перелік є досить великим та сама розробка вимагає багато часу. Також хочеться наголосити ще на тому, що увесь стек технологій базується виключно на мові програмування, здебільшого мається на увазі опис технологій для бекенду.

Мова програмування Java була обрана не випадково, приведено її переваги над іншими мовами, але найголовнішою проблемою є час розробки, а з цього випливає коштовність кінцевого продукту, але і в той самий час Java використовується повсемісно, де потрібно забезпечити можливість роботи з реально великою кількістю даних, серверів, запитів та інше. І виходячи з даних критеріїв краще мати більш дорогий у розробці продукт, але його не потрібно буде переписувати з нуля, якщо він матиме успіх та потрібно буде його розширювати з точки зору функціональності та кількості серверів.

2.3 Обґрунтування вибору стеку технологій для розробки

Рекомендації по використанню інформаційних технологій і архітектур вироблено, а тому потрібно обґрунтувати чому саме вони були обрані спираючись вже на сучасні тенденції та опитування серед фахівців в інформаційних технологіях.

Спираючись на мабуть найвідоміший портал з пошуку роботи та розміщення своїх оголошень linkedin можна з впевненістю казати, що мова програмування java на даний момент займає достатньо високу сходинку у рейтингу мов програмування та входить у топ 10 (рисунок 2.7).

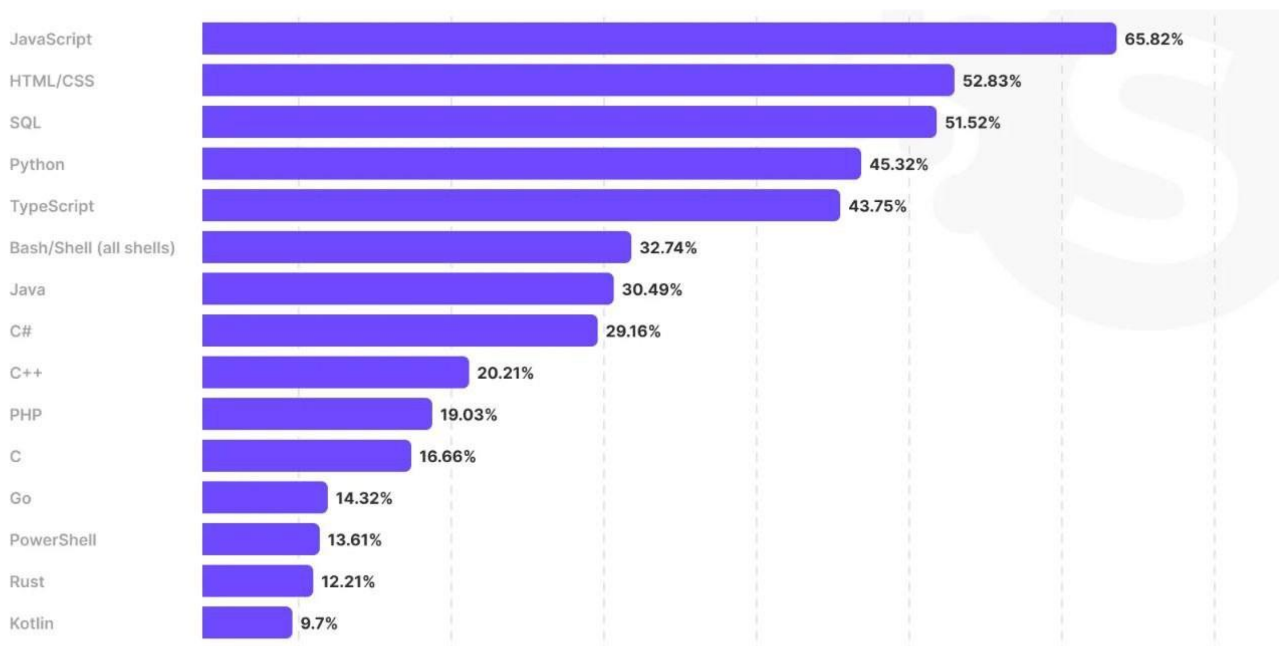

Рисунок 2.7 – Рейтинг мов програмування на 2024 від порталу linkedin

В рекомендаціях зазначалося, що фреймворком для розробки клієнт-серверної системи обслуговування виступає Spring та було приведено його характеристику, але потрібно вказати про що каже опитування про сучасні фреймворки серед розробників. І спираючись на статистику «Найбільш використовувані веб-фреймворки серед розробників у всьому світі - 2022 рік» з

Statistics & Data – це проект з досліджень, аналізу та візуалізації даних, який працює з 2019 року, який і провів дане дослідження. Результат на рисунку 2.8.

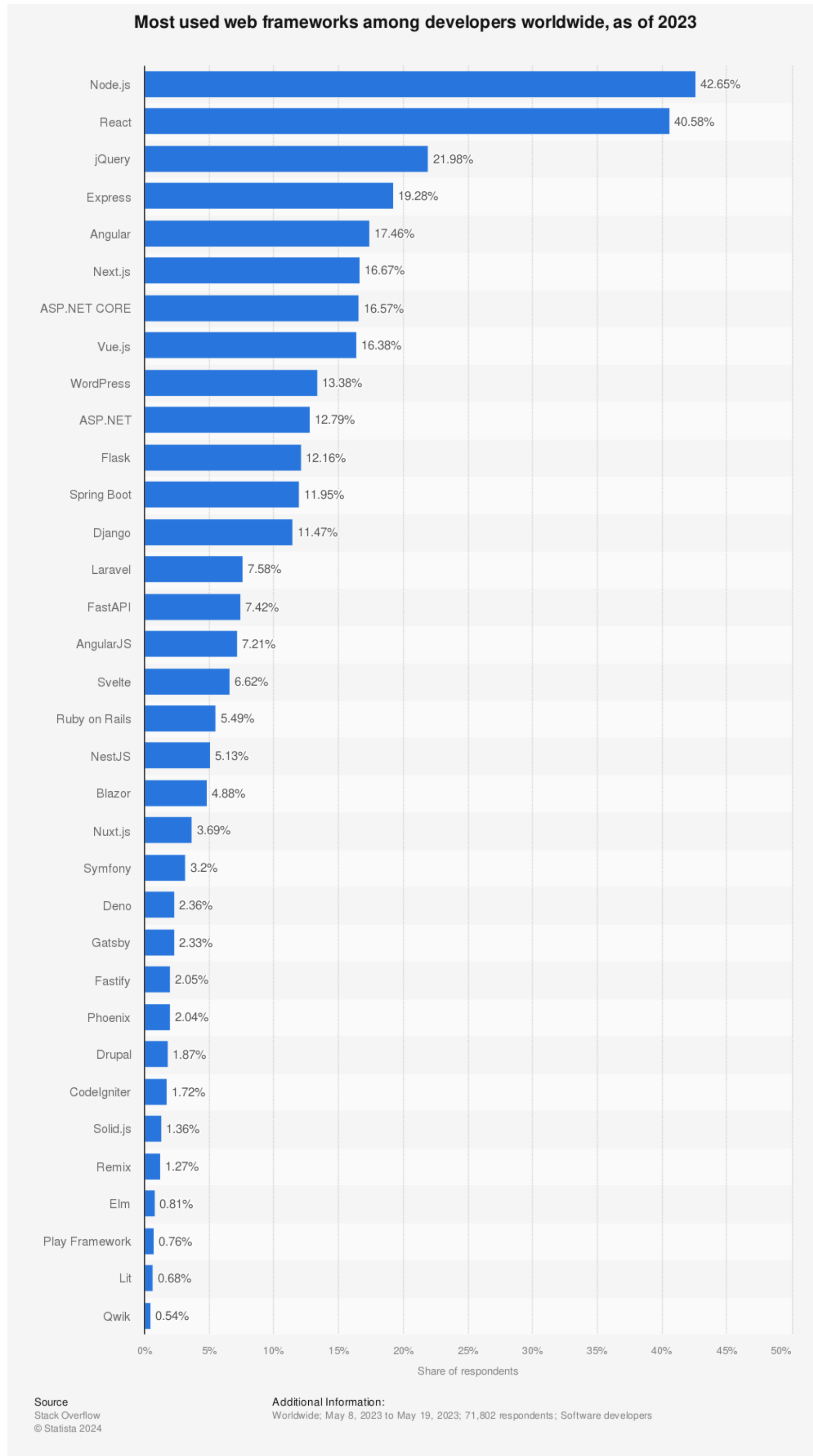

Рисунок 2.8 – Рейтинг найпопулярніших фреймворків на 2023 рік

Дане дослідження було замовлене StackOverflow і в якому приймало участь більше ніж 58 тисяч розробників з усього світу. Вони мали відповісти на питання про найбільш використовуваний ними фреймворк. Spring має 11.95 відсотків з усіх існуючих фреймворків у світі, але при цьому займає перше місце по використанню для мови програмування java. І як вже описано в минулих розділах усі подальші інформаційні технології вже напряму пов’язані з мовою програмування, а тому їх актуальність вже доведена.

2.4 Опис ходу розробки системи за виробленими рекомендаціями

Маючи вироблені рекомендації наступним етапом є проєктування конкретної клієнт-серверної системи обслуговування, аби на практиці перевірити доцільність цих рекомендацій та життєздатність розробленої системи при використанні її у реальних або наближених до реальних умов.

Для забезпечення виконання даної задачі:

– повинна бути спроєктовано клієнт-серверну систему з підбору комп’ютерних ігор відповідно до переваг користувача;

– визначено функціональні вимоги до системи;

– описано логічне та фізичне моделювання бази даних;

– описано хід виконання розробки системи;

– проведено аналіз даної клієнт-серверної системи обслуговування шляхом виконання навантажувального тестування та перевірки працездатності системи в цілому при оптимальних та критичних умовах використання.

В результаті повинно бути зрозуміло чи є вироблені рекомендації доцільними та чи можуть бути використані на практиці для виконання конкретних бізнес задач.

3 ПРОЄКТУВАННЯ АРХІТЕКТУРИ КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ

3.1 Опис схеми бази даних

Для реалізації серверної частини інформаційної системи було обрано платформу MySQL, виходячи з вироблених рекомендацій, для якої й розроблено базу даних.

Під час аналізу предметної області були виділені такі сутності: «main_products», «main_categories», «main_studios», «users», «main_oss», «main_product_category», «checkout_order», «checkout_orderhasproduct», «image», «user_roles» і «roles». Ці сутності було створено разом із проміжними таблицями для зв’язків "багато до багатьох" (рисунок 3.1).

Схема бази даних, як показано на рисунку, включає одинадцять сутностей, причому більшість з них не допускають нульових значень у своїх атрибутах.

Однак існують винятки, наприклад, у таблиці «checkout_order» атрибут «comment», що означає, що клієнт може залишити коментар за бажанням, наприклад, вказавши прохання про зворотний дзвінок. Ще один приклад — атрибут «reset_password_token», який зберігає токен для відновлення пароля. Після того, як користувач отримує посилання з токеном на електронну пошту і переходить за ним, система зпівставляє токен з наявним, дозволяючи змінити наявний пароль на новий.

Усі сутності містять первинні ключі, власні атрибути з визначеними типами даних та обмеженнями, а також зовнішні ключі для забезпечення зв’язків між дочірніми та батьківськими сутностями.

Для роботи з MySQL вирішено було використовувати інтегровану середу розробки MySQL Workbench. Це доволі потужний інструмент для створення, виконання та оптимізації SQL запитів і який також дозволяє візуально відображати схему бази даних та працювати з нею.

