Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

Переваги та недоліки

сучасних фреймворків черг повідомлень

Текстова частина до курсової роботи за спеціальністю

«Інженерія програмного забезпечення» 121

Керівник курсової роботи
Андрощук М. В.

_____

(підпис)

«__» _____ 2023 р.

Виконала студентка
Кудякова А. В.

_____

(підпис)

«__» _____ 2023 р.

Київ 2023

Індивідуальне завдання

на курсову роботу

Студентці Кудяковій Анні Вадимівні факультету інформатики 3 курсу

ТЕМА: Переваги та недоліки сучасних фреймворків черг повідомлень (ActiveMQ Artemis, RabbitMQ, Kafka, Redis)

Вихідні дані:

Зміст ТЧ до курсової роботи:

Календарний план

Вступ

Розділ 1: Загальний огляд черг повідомлень

Розділ 2: Порівняння

Розділ 3: Практична частина

Висновки

Список використаної літератури

Тема: Аналіз та використання патернів проектування

Календарний план виконання роботи:

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми курсової роботи	22.10.2022	
2.	Пошук тематичної літератури	5.11.2022	
3.	Ознайомлення з літературою	22.11.2022	
4.	Ознайомлення з теоретичним матеріалом	12.01.2023	
5.	Пошук прикладів реалізації проектів з досліджуваними фреймворками	15.02.2023	
6.	Написання першого розділу	28.02.2023	
7.	Написання другого розділу	9.03.2023	
8.	Реалізація програм для третього розділу	27.03.2023	
9.	Опис створених проектів	20.04.2023	
10.	Внесення змін до курсової роботи відповідно до зауважень наукового керівника	30.04.2023	
11.	Створення презентації	5.05.2023	
12.	Захист роботи	24.05.2023	

Студент Кудякова А. В.

Керівник Андрощук М. В.

“____” _____

Зміст

Зміст **4**
Вступ **5**
Розділ 1. Загальний огляд черг повідомлень **7**
1.1 Поняття черг повідомлень 7
1.2 Загальний огляд досліджуваних брокерів повідомлень 9
1.2.1 Kafka 9
1.2.2 RabbitMQ 11
1.2.3 ActiveMQ Artemis 12
1.2.4 Redis 13
Розділ 2. Порівняння **14**
2.1 Pull/Push – based approach 14
2.2 Збереження повідомлень 15
2.3 Маршрутизація повідомлень 16
2.4 Архітектура 17
2.5 Продуктивність 20
2.6 Випадки використання 21
Розділ 3. Реалізація проектів **25**
3.1 ActiveMQ 25
3.2 Kafka 29
3.3 RabbitMQ 34
3.4 Redis 38
3.5 Підсумки реалізації проектів 42
Висновок **44**
Список використаної літератури **46**

Вступ

У контексті сучасної епохи даних, ефективна та надійна обробка повідомлень є особливо актуальною. Черги повідомлень стали невід'ємною складовою в архітектурі високонавантажених додатків, а отже, є необхідними для опанування при роботі з останніми.

У цій роботі основна увага приділяється аналізу та порівнянню чотирьох найпопулярніших фреймворків черг повідомлень: RabbitMQ, ActiveMQ, Redis та Kafka. Вибір цих брокерів повідомлень обумовлений їх популярністю в середовищі розробників, високими показниками продуктивності та маштабованості, хоча кожен з них має свої особливості, переваги та недоліки.

Теоретична частина роботи зосереджується на детальному аналізі кожного з цих фреймворків, дослідженні їх архітектури, механізму доставки повідомлень, продуктивності та інших ключових характеристик, а також порівняння за цими параметрами.

В практичній частині роботи буде розроблено чотири Spring Boot проекти, кожен з яких включатиме споживача та виробника, що комунікують завдяки відповідному брокеру повідомлень. Ці проекти демонструватимуть практичне використання кожного з досліджуваних фреймворків, зручність їх використання та дозволять перевірити теоретичні висновки на практиці.

Основною метою цієї роботи є надання об'єктивного огляду кожного з розглянутих фреймворків, виявлення їх сильних та слабких сторін, а також демонстрація реальних прикладів їх використання.

Розділ 1. Загальний огляд черг повідомлень

1.1 Поняття черг повідомлень

Брокер повідомлень - програма-посередник, що дозволяє програмам, сервісам та системам комунікувати один з одним [16]. Брокер повідомлень перекладає повідомлення з формального протоколу обміну повідомленнями відправника на формальний протокол обміну повідомленнями одержувача. Таким чином сервіси можуть комунікувати між собою безпосередньо навіть якщо вони реалізовані на різних платформах чи написані на різних мовах [16].

Брокери повідомлень використовують для передачі, зберігання, перевірки та доставки повідомлень. Основна задача брокера повідомлень бути посередником між взаємозалежними сервісами, завдяки чому досягається відокремленість процесів всередині системи [7]. Відправник не знає скільком отримувачам прийде повідомлення, не має турбуватись про те, де вони знаходяться і чи будуть вони в цей час онлайн. [6] Так само отримувачу нічого не потрібно знати про відправника, його цікавить лише отримане повідомлення. Таким чином кожен сервіс може зосередитись на своїй основній роботі, не обтяжуючи себе додатковими функціями.

Основні поняття черг повідомлень :

Producer - ендпоінт, що надсилає повідомлення, які зберігаються в брокері повідомлень для подальшого розповсюдження.

Consumer - ендпоінт, що споживає повідомлення.

Queue/topic - тип даних, який брокер повідомлень використовує для збереження та впорядкування вхідних повідомлень

Існує два базові патерни комунікації з брокером повідомлень [16]

Point-to-Point

Спілкування один до одного між відправником та одержувачем, тобто в такому виді комунікації бере участь лише один відправник та лише один отримувач. Кожне повідомлення відправляється і споживається всього один раз. Для зберігання повідомлень в цьому патерні використовують черги повідомлень з логікою FIFO (first in first out). В цьому типі даних повідомлення зберігаються в тому ж порядку, в якому були додані до неї, а також, доти, доки одержувач не буде готовий отримати їх. За допомогою черги повідомлень досягається надійна гарантована доставка повідомлень, адже повідомлення дійде до адресата навіть якщо він не онлайн або у випадку збою програми. [14] Так, як черга повідомлень займається транспортуванням повідомлень і не обтяжує цим відправника і отримувача, вона дозволяє сервісам комунікувати асинхронно.

Point-to-Point патерн використовується в системах, де повідомлення має, по-перше, обов’язково бути отриманим одержувачем, по-друге, лише один раз.

