Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____Комп'ютерних наук_____
(повна назва)

Кафедра_____Програмної інженерії_____
(повна назва)

# АТЕСТАЦІЙНА РОБОТА
# Пояснювальна записка

рівень вищої освіти – другий (магістерський)

Дослідження технологій та методів обміну даних в корпоративних системах за допомогою брокерів повідомлень
(тема)

Виконав: студент 2 курсу, групи ПЗСм – 18 – 1___

_____Апухтін Владислав Геннадійович_____
(прізвище, ініціали)

спеціальності 121– Інженерія програмного забезпечення
(код і повна назва спеціальності)

_____Освітньо – професійної програми_____
(тип програми)

*Освітньо – професійної програми*
*Програмне забезпечення систем*

Керівник _____проф. Дудар З.В._____
Ініціали_____
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф. _____ З.В.Дудар

2019 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук_____

Кафедра_Програмної інженерії_____

Рівень вищої освіти – другий (магістерський)

Спеціальність _121 – Інженерія програмного забезпечення_____
(код і повна назва)

Тип програми освітньо – професійна програма

Освітня програма Програмне забезпечення систем_____

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

# ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові _Апухтіну Владиславу Геннадійовичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи _ Дослідження технологій та методів обміну даних в корпоративних системах за допомогою брокерів повідомлень ___
затверджена наказом університету від "___" _____20 ___ р № _____
заповнюється вручну після отримання наказу
2. Термін подання студентом роботи до екзаменаційної комісії
10 грудня 2019 р.
3. Вихідні дані до роботи мінімальні вимоги до функціональності програми, загальні вимоги до архітектури системи, результати математичних розрахунків, результати експериментів._____
4. Перелік питань, що потрібно опрацювати в роботі *мета роботи, аналіз проблемної галузі і постановка задачі, аналіз підходів розробки корпоративних застосувань, аналіз впровадження брокерів повідомлень, опис роботи системи до та після впровадження брокерів повідомлень, аналіз результатів впровадження брокерів повідомлень.* _____

5 Консультанти розділів роботи

| Найменування розділу | Консультант (посада, прізвище, ім’я, по батькові) | Позначка консультанта про виконання розділу | |
|---|---|---|---|
| | | підпис | дата |
| | | | |

## КАЛЕНДАРНИЙ ПЛАН

| № | Назва етапів роботи | Терміни виконання етапів роботи | Примітка * |
|---|---|---|---|
| 1. | Аналіз предметної галузі | 16 вересня 2019р. | |
| 2. | Формулювання проблеми | 24 вересня 2019р | |
| 3. | Постановка задачі | 29 вересня 2019р | |
| 4. | Проектування архітектури системи | 10 жовтня 2019р | |
| 5. | Опис досліджуваної моделі та аналіз | 20 листопада 2019р | |
| 6. | Підготовка презентації та доповіді | 28 листопада 2019р. | |
| 7. | Нормоконтроль, рецензування | 09 грудня 2019р. | |
| 8. | Занесення диплома в електронний архів | 11 грудня 2019р. | |
| 9. | Допуск до захисту у зав. кафедри | 13 грудня 2019р. | |
| * заповнюється вручну після виконання чергового пункту | | | |

Дата видачі завдання __2__ _вересня__2019 р.

Студент _____
(підпис)

Керівник роботи _____ __проф. Петров П.П_____
(підпис) (посада, прізвище, ініціали)

# РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить 89с., 17 рис., 6 таблиць, 22 джерела.

ОБМІН ПОВІДОМЛЕННЯМИ, АСИНХРОННІ ОПЕРАЦІЇ, ЗАКОН АМДАЛА, ПРОТОКОЛ, ІНТЕГРАЦІЯ ЗАСТОСУНКІВ.

У документі розглядаються основні протоколи брокерів повідомлень і шаблони, які використовуються для вирішення технічних проблем. Робота ознайомлює читача з типовими проблемами розробки програмного забезпечення, які можуть привести до концептуальних порушень та невідповідності нефункціональних вимог до програмного забезпечення, а саме: проблема високої зв’язності компонентів, втрата продуктивності через використання синхронних операцій і проблема інтеграції компонентів системи. Рішення, які з'являються в результаті застосування обміну повідомленнями, охоплюють кожну вищезазначену проблему, зокрема, встановлюють зв'язок між слабо зв'язаними застосунками, досліджують підвищення продуктивності додатків відповідно до закону Амдала і застосовують архітектурний шаблон Read Model, який використовує можливості брокерів повідомлень.

MESSAGING, ASYNCHRONOUS OPERATIONS, AMDAHL’S LAW, PROTOCOL, APPLICATION INTEGRATION.

The paper covers main message broker protocols and the patterns which are used to manage technical problems. The article provides the reader with typical software engineering design issues which might lead to conceptual violations, software non – functional requirements nonconformity, which is high coupling problem, synchronous operation performance loss, and system constituent integration problem. The solutions which come as a result of applying messaging are brought covering each problem above, in particular establishing the communication between loosely coupled applications, researching the application performance speedup with the Amdahl’s law and applying Read Model architectural pattern leveraging message brokers capabilities.

**ЗМІСТ**

# ВСТУП

Корпоративні програми – це програми, які відображають, маніпулюють та зберігають великі обсяги даних і підтримують або автоматизують бізнес – процеси з цими даними [1]. Такі програми зазвичай складаються з окремих модулів, сервісів або навіть кластерів сервісів, які по – різному виконують бізнес – задачі. Беручи до уваги складність архітектури, існує потреба в підтримці інтеграції між компонентами системи. Ідея брокерів повідомлень з'явилася як адекватне і гнучке рішення для цієї мети.

Брокери повідомлень є гнучким інструментом для вирішення типових проблем комп'ютерної системи. У розділі «Аналіз предметної галузі» описуються основні концепції обміну повідомленнями та розглядаються документи розглянутої наукової області. У розділі «Формулювання проблеми» представлені проблеми, які можуть виникнути в процесі розробки програми. Їх практичні рішення представлені в розділі «Опис проведення теоретичних і практичних досліджень», які спрямовані на вирішення проблем розподілених комунікаційних послуг на додаток до централізації розподілених даних за допомогою можливостей обміну повідомленнями.

# 1 АНАЛІЗ ПРОБЛЕМНОЇ ГАЛУЗІ

## 1.1 Аналіз предметної області

Брокер повідомлень – це проміжне програмне забезпечення, яке відповідає за відправку повідомлень в форматі, визначеному між програмними компонентами, з використанням певного протоколу, наприклад: MQTT (Message Queuing Telemetry Transport), STOMP (Streaming Text Oriented Messaging Protocol), AMQP (Advanced Message Queuing Protocol) [2].

Системи обміну повідомленнями зазвичай передбачають введення посередника між двома системами, які обмінюються даними, щоб додатково відокремити відправника від одержувача або одержувачів. При цьому система обміну повідомленнями дозволяє відправнику відсилати повідомлення, не знаючи про одержувача, активний він чи яка їх кількість існує [3]. Існують різні методи (іншими словами – шаблони) для досягнення цієї мети, а саме:

- Publisher – subscriber;
- Exclusive point – to – point;
- RPC (Remote procedure call).

Повідомлення – це обсяг даних, яким обмінюються відправник та отримувач. Зазвичай повідомлення обмежені за розміром. Крім того, існують різні режими публікації повідомлень:

- точно один раз, тобто повідомлення може бути доставлено тільки один раз;
- принаймні один раз, тобто повідомлення може бути доставлено кілька разів з дублюванням.

Повідомлення можуть бути постійними на непостійними, що означає, що після збою в роботі брокера повідомлень існує технічна можливість пересилати повідомлення чи ні відповідно.

Повідомлення може містити властивості і заголовки. Властивості повідомлення – це конкретні характеристики повідомлення, наприклад, пріоритет або тип вмісту.

Заголовки – це характеристики повідомлення, які можна гнучко змінювати. На відміну від властивостей повідомлення, заголовки можуть враховуватися брокером для дозволу одержувача.

Шаблон Publisher – subscriber об'єднує поняття відправника, одержувача, повідомлення і теми.

Видавець і одержувач – це вузли зв'язку, які існують у відношенні «багато до багатьох»; вузли відповідають за відправлення та одержання повідомлень відповідно, використовуючи теми обміну.

Теми – це строкові значення, які використовуються брокером повідомлень для визначення того, який одержувач призначений для отримання надісланого повідомлення. Видавець відправляє повідомлення посереднику з темою обміну, посередник повідомлень вирішує, які споживачі можуть отримати повідомлення.

Користувач може вказати ключ теми або в якості прямого значення імені теми, або відповідно до шаблону.

Передбачається, що одне повідомлення може бути доставлено до багатьох одержувачів.

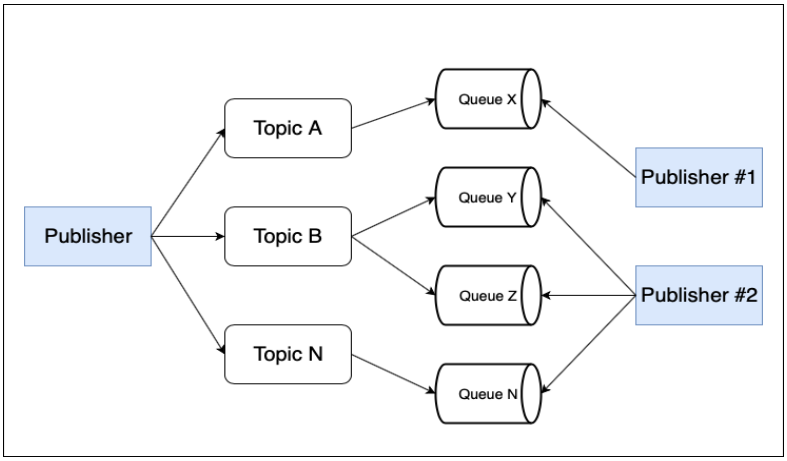


Рисунок 1 – Типова інфраструктура шаблону Publisher – Subscriber

Шаблон Exclusive point – to – point використовуються в тому випадку, коли видавцеві необхідно відправити повідомлення тільки одному одержувачу повідомлення. Незважаючи на те, що кілька споживачів можуть прослуховувати в черзі одне й те саме повідомлення, тільки один з цих потоків – одержувачів отримає повідомлення [4].

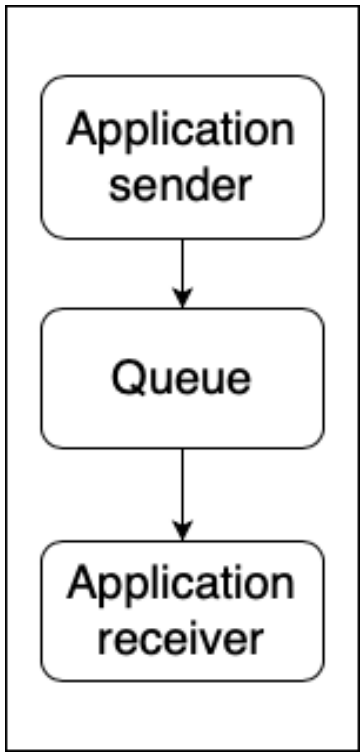


Рисунок 2 – Типова інфраструктура шаблону Exclusive point – to – point

Шаблон RPC (Remote Procedure Call) є шаблону exclusive point – to – point. На відміну від іншого згаданого шаблона обміну повідомленнями існує двосторонній зв'язок між відправником і отримувачем. Для цього зазвичай створюються два різних канали, зокрема, для повідомлень запиту і відповіді, які обробляються як один запит з використанням ідентифікатора – ідентифікатора кореляції.

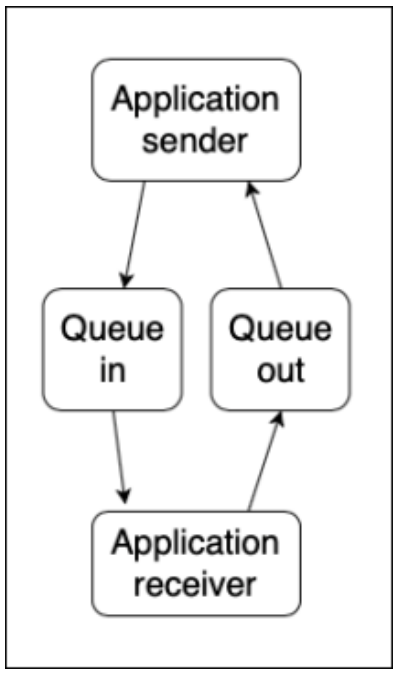


Рисунок 3 – Типова інфраструктура шаблону RPC

Протокол MQTT (розшифровується як Message Queuing Telemetry Transport) – це легковійний протокол обміну повідомленнями в реальному часі, що використовує шаблон Publisher – Subscriber для доставки повідомлень в бінарному форматі з використанням базового протоколу TCP / IP.

Легкість протоколу досягається за рахунок низького розміру пакета, який варіюється від 1 байта до 256 мегабайт. Крім того, реалізації призначені для підтримки низького енергоспоживання. Протокол також розроблений таким чином, щоб реалізація враховувала обмежену пропускну здатність мережі та високу затримку.

З урахуванням вищевикладеного, протокол зазвичай використовується для зв'язку між смартфонами, пристроями Інтернету речей (IoT), маршрутизаторами і т. д.

Реалізації протоколу MQTT включають, але не обмежуються ними, Eclipse Mosquitto, eqmtt, VerneMQ.

Протокол STOMP (Streaming Text Oriented Messaging Protocol) – протокол потокової передачі текстових повідомлень, як випливає з його назви, є протоколом

обміну текстовими повідомленнями. Протокол використовує патерн publisher – subscriber, і є простий у використанні. Протокол має безліч реалізацій, наприклад, ActiveMQ, RabbitMQ, Stampy.

Протокол AMQP (Advanced Message Queuing Protocol) – це бінарний протокол рівня додатка, який використовує базовий TCP (Transmission Control Protocol) для асинхронної потокової передачі повідомлень по мережі в бінарному форматі.

У порівнянні з MQTT і STOMP, AMQP надає широкий спектр можливостей для додатків.