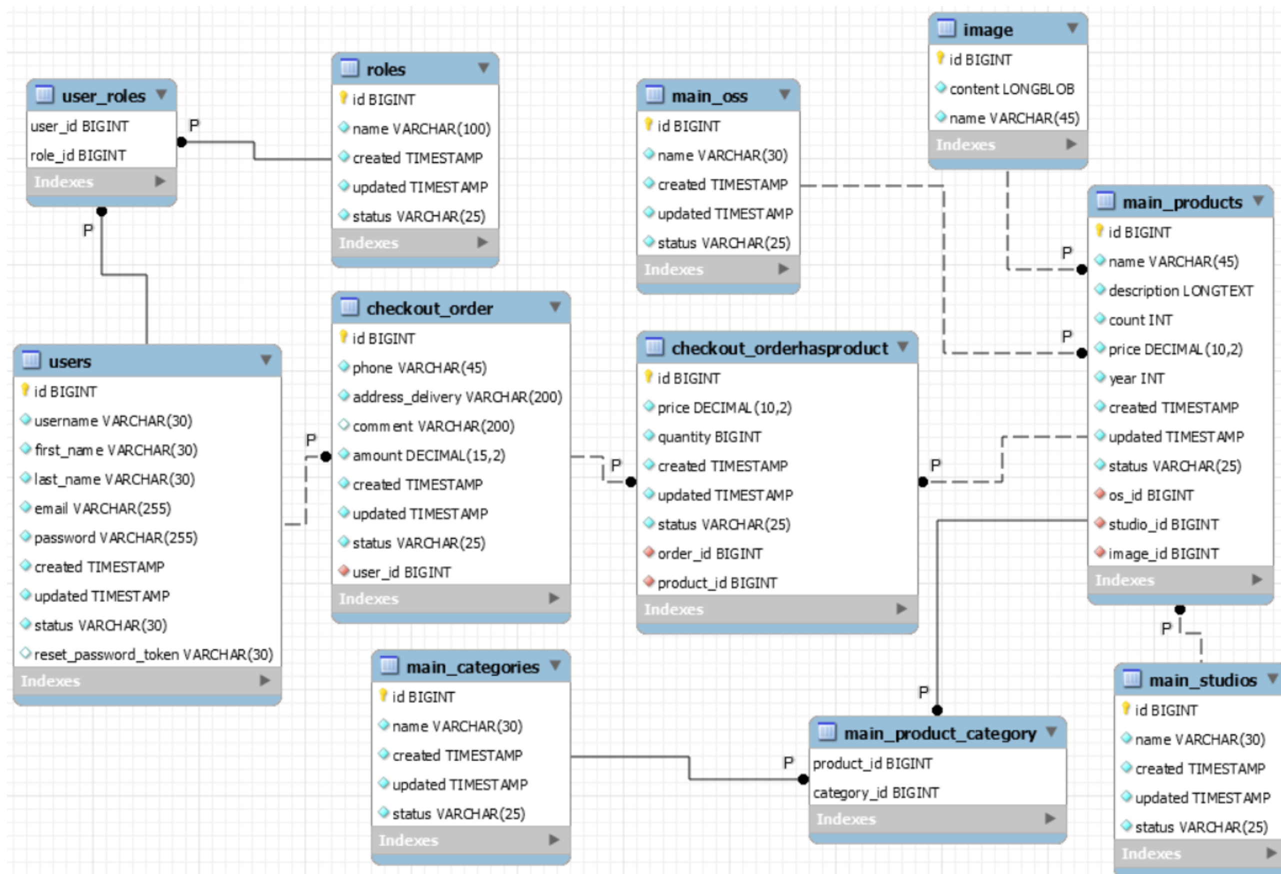

Рисунок 3.1 – Схема бази даних

Ця схема бази даних не включає повторюваних даних, а кожна сутність має лише ті атрибути, які її характеризують, без атрибутів інших сутностей. Таким чином, можна зробити висновок, що база даних пройшла нормалізацію.

3.2 Опис сервісів за предметною областю

Маючи мікросервісну архітектуру, потрібно виділити сервіси, які потрібні для проєктування клієнт-серверної системи за обраною предметною областю та привести їх детальний опис.

У таблиці 3.1 представлено сервіс «UserService» клієнт-серверної системи та опис його методів із зазначенням їх призначення.

Таблиця 3.1 – Сервіс «UserService» клієнт-серверної системи і методи цього сервісу з описом їх призначення

№	Назва сервісу	Найменування методу сервісу	Призначення методу
1	UserService	User save(User user);	Додати користувача до бази даних
2		User register(User user);	Зареєструвати користувача
3		User registerByAdmin(User user);	Зареєструвати користувача адміністратором
4		User update(User user);	Оновити дані користувача
5		List<User> getAll();	Отримати усіх користувачів
6		User findByUsername(String name);	Отримати користувача за ім’ям
7		User findById(Long id);	Отримати користувача за ідентифікаційним номером
8		List<User> findAllByStatus(Status status);	Отримати користувачів за статусом
9		List<User> findAllByRoles(Role role);	Отримати користувачів за роллю
10		void delete(Long id);	Видалити користувача
11		void updateResetPasswordToken(String token, String email);	Оновити пароль користувача за його токеном та поштою
12		User getResetPasswordToken(String token);	Отримати користувача за його токеном
13		void updatePassword(User user, String newPassword);	Оновити пароль користувача
14		void authUser(User user)	Авторизувати користувача

У таблиці 3.2 представлено сервіс «ProductService» клієнт-серверної системи та опис його методів із зазначенням їх призначення.

Таблиця 3.2 – Сервіс «ProductService» клієнт-серверної системи і методи цього сервісу з описом їх призначення

№	Назва сервісу	Найменування методу сервісу	Призначення методу
1	ProductService	Product save(Product product);	Додати гру до бази даних
2		List<ProductDtoOut> getAll();	Отримати усі ігри з бази даних
3		Product findByName(String name);	Отримати гру за її назвою
4		List<ProductDtoOut> findAllOrderByIdDesc();	Отримати усі ігри від максимального id до мінімал.
5		List<ProductDtoOut> findAllOrderByCount();	Отримати усі ігри за кількість від мін. до макс.
6		List<ProductDtoOut> findAllOrderByCountDesc();	Отримати усі ігри за спадаючою кількість
7		Product findById(Long id);	Отримати гру за ідентифікаційним номером
8		List<ProductDtoOut> findAllOrderByPrice (); List<ProductDtoOut> findAllOrderByPriceDesc();	Отримати усі ігри за ціною від мін. до макс. та навпаки
9		List<ProductDtoOut> findAllOrderByYear(); List<ProductDtoOut> findAllOrderByYearDesc();	Отримати усі ігри за роком від мін. до макс. та навпаки

№	Назва сервісу	Найменування методу сервісу	Призначення методу
10		void delete(Long id);	Видалити гру за ідентифікаційним номером
11		List<ProductDtoOut> findAllByStatus(Status s);	Отримати усі ігри за статусом
12		List<ProductDtoOut> findAllByCategoriesOsInAndStudioInAndPriceBetweenAndYearBetween(ProductByFilters productByFilters);	Отримати ігри відповідно до категорії, операційної системи, студії, ціни від мін. до макс. та року випуску від мін. до макс.
13		void updateProductCount(Long productId, Long count);	Оновити кількість гри за id та кількістю придбаної гри

У таблиці 3.3 представлено сервіс «CheckoutOrderService» клієнт-серверної системи та опис його методів із зазначенням їх призначення.

Таблиця 3.3 – Сервіс «CheckoutOrderService» клієнт-серверної системи і методи цього сервісу з описом їх призначення

№	Назва сервісу	Найменування методу сервісу	Призначення методу
1		CheckoutOrder save(CheckoutOrder checkoutOrder);	Додати замовлення до бази даних
2		List<CheckoutOrder> findAll();	Отримати усі замовлення з бази даних
3	CheckoutOrder Service	List<CheckoutOrder> findAllByUsername(String username);	Отримати замовлення за логіном клієнта

№	Назва сервісу	Найменування методу сервісу	Призначення методу
4		CheckoutOrder saveOrder(CheckoutOrderDto checkoutOrderDto, User user, List<Basket> basketList);	Створити замовлення для клієнта за його обраними товарами
5		CheckoutOrder findFirstByUser_IdOrderByCreatedDesc(Long userId);	Отримати останнє замовлення користувача за його id
7		CheckoutOrder findById(Long id);	Отримати замовлення за id
8		List<CheckoutOrder> findAllByIdOrderByCreated();	Отримати усі замовлення за датою створення від мін. до макс. та н
9		List<CheckoutOrder> findAllByIdOrderByCreatedDesc();	Отримати усі замовлення за датою від макс. до мін.
10		List<CheckoutOrder> findByUserId(Long userId);	Отримати усі замовлення клієнта за його id
11		List<CheckoutOrder> findAllByStatusOrderByCreated(OrderStatus orderStatus);	Отримати усі замовлення за статусом

У таблиці 3.4 наведено всі сервіси клієнт-серверної системи з описом їх призначення та вказівкою на відповідні таблиці бази даних, з якими вони взаємодіють.

Таблиця 3.4 – Перелік усіх сервісів в клієнт-серверної системі

№	Ім’я сервісу	Призначення сервісу	Взаємозв’язок з таблицею БД	Кількість методів
1	BasketService	Даний сервіс призначений для проведення операцій з об’єктом класу «Кошик»	«basket»	7
2	CategoryService	Даний сервіс призначений для проведення операцій з об’єктом класу «Категорія»	«main_categories»	3
3	CheckoutOrderHasProductService	Даний сервіс призначений для проведення операцій з об’єктами класу «Замовлення» та «Продукт»	«checkout_orderhasproduct»	2

№	Ім’я сервісу	Призначення сервісу	Взаємозв’язок з таблицею БД	Кількість методів
4	CheckoutOrderService	Даний сервіс призначений для проведення операцій з об’єктом класу «Замовлення»	«checkout_order»	10
5	ImageService	Даний сервіс призначений для проведення операцій з об’єктом класу «Зображення»	«image»	4
6	OsService	Даний сервіс призначений для проведення операцій з об’єктом класу «Операційна система»	«main_oss»	3
7	ProductService	Даний сервіс призначений для проведення операцій з об’єктом класу «Продукт»	«main_products»	16
8	StudioService	Даний сервіс призначений для проведення операцій з об’єктом класу «Студія»	«main_studios»	3
9	UserService	Даний сервіс призначений для проведення операцій з об’єктом класу «Користувач»	«users»	14

З таблиці зрозуміло, що створено дев'ять сервісів, кожен з яких відповідає за операції з конкретною таблицею бази даних. Для здійснення маніпуляцій з даними в цих таблицях було розроблено значну кількість методів для кожного сервісу, призначення якого описано в даних таблицях.

3.3 Проєктування системи за виробленими рекомендаціями

Маючи вироблені рекомендації та вже розроблену схему бази даних, яка відповідає їм, тепер можливо перейти до проєктування серверної та клієнтської частин системи.

Зазвичай розробка починається з серверної частини, бо вона описує об'єкти, які задіяні у процесі роботи системи. Як ці об'єкти повинні називатися та які атрибути мати вже описано на етапі розробки бази даних. Тому спочатку потрібно встановити підключення до неї та розробити засобами мови програмування java такі ж самі об'єкти.