До прикладу фінансова транзакція, або код для верифікації, який не повинен бути вірним, якщо спробувати його ввести для наступної спроби перевірки

Publish-Subscribe

Спілкування один до багатьох між відправником та одержувачем. В цьому патерні відправника називають publisher, а отримувача subscriber, саме тому патерн називають “pub/sub”. Відповідно до даної моделі повідомлень publisher (виробник) публікує повідомлення в topic (тему), а subscriber (підписник) підписується на ті теми, з яких він бажає отримувати повідомлення. Кожне повідомлення, що потрапило в певну тему, відправляється всім споживачам, що є підписаними на дану тему. Один або декілька виробників можуть публікувати повідомлення в одну тему; підписник отримує повідомлення від одного або багатьох відправників. Відправник не знає нічого про отримувачів, так само як підписники не отримують ніякої інформації про відправника. Така модель може бути використана в системах, одна і та ж інформація має надходити декільком сторонам одночасно [11]. До прикладу, електронне повідомлення від магазину про знижки всім покупцям, які підписалися на розсилку.

1.2 Загальний огляд досліджуваних брокерів повідомлень

1.2.1 Kafka

Kafka – це платформа розподіленої потокової передачі подій призначена для ефективної передачі, обробки та зберігання великих об’ємів інформації.

Початково розроблений LinkedIn-ом Kafka з 2011 стає open-source проектом. [2] Цей брокер повідомлень підтримує всі основні мови програмування деякі з яких Java, JavaScript, C/C++, .NET, Python, PHP, Ruby , та може ранитись як у локальному, так і в хмарному середовищах.

Для ефективної комунікації між серверами та клієнтами в Kafka імлементована розподілена система, що використовує мережевий протокол TCP. Це дозволяє передавати повідомлення з пропускною спроможністю з дуже низькою затримкою, всього 2 мілісекунди, що робить цей брокер повідомлень чудовим вибором для великомаштабних програм з потребою в обробці великої кількості даних.[8] Крім того, в кластері серверів, що реалізує розподілену систему, можна надійно зберігати повідомлення, з впевненістю, що вони нікуди не зникнуть. В систему Kafka також вбудовані механізми розподілу (partitioning) для горизонталнього маштабування та реплікації (replication) для можливості відновлення після відмови (failover capability) [10].

Порівнюючи Kafka з іншими брокерами повідомлень за продуктивністю, його показники, як правило, є вищими [15]. Саме тому його часто використовують у сценаріях швидкої роботи – у керуванні подіями та пайплайнами потокових даних у реальному часі. Цей меседж брокер застосовується такими популярними компаніями як Spotify, Uber, Slack, Netflix, Coursera, eBay, PayPal, та іншими, що може свідчити про високу якість продукту [2].

1.2.2 RabbitMQ

RabbitMQ – найпопулярніший брокер повідомлень з відкритим кодом і один з перших, що з'явились на ринку технологій (реліз відбувся у 2007 році). Спочатку імлементований протоколом AMQP (Advanced Message Queuing Protocol), згодом був розширений plug-in архітектурою для сумісності з протоколами MQTT (Message Queuing Telemetry Transport), STOMP (Streaming Text Oriented Messaging Protocol) та іншими [11]. RabbitMQ пітримує багато мов програмування, дозволяє міжмовний обмін повідомленнями та може ранитись як в ріноманітних операційних системах так і в хмарних середовищах. Перевагою цього брокера повідомлень є те, що його можна розгорнути як в об'єднаних так і в розподілених конфігураціях, що задовільніє різноманітні бізнес-потреби та гарантує високу маштабованість [5].

Однією з особливостей RabbitMQ є вбудований плагін [13], що доповнює брокер повідомлень графічним користувацьким інтерфейсом в браузері для більш зручного керування, моніторингу та можливості проглядати різноманітні стаитстики пов'язані з передачею повідомлень. Окрім нього, для керування брокером можна також використовувати HTTP-API, інструмент командного рядка.

Цей брокер повідомлень є user-friendly, інтуїтивно зрозумілим та надійним, що робить його дуже зручним у використанні. В цілому RabbitMQ має велику кількість різноманітних інструментів та плагінів для інтеграції та взаємодії з іншими системами.

1.2.3 ActiveMQ Artemis

Active MQ Artemis – брокер повідомлень з відкритим кодом розроблений Apache Software Foundation та написаний на Java [12]. Active MQ імлементує JMS API (Java Message Service), що визначає стандарт створення та передачі повідомлення. Підтримує найпопулярніші мови програмування (Java, JavaScript, C, C++, Python, .Net) та протоколи; сумісний з багатьма платформами через AMQP протокол [12]. Крім того, відомий через сумісність з багатьма міжмовними (cross language) клієнтами.

Active MQ є досить універсальним та гнучким, тож підійде для різних сценаріїв обміну повідомленнями. В ньому імлементовані такі функції як кластеризація, групи повідомлень, комбіновані черги та кешування.

На разі існують дві версії ActiveMQ: Classic і Artemis. Artemis є розширеною версією, і саме її я досліджуватиму в цій роботі [1]. Active MQ Artemis має неблокуючу архітектуру, що є необхідністю для програм нового покоління, які вимагають високої продуктивності.

Active MQ Artemis був створений для розробників та підійде для роботи з невеликою кількістю інформації. Він легкий в налаштуванні та підтримці і не вимагає великої кількості ресурсів. До того ж, цей меседж брокер є чудовим вибором для тих випадків, де необхідна одноразова доставка [1].

1.2.4 Redis

Redis – це in-memory сховище структури даних з відкритим кодом, що використовується як база даних, двигун кешування та потокової передачі, а також як брокер повідомлень [5]. Через те, що в першу чергу Redis це in-memory сховище даних, повідомлення в ньому не будуть зберігатись довго. Замість збереження даних, він скидує пам’ять в базу даних або на диск. Такий механізм чудово підійде для обробки інофрмації в реальному часі [5].

Раніше Redis можна було використовувати лише імплементуючи патерн point-to-point, але після релізу Redis 5.0 стало можливо також Pub/Sub обмін повідомленнями.

Redis підтримує найпопулярніші мови програмування, деякі з яких Java, C, C++, C#, Node.js, Python та інші. Він є прикладом дуже спрощеної технології, яка не вимагає багато часу чи зусиль для освоєння, не потребує великої кількості ресурсів, а також може похизуватись широкою підтримкою відкритої спільноти [11].

Розділ 2. Порівняння

2.1 Pull/Push – based approach

Є одна відміннсть, що відрізняє Kafka від трьох інших брокерів повідомлень, що ми розглядаємо. Це те, що вони використовують різні підходи, Kafka – pull-based, а ActiveMQ Artemis, RabbitMQ та Redis – push-based [2]. В першому випадку брокер повідомлень очікує на запит даних від споживача, в той час як в другому повідомлення надсилаються споживачу негайно, як тільки надійшли.