Передані дані можуть бути зашифровані з використанням TLS (Transport Layer Security). Протокол підтримує узгодженість шляхом реалізації функцій підтвердження доставки, атомарности в одній черзі, хоча і без повної підтримки транзакцій (наприклад, реалізація протоколу RabbitMQ забезпечує підтримку транзакцій при публікації, підтвердження, відхилення повідомлень при створенні / видаленні ресурсів, таких як черги і біржі не реалізовані, щоб бути транзакційними). AMQP також гарантує, що повідомлення, опубліковані в певному порядку, будуть доставлятися споживачам в строго однаковому порядку, незважаючи на кількість споживачів.

AMQP підтримує всі режими публікації повідомлень, які згадувалися раніше, а також шаблони.

Беручи до уваги все перераховане вище, цей протокол переважно використовувати в корпоративних додатках.

Типова реалізація – RabbitMQ.

## 1.2 Аналіз стану проблеми

Брокери повідомлень активно використовуються як в середовищі єдиної комп'ютерної системи, так і в розподіленої, що реалізує різні шаблони. Jun – Young

Byun у виданні [5] запропонував хмарну екосистему з базовою схемою publisher – subscriber. Автором було запропонована наступна стратегія роботи с брокером повідомлень у корпоративних застосуваннях. Замість одного екземпляру брокеру повідомлень доступного в системі, формується мережа інфраструктурних застосувань, кожен з яких виступає «обгорткою» брокеру повідомлень. Відправник має можливість публікувати повідомлення до кожної частини мережі брокерів – обгорток, отримувач кожного іншого вузлу гарантовано отримає повідомлення, особливо, підписавшись на тему каналу с того самого вузлу. Такого роду мережа має у собі центральний канал усіх повідомлень, який виступає у ролі роутера повідомлень на аплікаційному рівні.

При вищеописаному дизайні системи, система є більш захищеною від системних збоїв, а саме втрати інформації повідомлень у разі зупинки єдиного брокеру повідомлень. Відключений вузол обгорток автоматично провокує залежні від нього брокери – обгортки накопичувати повідомлення до того часу, коли зв’язок між складовими мережі брокерів буде відновлено.

З дослідження авторів, розгорнута інфраструктура також здатна швидко обробляти повідомлення, підвищуючи швидкість відгуку повідомлень в декількох регіонах.

Однак, незважаючи на створення великої служби обміну повідомленнями, інтеграція її у мікросервісну архітектуру призводить до того, що інформація буде розділена, а доступ до наскрізних даних мав на увазі додаткові витрати ресурсів.

Інше дослідження [6] було взяте до уваги. Автори роботи зробили аналіз існуючої системи брокерів повідомлень для пристроїв IOT (Internet of things) під назвою The Arrowhead Framework. Згідно з дослідженням, було розгорнута екосистема з декількох вузлів для тестування латентності мережі при передачі повідомлень між вузлами. Авторами було наведено детальний звіт затримок мережі. Комбінація інфраструктури Grizzly для HTTP – сервера і сервера Jersey була взята в досідження. Однак більша частина роботи залежала більшою мірою від програмних прикладних рішень Java, особливо в залежності від обмежень пулу потоків Java, а не від перенесення паралельних обчислень на екземпляри брокера

повідомлень. Тому наскрізні вимірювання затримки могли мати різні результати. Також були відсутні дані про виграш у часі від виконання асинхронних операцій брокером повідомлень.

Проаналізувавши роботи та предметну галузь, з ціллю зрозуміти доцільність введення брокерів повідомлень у систему, необхідно вивчити наступні проблеми: зв'язність компонентів, недолік синхронних операцій і складність інтеграції нових компонентів.

Проблема високої зв'язності. Ствердження «Висока згуртованість, низька зв'язність» є принципом General Responsibility Assignment Software Patterns. Чим нижче зв'язність між складовими системи, тим більше система вважається роз'єднаною. З точки зору корпоративних додатків, особливість представлено в наступному: кодові залежності, виконання віддалених запитів за допомогою протоколів, а саме HTTP [S], TCP, SOAP та інших.

Розглянемо компонент A, який посилається на компонент B, який посилається на компонент C і A, і слід виконати наступний ланцюжок викликів, щоб відправити довільне повідомлення компоненту Z. Така приблизна структура є типовим прикладом системи з високим ступенем зв'язності.

Проблема полягає в тому, як уникнути залежностей коду.

Синхронні операції. Синхронна операція A – це операція, яка виконується програмою, яка не дозволяє перейти до виконання програми до операції B, поки не буде виконана A. Іншими словами, синхронна операція блокує поточний потік. Прикладами є операції введення / виводу, HTTP [S] запит.

Асинхронна операція – це операція, яка не блокує поточний потік.

Роблячи тривалі операції асинхронними, додаток можна вважати таким, що швидко оброблює вхідну інформацію. Такі додатки використовують паралелізм або паралельні обчислення.

Паралельні обчислення – це тип обчислень, при якому операції можуть виконуватися одночасно (паралельно).

Що стосується корпоративних додатків, прикладом тривалої синхронної операції, яка займає відносно значний час, є відправка електронного листа.

Проблема полягає в тому, як зробити життєздатні асинхронні операції незалежно від того, чи належать компоненти, що містять тривалі операії.

Інтеграції нових компонентів. Проблема інтеграція нового компонента частково збігається з проблемою зв’язності компонентів. Корпоративні додатки рідкою мірою складаються з одного виконуваного екземпляра. Навпаки, конкретні автономні функції, які не пов'язані з основними, зазвичай розробляються в іншому екземплярі програми. В цьому випадку виникає архітектурна проблема: як новий компонент повинен обробляти події різного роду з основного додатку і навпаки?

Інша проблема інтеграції полягає в перенесенні даних з однієї бази даних, ймовірно, застарілої і не підтримуваної в іншу. Проблема виникає в додатках реального часу, які повинні обробляти поточні дані і мати можливість об'єднувати старі в реальному часі.

Обмін повідомленнями може бути використаний як інструмент для вирішення вищезазначених проблем.

## 1.3 Постановка задачі

У рамках магістерської роботи необхідно методом наукового дослідження та вивірення оцінити переваги використання брокерів повідомлень у корпоративних системах. Упродовж роботи необхідно:

- проаналізувати типові архітектурні рішення корпоративних застосувань;
- проаналізувати можливості впровадження брокерів повідомлень у корпоративні застосування;
- розробити архітектуру модельованої розподіленої системи без використання брокеру повідомлень;
- кількісно зробити заміри роботи системи за умови роботи користувача та бізнес процесу;

- математично передбачити приріст продуктивності системи та її компонентів;
- інтегрувати брокер повідомлень до розподіленої системи;
- кількісно зробити заміри роботи системи за умови роботи після інтеграції брокеру повідомлень;
- проаналізувати результати.

Розроблена система повинна бути реалізована таким чином, щоб її архітектурна модель могла ефективно зобразити проблеми, описані в частині 2.2.

Для реалізації програмної системи розробити екосистему мікросервісної архітектури на базі технології Java та бібліотеці Spring Cloud. У якості брокеру повідомлень системи використовувати RabbitMQ. Додатково, необхідно розробити емулятор застосування, яке повинно бути інтегровано в систему за допомогою брокеру повідомлень.

Система для розробки повинна формувати веб – застосунок, який інформує користувача нотифікацією про виникнення подій певного типу всередині системи. Нотифікації мають бути двох типів: ті, що можуть бути створеними миттєво, та такими, які потребують певний час на створення. Події, нотифікації та деталі виникнення подій необхідні знаходитися у різних мікросервісах. Таким чином можна відтворити проблему розподіленої даних у гетерогенній системі. Виникнення подій можливе з кожного сервісу системи, а генерація подій повинна бути тривалою за часом операцією, таким чином можна відтворити проблему синхронного виконання коду. Деталі нотифікації одного з сервісів системи повинні також мати джерело даних у зовнішньому, незалежному застосуванні, яке повинне працювати паралельно, таким чином можна відтворити проблему інтеграції застосувань.

# 2 ОПИС ПРОВЕДЕННЯ ТЕОРЕТИЧНІХ І ЕКСПЕРИМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

## 2.1 Вибір архітектурного рішення для корпоративного застосування

Сучасні програмні рішення рідко складаютсья с одного застосунку, а навпаки, утворюють розподіленні мережі [7].

Серед основних архітектурних підходів корпоративних застосуваннь можна виділити наступні:

- монолітні застосунки;
- розподілені мікросервісні застосунки.

Монолітний застосунок використовує перевагу простої багатокомпонентної комунікації в рамках проекту. Бізнес – логіка додатка, зберігання даних і виконання транзакцій, виконуються в рамках одного середовища, що, в свою чергу, знижує витрати на підтримку системи. Крім того, монолітні додатки можуть бути дешево масштабовані горизонтально, що означає додавання нового екземпляра додатка до балансувальника навантаження. Крім того, в такому додатку легко додати нові функціональні можливості, щоб побачити негайні результати завдяки узгодженості коду.

Зростання монолітного застосування в кінцевому підсумку призводить до ускладнення розробки нових функцій і доставки продукту. Збільшується час збирання, ланцюгові залежності компонентів збільшують підтримку додатків; Вихід з ладу компонента тягне за собою втрату працездатності всієї системи. Крім того, розробка програми обмежена технологічним стеком, який був обраний на початковому етапі життєвого циклу проекту.

Ще один спосіб створення надійного додатки – це розподілені хмарні застосунки. Такі застосування можуть мати мікросервісну архітектуру чи безсерверну (serverless) архітектуру. Цей термін є варіантом архітектурного стилю сервіс – орієнтованої архітектури (SOA), який структурує додаток як набір слабо пов'язаних сервісів. В архітектурі мікросервісов служби повинні бути деталізовані,

а протоколи – легкими. Розробка додатків в такий спосіб роз'єднує цілісність додатків. Оскільки додаток може складатися з декількох компонентів, які можуть містити абсолютно різні цілі і бізнес – логіку, виділення його в окремі компоненти робить заявлену архітектуру більш гнучкою. Оскільки кілька служб мають загальний відкритий інтерфейс програми (API), межі реалізації стираються разом зі стеком технологій. Наприклад, один сервіс може бути реалізований з використанням рішень .NET через інтенсивне використання продукту Microsoft Office, в той час як інший сервіс використовує мову програмування Python, що використовує бажані рамки машинного навчання. Поділ різних компонентів додатка призводить до використання можливостей взаємодії, де використовується легкий протокол передачі даних з мінімальною додатковою логікою або без неї. Зокрема, архітектура Microservices пропонує протокол REST на відміну від службової шини Enterpise (ESB), яка може містити істотну бізнес – логіку в механізмі протоколу передачі. Більш того, обговорення архітектури забезпечує рішення стресостійкість, що означає, що додаток може залишатися працездатним в разі збою одного з компонентів з різних причин.

Однак мікросервісная архітектура має свої недоліки. Інтенсивний зв'язок між службами може уповільнити загальну реакцію відгуку екземплярів системи. Крім того, з монолітної точки зору, розробники програмного забезпечення повинні набагато краще справлятися з розгортанням, тестуванням і моніторингом додатки, оскільки розподілена система вимагає витрат ресурсів для підтримки системи. Нарешті, виконання транзакцій баз даних є складним в реалізації і може привести до додаткових витрат системних ресурсів.

## 2.2 Методи впровадження брокеру повідомлень

Брокери повідомлень являють собою стороннє програмне забезпечення, що складається з клієнтської та серверної складової.

Серверна складова відповідальна за інфраструктурну обробку функціональних можливостей брокерів повідомлень. Серед основних можна виділити наступні:

- підтримка довговічності повідомлень у разі помилок, збоїв чи інших причин, за якої брокер може не працювати;
- правила доставки повідомлень від адресанта до адресата, згідно з конфігураціями підписки на теми;
- час життя повідомлень та обробка недоставлених повідомлень;
- журналювання та моніторинг серверу брокеру.

У розподілених системах сервер брокеру повідомлень може буди розгорнутий у локальній мережі кластеру, буди наданий сторонніми хмарними системами. Більш того, деякі брокери повідомлень, наприклад Apache ActiveMQ, дають можливість використовувати сервер у вбудованому (embedded) режимі – у такому випадку, брокер працює у процесі залежного від нього застосунку, у якості бібліотеки.

Друга основна складова брокерів повідомлень є клієнт. За допомогою його, залежні екземпляри застосувань мають технічну можливість відправляти повідомлення до обраної теми для брокеру, що розташований у якості вузлу мережі, за допомогою одного з підтримуваних протоколів, наприклад AMQP. Клієнт поставляється видавником брокеру повідомлень на бажану для розробників платформу.

Таким чином, брокери повідомлень можуть використовуватися для налагодження роботи складових програмної системи як всередині екземпляру системи, так і між складовими системи.

Розглянемо теоретичну можливість використання брокеру повідомлень в межах екземпляру аплікації.

Робота зі зменшенням зв’язності компонентів додатка призводить до звернення до брокера повідомлень замість самих компонентів. Компонент А більше не відповідає за визначення того, які вузли вважаються одержувачами повідомлення, як обробляти розбивку одержувачів, скільки існує одержувачів, чи повинен одержувач прийняти повідомлення чи ні. Замість цього компонент просто поміщає повідомлення в пряме ім'я теми або ім'я теми шаблону, а посередник повідомлень доставляє його компоненту В, С і т. Д. Компоненти – прослуховувачі теми, які можуть підписатися або відписатися від теми, визначають залежність від отримання повідомлення. сам. Збій вузла, що прослуховує повідомлення, змушує посередника повторно доставляти повідомлення стільки разів, скільки налаштовано [8].