Створені об'єкти зображені на рисунку 3.2

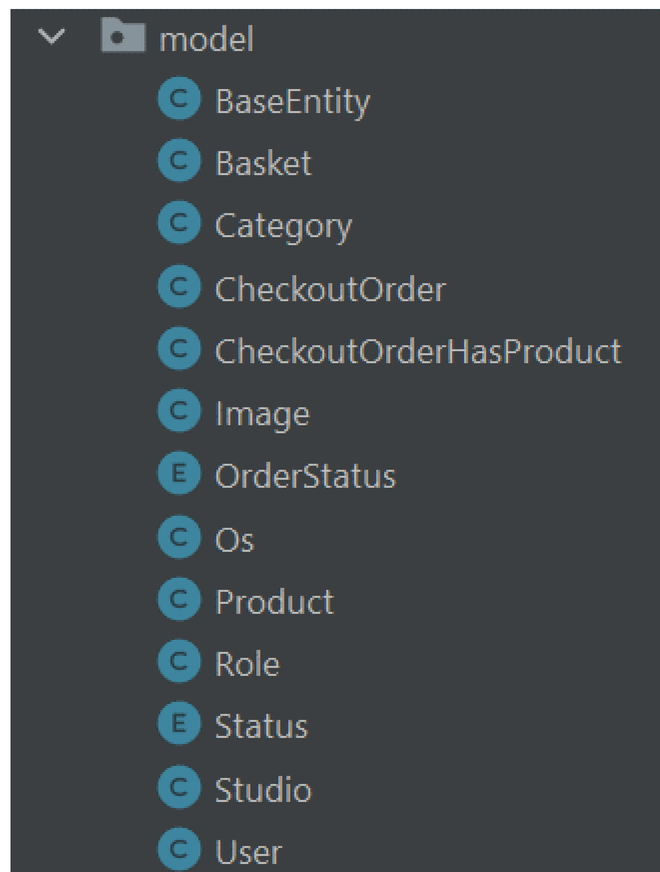

Рисунок 3.2 – Створені об'єкти засобами java

Згідно вироблених рекомендацій у підрозділі 2.2 потрібно реалізувати мікросервісну архітектуру для вже описаних мікросервісів у підрозділі 3.2 та реалізувати зв’язок для обміну даними між цими сервісами як показано на рисунку 3.3.

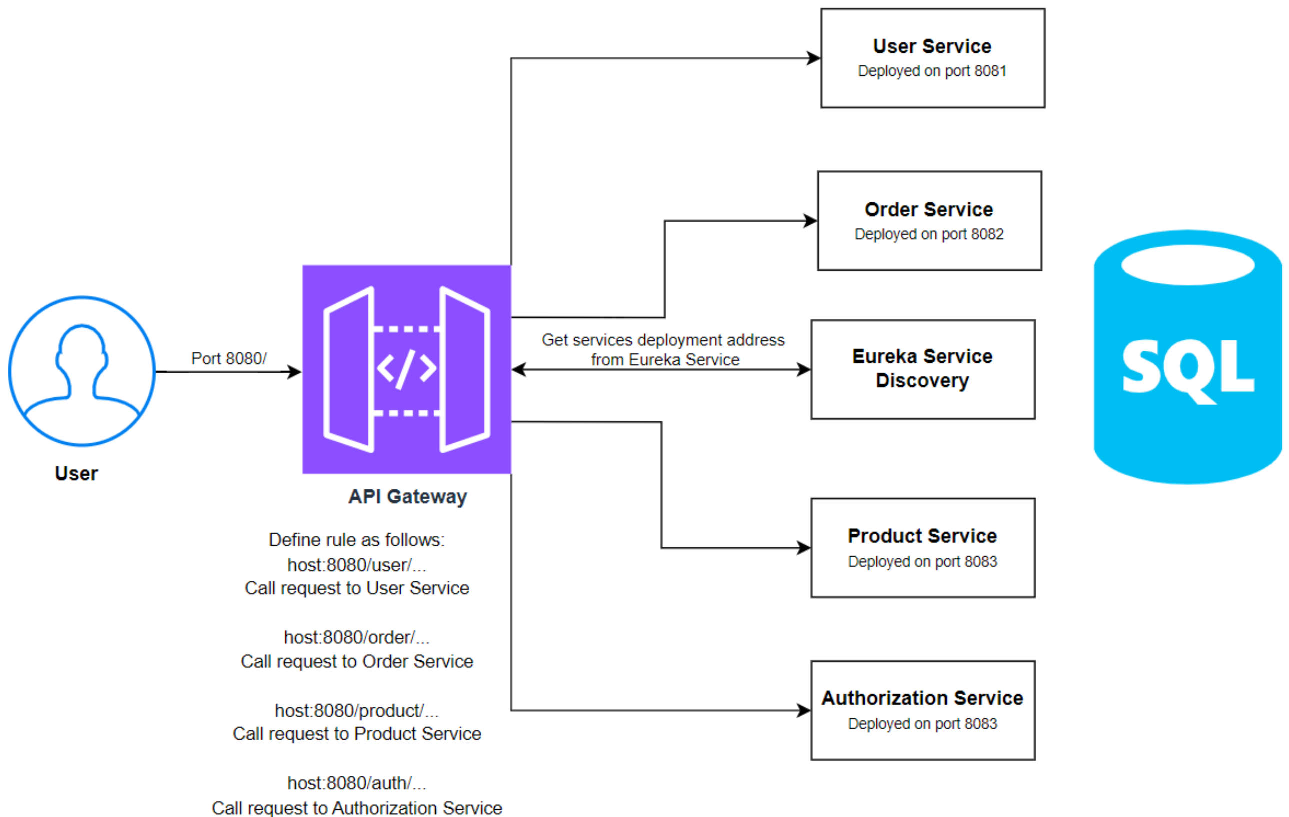

Рисунок 3.3 – Загальна схема робити системи

Тепер потрібно описати дані сервіси мовою програмування java спочатку їх інтерфейси, а вже потім реалізацію цих сервісів.

Кожний сервіс в мікросервісній архітектурі є standalone, тобто автономним, або самостійним, який не повинен залежати напряму від роботи інших сервісів. Звичайно, якщо у ході роботи сервісу він повинен працювати у парі з іншими сервісами, то задача полягає у тому, аби забезпечити стабільну роботу даного сервісу, навіть якщо інші в даний час не працюють взагалі, або мають проблеми у ході роботи системи.

Для цього створено нові класи, які відповідають назвам вже описаних сервісів і які мають вже реалізовані методи, що можуть використовуватися у ході роботи системи. Структура сервісів зображена на рисунку 3.4.

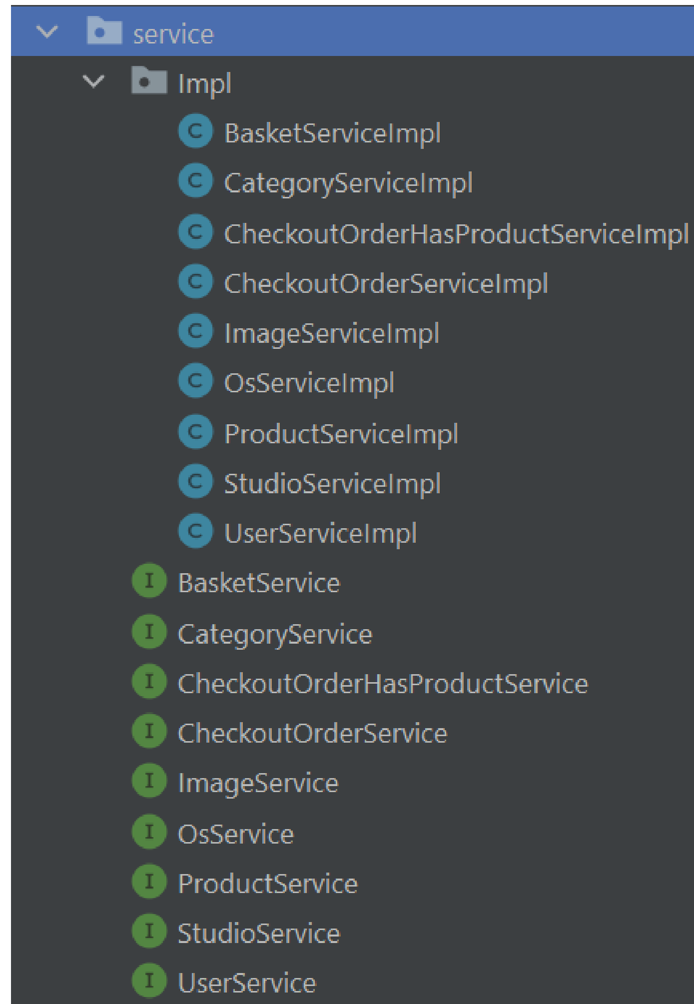

Рисунок 3.4 – Структура сервісів

Після того як сервіси були створені, потрібно налаштувати зв’язок між ними шляхом використання Eureka Service Discovery та правильно налаштованого API Gateway.

Так як кожний сервіс працює на своєму власному хості, то як наслідок він не може бачити навколо себе інші сервіси на відміну від монолітної системи, де завдяки тому, що все розробляється в рамках одного проєкту та запускається на одному сервері. Тому є необхідність зареєструвати кожен сервіс у Eureka Service Discovery, який дозволяє сервісам знаходити один одного та комунікувати один

з одним. Також завдяки Eureka Service Discovery відразу можна налаштувати балансувальник навантаження для того, аби маючи декілька однакових сервісів запити потрапляли на усі сервіси, а не постійно на один з них. Ще дуже важливим етапом є відстеження, тобто моніторинг і управління сервісами і Eureka Service Discovery дозволяє відслідковувати стан сервісів та надає інформацію, щодо поточного стану системи.

Тому для цього створимо окремий проєкт для написання реєстру сервісів. Для цього скористаємося сайтом https://start.spring.io/ і надамо потрібні дані як на рисунку 3.5.

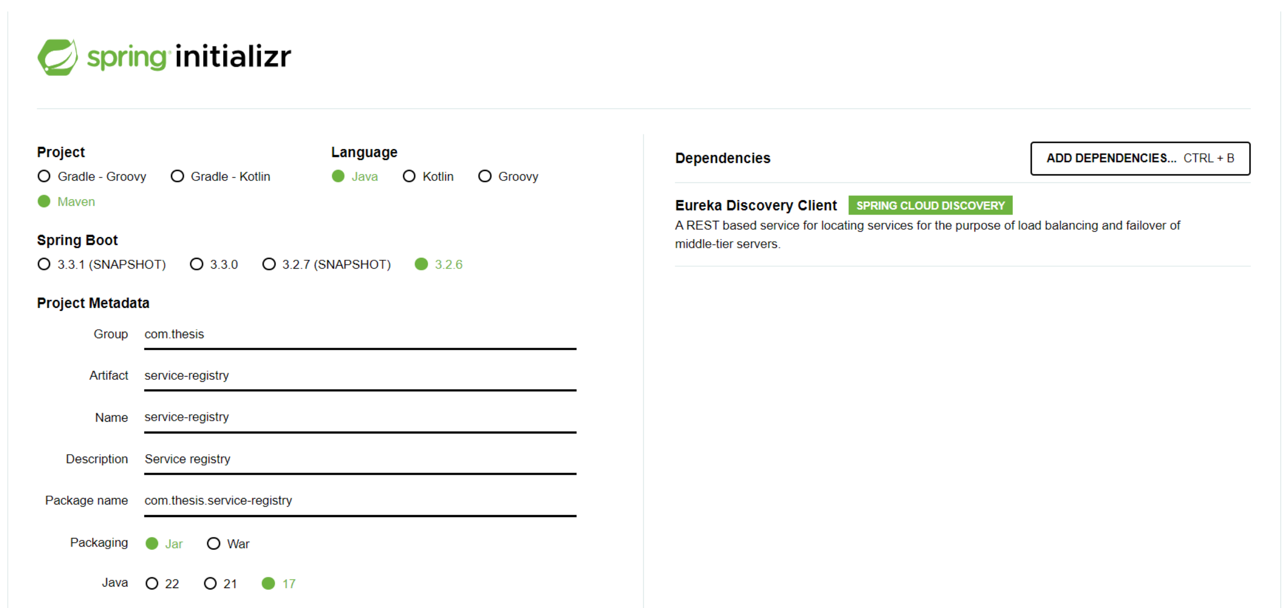

Рисунок 3.5 – Створення реєстру сервісів

Після створення початкового проєкту потрібно налаштувати реєстр сервісів вказавши йому порт на якому він буде працювати – в даному випадку це 8761 та ще декілька налаштувань як на рисунку 3.6.