Kafka використовує pull-based підхід, де споживач робить запит на пакет (batch) повідомлень з певного зміщення (лічильник останнього отриманого повідомення). Цей меседж брокер також забезпечує тривалий пул (long-pooling), тобто можливість налаштувати часовий інтервал, між пакетами подій, що надсилає виробник[2]. Це дозволяє забезпечити користувачам споживання подій з необхідним їм темпом, а також групувати повідомлення для кращої пропускної здатності.

Push-based брокер повідомлень вирішує, коли надсилати споживачам повідомлення. Попередньо налаштовується обмеження в кількості повідомлень за пенвий час (prefetch limit) для запобігання перенавантаження споживача повідомленнями [10]. Після обробки повідомлення споживачем, надсилається підтвердження, що гарантує отримання повідомлення. У випадку якщо приходить повідомлення про помилку повідомлення знову потрапляє в чергу і здійснюється ще одна спроба відправки повідомлення.

Така модель чудово підходить для передачі повідомлень з низькою затримкою.

Push-based модель має на меті рівномірно розподілити робоче навантаження між декількома споживачами. Через те, що повідомлення можуть оброблятись з різною швидкістю, порядок обробки, той в якому повідомлення надійшли в чергу, може бути неточним.

Зазвичай брокери повідомлень, включаючи RabbitMQ та ActiveMQ дозволяють використовувати як push-based підхід так і pull-based, проте другий використовувати не рекомендовано через потенційну неефективність і підвищену затримку, зважаючи на випадки використання [1].

2.2 Збереження повідомлень

Kafka повідомлення є постійними і не видаляються після отримання їх споживачем, адже Kafka працює як лог повідомлень. Дані, що надійшли до Kafka зберігаються там до вказаного періоду зберігання, який за замовчуванням дорівнює тижню. Особливістю цього брокера повідомлень є можливість використання стратегії ущільнення логування (Log compaction strategy), що передбачає збереження лише останнього значення для кожного ключа повідомлення в межах черги для єдиного розділу теми (topic partition), старіші ж версії повідомлення видаляються [17]. Її можна встановити в налаштуваннях.

В цей час в ActiveMQ і RabbitMQ повідомлення зберігаються до того часу, поки не будуть оброблені споживачам та підтверджені зворотнім повідомленням (ACK message – acknowledgment message) від нього [13]. Коли відправка повідомлення підтверджена, воно видаляється з черги. Якщо ж від споживача прийшло повідомлення негативного підтверження (NACK - negative acknowledgment) повідомлення потрапялє знову в чергу. Разом з тим, збереження повідомлень можна увімкнути, явно законфігурувавши його. В ActiveMQ це досягається файловим журналом на диску або ж JDBC-сумісній базі даних [11]. А RabbitMQ пропонує стійкі черги, метадата яких зберігається на диску [2].

У випадку Redis повідомлення миттєво відправляються споживачу та видаляються. Навіть якщо споживач не був доступний чи під’єднаний, повідомлення не зберігається і не може бути повторно відправленим.

2.3 Маршрутизація повідомлень

RabbitMQ пропонує широкі можливості для гнучкої маршрутизації повідомлень, спроможні задовільнити навіть важкі випадки. В його мехізм вбудовано декілька типів обміну для різної логіки маршрутизації: прямий (direct), тематичний (topic), фанаут (fanout) та обмін заголовками (header) [2]. Прямий обмін направляє повідомлення згідно з ключем маршрутизації (routing key), яке містить повідомлення. Тематичний обмін маршрутизує повідомлення в черги відповідно до збігу повного чи часткового з ключем маршрутизації [2]. Фанаут обмін направляє повідомлення в усі доступні черги та ігнорує ключ маршрутизації. Обмін заголовками відбувається на

основі заголовків повідомлення , що може вміщати більше атрибутів, аніж ключ маршрутизації. Для складнішої маршрутизації користувач може поєднати декілька типів обміну або навіть створити свій власний тип обміну у вигляді плагіну [8].

В ActiveMQ також імлементований механізм гнучкої маршрутизації за допомогою адрес підстановки (wildcard addresses), способу організації подій в ієрархії через використання символів узагальнення [10]. До прикладу, якщо адреса черги виглядає як queue.animal.#, то вона отримує всі повідомлення, надіслані адресам queue.animal.bear чи queue.animal.duck, та іншим, що відповідають шаблону. Окрім того, ActiveMQ підтримує селектори JMS (механізм, що дозволяє фільтрувати події на рівні брокера) [12].

Kafka не підтримує можливість фільтрування повідомлень перед їх запитом [17]. Теми розділені між розділами (partitions), що вміщають повідомлення в незмінному порядку. Споживач, підписаний на тему має виконувати фільтрацію на своєму рівні.

У Redis так само відсутня концепція маршрутизації. Повідомлення відправляються з каналу всім його підписникам без розбору.

2.4 Архітектура

В описі архітектури брокерів повідомлень розглянуто процес обміну повідомленнями за патерном publish/subscribe.

Кафка класично містить в собі виробників та споживачів. Виробники відправляють потоки подій в брокер повідомлень, а потім споживач пулає їх з брокера. Події стають в чергу в темах (topic), а потім пропорційно поділяються між розділами (partition) і надсилаються всім підписникам теми. [14] Тут слід детальніше пояснити наступні терміни :

- Тема (topic) – компонент, що логічно розподіляє повідомлення за їх типом, семантикою. Це зручно для групування повідомлень згідно їх призначення. До прикладу, в одну тему надходять повідомлення нагадування, у той час як в іншу – повідомлення про помилки

- Розділ (partition) – контейнер, в якому містяться повідомлення з певної теми. Повідомлення розподіляються рівномірно між усіма розділами та зберігаються в незмінному порядку. Розділ реплікується між багатьма брокерами і обмежується фактором реплікації, до прикладу, якщо фактор реплікації 4, то кожний розділ збережеться в 4 окремих брокерах. Серед цих брокерів, один визначається як головний, а всі інші вважатимуться послідовниками. Головний брокер повідомлень видаляє та додає повідомлення до розділу, а послідовники, у свою чергу, синхронізуються з ним.

В RabbitMQ виробник надсилає повідомлення в певний обмін (exchange) з 4-ьох, які були описані вище. У свою чергу, обмін направляє повідомлення у чергу, базючись на правилах прив’язки (binding rules), відповідно до типу обміну. Після цього споживачі, підписані на черги тримують повідомлення [2].

В цьому флові роботи варто описати такі терміни :

- Обмін (Exchange) – отримує повідомлення та, відповідно до структури отриманого повідомлення та\чи його атрибутів (залежно від типу обміну), визначає, куди повідомлення має бути направлено.