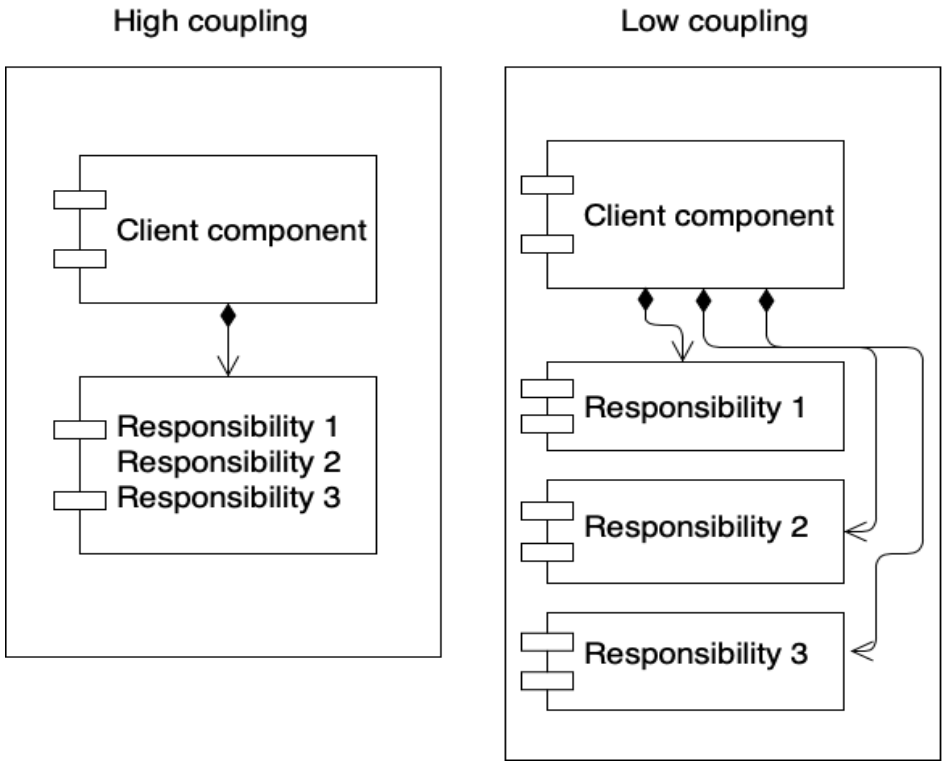


Рисунок 4 – Приклад сильно та слабко зв’язного коду

Брокери повідомлень є кандидатами для внесення асинхронної обробки в додаток. Реалізація прослуховувача каналу автоматично робить обробку асинхронною, іншими словами, клієнт посередника повідомлень привносить паралельну обробку у застосунок. Більш того, брокер – клієнт повідомлень знижує накладні витрати на реалізацію асинхронної обробки через кілька екземплярів з шаблоном publisher – subscriber.

Перевага паралелізму задач по типу завдання можна оцінити наступними способами:

- емпіричним методом;
- розрахувати за законом Амдала;
- розрахувати за законом Густавсона – Барсиса.

Емпіричний метод оцінки, на основі попередньої систематизації, формує більш узагальнене уявлення щодо приросту продуктивності. Саме тому, експериментальна частина роботи буде сфокусована на наступних законах.

Закон Амдала описує зв'язок між прискоренням і робочим навантаженням, яку можна очікувати від системи або компонента. Згідно з законом, представленим у 1967 році, приріст продуктивності системи, пов’язанний з введеням паралельних обчислень, обмежений сумарною кількістю виконання інструкцій послідовно.

Закон представлений в наступній формулі:

$$S = \frac{1}{(1-p) + \frac{p}{s}} \quad (1)$$

де S – теоретичне прискорення виконання завдання, p – частка часу виконання, частину якої, отримавши вигоду з поліпшених ресурсів, спочатку займала, s – прискорення тієї частини завдання, в якій використовуються поліпшені системні ресурси,.

Альтернативним варіантом якісної оцінки поліпшення продуктивності є законом Густавсона – Барсиса. Закон дозволяє обчислити досягаємий рівень

паралелізму базуючись на кількості потоків для обчислення та процент послідовних обчислень.

На відміну від попереднього закону, оцінюється об’єм вимірюваної задачі за фіксований проміжок часу, а не за рахунок зменшення витраченого часу на фіксований об’єм задачі.

Закон представлений формулою:

$$S_n = s + (1 - s)n \tag{2}$$

де s – доля послідовних обчислень у системі, n = кількість потоків у системі.

Брокери повідомлень корисні для інтеграції нових компонентів в існуючу систему, особливо в додатках з мікросервісною архітектурою. Асинхронна відповідь на існуючі системні бізнес – процеси може бути реалізована шляхом зміни нових компонентів таким чином, щоб нові компоненти програми були підписані на операції маніпулювання даними з основного додатка, щоб підтримувати узгодженість внутрішніх даних або відображати зовнішні зміни у внутрішньому додатку [9].

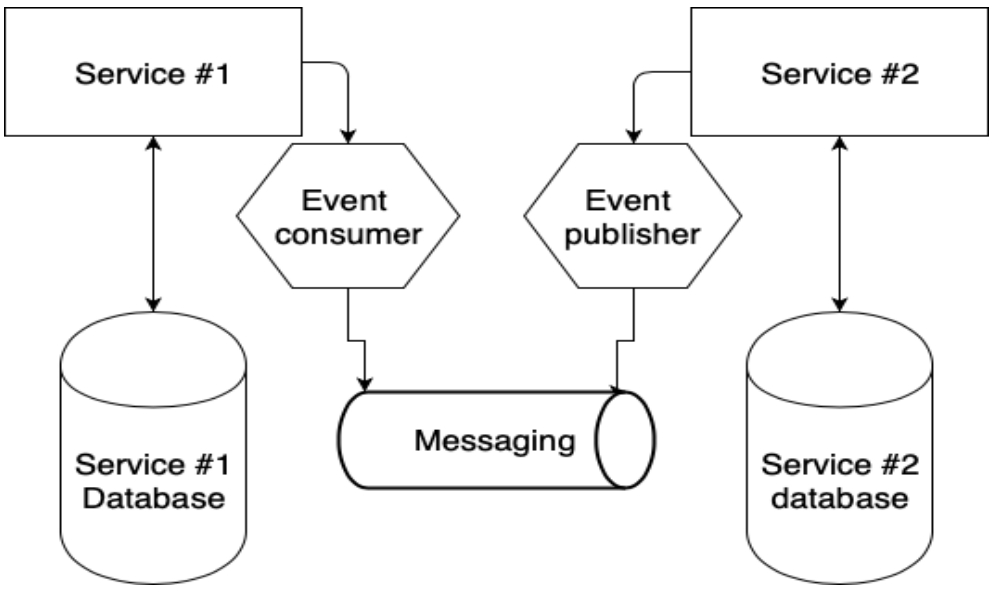


Рисунок 5 – Приклад інтеграції компонентів нової системи до вже існуючої

Докладніший приклад інтеграції нового компонента з обміном повідомленнями полягає в тому, що в прикладної системі, яка містить кілька екземплярів сервісів, поле домену перетинається між декількома сервісами. В цьому випадку дублювання даних стає незбіжним. Обмін повідомленнями може застосовуватися для реалізації моделі Read Model разом з підходом, керованим подіями. У цьому сценарії кожен мікросервіс, що має власну доменну логіку, повинен опублікувати події в разі успішного зміни даних на виділеному каналі. Таким чином, з'являється певна кількість подій, які в кінцевому підсумку можуть бути використані сервісом моделі читання. Цей мікросервіс буде відповідати за зберігання записів бази даних крос – даних, які можуть запитуватися декількома службами одночасно, забезпечуючи кінцеву узгодженість (eventual consistency) в системі, оскільки асинхронна обробка кожної події вимагає часу, яке необхідно доставити для читання служби моделі.

Міграція даних бази даних може бути досягнута в робочому середовищі мікросервіса, що використовує обмін повідомленнями. Кожен сервіс, якому необхідно перенести дані на додаток до вже існуючого, зазвичай здатний створювати дані такого ж типу. Це може стати точкою інтеграції, адаптованого до прослуховувача каналу, що доставить доменні об'єкти із застарілого джерела даних в новий. Для цього необхідно розгорнути додатковий інтеграційний мікросервіс, який буде публікувати дані по команді [10][11].

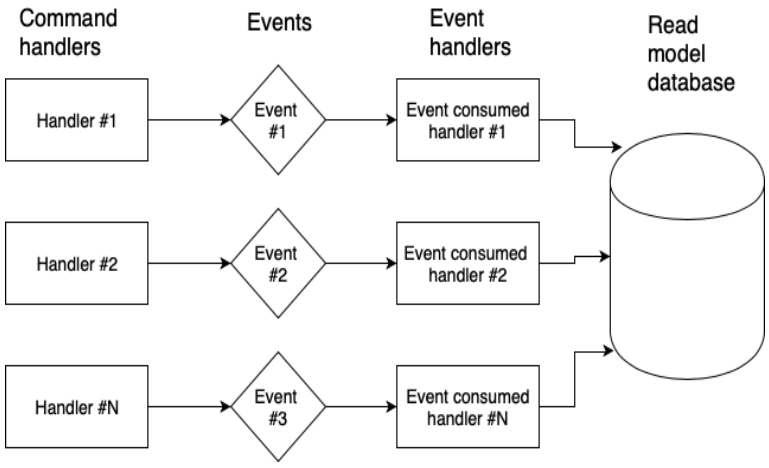


Рисунок 6 – Реалізація системи з моделью Read Model.

## 2.3 Проектування архітектури системи

Архітектура програмного забезпечення — спосіб структурування програмної або обчислювальної системи, абстракція елементів системи на певній фазі її роботи.

Для ефективного дослідження брокерів повідомлень був зроблений аналіз сучасних архітектурних рішень, оцінивши складність рішення та час відведений на виконання роботи, були імплементовані наступні архітектурні підходи.

Обраний архітектурний підхід розробки – мікросервісна архітектура. Виходячи з цього, наступні необхідні інфраструктурні сервіси повинні мати місце.

Сервіс зовнішнього шлюзу (Api – Gateway Service). Цей сервіс є вхідною точкою до системи, розподіляючи навантаження на систему у вигляді HTTPS запитів за допомогою балансера навантаження до функціональних та інших інфраструктурних сервісів. Сервіс буде використовувати серверно – орієнтоване відкриття (server – side discovery) для локалізації знаходження IP адреси інфраструктурного чи функціонального сервісу у локальній мережі оточення. Кількість сервісів: 1.

Сервіс користувача (User service). Цей сервіс акумулює відповідальність за автентифікацію та авторизацію. Повинен мати свою ізольовану базу даних. Кількість екземплярів: 1 – 2. Екземпляри баз даних повинні бути під’єднані до єдиної ізольованої бази даних.

Сервіс веб – додатку (Web service). Сервіс, відповідальність якого – надавати користувацький інтерфейс, який буде виступати у якості споживача публічного інтерфейсу застосунку. Технічно, сервіс буде надсилати запити до сервісу зовнішнього шлюзу та на основі відповідей генерувати відповідний контент. Оскільки мета наукової роботи – кількісно дослідити переваги використання брокерів повідомлень, даний сервіс буде імплементовано з найнижчим пріоритетом. Кількість сервісів: 1. Екземпляр баз даних повинні бути під’єднані до єдиної ізольованої бази даних.

Функціональні сервіси системи – такі сервіси, які відповідальні зі бізнес – логіку застосунку. Нижче наведені функціональні сервіси системи:

Сервіс домену (domain service). Даний сервіс відповідальний за операції створення, обновлення, читання та видалення даних уявної доменної сутності. У реальній системі сервіс міг би бути представлений більш реальним сервісом, залежно від доменної області системи, наприклад сервіс документів, сервіс замовлень, сервіс доставки, сервіс коментарів, сервіс транзакцій тощо. У дослідницькому проекті сервіс буде оперувати сутністю «Сутність». Більш детальний опис доменної області буде зроблений у розділі «Опис роботи досліджуваної моделі без наявності брокеру повідомлень» Екземпляр баз даних повинні бути під’єднані до єдиної ізольованої бази даних.

Інтеграційний сервіс (integration service). Даний сервіс буде відповідальний за інтеграцію з уявною системою, яку необхідно інтегрувати в досліджувано, виходячи з постановки задачі. Цей сервіс буде разом за сервісом нотифікацій буде центральним об’єктом дослідження, адже вони обидва будуть активно використовувати можливості брокерів повідомлень для вирішення задач.

Сервіс нотифікацій (Notification service). Сервіс відповідальний за відправку нотифікацій користувачеві. Технічно, сервіс буде налаштовано на прослуховування каналу подій, на який інші сервіси будуть відправляти повідомлення з відповідними функціональними подіями. Сервіс повинен мати ізольовану базу даних. Кількість одночасно працюючих сервісів: 1 – 3. Екземпляри баз даних повинні бути під’єднані до єдиної ізольованої бази даних.

Розглянемо архітектуру сервісу нотифікацій більш детально. Оскільки центральною функціональністю потенційного застосунку є нотифікації, необхідно розглянути підхід, за якого застосунок може оповіщати клієнтську сторону застосунку щодо нових нотифікацій. Оскільки протокол HTTP розроблявся за шаблоном запит – відповідь між клієнтом та сервером,. тобто рух в одну сторону, для реалізації відправки інформації від серверу до клієнту необхідно залучити спеціальні підходи чи технології. Для швидкого відклику досліджуємого

застосунку можна розглянути один з наступних типів опитування сервісу на наявність нових нотифікацій [12]:

- опитування з інтервалом;
- довгі опитування (long poling);
- веб – сокет канал (web socket).

Довгі опитування – шаблон розробки, за яким клієнт періодично опитує API на наявність нових повідомлень, але при якому з'єднання залишається відкритим до тих пір, коли сервіс поверне відповідь [13]. Іншими словами, замість негайної відповіді з повідомленням щодо відсутності даних, відповідь від сервісу повертається тільки у двох випадках:

- дані знайдені;
- дані не знайдені по проходження виділеного часу на відповідь.

Даний підхід має наступні переваги. Кількість одночасних запитів до веб – сервісу зменшуються, що зменшує навантаження до сервісу та збільшує потенціальну кількість клієнтів, що можуть обслуговуватися у одиницю часу. Ця кількість на кожний екземпляр сервісу обмежена лише кількістю відкритих з'єднань одночасно, що фасилітує «вертикальне» масштабування. З іншої сторони, існують наступні недоліки. Горизонтальне масштабування ускладняється наступним фактором: кожен доданий ексезмпляр веб – сервісу повинен бути прив'язаний до сесії користувача. Також, помилки обробки, пов'язані з мережевими проблемами, потребують більших ресурсів на обробку, як от: зміна мережі, обрив з'єднання, декілька одночасно працюючих клієнтів з одного вузла мережі. Виходячи з вищезгаданого, імплементація даного шаблона потребує більш деталізованого розгляду та проведення дослідження доцільності впровадження, оглядаючи на архітектуру тієї чи іншої системи. В даному дослідженні використання цього шаблону не є доцільним, оскільки є предметом іншого дослідження.