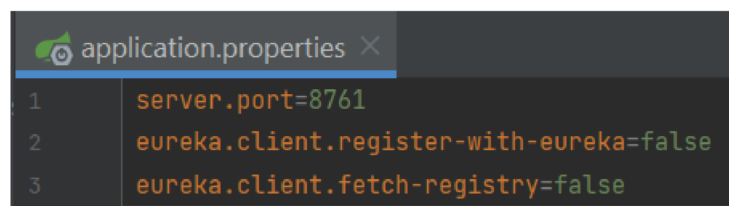

Рисунок 3.6 – Налаштування реєстру сервісів

Тепер потрібно вказати вже розробленим сервісам про те, що для них є реєстр сервісів і для цього потрібно також додати конфігураційні властивості до їх файлів конфігурації як це було зроблено для сервісу UserService. Приклад налаштування зображено на рисунку 3.7.

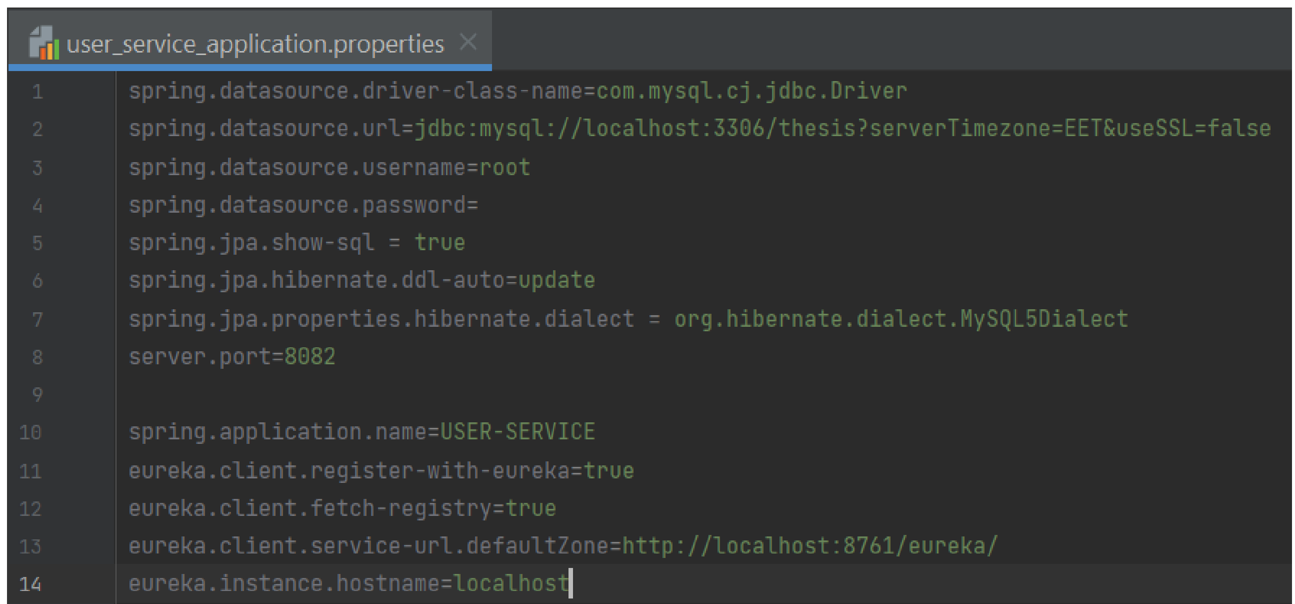

Рисунок 3.7 – Налаштування конфігурації сервісів

Те ж саме потрібно було зробити і для усіх інших сервісів, але вказанням їх порту та назви за якою вони будуть визначатися, а також додаванням потрібних анотацій вже застосовуючи java.

На даному етапі вже є розроблені сервіси та реєстр для цих сервісів і наступним етапом є створення API Gateway. Він відіграє роль посередника між мікросервісами та клієнтськими додатками, що працюють з ними. Потрібен він для забезпечення єдиної точки входу для усіх запитів надаючи єдиний інтерфейс для мікросервісів та таким чином спрощує клієнтську частину.

Для реалізації цього кроку було створено ще один проєкт з назвою «thesis-cloud-gateway». Порт на якому він буде працювати 8080 і цей же порт і буде використовуватися як єдиний інтерфейс для усіх запитів до системи.

Приклад налаштування сервісів UserService та ProductService зображено на рисунку 3.8.

Рисунок 3.7 – Налаштування шляхів для пересилки вхідних запитів

Усі запити які будуть містити у собі «/use/**» будуть перенаправлені до сервісу UserService і так само аналогічно з іншими сервісами.

Наразі с розробкою серверної частини можна зупинитися і перейти до розробки клієнтської частини, де розробка відбувається за допомогою фреймворку React, який вказаний у вироблених рекомендаціях.

Хід розробки UI є доволі стандартним і результати його можна побачити у додатку А.

Після того як UI є готовим до роботи можна повернутися до серверної частити та провести деякі зміни, а саме хід обробки запитів у системі. І для того аби система працювала швидше та була більш стабільна виходячи з рекомендацій необхідно використовувати event-processing, який забезпечується Kafka. Для цього необхідно мати Docker з docker-compose.yml файлом, який описує усі необхідні конфігурації і після запуску якого Kafka стане доступною для користування. Файл конфігурації зображений на рисунку 3.8.

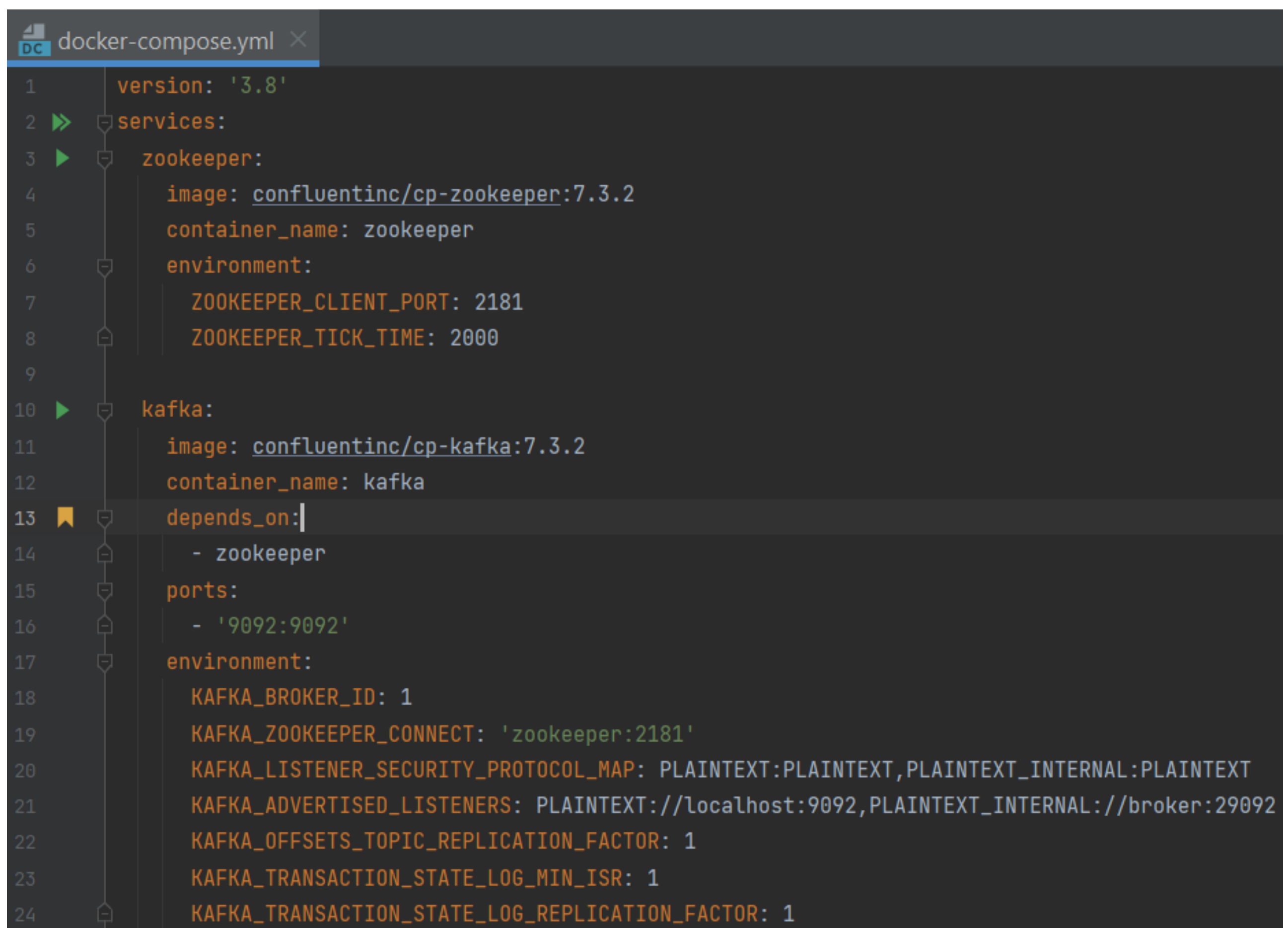

Рисунок 3.8 – Файл конфігурації для Kafka

В результаті тепер можливо налаштувати consumers та producers для роботи з подіями у системі. Їх використання полягає у тому, що коли у систему приходить запит, то він перекладається у вигляді події «event» до Kafka topic. І

вже потім відповідний consumer починає забирати цю неопрацьовану подію з Kafka topic і починає її опрацьовувати. Це значно полегшує роботу системи в цілому, бо не потрібно чекати наступному запит, коли попередній повністю завершиться, він так само швидко за допомогою Kafka producer буде відправлений до відповідного Kafka topic і потім буде опрацьований. Також, якщо у системі стався збій, то після повернення до нормального режиму Kafka consumer почне свою роботу з того місця, де він закінчив, що гарантує опрацювання подій, які ще знаходилися у Kafka topic до появи проблем у системі. Візуальне відображення контейнерів для Kafka та Zookeeper на рисунку 3.9.

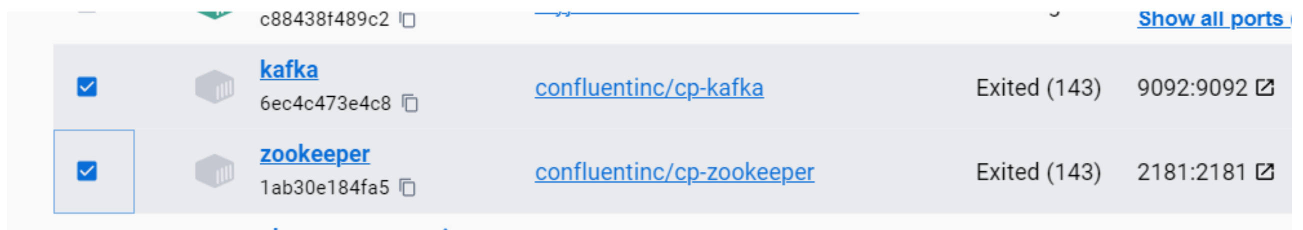

Рисунок 3.9 – Візуальне відображення контейнерів для Kafka та Zookeeper

Також для пришвидшення роботи системи та зменшення запитів до бази даних було вирішено використовувати кеш GuavaCache. Приклад як саме виглядає імплементація кешу представлено на рисунку 3.10.

```
private static LoadingCache<Long, User> userCache = CacheBuilder.newBuilder()
        .concurrencyLevel(4)
        .maximumSize(10000)
        .expireAfterWrite(15, TimeUnit.MINUTES)
        .build(new CacheLoader<Long, Optional<User>>() {

            @Override
            public Optional<User> load(Integer key)
                    throws Exception {
                User result = userRepository.findById(id).orElse(null);
                if (result == null) {
                    log.warn("IN findById - no user found {}", id);
                    return null;
                }
                log.info("IN findById - user: {} successfully found by id", result.getUsername());
                return result;
            }
        });
```

Рисунок 3.10 – Імплементація кешу для отримання інформації про користувача

Це доволі зручна бібліотека з великою кількістю можливостей для налаштування часу зберігання, кількості потоків, загального розміру кешу та функції, яка повинна виконатися у разі відсутності ключа в кеші, аби повернути відповідне до ключа значення.