- Правила прив’язки (Binding rules) – визначають в яку чергу повідомлень потрібно направити повідомлення, а також фільтрують повідомлення, які можуть бути направлені в ту чи іншу чергу для певних типів обміну.

В ActiveMQ повідомлення надходять в адреси звідки надсилаються чергами до всіх споживачів, підписаних на відповідну адресу. У кожної адреси є тип маршрутизації [1].

Адреса (address) – це ендпоінт, в який надходять повідомлення. Кожна адреса має ім’я, одну чи більше чергу і тип маршрутизації.

Тип маршрутизації (routing type) – визначає в які черги будуть направлені повідомлення. Існує два типи маршрутизації : anycast і multicast. Anycast тип використовується у випадку, якщо потрібно направити повідомлення лише в одну чергу, отже у випадку point-to-point обміну повідомленнями. Натомість, multicast використовується якщо повідомлення має бути направлено в усі черги повідомлень в межах однієї адреси, тобто в publish/subscribe сценарії.

В Redis виробник надсилає повідомлення в певний канал (channel) , яке потім направляється всім підписниам цього каналу [14].

2.5 Продуктивність

Kafka розподілена система, яка дозволяє оброблювати велику кількість даних та досягати високої пропускної здатності (мільйон повідомлень за секунду) з обмеженими ресурсами [15]. Для підвищення продуктивності, цей брокер повідомлень використовує послідовний дисковий I/O (ввід – вивід).

В той же час RabbitMQ може обробити 4-10 K повідомлень з такими самими ресурсами, як Kafka [15]. Цей брокер повідомлень може досягти мільйона повідомлень за секунду, проте вимагатиме значно більше ресурсів, приблизно 30 вузлів.

Так як ActiveMQ гарантує відправку кожного повідомлення, що займає час, показник пропускної здатності у нього є нижчим. Натомість, цей брокер повідомлень є швидким у випадках маленької кількості даних, тому є чудовим вибором для систем з нижчою пропускною здатністю повідомлень [15].

Redis обирають для обміну повідомлень в реальному часі без потреби в збереженні повідомлень. Цей брокер повідомлень пропонує низьку затримку повідомлень та підходить для роботи з невеликою кількістю даних у випадках, де необхідна швидка відправка без потреби в надійному збереженні повідомлень [9].

Зважаючи на те, що для великої кількості даних та комплексних систем обміну повідомлення найкраще використовувати Kafka та RabbitMQ саме порівняння їх продуктивності є найбільш доцільним. Нещодавно команда компанії Confluent провела ряд тестів, щоб порівняти продуктивність цих брокерів повідомлень [18]. За їх результатами Kafka значно перевершує RabbitMQ у кількох ключових сферах.

Kafka демонструє чудову пропускну здатність, записуючи дані в 15 разів швидше, ніж RabbitMQ. Ця можливість високошвидкісної передачі даних робить Kafka більш ефективною системою при роботі з великими обсягами даних. З точки зору затримки, Kafka також перевершує RabbitMQ при вищій пропускній здатності, зберігаючи вищу швидкість навіть за великого навантаження. Однак варто зазначити, що RabbitMQ може досягти нижчої наскрізної затримки, ніж Kafka, але лише за значно нижчої пропускної здатності.

2.6 Випадки використання

Kafka

- Джерело подій (Event Sourcing) – представляє всі зміни в додатку як послідовність подій. Використовується для можливості відтворити попередні дії та для збереження історії подій. Використовується за необхідності вести історію повідомлень для того, щоб за потреби можна було їх відтворити [8]. Таким чином ні одна подія не буде загублена, а у випадку неспроможності споживача прийняти

повідомлення, воно буде відправлене знову. Гарним прикладом такого концепту є історія транзакцій у баньківських додатках, за допомогою якої можна легко відслідкувати, коли відбулась певна транзакція та виявити можливі злочиницькі дії.

- Потокова обробка – випадки, коли необхідно зберігати історію повідомлень для можливості відтворення та обробки отриманих даних у багаторівневих пайплайнах [13].

- Відслідкування активності з високою порпускною здатністю (100 К/сек) в реальному часі – Kafka дуже потужний інструмент, який ідеально підійде для швидкої обробки великох кількосіт даних. Прикладами використатння є менеджмент розкладу пацієнтів, ортимання та обробка даних з IoT сенсорів, відстеження активності веб-сайтів та інше [8].

- Моніторинг - обробка та розміження даних. Арграція логування, ключові показники ефективності (KPIs key performance indicators)) є одними з багатьох випадків використання [17].

RabbitMQ

- Випадки, в яких необхідно отримати підтвердеження отримання повідомлення від споживача.

- Якщо кожне повідомлення потребує особливої обробки, гарантій, послідовності.

- Мікросервісна архітектура, системи, де необхідний комплексний механізм маршрутизації. За допогою послідовного хешованого обміну балансується обробка навантаження, окрім того, є можливість використоувати альтернативні типи обмінів для певної частини повідомлень [5].

- Так як RabbitMQ підтримує такі застарілі протоколи, як MQTT, AMQP, STOMP, він підійде для підтримки застрарілих додатків та їх комунікації зі споживачами [2]. Гарним прикладом є використання JMS (Java Message Service) плагіну та клієнтської бібліотеким JMS для спілкування з JMS додатками.

ActimeMQ Artemis

- Трансофрмація даних on-the-fly (змінення динамічних даних, що містяться в часовому кроці траєкторії, зазвичай координати, коли вони завантажуються в пам’ять), імплементація процесу Extract, transform, and load (ETL) (поєднання даних з декількох джерел в один великий репозиторій)

- Випадки, які вимагають високого рівня транзакційності та надійності, коли одне повідомлення має бути однозначно отримане лише один раз.

Прикладами використання є безліч грошових операцій – зняття грошей, перекази, депозити і таке інше.

- Обробка маленької кількості повідомлень за день [10].

Redis

- у випадках миттєвої відправки повідомлень в невеликих кількостях та з допустимістю можливих втрат. Redis обробляє повідомлення в реальному часі з мінімальною затримкою, проте не має достатньо потужності для обробки великих об’ємів інформації ні складності для роботи з комплексними системами обміну повідомлень. [12]

Розділ 3. Реалізація проектів

Для того, щоб перевірити на практиці зручність використання розглянутих брокерів повідомлень я створила проект для кожного брокера повідомлень, що складається з 2 Spring boot проектів, виробника та споживача, які комунікують через систему відповідного брокера повідомлень, що раниться в Docker контейнері. Кожен проект реалізовує патерн publish-subscribe, який був описаний вище.