Наступним шаблоном, який можна розглянути щодо використання сервер – клієнтської комунікації є використання протоколу WebSocket. У той час, як попередні розгляну і шаблони використовують протокол HTTP[s], протокол

WebSocket – самостійний протокол, який працює на базі протоколу TCP/IP, який має схожу структуру запитів запитів та відповідей з HTTP[S], але працює двонаправленно, тобто клієнт та сервер мають можливість надсилати дані один одному без зайвих зусиль. Беручи до уваги наведену особливість, недоліки простого опитування та довгого опитування нівелюються. Однак не усі браузери мають підтримку даного протоколу. Більш того, нещодавно протокол було розкритикований через знайдених вразливостей безпеки, пов'язаних з підміною даних з використання прозорих проксі – серверів, що призвело до того, що підтримка протоколу вимкнена у частині браузерів за замовчуванням. Це призводить до ускладнення впровадження протоколу та ставить під сумнів подальше використання протоколу як стандарт розробки корпоративних застосувань. Зважаючи на це, цей протокол не буде взято до розробки веб – сервісу, який досліджується у даній роботі.

Опитування з інтервалом є найпростішим підходом отримання інформації щодо нових нотифікацій с точки зору реалізації. Кожний клієнт застосунку із заданою періодичністю опитує сервер на наявність нових нотифікації з певної точки часу. Якщо такі є – застосунок вертає знайдені у тілі запиту, в іншому разі, у якості альтернативи, можна повертати статус HTTP запиту 304 – Not modified, тобто сигнал щодо того, що ресурс не був змінений на стороні серверу (ресурсом у даному контексті виступають нові нотифікації).

Попри простоту реалізації, підхід перевантажує сервер марними запитами. Однак, цей протокол буде обраний для розробки у веб – сервісу, як компромісне рішення між безпекою та простотою обслуговування.

Окремим вузлом системи буде екземпляр меседж брокеру. Розгорнутий у єдиному екземплярі, буде вважатися, що екземпляр є завжди доступний без можливості виходу його з ладу.

Сервіс читацької моделі (read model service). Даний сервіс створений агрегувати дані крос – сервісного доменного поля. Сервіс повинен мати базу даних у єдиному екземплярі. Кількість одночасних екземплярів сервісу – 1. Сервіс повинен буде налаштований на доступ до брокеру повідомлень. Сервіс буде

відписаний на канали меседж брокеру, по яким буде передаватися повідомлення щодо зміни даних з кожного функціонального сервісу. Передача повідомлень повинна працювати за шаблоном проектування – транзакційний кошик (transactional outbox) – це шаблон для надійної публікації повідомлень без використання розподілених транзакцій. Він складається з двох частин: «Вихідні» та диспетчер повідомлень, які працюють разом для надійного збереження стану і публікації повідомлень [14].

Шаблон вихідних транзакцій використовує первинний рівень персистентності (база даних, SQL або NoSQL) у якості тимчасовий черги повідомлень. Служба, яка відправляє повідомлення, має таблицю бази даних OUTBOX. Як частина транзакції бази даних, яка створює, оновлює і видаляє об'єкти, служба відправляє повідомлення, вставляючи їх в таблицю OUTBOX. Валентність гарантується, тому що створюэться локальна транзакція ACID. У разі бази даних NoSql кожен агрегат, що зберігається у вигляді документа в базі даних, має атрибут, який представляє собою набір повідомлень, які необхідно опублікувати. Коли служба оновлює сутність в базі даних, вона додає повідомлення в цей список. Це атомарному, тому що це робиться за допомогою однієї операції з базою даних. Потім компонент реле (Message Relay) повідомлення зчитує таблицю OUTBOX або колекцію вихідних повідомлень в документі і публікує повідомлення в посереднику повідомлень. Message Relay публікує повідомлення з папки «Вихідні» в брокер повідомлень, відправляючи їх в свій канал повідомлень призначення. Він видаляє ці повідомлення з таблиці OUTBOX. Message Relay може бути побудований як видавець опитування або з використанням шаблону прив'язки журналу транзакцій. Видавець опитування публікує повідомлення, опитуючи вихідні повідомлення в базі даних. Опитування бази даних працює досить добре при невеликому масштабі, однак часто опитування бази даних не масштабується добре. Крім того, цей підхід погано працює з базами даних NoSQl, де шаблон запиту є більш складним. У базі даних NoSQL додаток повинен запитувати документи, що містять агрегати, і це може або не може бути можливо зробити ефективно [15][16].

Комунікація між сервісами всередині локальної мережі буде проводитися з протоколом HTTP.

Графічно архітектура системи буде рисунком нижче.

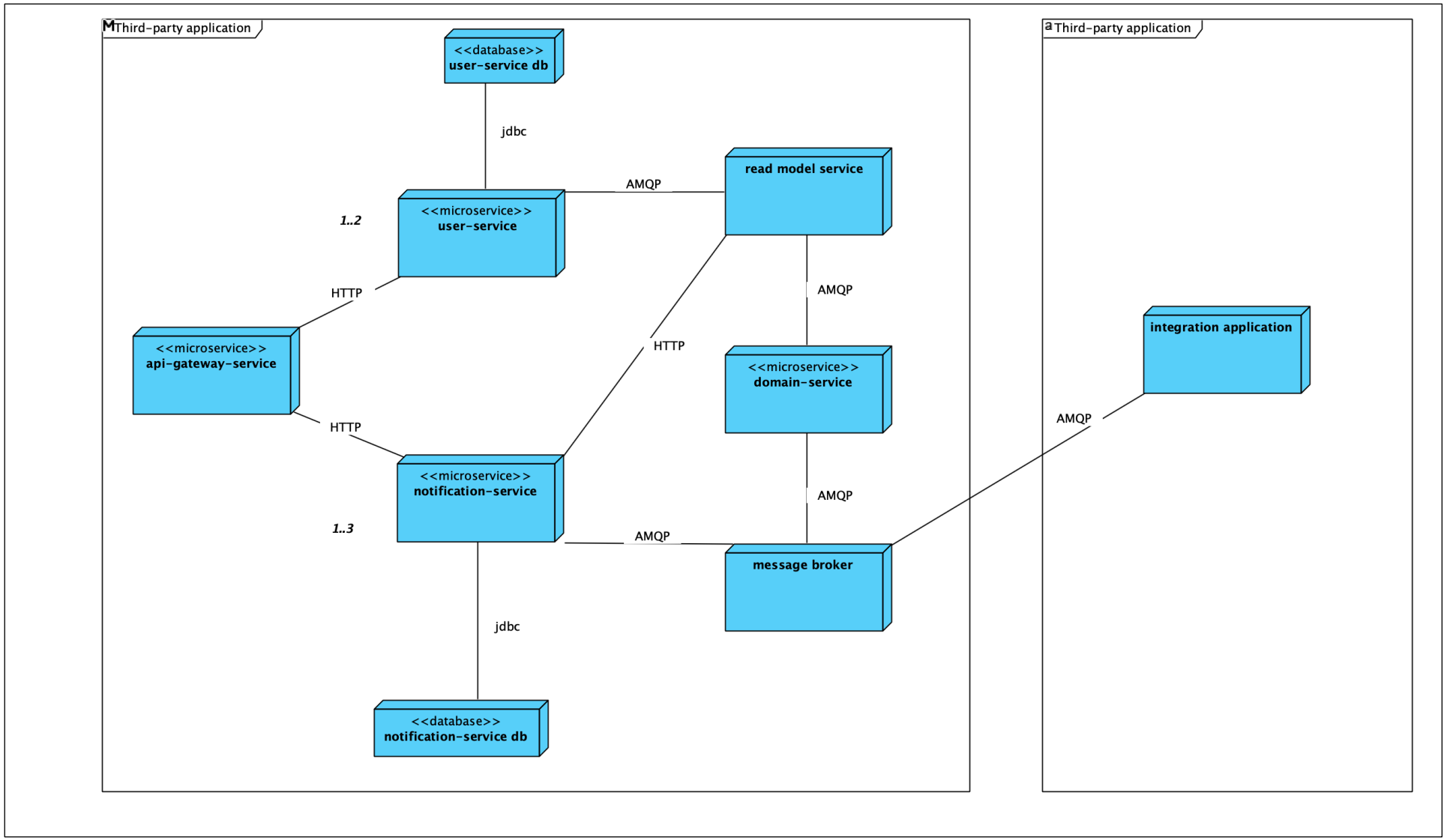


Рисунок 7 – Архітектура досліджуваної системи

Для аналізу необхідності впровадження брокеру повідомлень: необхідно розглянути роботу систем та її складових, беручи до уваги постановку задачі та архітектуру системи.

У ході роботи буде створено дві версії системи: v1, v2. Версія v1 буде розроблена без введення брокеру повідомлень, у той час як версій v2 буде розроблена з впровадженим брокером повідомлень. Архітектура та нефункціональні властивості двох версій системи будуть проаналізовані у розділах «Кількісні заміри часу роботи системи без введення брокерів повідомлень» та «Кількісні заміри часу роботи системи після введення брокерів повідомлень» , а саме час обробки певної кількості даних.

## 2.4 Опис роботи досліджуваної моделі без наявності брокеру повідомлень

Розглянемо інтеграційний сервіс системи. Основна його ціль – доставляти інформацію з уявної сторонньої системи до досліджуваної. Кількість інформації, що відправляється, не обмежена за обсягом та часом. Згідно з постановкою задач, комунікація між інтеграційним сервісом на іншим сервісом системи є безперервною, іншими словами, доменний сервіс повинен мати підтримку маніпулювання даними як з зовнішнього інтерфейсу користування, так і з інтеграційним сервісом. Потенційно, інтеграційний сервіс може виходити з ладу, що не повинно впливати ні на роботу системі в цілому, ні на роботу інтегрованого сервісу.

Принцип комунікації інтеграційного сервісу з доменним сервісом наступний: інтеграційний сервіс використовує кінцеву точку публічного інтерфейсу застосування, налаштовані на створення сутності «Сутність». Точка публічного доступу представлена HTTP запитом POST на ресурс /domain. У разі вдалого створення сутності: доменний сервіс повертає код відповіді 200 – успіх! У разі невдачі за тієї чи іншої причини, доменний сервіс повинен

повернути код відповіді 500 – помилка серверу. Кожна невдала спроба відіслати сутність до доменного сервісу буде супроводжуватися спробами відправити сутність знову – так званий шаблон спроб (retry pattern). Суть шаблону у тому – щоб виявити транзитивні помилки запитів та при їх наявності – зробити спробу відправки ще раз. Транзитивні помилки – таки помилки, які виникли у результаті таких проблем, які можуть бути не виявлені найближчим часом. Такими є, наприклад, інфраструктурні чи мережеві помилки. У якості стратегії виявлення необхідної затримки перед відправленням повторного запиту є:

- фіксована затримка;
- експоненціальна затримка;
- псевдо раптова затримка.

Фіксована затримка характеризує фіксований час між спробами отримати успішну відповідь. Кількість спроб обмежена конфігурацією. Після того, як доступна кількість спроб була вичерпана, відправлення помічається як невдале.

Експоненціальна затримка характеризує час між спробами отримати успішну відповідь, який визначається за експоненційною функцією. Таким чином, розрахункові ресурси сервісу не витрачаються марно. Кількість спроб обмежена конфігурацією. Після того, як доступна кількість спроб була вичерпана, відправлення помічається як невдале.

Псевдо випадкова затримка характеризує не фіксований та випадковий час між спробами отримати успішну відповідь. Кількість спроб обмежена конфігурацією. Після того, як доступна кількість спроб була вичерпана, відправлення помічається як невдале.

У міжсервесній комунікації буде використовуватися псевдо випадкова затримка і надалі у роботі розглядатися окремо не буде.

Інші деталі інтеграційного сервісу визначені вимогами стороннього уявного застосунку і не мають відношення до досліджуваної роботі.

Доменний сервіс представляє собою сервіс з доменною сутністю, яка є центральною в розглядаємій системі. За постановкою задачі, ця сутність є

центральною і є єдиною сутністю для спрощення ведення дослідницьких активностей.

Доменний сервіс надає публічний додатковий інтерфейс у вигляді POST, GET, PUT, DELETE HTTP запитів для створення, читання, замінювання та видалення сутності з сервісу. Доступ до даних сервісу є можливим як у внутрішній мережі мікросервісів, так і через публічний доступ ззовні через api – gateway сервіс. Доступ до даних у системі буде проводитися сервісом notification service задля отримання детальної інформації сутності.

Обробка даних потребує певних ресурсів. Дані доменного сервісу будуть зберігатися в реляційній базі даних, що означає певні ресурсні затрати на зчитування, видалення та запис даних задля підтримки даних у консистентному виді. Це означає, що база даних є вирішальною у визначенні пропускної спроможності сервісу.

Сервіс нотифікацій – це сервіс, основна робота якого – повертати доступний список непрочитаних нотифікацій для користувача. Нотифікація – це сутність, яка складається з наступних полів:

- ідентифікатор нотифікації;
- час створення нотифікації;
- ідентифікатор сутності;
- тип нотифікації (веб нотифікація чи sms нотифікація);
- ідентифікатор користувача;
- повідомлення нотифікації.

Як зазначено в сутності нотифікації, будуть підтримуватися два види нотифікацій: веб – нотифікації та sms нотифікація.

Згідно з постановкою задачі, задля розкриття можливостей асинхронної обробки, в системі існуватиме два типи нотифікацій: миттєві нотифікації та нотифікації, на обробку яких необхідно витратити певний час. Виходячи з цього, випливає наступне: в системі веб – нотифікації будуть виступати миттєвими нотифікаціями, sms нотифікації будуть такими, на створення яких необхідний час.

Веб – нотифікації будуть зчитані веб – сервісом з сервісу нотифікації як запис у таблиці базі даних. Запис нотифікації буде створений у той час, коли нотифікація була створена у сервісі нотифікацій.

SMS нотифікація буде уособлювати відправку користувачам SMS по створенню нотифікаці. Оскільки робота з відправкою SMS, використання відповідного протоколу тощо не об’єктом дослідження цієї роботи, буде використовуватися фіктивна логіка відправки SMS, а саме: до обробки створення SMS нотифікації буде додано фіксований час – 3 секунди, який буде емулювати реальну логіку. У результаті створення фіктивного SMS повідомлення, буде створена веб – нотифікація з поміткою SMS, яка буде відібражати уявне SMS повідомлення у веб – інтерфейсі. Послідовна обробка повідомлень, що потребують час на обробку, описує типову проблему недоліку синхронних викликів.