Один з кешів, реалізований для отримання повної інформації про користувача за його ідентифікаційним номером, що значно пришвидшило роботу системи та зменшило кількість запитів до бази даних.

Описано основні етапи у проєктуванні системи, які відіграють значну роль при створенні та налаштуванні системи.

3.4 Моніторинг та аналіз результатів проєктування і роботи системи

Після того, як будь-яка система була створена потрібно відслідковувати її поточний стан. Отримання реальних даних на дану секунду дуже важливо, бо допомагає побачити, у свою чергу допомагає виявити проблему відразу і почати її вирішувати. Даний спосіб спостереження називається моніторинг.

Моніторинг - це систематичний і безперервний збір та аналіз інформації про хід реалізації інтервенції з розвитку. Моніторинг проводиться для того, щоб забезпечити належне інформування всіх людей, які повинні знати про втручання, і своєчасне прийняття рішень. Існує багато різних видів моніторингу, включаючи фінансовий моніторинг, моніторинг процесу та моніторинг впливу [18, 19].

Існує багато різних видів моніторингу.

Моніторинг процесу або результативності зосереджується на діяльності, що здійснюється в рамках втручання з розвитку. Він покликаний оцінити, чи та/або наскільки добре ці заходи впроваджуються. Він також охоплює використання ресурсів. Моніторинг процесу покликаний забезпечити інформацію, необхідну для постійного планування та аналізу роботи, оцінки успішності або неуспішності реалізації проектів і програм, виявлення та

вирішення проблем і викликів, а також використання можливостей, що з'являються в міру їх виникнення [20].

Виходячи з реалізації обраної архітектури та рекомендацій до проєктування клієнт-серверних систем обслуговування, саме мікросервісної архітектури для роботи якої використовується event-processing, який забезпечується Apache Kafka, важливим фактором є відслідковування кількості неопрацьованих подій у топіку. Важливим це є, бо зі збільшенням навантаження системи, тобто зі збільшення кількості запитів, які потрібно опрацювати, лаг буде збільшуватися і треба відслідковувати даний ріст і вносити певні зміни до працюючої системи.

Є декілька популярних методів, як можна працювати в умовах коли мається постійний ріст кількості запитів на опрацювання або постійна, але все одно велика їх кількість, це або блокування нових запитів, або збільшення кількості серверів для опрацювання цих запитів. Головною задачею є не допустити перенавантаження системи, бо у такому разі взагалі нічого працювати не буде. І тому, аби мати змогу бачити в реальному часі, як система працює з новими подіями, яка кількість з них ще не опрацьована, та як взагалі система працює, та чи потрібно вносити якість корективи у роботу, було вирішено розробити метрику, яка б показувала чи потрібне втручання у роботу системи.

Дана метрика включає в себе декілька показників системи – це середнє навантаження процесору за останню хвилину, середня кількість часу на опрацювання події та коефіцієнт, який в результаті покаже чи наблизився стан системи до критичного (формула 3.1).

$$t = \mathrm{round}(\mathrm{c} + \mathrm{e} * 2) * \exp\left(\frac{c}{55}\right) * 80 \tag{3.1}$$

де c – середнє навантаження процесору за останню хвилину;

e – середня кількість часу обробки подій у системі за останню хвилину;

t – результуючий загальний комбінований лаг в системі;

80 – константа для наближення результату до потрібного значення.

В даній формулі використовується експоненційне масштабування: використання експоненти може бути корисним для перетворення значень у випадках, коли потрібно забезпечити швидке зростання або спадання в залежності від зміни завантаження процесора. У даному випадку для отримання значення експоненти використовується середнє навантаження процесору за останню хвилину, яке перебуває в діапазоні від 1 до 100, де граничним значенням береться число 55. Це у свою чергу означає, що значення експоненціальної функції буде зростати не швидко до значення параметра «c», яке менше за 55 і буде зростати набагато швидше, якщо його значення буде більше за 55. Потрібно це аби наочно відразу побачити наявні проблеми у системі, бо при проходженні граничного значення числовий результат функції значно почне збільшуватися.

Для того аби продемонструвати як саме це працює, було вирішено підключити сервіс Amazon Cloud. Даний сервіс є платний і за кожну відправку значення на нього потрібно платити, але для демонстративних цілей та як саме можна його використовувати і було вирішено підключити саме його.

Amazon CloudWatch – це сервіс, який відстежує додатки, реагує на зміни продуктивності, оптимізує використання ресурсів та надає інформацію про стан роботи. Збираючи дані з усіх ресурсів AWS, CloudWatch дає уявлення про продуктивність всієї системи і дозволяє користувачам встановлювати тривоги, автоматично реагувати на зміни та отримувати уніфіковане уявлення про стан роботи [21].

Приклад як працює, та навіщо потрібен Amazon CloudWatch (рис 3.11).

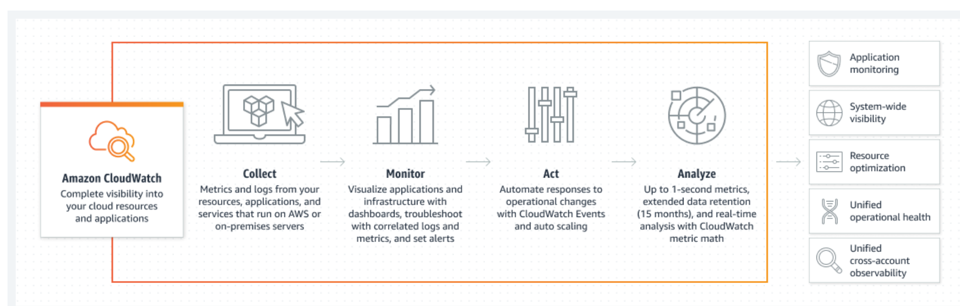

Рисунок 3.11 – Схема роботи Amazon CloudWatch

Amazon CloudWatch збирає та візуалізує журнали, метрики та дані про події в режимі реального часу на автоматизованих інформаційних панелях, щоб спростити обслуговування вашої інфраструктури та додатків [22].

Для того, аби створити дійсно велике навантаження було вирішено почати виконувати постійні запити на опрацювання оновлення профілей користовучів та створення нових користувачів у циклі, тобто щось подібне до DDoS-атаки.

Розподілена атака на відмову в обслуговуванні (DDoS) - це зловмисна спроба порушити нормальний трафік цільового сервера, сервісу або мережі, перевантаживши ціль або навколишню інфраструктуру потоком інтернет-трафіку [23].

Ефективність DDoS-атак досягається завдяки використанню декількох скомпрометованих комп'ютерних систем як джерел атакуючого трафіку. З високого рівня, DDoS-атака схожа на несподівану пробку, яка забиває шосе, не даючи звичайному трафіку прибути до місця призначення [24].

При проведенні навантаження системи, результуючі дані кожної хвилини записуються в Amazon CloudWatch. Для цього в java було створено потік (thread), який спрацьовує кожну хвилину і задача якого зібрати потрібні дані та відправити до сервісу. Результат проведення навантаження можна спостерігати на рисунку 3.12.

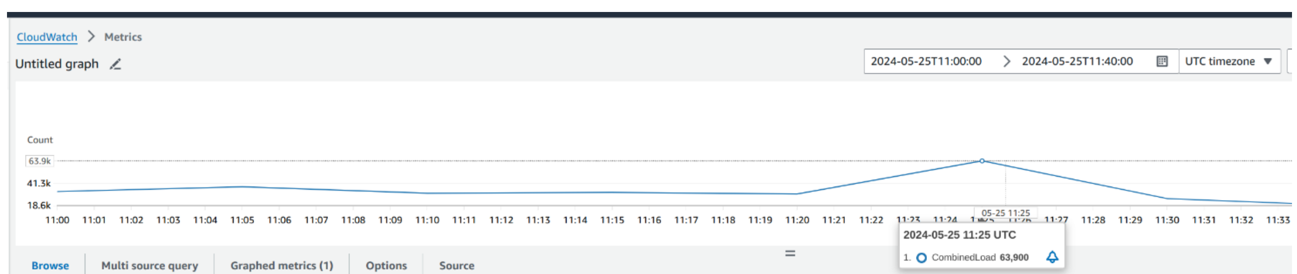

Рисунок 3.12 – Графік зміни загального комбінованого лагу в системі

Використовуючи формулу (формула 3.1) потік за розписом відпрацьовував і надсилав дані до Amazon CloudWatch. На даному проміжку часу можна побачити, що загальний комбінований лаг вже був трохи більшим ніж 30 тисяч, потім трохи зменшився, а коли запити різко почали зростати, бо паралельно відпрацьовувало відправлення запитів з різних джерел, то комбінований лаг стрімко пішов у гору. Сталося це тому, що навантаження на процесор збільшилося значно, а отже і результат функції також, і як результат на графіку добре видно, коли саме навантаження на систему різко зросло.

Платформа Amazon дозволяє досить гнучко та зручно використовувати її сервіси один з одним. Наприклад, CloudWatch сервіс можна використовувати як засіб для відслідковування проблем нестачі або навпаки простою серверів і відповідно до умов використовувати результуючу метрику, таку як загальний комбінований лаг, для autoscaling (автомасштабування).

Автомасштабування (autoscaling) - це функція хмарних обчислень, яка дозволяє організаціям автоматично масштабувати хмарні сервіси, такі як серверні потужності або віртуальні машини, залежно від певних ситуацій, таких як рівень трафіку або використання. Провайдери хмарних обчислень, такі як Amazon Web Services (AWS), Microsoft Azure та Google Cloud Platform (GCP), пропонують інструменти автомасштабування.

Базові функції автомасштабування також дозволяють знизити витрати та підвищити надійність роботи, плавно збільшуючи та зменшуючи кількість нових екземплярів, коли попит на них зростає або падає. Таким чином, автомасштабування забезпечує узгодженість, незважаючи на динамічний і часом непередбачуваний попит на додатки.

Загальна перевага автомасштабування полягає в тому, що воно усуває необхідність реагувати вручну в режимі реального часу на сплески трафіку, які вимагають нових ресурсів та екземплярів, шляхом автоматичної зміни активної кількості серверів. Кожен з цих серверів потребує конфігурації, моніторингу та виведення з експлуатації, що є основою автомасштабування.

Наприклад, коли такий сплеск викликаний розподіленою атакою на відмову в обслуговуванні (DDoS), його може бути важко розпізнати. Ефективніший моніторинг показників автомасштабування та кращі політики автомасштабування іноді можуть допомогти системі швидко відреагувати на цю проблему. Аналогічно, база даних з автоматичним масштабуванням автоматично збільшує або зменшує ємність, запускається або вимикається залежно від потреб програми [25].