Основний стек технологій, використаних для створення проектів:

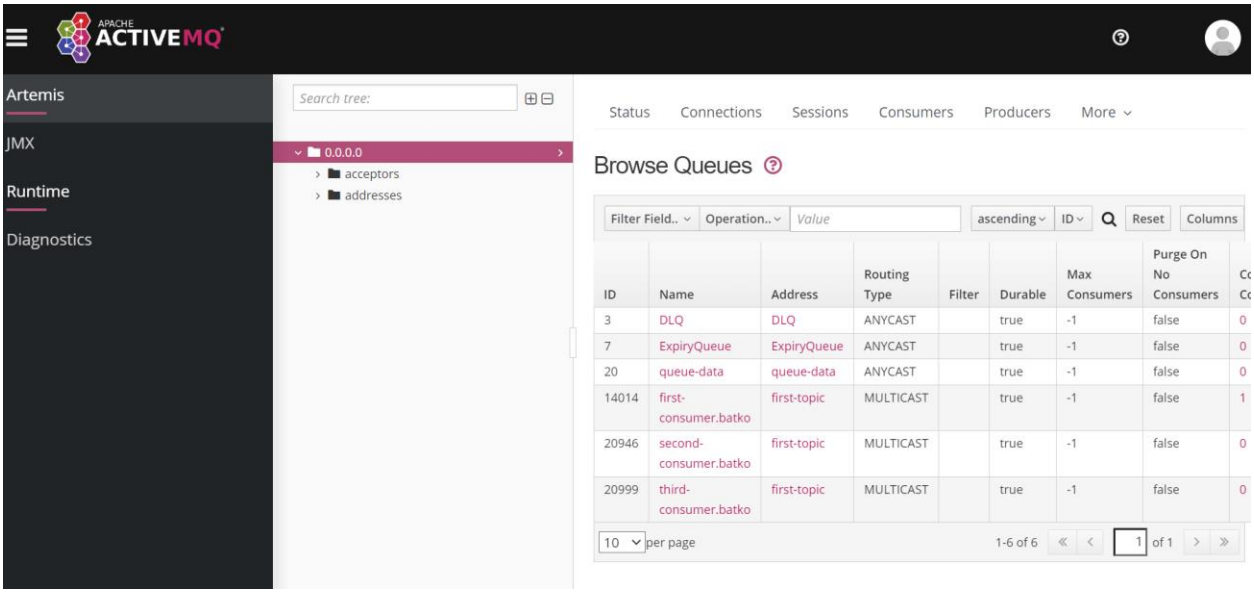
- Java
- Spring boot
- Maven
- Docker
- Postman

3.1 ActiveMQ

Перед початком роботи з ActiveMQ Artemis, необхідно локально встановити його на машину та запустити з командного рядку за допомогою команди artemis run.

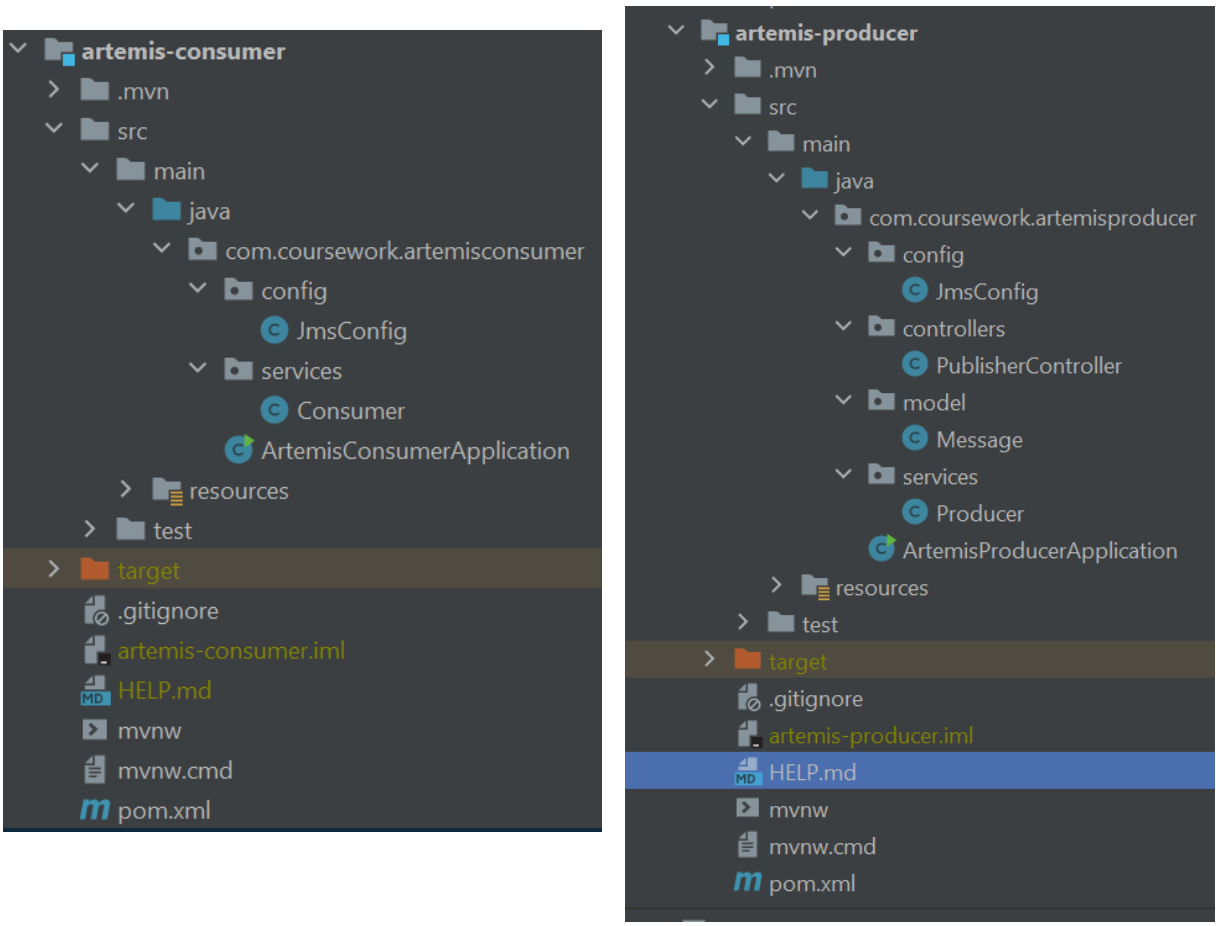
Після того, як брокер повідомлень успішно запустився ми можемо доступитись до графічної консолі Artemis через ендпоінт http://localhost:8161/console/auth/login ввівши відповідні креденшели. Тут

можна відслідковувати всі процеси, що відбуваються в брокері повідомлень : створені черги, виробників, споживачів, підрахунок повідомлень та інше.