Згідно з постановкою задачі, необхідно змоделювати таку систему, у якій будуть мати місце розподілені дані. Моделюючи систему, такими можуть бути дані щодо імені користувача та імені сутності, що будуть фігурувати и повідомленні нотифікації. Таким чином, сервіс має наступні альтернативи. По – перше, сервіс нотифікацій може відправляти HTTP запити до кожного с сервісів, щоб отримати детальну інформацію, але така поведінка можлива тільки у простому читанні даних. У більш складних випадках, наприклад, фільтрація даних, час виконання невпинно зростає разом із ресурсами, необхідними для вирішення даної задачі. Для цього, сервіс нотифікацій буде дублювати дані, при їх відсутності у сервісі нотифікацій, задля більш гнучкого доступу до них. Такий підхід відтворює проблему дублювання даних у гетерогенний системах.

Нотифікації зберігаються у сервісній базі даних, та будуть доступні ззовні через сервіс зовнішнього шлюзу через ресурс /notifications за допомогою HTTP запиту GET. Створення, видалення нотифікацій реалізовано за допомогою запитів CREATE, DELETE відповідно.

Створення нових нотифікацій реалізовано наступним чином. Доменний сервіс, по створенню сутності «Сутність» відправляють HTTP POST запит до сервісу нотифікацій.

Доступ клієнта до нотифікацій здійснюється за допомогою простого опитування з інтервалом. По успішному поверненню нотифікацій користувачеві, вони видаляються з бази даних. Сервісу нотифікацій.

Сервіс користувача – такий сервіс, який є відповідальним за зберігання сутності «користувач» у системі та за автентифікацію і авторизацію користувача. Сутність користувача має наступні властивості:

- ідентифікатор користувача;
- ім’я користувача;
- пароль користувача.

Сутність користувачів зберігатиметься в сервісній базі даних, створення, читання, видалення та оновлення сутності відбуватиметься за допомогою ресурсу /users та типів запиту CREATE, GET, DELETE, PUT відповідно.

Автентифікація користувача відбуватиметься за допомогою технології OAuth2 – відповідний токен з користувацькою інформацією буде відсилатися серед сервісів в мережі – таким чином, кожен з сервісів буде мати доступ до користувача, який відіслав запит, при цьому придержуючись підходу без стану (stateless). Сервіс зовнішнього шлюзу, при відсутності токену, перенаправляє запит до сервісу користувача.

Сервіс зовнішнього шлюзу є вхідною точкою усіх запитів, відправлених до системи, проксування їх до кожного сервісу системи.

Згідно з постановкою задачі та архітектурного дизайну, веб – сервіс є зовнішньою репрезентацією роботи системи.(продолжить). Веб – сервіс буде представляти собою застосування єдиної сторінки (single page application), що буде мати наступну логіку. Автентифікація користувача, після якого система визначить користувача таким, що зможе отримувати нотифікації. Після успішної нотифікації, застосування буде за фіксованим інтервалом опитувати застосунок на наявність нотифікацій за ресурсом /notifications. Токен, який був

отриманий у результаті автентифікації, зберігає інформацію, необхідну для авторизації, у тому числі – для можливості сервісу вивести відправника запиту. Таким чином, сервіс, згідно з внутрішньою логікою, поверне повний список нових нотифікацій, необхідних даному користувачеві. Повертаємий об’єкт буде агрегувати всю інформацію сутностей «Нотифікація», «Сутність». Веб – сервіс буде надавати функціональність створення нотифікації для того чи іншого користувача за допомогою інтерфейсу. Така можливість введена для більш наглядного тестування системи. Таким чином, веб – версія системи буде відібражати нотифікації створені вручну та за допомогою інтеграційного сервісу.

## 2.5 Опис роботи досліджуваної моделі за наявності брокеру повідомлень

Досліджуєма модель системи з брокером повідомлень ідентична моделі системи без прокеру повідомлень, за винятком наступних сервісів.

Інтеграційний та доменний сервіс. Створення нових сутностей «Сутність» буде проводитися наступним чином: до обидвох сервісів буде підключена залежність клієнту брокеру повідомлень, який надає технічні можливості комунікації з сервісом брокеру повідомлень, серед яких:

- підписка на канал повідомлень за темою;
- відписка від каналу за темою;
- відправка повідомлення до каналу за темою;
- конфігурація повідомлення для читання повідомлення;
- конфігурація повідомлення для відправки повідомлення;
- число одночасно – можливого завантажених для обробки повідомлень.

Обидва сервіси будуть працювати, беручи за основу спільну конфігурацію каналів повідомлень. Інтеграційний сервіс відправлятиме нові

повідомлення з сутностями до каналу з назвою integration.entity.N, де N – версія каналу.

Версіонування каналів є необхідною складовою розробки застосувань, які використовують можливості брокеру повідомлень. Канали та їх складові є такими, що не можуть бути змінені після створення. З цього випливає, що після змін у імплементації сервісів, необхідно створювати новий канал з його складовими. Завдяки такій особливості, гарантовано, що робота двух або більше застосунків одночасно з одним логічною дефініцією каналу не призведе до раптових змін конфігурації каналу, що в свою чергу не призведе до неочікуваних змін у роботі інших сервісів. Версіонування каналів буде проводитися до всіх каналів у системі, тому не буде розглядатися надалі.

Домений сервіс буде підписаний на оновлення каналу integration.entity.N, задекларуючи асинхронну обробку вхідних повідомлень. Максимальна кількість одночасних транзитних повідомлень обмежена конфігурацією брокеру повідомлень.

Сервіс брокеру повідомлень. Згідно з постановкою задачі, екземпляр брокеру повідомлень буде розгорнутий у системі у єдиному екземплярі. Сервіс буде постачатися з клієнтом відповідної версії, яка підключатиметься до інших мікросервісів для взаємодії. Розглянемо складові брокерів повідомлень та їх роботу у системі.

Брокер повідомлень буде поставлений у роботу разом із панелю управління. Панель управління надасть змогу візуалізувати навантаження брокеру, а саме:

- щільність заповнення черг каналів, кількість повідомлень, обробка яких була помічена як невдала;
- кількість повідомлень, які знаходяться у обробці;
- кількість повідомлень, які не були оброблені;
- налаштування каналів на обмінник (exchange);
- властивості каналів.

Налаштування обмінників буде проводитися у бібліотеці – клієнту і буде містити у собі дані, необхідні для підключення того чи іншого мікросервісу, який використовує клієнт.

Налаштування черг у каналів буде проводитися у також у бібліотеці клієнту. Кожна черга буде мати власну версію, що буде відповідати чинній версії клієнту. Кожна черга каналу буде мати налаштованого обмінника скасованих повідомлень (deal letter exchange), який буде відправляти повідомлення, що не були оброблені 3 рази підряд, або обробка яких закінчилася помилкою, або які не можуть бути оброблені через переповнення чинної черги повідомлень у каналі.

Бібліотека – клієнт – це бібліотека, яка буде поставлятися як залежність до інших мікросервісів. Розробка бібліотеки буде проводитися у рамках розробки сервісу нотифікацій. Створення бібліотеки вмотивовано агрегацією спільних конфігурацій та крос – сервісної логіки у єдиний інтерфейс, інкапсулюючи роботу з клієнтом брокеру повідомлень.

Сервіс нотифікацій є центральним сервісом дослідження, тому його робота буде змінена сильною мірою. Створення нових нотифікацій буде можливо лише за допомогою відправлення повідомлення до обміннику з темою notification.N, де N – версія обмінника. Можливість створювати нотифікації за допомогою HTTP POST запиту буде неможлива. Повідомлення буде відправлено на один із двох внутрішніх каналів у сервісі: web.notification.N, та sms.notification.N – канали обробки веб на SMS нотифікацій відповідно. Обидва канали впровадженні для зменшення зв’язності серед компонентів сервісу нотифікацій. Сервіс буде мати два обробника каналів, кожен з яких буде створювати запис у таблиці нотифікацій. Обробник SMS нотифікацій повинен мати 0,5 секундну затримку задля емуляції відправки SMS повідомлення.

Сервіс читацької моделі (read model service). На відміну від попереднього стану системи, вводиться новий сервіс, який буде агрегувати дані системи, необхідні для централізації даних. Таким чином, сервіс буде агрегувати дані сутностей «Користувач», «Сутність». Для цього, сервіс буде обробляти

створені для такої цілі канали під назвою readmodel.user.N, readmodel.entity.N, де N – версія обмінника.

До сервісу користувача та доменного сервісу будуть підключені клієнти задля відправлення події створення, зміни або видалення сутностей до каналів readmodel.user.N, readmodel.entity.N відповідно.

Слід відзначити, що відправка повідомлення буде імплементована за допомогою шаблону вихідних повідомлень (outbox pattern), описаних у розділі архітектурного опису дослідження.

Таблиця 1 повною мірою відображає відмінності двох систем.

Таблиця 1 – Відмінності дослідницької системи

| Назва сервісу | Без брокеру повідомлень | З брокером повідомлень |
|---|---|---|
| Сервіс користувача | – | Підправка змін сутності «Користувач» до каналу читацької моделі readmodel.user.N |
| Інтеграційний сервіс | Створення сутностей у доменному сервісі за допомогою синхронних HTTP запитів | Інтеграційний сервіс не відповідальний за створення сутностей. Проходить відправка повідомлень до каналу integration.entity.N. |
| Доменний сервіс | Опублікування HTTP інтерфейсу для можливості стороннім системам створювати сутність у сервісі. | Створення сутності за допомогою підписки на канал integration.entity.N. Відсилка повідомлення новоствореної сутності «Користувач» до каналу |

Кінець таблиці 1

| Назва сервісу | Без брокеру повідомлень | З брокером повідомлень |
|---|---|---|
| | | читацької моделі readmodel.user.N |
| Сервіс нотифікацій | Створення нотифікацій проводиться у | Створення нотифікацій проводиться |
| | користувацьких сервісах за допомогою синхронних HTTP запитів. Читання розподілених даних послідовними HTTP запитами до кількох функціональних сервісів. | обробником каналу web.notification.N, sms.notification.N, повідомлення до яких маршрутизується від каналу notification.N. Читання розподілених даних послідовними HTTP запитами до одного сервісу читацької моделі. |
| Сервіс читацької моделі | Відсутній у системі | Підписаний на канали оновлення даних з функціональних сервісів readmodel.user.N, readmodel.entity.N. |
| Брокер повідомлень | Відсутній у системі | Розгорнутий у єдиному екземплярі с приватній мережі системи. |

У даному розділі було проведено моделювання поведінки та роботи системи. У наступному розділі, згідно з постановкою задачі, необхідно зробити кількісні заміри роботи системі версії v1.

## 2.6 Кількісні заміри часу роботи системи без введення брокерів повідомлень

Одна з досліджуємих у роботі проблем є проблема використання синхронних операцій, яка має вирішення у впровадженні асинхронних операцій. У роботі буде проаналізовано, передбачено та порівнено час роботи системи у версії з та без брокеру повідомлень для виведення тенденцій змін у часу роботи системи у тих чи інших бізнес процесах.

Для дослідження була обрана ЕОМ Apple MacBook Pro 2017 с такими характеристиками:

- екран 15 Retina глянцевий;
- Intel Core i7, 4 ядра 2.9 ГГц;
- RAM 16 ГБ;
- SSD 512 ГБ;
- Intel Iris Plus Graphics 640;
- macOS Catalina.

У якості засобів виміряння часу роботи було обрано інструмент Oracle VisualVM.

VisualVM – це інструмент, який надає візуальний інтерфейс для перегляду детальної інформації про додатки на основі технології Java (додатках Java), що працюють на віртуальній машині Java (JVM). VisualVM організовує дані про програмне забезпечення JVM, які надходять з інструментами Java Development Kit (JDK), і представляє інформацію таким чином, щоб було можливим переглядати дані в декількох додатках Java. Також застосунок надає можливість переглядати дані про локальні додатках і додатках, що працюють на віддалених вузлах. Застосунок надає можливість збирати дані про примірники програмного забезпечення JVM і зберігати їх в локальній системі, а потім переглядати їх або ділитися ними з іншими. Таким чином, інструмент можна використовувати з метою профайлингу [21].

Розглянемо об’єкт дослідження часу роботи. Об’єктом дослідження часу роботи є точка інтеграції сервісу інтеграції та доменного сервісу, а також створення нотифікації у сервісі нотифікації.

У ході виміру часу необхідно зробити заміри таких операцій у секундах доменному сервісі та інтеграційному сервісі, присвоївши виміри таким змінним:

- час відправки однієї сутності з інтеграційного сервісу, середнє значення, a1;
- час відправки усіх сутностей з інтеграційного сервісу, a2;
- час передачі сутності мережею між доменним та інтеграційним сервісом, середнє значення, a3;
- час обробки усіх сутностей доменним сервісом, a4;
- час обробки однієї сутності доменним сервісом, середнє значення, a5;
- кількість відправлених повідомлень в 100 мілісекунд, a6;
- кількість відправлених повідомлень в 1 секунду, a7.

Також, у ході виміру часу необхідно зробити заміри таких операцій у доменному сервісі та сервісі нотифікацій:

- час відправки однієї сутності з доменного сервісу, b1;
- час відправки усіх сутностей з доменного сервісу, b2 ;
- час обробки однієї сутності сервісом нотифікацій для веб нотифікацій, b3;
- час обробки однієї сутності сервісом нотифікацій для sms нотифікацій, b3.1;
- час обробки усіх сутностей сервісом нотифікацій, b4;
- час передачі сутності мережею між сервісами, b5;
- кількість відправлених сутностей в 100 мілісекунд, b6;
- кількість відправлених сутносей в 1 секунду, b7.

Для проведення дослідження, налаштуємо інтеграційний сервіс на два режими роботи. Перший режим роботи відправляє фіксовану кількість запитів

до сервісу нотифікацій. Таким чином навантаження є рівномірним. Надалі у роботі такий режим буде іменуватися рівномірним режимом роботи інтеграційного сервісу. Другий режим роботи включає ненормоване навантаження. У такому режимі можна конфігурувати загальну кількість повідомлень, яка буде відправлена за певний проміжок часу, у той час, як сервіс буде псевдовипадковою мірою відправляти запити до сервісу нотифікацій. Таким чином, можна зробити емуляцію навантаження роботи сервісу близьку до реальної роботи сервісу. Надалі такий режим роботи буде іменуватися нерівномірним режимом роботи інтеграційного сервісу.