Для тестування розробленої комбінованої метрики для виявлення лагу у системі проводилося використання різних значень показників для виявлення їх впливу на результуюче значення метрики рисунки 3.13 – 3.15.

```
cpu=0, lag=0, duration=2 f2(cpu,lag,duration)=320.0
cpu=10, lag=0, duration=2 f2(cpu,lag,duration)=1343.0
cpu=20, lag=0, duration=2 f2(cpu,lag,duration)=2762.0
cpu=30, lag=0, duration=2 f2(cpu,lag,duration)=4693.0
cpu=40, lag=0, duration=2 f2(cpu,lag,duration)=7284.0
cpu=50, lag=0, duration=2 f2(cpu,lag,duration)=10723.0
cpu=60, lag=0, duration=2 f2(cpu,lag,duration)=15242.0
cpu=70, lag=0, duration=2 f2(cpu,lag,duration)=21138.0
cpu=80, lag=0, duration=2 f2(cpu,lag,duration)=28779.0
cpu=85, lag=0, duration=2 f2(cpu,lag,duration)=33394.0
cpu=90, lag=0, duration=2 f2(cpu,lag,duration)=38626.0
cpu=95, lag=0, duration=2 f2(cpu,lag,duration)=44552.0
cpu=100, lag=0, duration=2 f2(cpu,lag,duration)=51257.0
cpu=0, lag=0, duration=2 f2(cpu,lag,duration)=320.0
cpu=0, lag=0, duration=3 f2(cpu,lag,duration)=480.0
cpu=0, lag=0, duration=4 f2(cpu,lag,duration)=640.0
cpu=0, lag=0, duration=5 f2(cpu,lag,duration)=800.0
cpu=0, lag=0, duration=6 f2(cpu,lag,duration)=960.0
cpu=0, lag=0, duration=7 f2(cpu,lag,duration)=1120.0
cpu=0, lag=0, duration=8 f2(cpu,lag,duration)=1280.0
cpu=0, lag=0, duration=9 f2(cpu,lag,duration)=1440.0
cpu=0, lag=0, duration=10 f2(cpu,lag,duration)=1600.0
cpu=0, lag=0, duration=11 f2(cpu,lag,duration)=1760.0
cpu=0, lag=0, duration=5 f2(cpu,lag,duration)=800.0
cpu=0, lag=0, duration=5 f2(cpu,lag,duration)=800.0
cpu=0, lag=5000, duration=3 f2(cpu,lag,duration)=480.0
cpu=10, lag=5000, duration=3 f2(cpu,lag,duration)=1535.0
cpu=20, lag=5000, duration=3 f2(cpu,lag,duration)=2992.0
cpu=30, lag=5000, duration=3 f2(cpu,lag,duration)=4969.0
cpu=40, lag=5000, duration=3 f2(cpu,lag,duration)=7615.0
```

```
cpu=50, lag=5000, duration=3 f2(cpu,lag,duration)=11120.0
cpu=60, lag=5000, duration=3 f2(cpu,lag,duration)=15718.0
cpu=70, lag=5000, duration=3 f2(cpu,lag,duration)=21709.0
cpu=80, lag=5000, duration=3 f2(cpu,lag,duration)=29464.0
cpu=85, lag=5000, duration=3 f2(cpu,lag,duration)=34144.0
cpu=90, lag=5000, duration=3 f2(cpu,lag,duration)=39448.0
cpu=95, lag=5000, duration=3 f2(cpu,lag,duration)=45452.0
cpu=100, lag=5000, duration=3 f2(cpu,lag,duration)=52242.0
cpu=10, lag=5000, duration=2 f2(cpu,lag,duration)=1343.0
cpu=10, lag=5000, duration=3 f2(cpu,lag,duration)=1535.0
cpu=10, lag=5000, duration=4 f2(cpu,lag,duration)=1727.0
cpu=10, lag=5000, duration=5 f2(cpu,lag,duration)=1919.0
cpu=10, lag=5000, duration=6 f2(cpu,lag,duration)=2111.0
cpu=10, lag=5000, duration=7 f2(cpu,lag,duration)=2303.0
cpu=10, lag=5000, duration=8 f2(cpu,lag,duration)=2495.0
cpu=10, lag=5000, duration=9 f2(cpu,lag,duration)=2687.0
cpu=10, lag=5000, duration=10 f2(cpu,lag,duration)=2879.0
cpu=10, lag=5000, duration=11 f2(cpu,lag,duration)=3070.0
cpu=10, lag=5000, duration=5 f2(cpu,lag,duration)=1919.0
cpu=10, lag=5000, duration=5 f2(cpu,lag,duration)=1919.0
cpu=0, lag=10000, duration=4 f2(cpu,lag,duration)=640.0
cpu=10, lag=10000, duration=4 f2(cpu,lag,duration)=1727.0
cpu=20, lag=10000, duration=4 f2(cpu,lag,duration)=3222.0
cpu=30, lag=10000, duration=4 f2(cpu,lag,duration)=5245.0
cpu=40, lag=10000, duration=4 f2(cpu,lag,duration)=7947.0
cpu=50, lag=10000, duration=4 f2(cpu,lag,duration)=11517.0
cpu=60, lag=10000, duration=4 f2(cpu,lag,duration)=16195.0
cpu=70, lag=10000, duration=4 f2(cpu,lag,duration)=22280.0
cpu=80, lag=10000, duration=4 f2(cpu,lag,duration)=30149.0
cpu=85, lag=10000, duration=4 f2(cpu,lag,duration)=34894.0
cpu=90, lag=10000, duration=4 f2(cpu,lag,duration)=40270.0
cpu=95, lag=10000, duration=4 f2(cpu,lag,duration)=46352.0
```

Рисунок 3.13 – Результати тестування метрики

```
cpu=20, lag=10000, duration=2 f2(cpu,lag,duration)=2762.0
cpu=20, lag=10000, duration=3 f2(cpu,lag,duration)=2992.0
cpu=20, lag=10000, duration=4 f2(cpu,lag,duration)=3222.0
cpu=20, lag=10000, duration=5 f2(cpu,lag,duration)=3453.0
cpu=20, lag=10000, duration=6 f2(cpu,lag,duration)=3683.0
cpu=20, lag=10000, duration=7 f2(cpu,lag,duration)=3913.0
cpu=20, lag=10000, duration=8 f2(cpu,lag,duration)=4143.0
cpu=20, lag=10000, duration=9 f2(cpu,lag,duration)=4373.0
cpu=20, lag=10000, duration=10 f2(cpu,lag,duration)=4603.0
cpu=20, lag=10000, duration=11 f2(cpu,lag,duration)=4834.0
cpu=20, lag=10000, duration=5 f2(cpu,lag,duration)=3453.0
cpu=20, lag=10000, duration=5 f2(cpu,lag,duration)=3453.0
cpu=0, lag=15000, duration=5 f2(cpu,lag,duration)=800.0
cpu=10, lag=15000, duration=5 f2(cpu,lag,duration)=1919.0
cpu=20, lag=15000, duration=5 f2(cpu,lag,duration)=3453.0
cpu=30, lag=15000, duration=5 f2(cpu,lag,duration)=5521.0
cpu=40, lag=15000, duration=5 f2(cpu,lag,duration)=8278.0
cpu=50, lag=15000, duration=5 f2(cpu,lag,duration)=11914.0
cpu=60, lag=15000, duration=5 f2(cpu,lag,duration)=16671.0
cpu=70, lag=15000, duration=5 f2(cpu,lag,duration)=22852.0
cpu=80, lag=15000, duration=5 f2(cpu,lag,duration)=30834.0
cpu=85, lag=15000, duration=5 f2(cpu,lag,duration)=35645.0
cpu=90, lag=15000, duration=5 f2(cpu,lag,duration)=41092.0
cpu=95, lag=15000, duration=5 f2(cpu,lag,duration)=47252.0
cpu=100, lag=15000, duration=5 f2(cpu,lag,duration)=54214.0
cpu=30, lag=15000, duration=2 f2(cpu,lag,duration)=4693.0
cpu=30, lag=15000, duration=3 f2(cpu,lag,duration)=4969.0
cpu=30, lag=15000, duration=4 f2(cpu,lag,duration)=5245.0
cpu=30, lag=15000, duration=5 f2(cpu,lag,duration)=5521.0
cpu=30, lag=15000, duration=6 f2(cpu,lag,duration)=5797.0
cpu=30, lag=15000, duration=7 f2(cpu,lag,duration)=6073.0
```

```
cpu=30, lag=15000, duration=10 f2(cpu,lag,duration)=6902.0
cpu=30, lag=15000, duration=11 f2(cpu,lag,duration)=7178.0
cpu=30, lag=15000, duration=5 f2(cpu,lag,duration)=5521.0
cpu=30, lag=15000, duration=5 f2(cpu,lag,duration)=5521.0
cpu=0, lag=20000, duration=6 f2(cpu,lag,duration)=960.0
cpu=10, lag=20000, duration=6 f2(cpu,lag,duration)=2111.0
cpu=20, lag=20000, duration=6 f2(cpu,lag,duration)=3683.0
cpu=30, lag=20000, duration=6 f2(cpu,lag,duration)=5797.0
cpu=40, lag=20000, duration=6 f2(cpu,lag,duration)=8609.0
cpu=50, lag=20000, duration=6 f2(cpu,lag,duration)=12311.0
cpu=60, lag=20000, duration=6 f2(cpu,lag,duration)=17147.0
cpu=70, lag=20000, duration=6 f2(cpu,lag,duration)=23423.0
cpu=80, lag=20000, duration=6 f2(cpu,lag,duration)=31519.0
cpu=85, lag=20000, duration=6 f2(cpu,lag,duration)=36395.0
cpu=90, lag=20000, duration=6 f2(cpu,lag,duration)=41913.0
cpu=95, lag=20000, duration=6 f2(cpu,lag,duration)=48152.0
cpu=100, lag=20000, duration=6 f2(cpu,lag,duration)=55199.0
cpu=40, lag=20000, duration=2 f2(cpu,lag,duration)=7284.0
cpu=40, lag=20000, duration=3 f2(cpu,lag,duration)=7615.0
cpu=40, lag=20000, duration=4 f2(cpu,lag,duration)=7947.0
cpu=40, lag=20000, duration=5 f2(cpu,lag,duration)=8278.0
cpu=40, lag=20000, duration=6 f2(cpu,lag,duration)=8609.0
cpu=40, lag=20000, duration=7 f2(cpu,lag,duration)=8940.0
cpu=40, lag=20000, duration=8 f2(cpu,lag,duration)=9271.0
cpu=40, lag=20000, duration=9 f2(cpu,lag,duration)=9602.0
cpu=40, lag=20000, duration=10 f2(cpu,lag,duration)=9933.0
cpu=40, lag=20000, duration=11 f2(cpu,lag,duration)=10264.0
cpu=40, lag=20000, duration=5 f2(cpu,lag,duration)=8278.0
cpu=40, lag=20000, duration=5 f2(cpu,lag,duration)=8278.0
cpu=0, lag=25000, duration=7 f2(cpu,lag,duration)=1120.0
cpu=10, lag=25000, duration=7 f2(cpu,lag,duration)=2303.0
cpu=20, lag=25000, duration=7 f2(cpu,lag,duration)=3913.0
```

```
cpu=40, lag=25000, duration=7 f2(cpu,lag,duration)=8940.0
cpu=50, lag=25000, duration=7 f2(cpu,lag,duration)=12708.0
cpu=60, lag=25000, duration=7 f2(cpu,lag,duration)=17624.0
cpu=70, lag=25000, duration=7 f2(cpu,lag,duration)=23994.0
cpu=80, lag=25000, duration=7 f2(cpu,lag,duration)=32205.0
cpu=85, lag=25000, duration=7 f2(cpu,lag,duration)=37146.0
cpu=90, lag=25000, duration=7 f2(cpu,lag,duration)=42735.0
cpu=95, lag=25000, duration=7 f2(cpu,lag,duration)=49053.0
cpu=100, lag=25000, duration=7 f2(cpu,lag,duration)=56185.0
cpu=50, lag=25000, duration=2 f2(cpu,lag,duration)=10723.0
cpu=50, lag=25000, duration=3 f2(cpu,lag,duration)=11120.0
cpu=50, lag=25000, duration=4 f2(cpu,lag,duration)=11517.0
cpu=50, lag=25000, duration=5 f2(cpu,lag,duration)=11914.0
cpu=50, lag=25000, duration=6 f2(cpu,lag,duration)=12311.0
cpu=50, lag=25000, duration=7 f2(cpu,lag,duration)=12708.0
cpu=50, lag=25000, duration=8 f2(cpu,lag,duration)=13105.0
cpu=50, lag=25000, duration=9 f2(cpu,lag,duration)=13502.0
cpu=50, lag=25000, duration=10 f2(cpu,lag,duration)=13900.0
cpu=50, lag=25000, duration=11 f2(cpu,lag,duration)=14297.0
cpu=50, lag=25000, duration=5 f2(cpu,lag,duration)=11914.0
cpu=50, lag=25000, duration=5 f2(cpu,lag,duration)=11914.0
cpu=0, lag=30000, duration=8 f2(cpu,lag,duration)=1280.0
cpu=10, lag=30000, duration=8 f2(cpu,lag,duration)=2495.0
cpu=20, lag=30000, duration=8 f2(cpu,lag,duration)=4143.0
cpu=30, lag=30000, duration=8 f2(cpu,lag,duration)=6349.0
cpu=40, lag=30000, duration=8 f2(cpu,lag,duration)=9271.0
cpu=50, lag=30000, duration=8 f2(cpu,lag,duration)=13105.0
cpu=60, lag=30000, duration=8 f2(cpu,lag,duration)=18100.0
cpu=70, lag=30000, duration=8 f2(cpu,lag,duration)=24566.0
cpu=80, lag=30000, duration=8 f2(cpu,lag,duration)=32890.0
cpu=85, lag=30000, duration=8 f2(cpu,lag,duration)=37896.0
cpu=90, lag=30000, duration=8 f2(cpu,lag,duration)=43557.0
```

```
cpu=85, lag=30000, duration=8 f2(cpu,lag,duration)=37896.0
cpu=90, lag=30000, duration=8 f2(cpu,lag,duration)=43557.0
cpu=95, lag=30000, duration=8 f2(cpu,lag,duration)=49953.0
cpu=100, lag=30000, duration=8 f2(cpu,lag,duration)=57171.0
cpu=60, lag=30000, duration=2 f2(cpu,lag,duration)=15242.0
cpu=60, lag=30000, duration=3 f2(cpu,lag,duration)=15718.0
cpu=60, lag=30000, duration=4 f2(cpu,lag,duration)=16195.0
cpu=60, lag=30000, duration=5 f2(cpu,lag,duration)=16671.0
cpu=60, lag=30000, duration=6 f2(cpu,lag,duration)=17147.0
cpu=60, lag=30000, duration=7 f2(cpu,lag,duration)=17624.0
cpu=60, lag=30000, duration=8 f2(cpu,lag,duration)=18100.0
cpu=60, lag=30000, duration=9 f2(cpu,lag,duration)=18576.0
cpu=60, lag=30000, duration=10 f2(cpu,lag,duration)=19053.0
cpu=60, lag=30000, duration=11 f2(cpu,lag,duration)=19529.0
cpu=60, lag=30000, duration=5 f2(cpu,lag,duration)=16671.0
cpu=60, lag=30000, duration=5 f2(cpu,lag,duration)=16671.0
cpu=0, lag=35000, duration=9 f2(cpu,lag,duration)=1440.0
cpu=10, lag=35000, duration=9 f2(cpu,lag,duration)=2687.0
cpu=20, lag=35000, duration=9 f2(cpu,lag,duration)=4373.0
cpu=30, lag=35000, duration=9 f2(cpu,lag,duration)=6626.0
cpu=40, lag=35000, duration=9 f2(cpu,lag,duration)=9602.0
cpu=50, lag=35000, duration=9 f2(cpu,lag,duration)=13502.0
cpu=60, lag=35000, duration=9 f2(cpu,lag,duration)=18576.0
cpu=70, lag=35000, duration=9 f2(cpu,lag,duration)=25137.0
cpu=80, lag=35000, duration=9 f2(cpu,lag,duration)=33575.0
cpu=85, lag=35000, duration=9 f2(cpu,lag,duration)=38646.0
cpu=90, lag=35000, duration=9 f2(cpu,lag,duration)=44379.0
cpu=95, lag=35000, duration=9 f2(cpu,lag,duration)=50853.0
cpu=100, lag=35000, duration=9 f2(cpu,lag,duration)=58157.0
cpu=70, lag=35000, duration=2 f2(cpu,lag,duration)=21138.0
cpu=70, lag=35000, duration=3 f2(cpu,lag,duration)=21709.0
cpu=70, lag=35000, duration=4 f2(cpu,lag,duration)=22280.0
```