(Рисунок 3.1 – графічна консоль Artemis)

Структура Maven проектів споживача та виробника виглядає наступним чином :



(Рисунок 3.2 – Структура проектів ActiveMQ споживача та виробника)

В конфігураційних класах містяться bean-компоненти для увімкення служби електронної пошти для Java через Spring, налаштування виробника та користувача.

У проекті споживача також є клас, основна задача якого отримання повідомлень з черги. За отримання повідомлень відповідає метод messageListener з анотацією @JmsListener, яка вказує на те, що метод має бути викликаний щоразу, як було отримано нове повідомлення з вказаного пункту призначення (атрибут анотації)

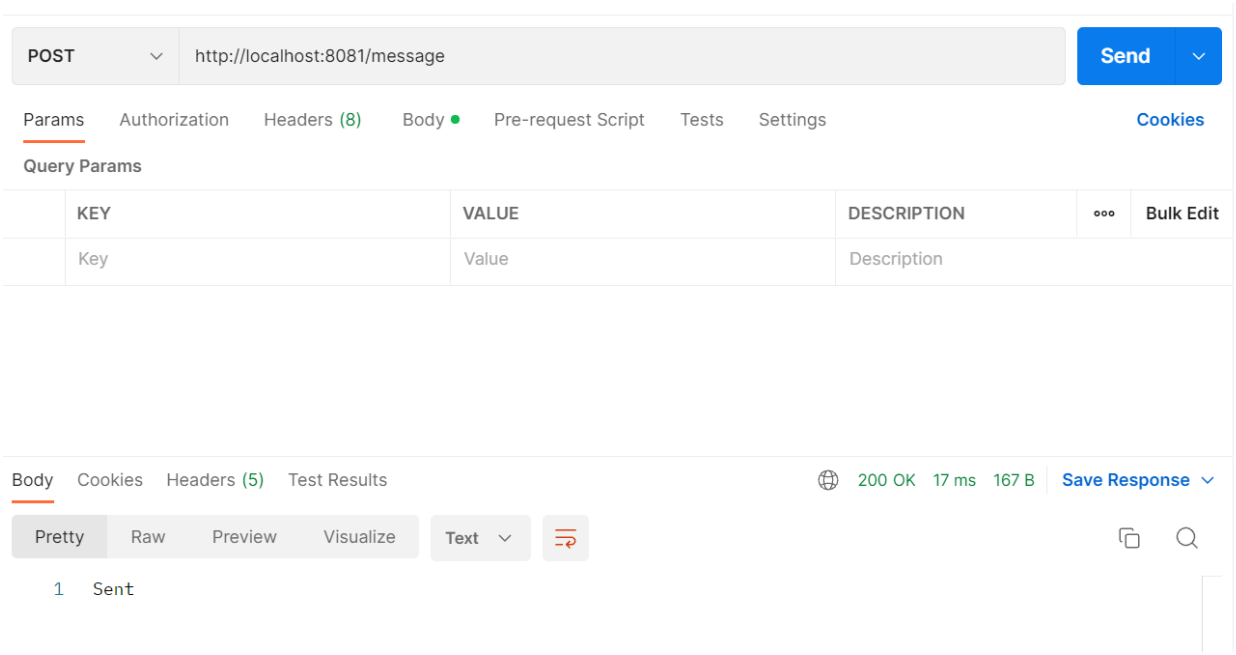
```
@JmsListener( destination = "${spring.activemq.topic-name}", subscription = "batko")
  public void messageListener(String message){
     log.info("Message received, {}",message);
  }
```

Перейдемо до сервісу, що є відповідальним за надсилання повідомлень. За цю задачу відповідальний метод sendToTopic

```
  public void sendToTopic(Message message) throws JsonProcessingException {
     jmsTemplate.convertAndSend(topic, mapper.writeValueAsString(message));
  }
```

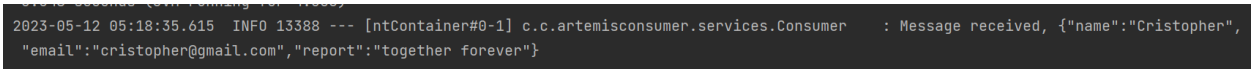
Також в проекті споживача є клас контролера, в якому визначений HTTP POST ендпоінт з якого можна надсилати повідомлення.

Для тестування REST ендпоінту використовується Postman. Нижче наведений тестова відправка.



(Рисунок 3.3 – тестування POST запиту в Postman)

Отримане повідомлення споживачем.



(Рисунок 3.4 – результат отримання повідомлення)

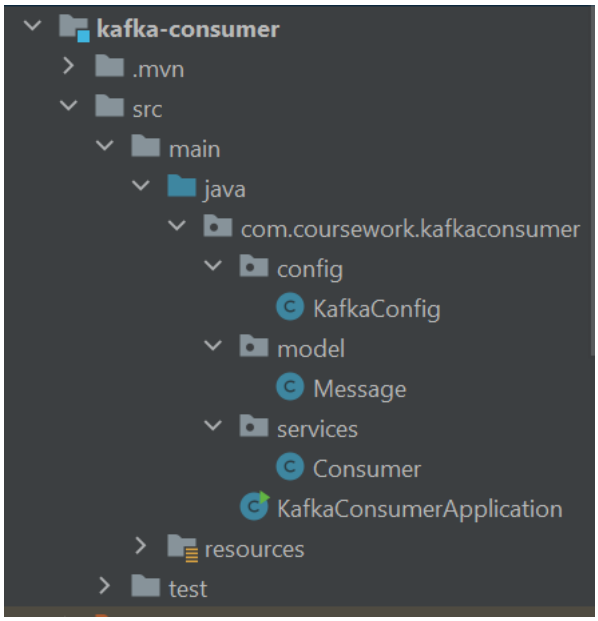
3.2 Kafka

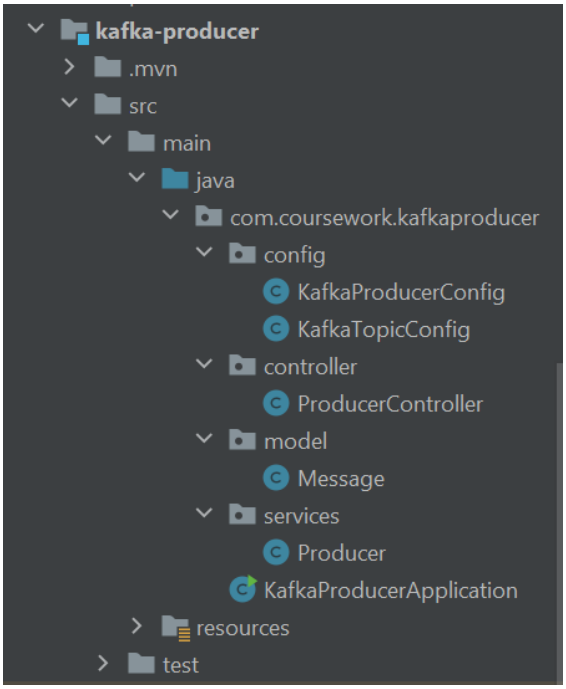
Програма має наступну структуру : проект споживача, проект виробника та файл docker-compose.yml для того, щоб розпочати сервер Kafka в докер контейнері.