Розглянемо рівномірний режим роботи інтеграційного сервісу. Інтеграційний сервіс буде конфігуровано на створення 50 сутностей за 10 секунд. Кожна з конфігурацій буде. Результат часу роботи кожної з конфігурації буде виміряний 10 разів, після чого середній результат буде обраний у якості фінального.

Провівши випробування, були отримані наступні результати, що відображені у таблиці 2.

Таблиця 2 – Виміри параметрів доменного сервісу та інтеграційного сервісів

| Секунда провед. досл. | a1, с | a2, с | a3, с | a4, с | a5, с |
|---|---|---|---|---|---|
| 1 | 0,153 | 8,35 | 0,122 | 6,01 | 0,019 |
| 2 | 0,147 | 7,41 | 0,144 | 7,2 | 0,003 |
| 3 | 0,15 | 7,44 | 0,135 | 6,79 | 0,015 |
| 4 | 0,131 | 7,87 | 0,13 | 6,5 | 0,001 |
| 5 | 0,155 | 7,75 | 0,148 | 7,8 | 0,007 |
| 6 | 0,154 | 7,7 | 0,149 | 7,65 | 0,005 |
| 7 | 0,139 | 6,99 | 0,135 | 6,85 | 0,004 |
| 8 | 0,141 | 7,15 | 0,129 | 6,49 | 0,012 |
| 9 | 0,135 | 6,77 | 0,119 | 5,98 | 0,016 |
| 10 | 0,149 | 7,46 | 0,121 | 6,15 | 0,028 |

Нижче наведені діаграми часу відправки однієї сутності залежно від кількості сутностей за одиницю часу.

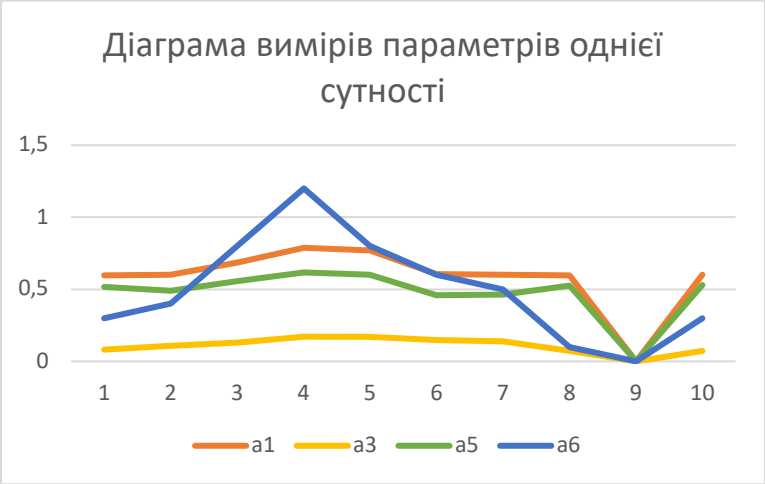


Рисунок 8 – Діаграма вимірів параметрів часу відправки однієї сутності під час режиму рівномірного навантаження

Нижче наведені діаграми часу відправки однієї сутності залежно від кількості сутностей за одиницю часу.

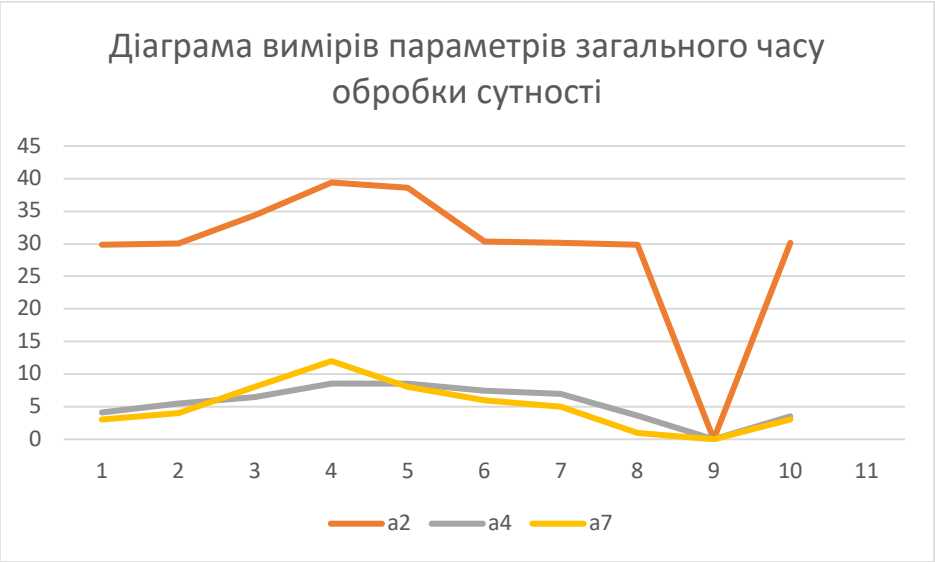


Рисунок 9 – Діаграма вимірів параметрів загального часу обробки сутності під час режиму рівномірного навантаження

Зробімо розбір отриманих даних.

Як видно з таблиці та графіків, обробка створення сутності на стороні доменного сервісу займає певний час, який витрачається здебільшого на запис рядку у таблиці бази даних. Слід зазначити, що зі суттєвим збільшенням збережених даних, кількість часу, витрачаємого на створення нового рядка буде зростати.

Необхідно також зауважити різницю часу між відправкою запиту на створення сутності, та безпосереднього часу на створення сутності у доменному сервісі. Це пояснюється латентністю мережі.

Графіки зображують тенденцію залежності витрачаємих ресурсів інтеграційного сервісу до доменного сервісу, оскільки є технічна необхідність очікувати на успішну відповідь від доменного сервісу. Також досліджені данні відібражають лінійну залежність між кількістю повідомлень за одиницю часу та загальним часом обробки сутностей. Коливання графіків може бути зумовленим оптимізацією Java повторюваних операцій[17][18].

Розглянемо нерівномірний режим роботи інтеграційного сервісу. Інтеграційний сервіс буде зконфігуровано на створення 50 сутностей за 10 секунд.

Результат часу роботи кожної з конфігурації буде виміряний 10 разів, після чого середній результат буде обраний у якості фінального.

Провівши випробування були отримані наступні результати, відображені у таблиці 3.

Таблиця 3 – Виміри параметрів доменного сервісу та інтеграційного сервісів.

| Секунда провед. досл. | a1 | a2 | a3 | a4 | a5 | a6 | a7 |
|---|---|---|---|---|---|---|---|
| 1 | 0,598 | 29,9 | 0,083 | 4,15 | 0,515 | 0,3 | 3 |
| 2 | 0,601 | 30,1 | 0,109 | 5,45 | 0,492 | 0,4 | 4 |
| 3 | 0,687 | 34,4 | 0,129 | 6,45 | 0,558 | 0,8 | 8 |
| 4 | 0,788 | 39,4 | 0,171 | 8,55 | 0,617 | 1,2 | 12 |
| 5 | 0,772 | 38,6 | 0,17 | 8,5 | 0,602 | 0,8 | 8 |
| 6 | 0,607 | 30,4 | 0,149 | 7,45 | 0,458 | 0,6 | 6 |
| 7 | 0,603 | 30,2 | 0,14 | 7 | 0,463 | 0,5 | 5 |

Кінець таблиці 3

| Секунда провед. досл. | a1 | a2 | a3 | a4 | a5 | a6 | a7 |
|---|---|---|---|---|---|---|---|
| 8 | 0,598 | 29,9 | 0,072 | 3,6 | 0,526 | 0,1 | 1 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 0,602 | 30,1 | 0,071 | 3,55 | 0,531 | 0,3 | 3 |

Нижче наведені діаграми часу відправки однієї сутності залежно від кількості сутностей за одиницю часу.

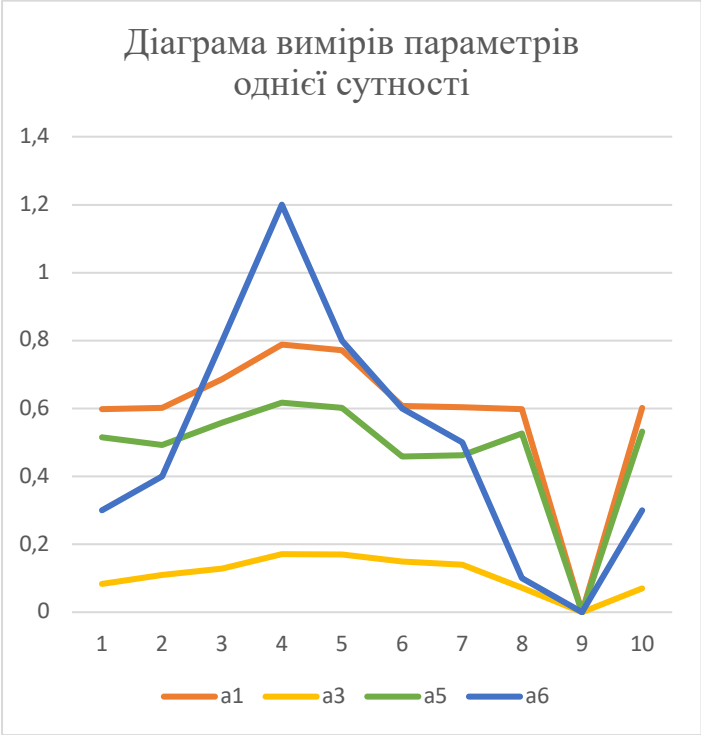


Рисунок 10 – Діаграма вимірів параметрів часу відправки однієї сутності під час режиму нерівномірного навантаження

Нижче наведені діаграми часу відправки однієї сутності залежно від кількості сутностей за одиницю часу.

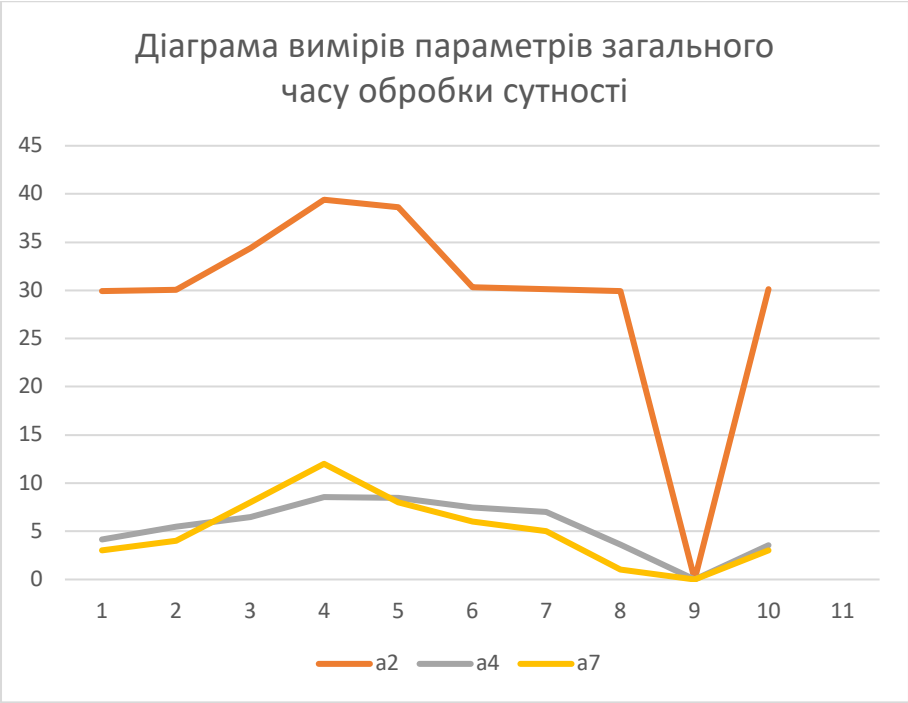


Рисунок 11 – Діаграма вимірів параметрів загального часу обробки сутності під час режиму нерівномірного навантаження

Зробімо розбір отриманих даних. Як видно з таблиці та графіків, навантаження на доменний сервіс зростає відповідно зі зростанням кількості створюваних сутностей. Також відслідковується залежність навантаження інтеграційного сервісу від навантаження доменного сервісу – чим більше час обробки створюваного повідомлення доменним сервісом, тим більше витрачається час на відправку запитів зі сторони інтеграційного сервісу. Також, на відміну від режиму рівномірного навантаження, можна відслідкувати приріст часу, який витрачається на транспорт сутностей по мережі. У той час, як у рівномірному режимі такий приріст був точковий, що можна віднести до статистичної погрішності, під час нерівномірного навантаження час транспорту збільшувався разом із збільшенням кількості одночасних сутностей. Така поведінка також зумовлюється латентністю мережі: збільшення кількості повідомлень зменшує кількість передачі їх одночасно.

Розглянемо вплив нерівномірного навантаження на взаємодію доменного сервісу та сервісу нотифікацій вказані у таблиці 4.

Таблиця 4 – Вимір параметрів доменого сервісу та сервісу нотифікацій.

| Секунда пров. досл. | b1 | b2 | b3 | b3.1 | b4 | b5 | b6 | b7 |
|---|---|---|---|---|---|---|---|---|
| 1 | 0,505 | 26,3 | 0,169 | 0,5 | 25,22 | 0,005 | 0,3 | 3 |
| 2 | 0,508 | 26,4 | 0,169 | 0,5 | 25,13 | 0,008 | 0,4 | 4 |
| 3 | 0,594 | 31,7 | 0,181 | 0,5 | 25,27 | 0,010 | 0,8 | 8 |
| 4 | 0,695 | 37,8 | 0,189 | 0,5 | 25,25 | 0,012 | 1,2 | 12 |
| 5 | 0,679 | 37 | 0,179 | 0,5 | 25,23 | 0,029 | 0,8 | 8 |
| 6 | 0,514 | 26,7 | 0,173 | 0,5 | 25,13 | 0,014 | 0,6 | 6 |
| 7 | 0,51 | 26,5 | 0,17 | 0,5 | 25,22 | 0,010 | 0,5 | 5 |
| 8 | 0,505 | 26,3 | 0,12 | 0,5 | 25,22 | 0,005 | 0,1 | 1 |
| 9 | 0 | 0 | 0 | 0 | 0,00 | 0,000 | 0,0 | 0 |
| 10 | 0,509 | 27,5 | 0,123 | 0,5 | 25,23 | 0,009 | 0,3 | 3 |

Нижче наведені діаграми часу відправки однієї сутності залежно від кількості сутностей за одиницю часу.