Рисунок 3.14 – Результати тестування метрики

```
cpu=70, lag=35000, duration=6 f2(cpu,lag,duration)=23423.0
cpu=70, lag=35000, duration=7 f2(cpu,lag,duration)=23994.0
cpu=70, lag=35000, duration=8 f2(cpu,lag,duration)=24566.0
cpu=70, lag=35000, duration=9 f2(cpu,lag,duration)=25137.0
cpu=70, lag=35000, duration=10 f2(cpu,lag,duration)=25708.0
cpu=70, lag=35000, duration=11 f2(cpu,lag,duration)=26279.0
cpu=70, lag=35000, duration=5 f2(cpu,lag,duration)=22852.0
cpu=70, lag=35000, duration=5 f2(cpu,lag,duration)=22852.0
cpu=0, lag=40000, duration=10 f2(cpu,lag,duration)=1600.0
cpu=10, lag=40000, duration=10 f2(cpu,lag,duration)=2879.0
cpu=20, lag=40000, duration=10 f2(cpu,lag,duration)=4603.0
cpu=30, lag=40000, duration=10 f2(cpu,lag,duration)=6902.0
cpu=40, lag=40000, duration=10 f2(cpu,lag,duration)=9933.0
cpu=50, lag=40000, duration=10 f2(cpu,lag,duration)=13900.0
cpu=60, lag=40000, duration=10 f2(cpu,lag,duration)=19053.0
cpu=70, lag=40000, duration=10 f2(cpu,lag,duration)=25708.0
cpu=80, lag=40000, duration=10 f2(cpu,lag,duration)=34260.0
cpu=85, lag=40000, duration=10 f2(cpu,lag,duration)=39397.0
cpu=90, lag=40000, duration=10 f2(cpu,lag,duration)=45201.0
cpu=95, lag=40000, duration=10 f2(cpu,lag,duration)=51753.0
cpu=100, lag=40000, duration=10 f2(cpu,lag,duration)=59142.0
cpu=80, lag=40000, duration=2 f2(cpu,lag,duration)=28779.0
cpu=80, lag=40000, duration=3 f2(cpu,lag,duration)=29464.0
cpu=80, lag=40000, duration=4 f2(cpu,lag,duration)=30149.0
cpu=80, lag=40000, duration=5 f2(cpu,lag,duration)=30834.0
cpu=80, lag=40000, duration=6 f2(cpu,lag,duration)=31519.0
cpu=80, lag=40000, duration=7 f2(cpu,lag,duration)=32205.0
cpu=80, lag=40000, duration=8 f2(cpu,lag,duration)=32890.0
cpu=80, lag=40000, duration=9 f2(cpu,lag,duration)=33575.0
cpu=80, lag=40000, duration=10 f2(cpu,lag,duration)=34260.0
cpu=80, lag=40000, duration=11 f2(cpu,lag,duration)=34945.0
cpu=80, lag=40000, duration=5 f2(cpu,lag,duration)=30834.0
```

```
cpu=0, lag=45000, duration=11 f2(cpu,lag,duration)=1760.0
cpu=10, lag=45000, duration=11 f2(cpu,lag,duration)=3070.0
cpu=20, lag=45000, duration=11 f2(cpu,lag,duration)=4834.0
cpu=30, lag=45000, duration=11 f2(cpu,lag,duration)=7178.0
cpu=40, lag=45000, duration=11 f2(cpu,lag,duration)=10264.0
cpu=50, lag=45000, duration=11 f2(cpu,lag,duration)=14297.0
cpu=60, lag=45000, duration=11 f2(cpu,lag,duration)=19529.0
cpu=70, lag=45000, duration=11 f2(cpu,lag,duration)=26279.0
cpu=80, lag=45000, duration=11 f2(cpu,lag,duration)=34945.0
cpu=85, lag=45000, duration=11 f2(cpu,lag,duration)=40147.0
cpu=90, lag=45000, duration=11 f2(cpu,lag,duration)=46023.0
cpu=95, lag=45000, duration=11 f2(cpu,lag,duration)=52653.0
cpu=100, lag=45000, duration=11 f2(cpu,lag,duration)=60128.0
cpu=85, lag=45000, duration=2 f2(cpu,lag,duration)=33394.0
cpu=85, lag=45000, duration=3 f2(cpu,lag,duration)=34144.0
cpu=85, lag=45000, duration=4 f2(cpu,lag,duration)=34894.0
cpu=85, lag=45000, duration=5 f2(cpu,lag,duration)=35645.0
cpu=85, lag=45000, duration=6 f2(cpu,lag,duration)=36395.0
cpu=85, lag=45000, duration=7 f2(cpu,lag,duration)=37146.0
cpu=85, lag=45000, duration=8 f2(cpu,lag,duration)=37896.0
cpu=85, lag=45000, duration=9 f2(cpu,lag,duration)=38646.0
cpu=85, lag=45000, duration=10 f2(cpu,lag,duration)=39397.0
cpu=85, lag=45000, duration=11 f2(cpu,lag,duration)=40147.0
cpu=85, lag=45000, duration=5 f2(cpu,lag,duration)=35645.0
cpu=85, lag=45000, duration=5 f2(cpu,lag,duration)=35645.0
cpu=0, lag=50000, duration=5 f2(cpu,lag,duration)=50000.0
cpu=10, lag=50000, duration=5 f2(cpu,lag,duration)=50000.0
cpu=20, lag=50000, duration=5 f2(cpu,lag,duration)=50000.0
cpu=30, lag=50000, duration=5 f2(cpu,lag,duration)=50000.0
cpu=40, lag=50000, duration=5 f2(cpu,lag,duration)=50000.0
cpu=50, lag=50000, duration=5 f2(cpu,lag,duration)=50000.0
cpu=60, lag=50000, duration=5 f2(cpu,lag,duration)=50000.0
```

```
cpu=90, lag=50000, duration=8 f2(cpu,lag,duration)=50000.0
cpu=90, lag=50000, duration=9 f2(cpu,lag,duration)=50000.0
cpu=90, lag=50000, duration=10 f2(cpu,lag,duration)=50000.0
cpu=90, lag=50000, duration=11 f2(cpu,lag,duration)=50000.0
cpu=90, lag=50000, duration=5 f2(cpu,lag,duration)=50000.0
cpu=90, lag=50000, duration=5 f2(cpu,lag,duration)=50000.0
cpu=0, lag=55000, duration=5 f2(cpu,lag,duration)=55000.0
cpu=10, lag=55000, duration=5 f2(cpu,lag,duration)=55000.0
cpu=20, lag=55000, duration=5 f2(cpu,lag,duration)=55000.0
cpu=30, lag=55000, duration=5 f2(cpu,lag,duration)=55000.0
cpu=40, lag=55000, duration=5 f2(cpu,lag,duration)=55000.0
cpu=50, lag=55000, duration=5 f2(cpu,lag,duration)=55000.0
cpu=60, lag=55000, duration=5 f2(cpu,lag,duration)=55000.0
cpu=70, lag=55000, duration=5 f2(cpu,lag,duration)=55000.0
cpu=80, lag=55000, duration=5 f2(cpu,lag,duration)=55000.0
```

```
cpu=85, lag=55000, duration=5 f2(cpu,lag,duration)=55000.0
cpu=90, lag=55000, duration=5 f2(cpu,lag,duration)=55000.0
cpu=95, lag=55000, duration=5 f2(cpu,lag,duration)=55000.0
cpu=100, lag=55000, duration=5 f2(cpu,lag,duration)=55000.0
cpu=95, lag=55000, duration=2 f2(cpu,lag,duration)=55000.0
cpu=95, lag=55000, duration=3 f2(cpu,lag,duration)=55000.0
cpu=95, lag=55000, duration=4 f2(cpu,lag,duration)=55000.0
cpu=95, lag=55000, duration=5 f2(cpu,lag,duration)=55000.0
cpu=95, lag=55000, duration=6 f2(cpu,lag,duration)=55000.0
cpu=95, lag=55000, duration=7 f2(cpu,lag,duration)=55000.0
cpu=95, lag=55000, duration=8 f2(cpu,lag,duration)=55000.0
cpu=95, lag=55000, duration=9 f2(cpu,lag,duration)=55000.0
cpu=95, lag=55000, duration=10 f2(cpu,lag,duration)=55000.0
cpu=95, lag=55000, duration=11 f2(cpu,lag,duration)=55000.0
cpu=95, lag=55000, duration=5 f2(cpu,lag,duration)=55000.0
cpu=95, lag=55000, duration=5 f2(cpu,lag,duration)=55000.0
```