(Рисунок 3.5 – структура програми для брокера повідомлення Kafka)

В файлі docker-compose.yml налаштовує мережу Docker і розгортає два контейнери ZooKeeper і Kafka. ZooKeeper виконує роль централізованого контролера для керування всіми даними про Kafka брокерів, виробників та споживачів. Контейнери під'єднані до мережі kafka-net, що дозволяє їм спілкуватися один з одним за допомогою імен своїх сервісів.

Споживач та виробник мають наступну структуру ::





(Рисунок 3.6 – Структура проектів Kafka споживача та виробника)

В конфігураційних класах оголошені біни для налаштування Kafka виробника, споживача та створення тем.

Клас Message описує структура повідомлень, які виробник надсилатиме споживачу.

Метод за допомогою якого Kafka виробник надсилає повідомлення виглядає наступним чином

```
public void sendMessage(Message message) {
        messageKafkaTemplate.send(topicName, message);
 }
```

Метод в споживачі анотований @KafkaListener є слухачем повідомлень

```
  @KafkaListener(topicPartitions = @TopicPartition(topic = "${topic.message.name}",
          partitions = { "0", "3" }),
          groupId = "messages",
          containerFactory = "messageKafkaListenerContainerFactory")
```

```
    public void listenMessage(@Payload Message message,
                              @Header(KafkaHeaders.RECEIVED_PARTITION_ID) int partition)
{
        log.info(String.format("\nReporter : %s\nEmail: %s\nReport: %s\nPartition: %s",
                message.getName(), message.getEmail(), message.getReport(), partition));
    }
```

Анотація також має в собі ряд атрибутів, які можна задати

topicPartitions - використовується для визначення теми та розділів для прослуховування. Він встановлюється за допомогою анотації @TopicPartition. Назва теми отримується з файлу властивостей програми Spring за допомогою ${topic.message.name}, а розділи вказуються як масив зі значеннями «0» і «3». Це означає, що слухач споживатиме повідомлення лише з розділів 0 і 3 зазначеної теми.

groupId - визначає ідентифікатор групи споживачів. Група споживачів — це спосіб об’єднати споживачів і розподілити між ними робоче навантаження для певної теми.

containerFactory визначає ім’я bean-компонента, який надає фабрику контейнерів слухача Kafka (оголошено в конфігураційному файлі).

Також функція має два анотованих параметри message і partition

message анотований @Payload, що репрезентує корисне навантаження повідомлення. В конкретному випадку має тип Message, вказуючи, що повідомлення має бути десеріалізовано в об’єкт цього типу.

partition анотований @Header і вказує, що отриманий ідентифікатор розділу слід додати до цього параметра. Це дозволяє отримати доступ до розділу, з якого було спожито повідомлення.

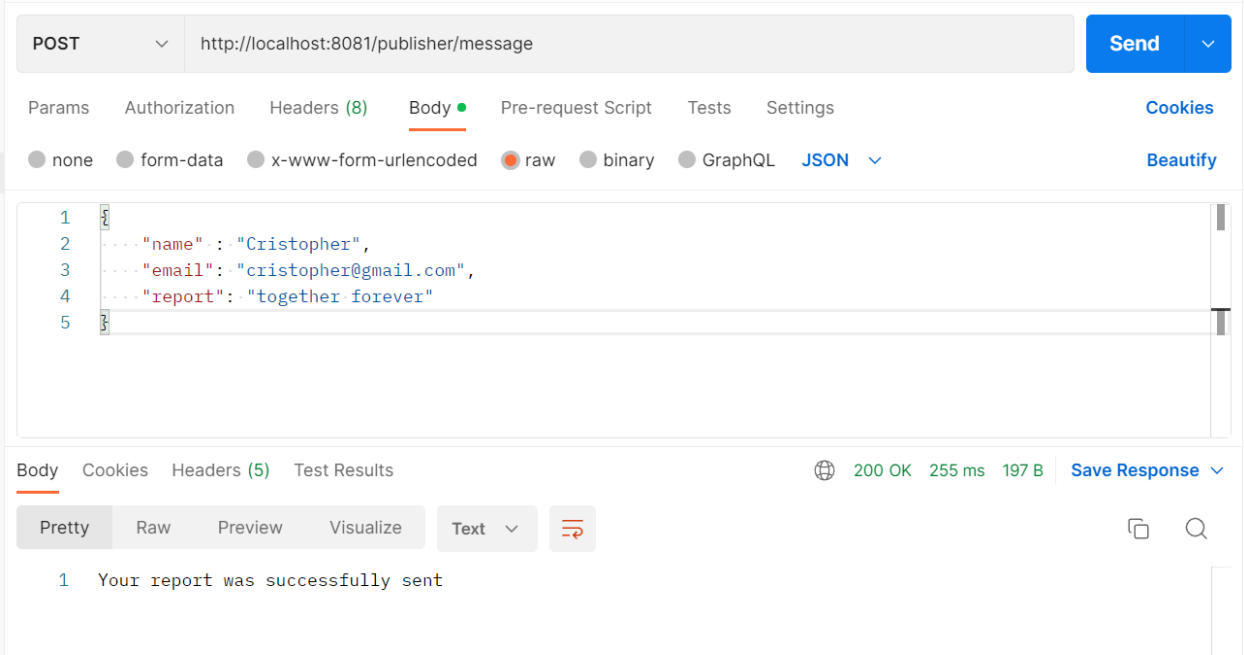
Виробник також містить клас ProducerController для обробки запитів HTTP, що відповідають за публікацію повідомлень.

Тестування застосунку

Для того, щоб запустити брокер повідомлень в терміналі в директорії в якій знаходиться файл docker-compose.yml небохідно виконати команду docker compose up, що заранить брокер повідомлень.

Далі необхідно запустити порграму користувача та програму споживача.

Для тестування я використовуватиму Postman роблячи запит на @POST ендпоінт, визначений в контролері.