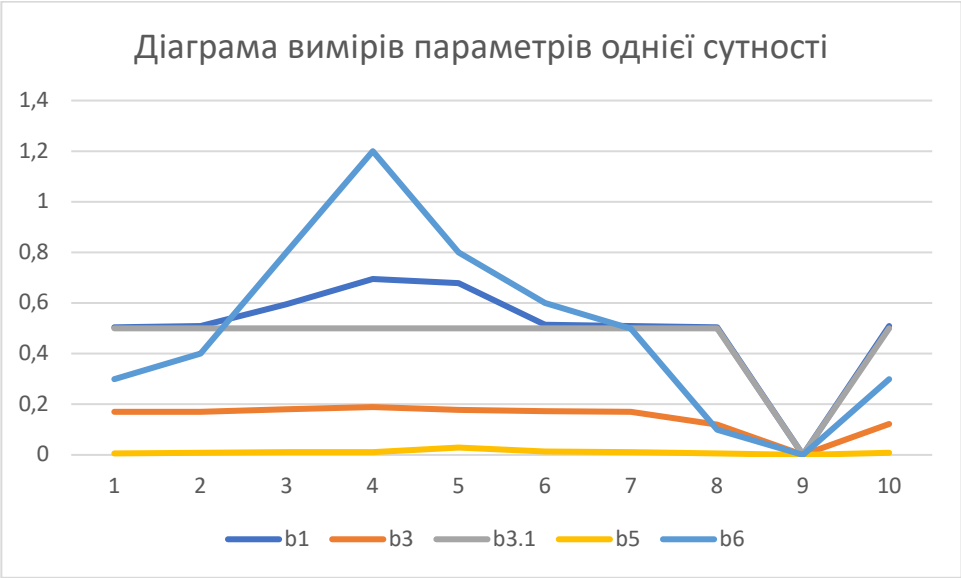


Рисунок 12 – Діаграма вимірів параметрів часу відправки однієї сутності під час режиму нерівномірного навантаження

Нижче наведені діаграми загального часу обробки сутності під час режиму нерівномірного навантаження

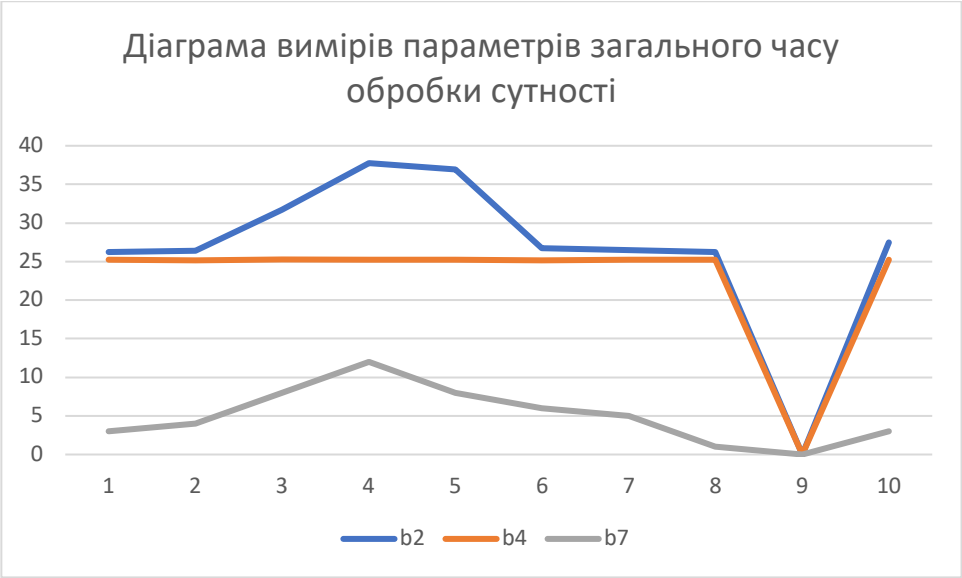


Рисунок 13 – Діаграма вимірів параметрів загального часу обробки сутності під час режиму нерівномірного навантаження

Зробимо розбір отриманих даних. Виходячи з даних параметру b1, обробка створення сутності нотифікацій напряму впливала на час виконання запиту не тільки в сервісі нотифікацій, а й на доменний та інтеграційний сервіс. Також можна прослідкувати, що більшість часу обробки створення нотифікацій займає створення sms нотифікація, яка, згідно с описом роботи системи, представляє собою заглушку, що виконується 500 мс. Слід зауважити, що, незважаючи на те, що серед двох паралельних операцій, загальна кількість виконання повинна бути рівною максимальному часу виконанню, це не завжди було так. На 3 – 5 секундах виконання експерименту видно, що задачі виконувалися майже еквівалентно послідовному виконанню. Це може бути результатом відсутності вільних потоків на виконання тієї чи іншої задачі у застосуванні, що свідчить, що певні задачі очікували на звільнення ресурсів. У розгляді даних сервісів також можна побачити кореляцію між великою кількістю запитів та збільшенням значення латентності мережі.

## 2.7 Математичній розрахунок приросту продуктивності через введення брокеру повідомлень

На вимірах часу виконання системи було зафіксовано великий рівень споживаємих ресурсів доменного сервісу через синхронну роботу ланцюгу сервісів, а саме –інтеграційного сервісу, доменного сервісу та сервісу нотифікацій.

Необхідно розглянути приріст продуктивності при введенні асинхронних операцій у даному ланцюгу, а саме:

- асинхронне створення доменної сутності;
- асинхронне створення нотифікацій.

Для розрахунків буде використовуватися закон Амдала.

Закон представлений в наступній формулі:

$$S = \frac{1}{(1-p) + \frac{p}{s}} \tag{3}$$

де S – теоретичне прискорення виконання завдання, s – прискорення тієї частини завдання, в якій використовуються поліпшені системні ресурси, p – частка часу виконання, частину якої, отримавши вигоду з поліпшених ресурсів, спочатку займала.

Розглянемо роботу доменного сервісу. Визначемо змінні. Мінімально необхідна синхронна робота сервісу, яку необхідно витратити – завдання A. виконання запиту на створення нотифікації– завдання B, створення доменної сутності – завдання C. Тоді виконання завдання A  триває протягом 0,2 секунди, задача B триває 0,5 секунди, і задача C триває 0,3 секунди. Припустимо, що завдання B і C можуть виконуватися паралельно. В цілому завдання B і C виконуються протягом 0,8 секунди, отже, вони становлять 80% завдання X. Отже, з рівняння (1) p = 0,8. Оскільки завдання, що виконуються паралельно B і

С, закінчуються не раніше, ніж час виконання самої трудомісткою задачі (тут – протягом 0,5 секунд), прискорення паралелізації завдань можна оцінити як 0,5 в порівнянні з 0,8 секундами послідовного виконання. Таким чином, частка часу виконання, яку прискорення можуть бути обчислені як 1 + (0,5 / 0,8) = 1,625, яка є з рівняння (1). Беручи до уваги вищесказане, закон Амдала говорить, що загальне прискорення застосування поліпшення буде:

$$S=\frac{1}{(1-0.8)+\frac{0.8}{1.625}} \approx 1.44 \approx 44\% \quad (3)$$

Згідно з розрахунками, теоретичне прискорення після введення асинхронної операції буде становити сорок чотири відсотки.

## 2.8 Кількісні заміри часу роботи системи після введення брокеру повідомлень

Об’єкт дослідження системи з брокером повідомлень ідентичний з дослідженням системи без брокеру повідомленню. Налаштування інтеграційного сервісу також ідентичні. Розглянемо нерівномірний режим роботу інтеграційного сервісу.

Нижче наведені діаграми часу відправки однієї сутності залежно від кількості сутностей за одиницю часу, відображені у таблиці 5.

Таблиця 5 – Виміри параметрів доменного сервісу та інтеграційного сервісів.

| Секунда пров. досл. | a1 | a2 | a3 | a4 | a5 | a6 | a7 |
|---|---|---|---|---|---|---|---|
| 1 | 0,083 | 4,15 | 0,081 | 4,05 | 0,002 | 0,3 | 3 |
| 2 | 0,131 | 6,55 | 0,129 | 6,45 | 0,002 | 0,4 | 4 |
| 3 | 0,141 | 7,05 | 0,138 | 6,9 | 0,003 | 0,8 | 8 |
| 4 | 0,185 | 9,25 | 0,181 | 9,05 | 0,004 | 1,2 | 12 |
| 5 | 0,171 | 8,55 | 0,17 | 8,5 | 0,001 | 0,8 | 8 |
| 6 | 0,153 | 7,65 | 0,15 | 7,5 | 0,003 | 0,6 | 6 |
| 7 | 0,15 | 7,5 | 0,149 | 7,45 | 0,001 | 0,5 | 5 |
| 8 | 0,073 | 3,65 | 0,072 | 3,6 | 0,001 | 0,1 | 1 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 0,071 | 3,55 | 0,07 | 3,5 | 0,001 | 0,3 | 3 |

Нижче наведені діаграми часу відправки однієї сутності залежно від кількості сутностей за одиницю часу.

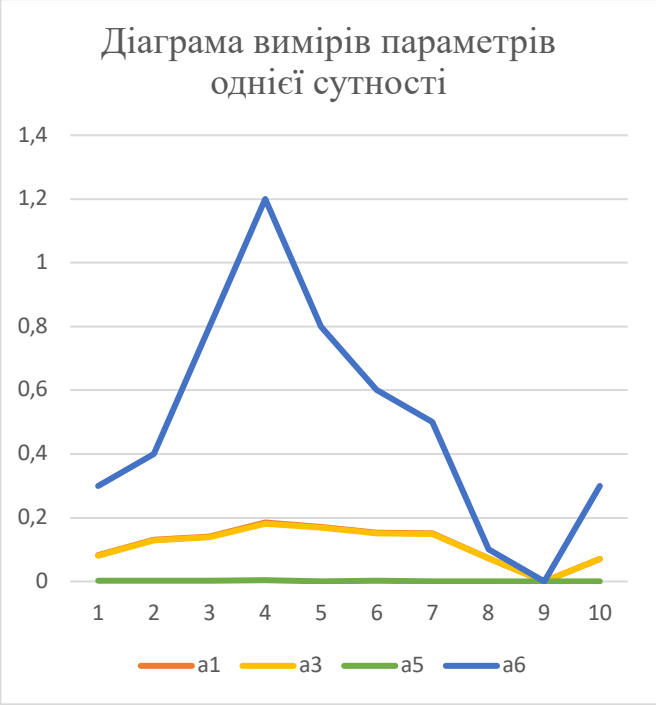


Рисунок 14 – Діаграма вимірів параметрів часу відправки однієї сутності під час режиму нерівномірного навантаження

Нижче наведені діаграми часу відправки однієї сутності залежно від кількості сутностей за одиницю часу.

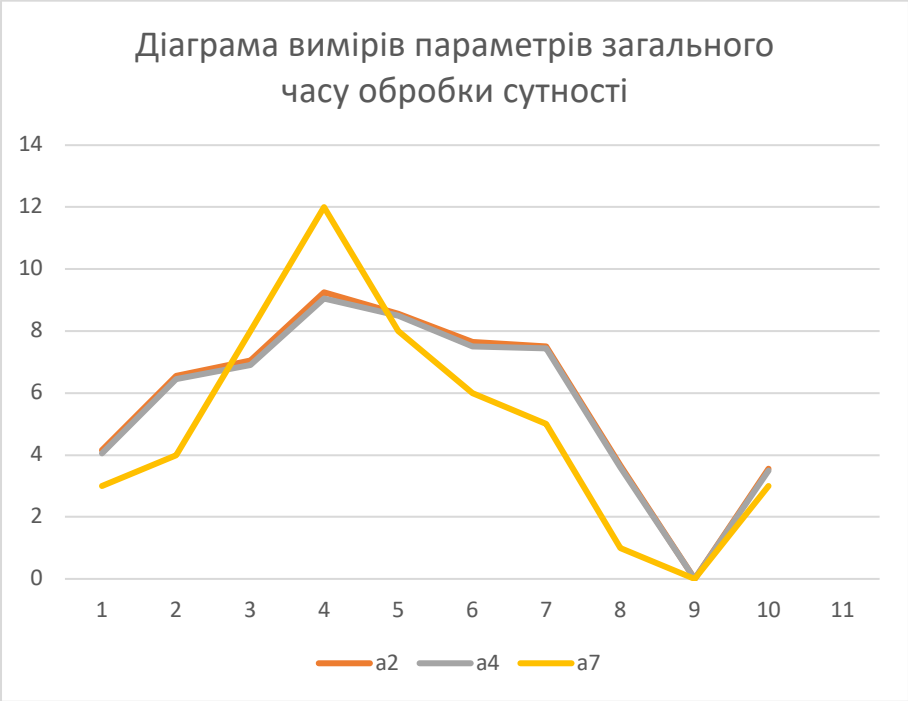


Рисунок 15 – Діаграма вимірів параметрів загального часу обробки сутності під час режиму нерівномірного навантаження

Зробимо розбір отриманих даних. Як видно з таблиці та діаграм, значення параметру a1 зменшилося. Дане явище зумовлюється зникненням синхронної залежності доменного сервісу від сервісу нотифікацій та інтеграційного сервісу від доменного сервісу. Також можна побачити зменшення впливу латентності мережі на швидкість обробки, навіть при необхідності відсилати повідомлення до ще одного сервісу – доменного сервісу. Це є результатом особливостей роботи AMQP протоколу.

Розглянемо заміри взаємодії сервісу нотифікацій та доменного сервісу, вказані у таблиці 6.