Рисунок 3.15 – Результати тестування метрики

Для тестування були узяті різні значення для даних трьох складових змінних формули, які були використанні одна з одною для отримання результуючого значення. Головною метою було досягнення початкового ліміту в 50000 при умовах, що середнє навантаження процесору починаючи з 90 відсотків може повертати такий або більший заданий ліміт. З представлених рисунків 3.2 – 3.4 наочно можна побачити, що результат відповідає очікуваному.

Так, наприклад, при навантаженні процесору 90 або більше процентів та з середнім часом обробки нових подій у системі більше ніж 11 мілісекунд пороговий ліміт буде досягнутий. Також, якщо в системі кількість неопрацьованих подій, тобто це та кількість, що залишається у Kafka topic для опрацювання більше за або дорівнює 50000, то ця кількість і повертається як результат метрики. І задача полягала саме в тому, аби отримувати з потрібних для розрахунку змінних результат, який би при бажаних значеннях цих змінних відображав таку ж саме інформацію еквівалентну до показника неопрацьованих подій у Kafka topic.

Дивлячись на результат, можна прийти до висновку, що поставлена задача була виконана та потрібна формула для розрахунку комбінованої метрики була розроблена та успішно пройшла тестування.

3.5 Проблеми та рекомендації до використання

Розроблена клієнт-серверна система обслуговування за виробленими рекомендаціями підтвердила свою життєздатність та можливість бути використаною для вирішення конкретних бізнес задач, а тому і самі рекомендації також були правильні та доцільні. Була розроблена спеціальна формула для розрахунку метрики, що також довела свою можливість бути використаною на практиці. Проте у ході дослідження не було використано на практиці розроблена формула та сама метрика для того, для чого вона була і задумана. Для її винаходження було витрачено багато часу та зусиль і головною метою є її використання при пікових навантаженнях, аби було чітко зрозуміло, що є наразі проблема у системі, бо надто велика кількість запитів і просто не має можливості їх усі опрацювати швидко. А тому потрібно мати показник, що б відображав таку проблему і застосовуючи цей показник невідкладно вдаватися до певних дій.

У ході проведення дослідження даний показник використовувався на платформі Amazon з сервісом CloudWatch. Даний сервіс дозволяє досить зручно взаємодіяти з іншими сервісами на платформі. І задумка була у тому, що маючи мікросервісну архітектуру, і маючи даний показник розгортати ще один або декілька таких сервісів для забезпечення стабільності системи і боротьби з піковими навантаженнями. А коли навантаження спаде і даний показник покаже певне граничне значення, то додаткові сервіси можна зупинити, так як вони вже більше не потрібні наразі.

Проблема полягала у тому, аби налаштувати таку поведінку системи, а це вже дуже схоже на те, як проходять процеси у великих компаніях. І налаштування сервісів від Amazon таких як S3, RDS, EC2, CloudWatch та інших вимагає не тільки достатньо великих знань у сфері devops, але й ще багато часу та грошового забезпечення. Проте частина даного процесу все ж таки була показана у дослідженні.

Рекомендацією по використанню є застосування виробленої метрики на практиці, як це описано висче разом з клієнт-серверною системою обслуговування з мікросервісною архітектурою. Як наслідок, при правильному налаштуванні усіх сервісів можна отримати дійсно надійну систему, яка динамічно реагуватиме на навантаження на неї. Бо з проведених тестувань наочно можна було побачити, що система є доволі відмовостійкою, бо запити йдуть на різні сервіси, а не на один, як це було б при монолітній системі, а тому кількість опрацьованих запитів буде висче набагато та сам час на їх опрацювання буде мінімальним знову ж таки, порівнюючи з монолітною системою, де усі запити проходять через один й той самий шлях, що представляє собою один сервер для опрацювання усіх подій у системі.

Потенціал та можливість використання результатів даного дослідження є досить великим, що показує його актуальність на даний час.

ВИСНОВКИ

В результаті проведення даного дослідження було порушено багато питань. Ці питання стосувалися сучасних інформаційних технологій для розробки клієнт-серверних систем обслуговування. Сучасних архітектур для використання у розробці таких систем. Порівняння усім відомої монолітної архітектури з мікросервісною та сервіс-орієнтованою архітектурами, де при поставленій задачі дослідження перевага була віддана мікросервісній архітектурі. Описано їх сильні та слабкі сторони, виходячи з яких і були вироблені рекомендації з використання.

Використовуючи вироблені рекомендації, які також включали в себе опис сучасних інформаційних технологій, була поставлена задача на практиці спроєктувати клієнт-серверну систему обслуговування з підбору комп'ютерних ігор відповідно до вимог користувача та описати процес реалізації даної задачі. В результаті проведено аналіз, де показано переваги розробленої системи відповідно до вироблених рекомендації. Розроблено формулу комбінованої метрики для використання її у роботі з даною системою разом. Описано як саме можна її застосувати і як було її використано для проведення аналізу розробленої системи.

Достатньо важливим етапом будь-якої роботи, а в даному випадку дослідження є розуміння проблем, які були виявлені у ході роботи, детальний опис для подальшого їх опрацювання. Це допоможе у майбутніх дослідженнях, бо буде відразу зрозуміло з чим потрібно буде розбиратися та працювати.

Розроблено рекомендації до використання, що також є важливим етапом, якщо дане дослідження буде розглянуто як підґрунтя для подальших досліджень за даним напрямком.

Як результат, можна стверджувати, що вироблені рекомендації повністю себе виправдали і можуть бути використані на практиці, як це описано у підрозділі 3.5.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Hugo F. O. R. Practical Event-Driven Microservices Architecture: Building Sustainable and Highly Scalable Event-Driven Microservices, 2021. 32-34 p.

2. Hugo F. O. R. Practical Event-Driven Microservices Architecture: Building Sustainable and Highly Scalable Event-Driven Microservices, 2021. 36-37 p.

3. Advantages and Disadvantages of SQL: Complete Breakdown https://www.almabetter.com/bytes/articles/advantages-and-disadvantages-of-sql (дата звернення 20.03.2024).

4. Advantages of microservices and disadvantages to know https://www.atlassian.com/microservices/cloud-computing/advantages-of-microservice (дата звернення 22.03.2024).

5. Kafka 3.7 Official source documentation https://kafka.apache.org/documentation/#gettingStarted (дата звернення 28.03.2024).

6. Neha Narkhede, Gwen Shapira, Todd Palino Kafka - The Definitive Guide, 2019. 28 p.

7. Gwen Shapira, Todd Palino, Rajini Sivaram, Krit Petty Kafka: The Definitive Guide: Real-Time Data and Stream Processing at Scale 2021. 54 p.

8. Hugo F. O. R. Practical Event-Driven Microservices Architecture: Building Sustainable and Highly Scalable Event-Driven Microservices, 2021. 279-280 p.

9. Hugo F. O. R. Practical Event-Driven Microservices Architecture: Building Sustainable and Highly Scalable Event-Driven Microservices, 2021. 281 p.

10. Service Registration and Discovery https://spring.io/guides/gs/service-registration-and-discovery (дата звернення 04.05.2024).

11. Get started with microservices using Spring Boot https://medium.com/ms-club-of-sliit/lets-build-a-microservice-with-spring-boot-faf39b968857 (дата звернення 05.05.2024)

12. Відкрита JS бібліотека React https://web-developer.in.ua/assets/articles/react/react-js-library/react-js-library.html (дата звернення 08.05.2024)

13. The main principles or features of React https://legacy.reactjs.org/docs/design-principles.html (дата звернення 11.05.2024)

14. Getting Started with Redux https://redux.js.org/introduction/getting-started (дата звернення 12.05.2024)

15. Usage Guides https://redux.js.org/usage/ (дата звернення 13.05.2024)

16. Writing Logic with Thunks https://redux.js.org/usage/writing-logic-thunks (дата звернення 13.05.2024)

17. Why use Redux-Thunk https://redux.js.org/usage/writing-logic-thunks#why-use-thunks (дата звернення 14.05.2024)

18. Intrac for civil society. Monitoring, 2021. 6 p.

19. Pratt, B and Boyden, J (eds.) The Field Directors' Handbook: An Oxfam Manual for Development Workers. Oxford University Press, Oxford, 2019. 241 p.

20. Simister, N (2000). Laying the Foundations: The role of data collection in the monitoring systems of development NGOs. Occasional paper 01/00. Bath, Centre for Development Studies, University of Bath, 2020 152 p.

21. By Andreas Wittig and Michael Wittig Amazon Web Services in Action, 2018. 383 p.

22. Amazon CloudWatch https://aws.amazon.com/cloudwatch/ (дата звернення 16.05.2024)

23. What is a DDoS attack https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/ (дата звернення 17.05.2024)

24. By Stephen Cole, Sara Perrott AWS Certified SysOps Administrator Official Study Guide: Associate Exam First Edition, 2017. 84 p.

25. What is a DDoS attack https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/ (дата звернення 19.05.2024)

26. Методичні вказівки до організації виконання та захисту кваліфікаційної роботи на здобуття другого (магістерського) рівня вищої освіти спеціальності 122 Комп’ютерні науки, освітньо-наукова програма «Системне проєктування» / Упорядники: І.В. Гребеннік, Н.І. Калита, І.М. Рябченко, З.А Імангулова, О.Б. Колесник, М.Ю. Вишняк. Харків: ХНУРЕ, 2021. 54 с.

27. JB. Clarion, EZ-Wheel, J. François, I. Grebennik, R. Dupas A simulated annealing approach for optimizing layout design of reconfigurable manufacturing system based on the workstation properties // Proc. 10th IFAC Conference on manufacturing modelling, management and control – Nantes, France, 22 – 24 June 2022, pp. 1657 – 1662.