Таблиця 6 – Вимір параметрів доменного сервісу та сервісу нотифікацій.

| Секунда пров. досл. | b1 | b2 | b3 | b3.1 | b4 | b5 | b6 | b7 |
|---|---|---|---|---|---|---|---|---|
| 1 | 0,501 | 25,1 | 0,159 | 0,5 | 25,24 | 0,001 | 0,3 | 3 |
| 2 | 0,503 | 25,2 | 0,164 | 0,5 | 25,12 | 0,003 | 0,4 | 4 |
| 3 | 0,514 | 27,7 | 0,18 | 0,5 | 25,27 | 0,001 | 0,8 | 8 |
| 4 | 0,515 | 27,8 | 0,18 | 0,5 | 25,22 | 0,003 | 1,2 | 12 |
| 5 | 0,519 | 28 | 0,171 | 0,5 | 25,33 | 0,004 | 0,8 | 8 |
| 6 | 0,514 | 25,7 | 0,183 | 0,5 | 25,12 | 0,002 | 0,6 | 6 |
| 7 | 0,51 | 28,5 | 0,17 | 0,5 | 25,24 | 0,003 | 0,5 | 5 |
| 8 | 0,501 | 25,1 | 0,12 | 0,5 | 25,13 | 0,001 | 0,1 | 1 |
| 9 | 0 | 3 | 0 | 0 | 0,00 | 0,000 | 0,0 | 0 |
| 10 | 0,509 | 27,5 | 0,124 | 0,5 | 25,23 | 0,003 | 0,3 | 3 |

Нижче наведені діаграми загального часу обробки сутності під час режиму нерівномірного навантаження.

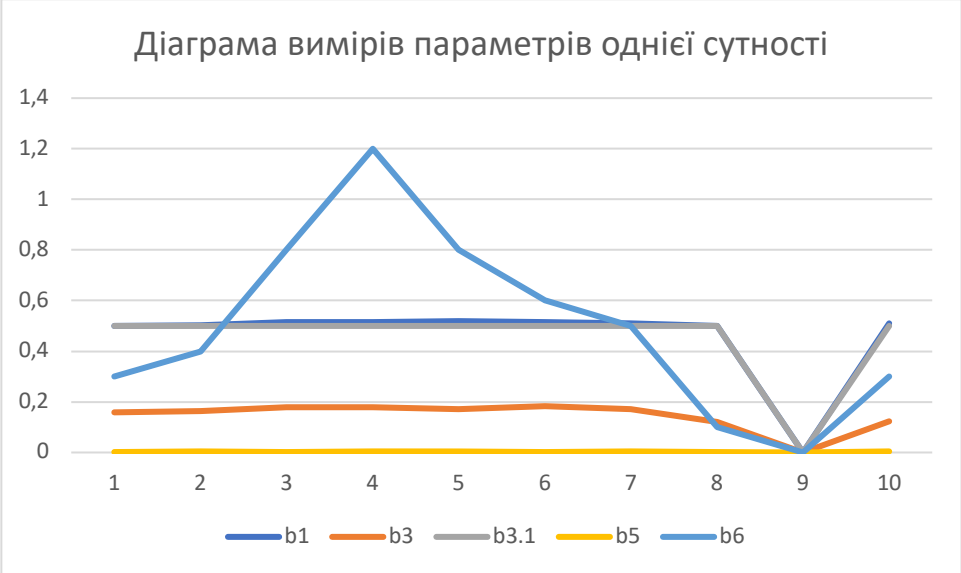


Рисунок 16 – Діаграма вимірів параметрів часу обробки однієї сутності під час режиму нерівномірного навантаження

Нижче наведені діаграми часу відправки однієї сутності залежно від кількості сутностей за одиницю часу.

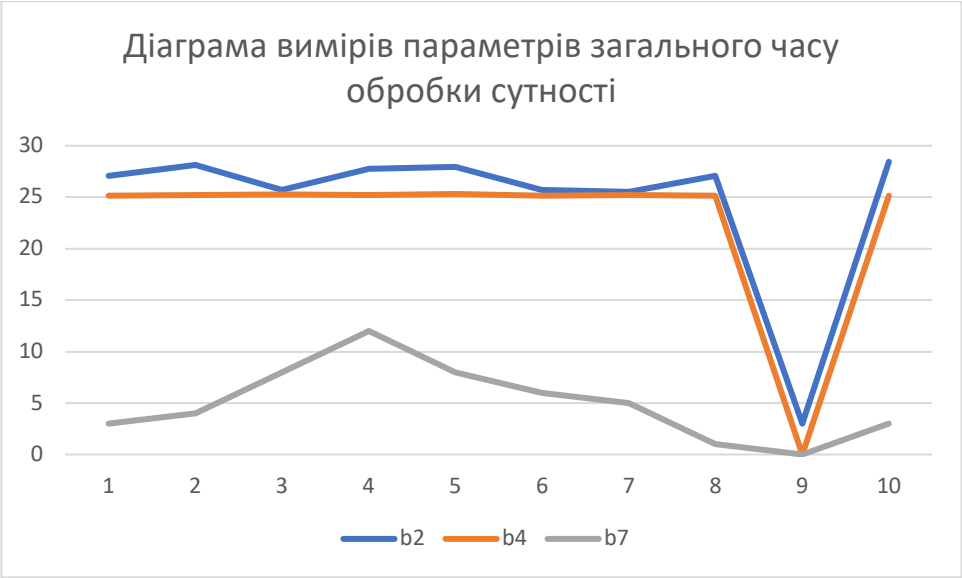


Рисунок 17 – Діаграма вимірів параметрів загального часу обробки сутності під час режиму нерівномірного навантаження

Зробимо аналіз отриманих даних. Виходячи з таблиці значень, вплив латентності мережі зменшився, як і у випадку з попередніми результатами. Також необхідно зазначити зникнення збільшення часу обробки нотифікацій при великому навантаженні. Дані показники є результатом зникнення затримок, пов'язаних з задачами, що очікують пул потоків на виконання. Дане явище є можливим результатом реалізації брокеру повідомлень на мові програмування Erlang, яка має відмінні від Java парадигми програмування, які краще підходять для даних цілей.

## 2.9 Аналіз результатів

В ході дослідження було розгорнуте єдине середовище з застосуванням підходу розробки на основі мікросервісної архітектури, в результаті якого були зроблені наступні висновки щодо проблем, описаних у розділі 2.2.

Сервіс нотифікацій використовував брокер повідомлень для зменшення зв’язності коду. Дане рішення зменшило зв’язаність коду між компонентами сервісу нотифікацій, що призвело до покращення можливостей підтримки застосування, можливості конфігурувати міжкомпонентну взаємодію, використовувати політику обробки помилок. Більш того, виходячи з даних експерименту, введення брокеру повідомлень вирішило проблему задач, які простоюють, під час паралельної обробки веб та sms нотифікацій. Говорячи про поліпшення інтеграції, у брокерів повідомлень є такі недоліки: підвищена складність підтримки інфраструктури та додаткова точка відмови в додаток до денормалізованим даними, зазвичай знаходяться в окремих мікросервісах.

Під час введення брокеру повідомлень до системи був введений новий сервіс – сервіс читацької моделі. На прикладі сервісу нотифікацій, сервіс читацької моделі зміг зменшити кількість розподілених викликів з одного сервісу до інших разом із вирішенням проблеми дуплікації даних. Виходячи з даних експерименту, введення майже не вплинуло на загальний час обробки сутностей. Що стосується використання посередників повідомлень для застосування моделі Read Model – недоліком заявленого підходу є підвищений попит на сховище подій обміну повідомленнями, а також необхідність додаткового моніторингу доступності посередника повідомлень. Властивості транзакцій ACID також не можуть бути досягнуті (атомарність, узгодженість, ізоляція, довговічність), залишаючись дією узгодженості системи у властивостях узгодженості BASE.

Після зміни типу комунікації між інтеграційним, доменним сервісом та сервісом нотифікацій, асинхронні операції були задіяні для комунікації між сервісами. У результаті цього, запит з інтеграційного сервісу більше не очікував на завершення запиту доменного сервісу, який у свою чергу очікував на завершення запиту на обробку сервісу нотифікацій. Прорахувавши за законом Амдала ймовірне прискорення роботи доменного сервісу, було отримано приріст швидкості оброки сутності у 44%. Зробивши заміри роботи доменного сервісу зокрема, можна вивести актуальний відсоток прискорення. Велика частина

ресурсів доменного сервісу витрачалася на синхронне очікування запитів обробки сервісу нотифікацій, які, як мінімум, виконувалися не менше ніж 500 мс через суттєвий час відправки sms нотифікацій. Зробивши співвідношення середнього часу обробки сутності у версії системи v2, фактичне прискорення системи дорівнює = 100 * 0,5856 / 0,1158 = 50,6%. Невелика різниця між теоретичним розрахунком та фактичним заміром може бути зумовлена часткою часу, яку було закладено під час математичних прорахунків на частку необхідних синхронних операцій з боку доменного сервісу. Введення асинхронних операцій має свої недоліки, а саме:

– збільшення складності розробки системи;
– обмеження на максимальну кількість повідомлень.

Загалом, введення брокеру повідомлень до системи, попри внесення певної кількості обмежень та необхідності у додаткових ресурсних внесень у систему, робить корпоративне застосування адаптивним, таким, що має можливість справлятися з викликами, які притаманні високонавантаженим системам сучасності. Брокери повідомлень забезпечують збереження даних і скорочують помилки, які відбуваються при відключенні різних частин системи [22]. За рахунок поділу різних компонентів за допомогою черг повідомлень можна підвищити відмовостійкість. Якщо одна частина системи стала недоступна, інша може продовжувати взаємодіяти з чергою. Можна створити дзеркальну копію самої черги для забезпечення ще більшої доступності.

Після створення бессерверної системи мікросервісів, розгортання на серверах або установки будь – якого програмного забезпечення можна використовувати черги повідомлень для створення надійних і масштабованих бессерверной сповіщень, організації взаємодії між процесами і забезпечення видимості бессерверной функцій і PaaS.

# ВИСНОВКИ

У міру зростання комп'ютерних систем зростає потреба в адаптивних, масштабованих і підтримуваних додатках. Підхід обміну повідомленнями був розроблений з урахуванням класичних і сучасних шаблонів програмних додатків, які використовуються для підтримки технічної зрілості.

Обмін повідомленнями забезпечує нові архітектурні шаблони, які вимагають додаткових зусиль для впровадження та обслуговування. Однак брокер повідомлень вимагає вжиття заходів для забезпечення високої доступності та моніторингу примірників брокера.

У роботі було розглянуто огляд дизайну брокера повідомлень і практичні приклади проектних рішень, які можна інтегрувати в корпоративне додаток. З практичної точки зору, дослідження продуктивності між різними шаблонами обміну повідомленнями має враховуватися при впровадженні одного в існуючу програму, так як програмна система варіюється в залежності від предметної області додатки і функціональних вимог. Крім того, тестування системних сервісів, які мають основне використання обміну повідомленнями, є предметом розгляду дій в рамках функціональних вимог програми.

Результати роботи можуть бути використані в області інтеграції гетерогенних систем; середовища, які вимагають переваг використання паралельних обчислень.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ


1. M. Fowler  Patterns of Enterprise Application Architecture. – Boston: Addison Wesley, 2002. – 21 – 30 с.
2. J. Korab Understanding Message Brokers. Learn the Mechanics of Messaging though ActiveMQ and Kafka. – Sebastopol: O'Reilly, 2017. – 27 – 40с.
3. ISO/IEC TR 19759 Software Engineering — Guide to the Software Engineering Body of Knowledge (SWEBOK), 2005.
4. R. Monson – Haefel, M. Richards and D. A Chappell Java Message Service, 2nd Edition. – Sebastopol: O'Reilly, 2009. – 130 – 134с.
5. J. – Young Byun. A Real – time Message Delivery Method of Publish/Subscribe Model in Distributed Cloud Environment //, M.S. thesis, Information Technology Research Center, Univ. Information & communications Technology Promotion, Daejeon, Republic of Korea, 2017.
6. R. Rocha Improving the performance of a Publish – Subscribe message broker // IEEE 22nd International Symposium on Real – Time Distributed Computing (ISORC)., 2019.
7. Бондаренко М.Ф., Дудар З.В., Збітнєва М.В. Методи і алгоритми аналізу конфігурації мережі // Вісник Житомирського інженерно-технологічного інституту. - Житомир, 2002. - №3. - С. 104-113.
8. Fielding Roy. Architectural Styles and the Design of Network – based Software Architectures. – Сакраменто: Каліфорнійський університет, 2000.
9. Jansen, A., Bosch, J. Software Architecture as a Set of Architectural Design Decisions // 5th Working IEEE/IFIP Conference on Software Architecture, 2005
10. Tanenbaum, Andrew S.; Van Steen, Maarten. Distributed Systems: Principles and Paradigms. Prentice Hall, 2002.
11. Garg, Vijay K. Elements of Distributed Computing. Wiley: IEEE Press, 2002.

12. Хендерсон К. Building Scalable Web Sites / К. Хендерсон. Sebastopol: O`Reilly Media, 2006. – 323 с

13. Gregor Hohpe, Bobby Woolf, Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. – Boston: Addison – Wesley Professional, 2003 – 122c.

14. Jacobson I. Object Oriented Software Engineering – A use case driven approach. – Boston: Addison – Wesley, 1992. – 464 с

15. Фаулер, М. UML. Основы: пер. з англ. – СПб.: Символ, 2006. – 184 с.

16. Кватрани, Т. Rational Rose 2000 к UML. Визуальное моделирование.– М.: ДМК Пресс, 2001. – 176 с.

17. Еккель, Б. Философія Java: пер. з англ. – СПб: 2001. – 880с

18. Шилдт Г. Java: Полное руководство Java SE 8 [Текст] / Г. Шилдт. – 9 – е изд. – М.: Вильямс, 2015. – 1104 с.

19. Хемраджани, А. Гибкая разработка приложений на Java с помощью Spring, Hibernate и Eclipse / А. Хемраджани. − М.: Вильямс, 2008. − 319 с.

20. Хо К. Spring 3 для профессионалов [Текст] / К. Хо, Р. Харроп. – М.: Вильямс, 2012. – 880 с

21. VisualVM All – in – One Java Troubleshooting Tool / Oracle Corporation and/or its affiliates. URL: https://visualvm.github.io (дата звернення – 2 жовтня 2019 р.).

22. V. Apukhtin. The Relevance of Using Message Brokers in Robust Enterprise Applications // IEEE International Scientific and Practical Conference, PIC S&T, Kyiv, Ukraine, 2019.