(Рисунок 3.7 – тестування POST запиту в Postman)

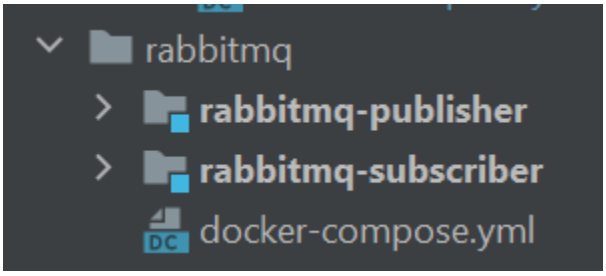
Отримане повідомлення в консолі споживача

```
2023-05-12 22:00:50.107  INFO 26552 --- [ntainer#0-0-C-1] c.c.kafkaconsumer.services.Consumer
Reporter : Cristopher
Email: cristopher@gmail.com
Report: together forever
Partition: 3
```

(Рисунок 3.8 – результат отримання повідомлення)

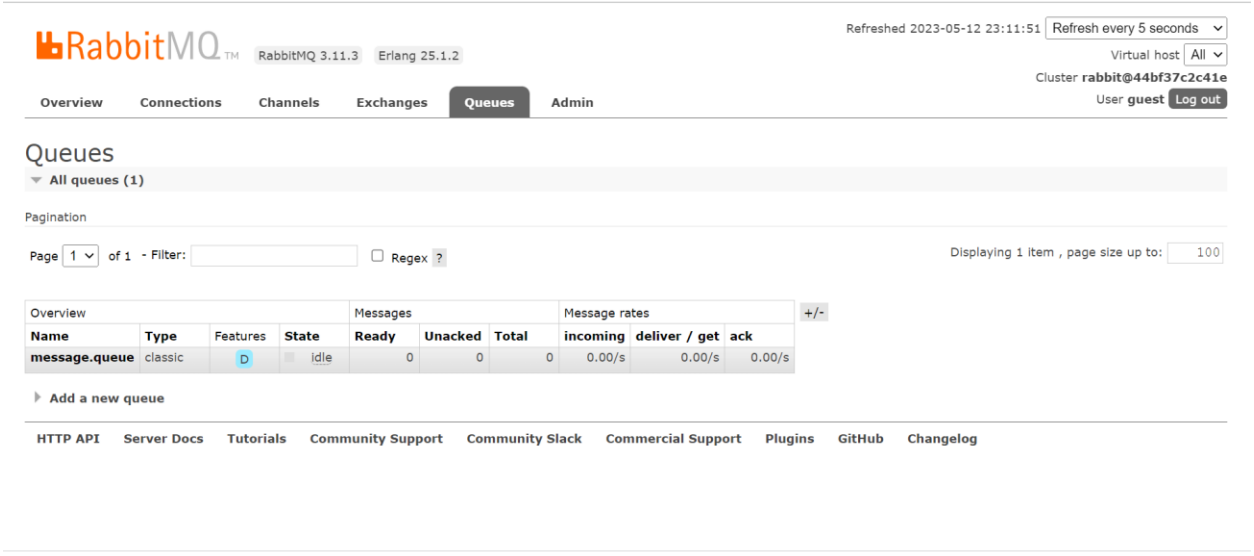
3.3 RabbitMQ

Програма має наступну структуру : проект споживача, проект виробника та файл docker-compose.yml для того, щоб розпочати сервер RabbitMQ в докер контейнері.



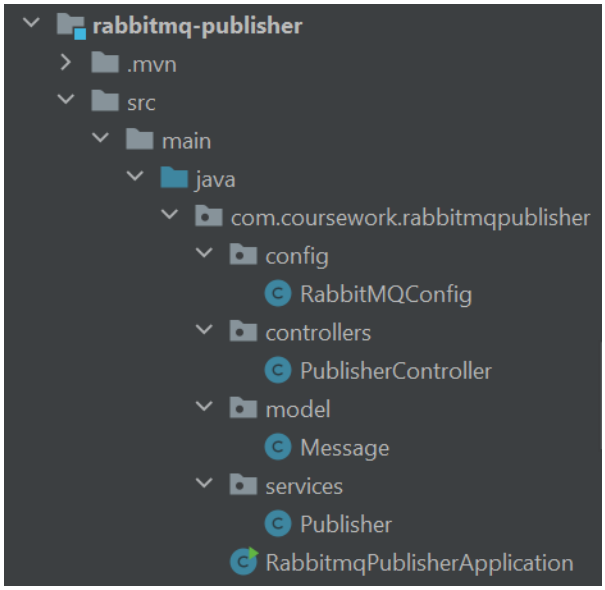
(Рисунок 3.9 – структура програми для брокера повідомлення RabbitMQ)

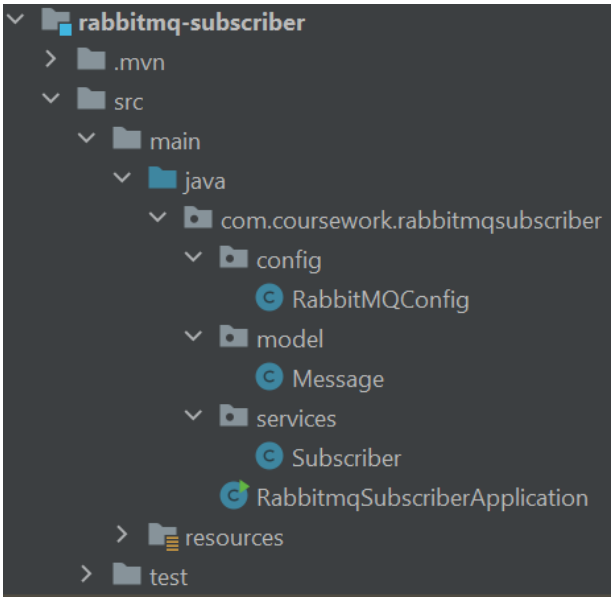
Після запуску брокеру повідомлень за допомогою команди docker compose up, за адремою localhost:15672 стає доступним RabbitMQ management UI, за допомогою якого можна зручно керувати та переглядати вузли і кластери RabbitMQ разом із інтерфейсом користувача



(Рисунок 3.10 – графічна консоль RabbitMQ)

Програми виробника та споживача мають наступну структуру





(Рисунок 3.11 – Структура проектів RabbitMQ споживача та виробника)

В конфігураційних класах оголошені bean-компоненти для з’єднання з RabbitMQ: створення фабрики з’єднань, черг, обмінів, правил зв’язування, RabbitTemplate для надсилання та отримання повідомлень, конвертеру для сервалізації JSON-ів.

Щодо безпоередньої відправки та отримання повідомлень розглянемо наступні методи

Метод в виробнику відповідальний за відправку повідомлень

```
public void publish(Message message) throws JsonProcessingException {

    rabbitTemplate.convertAndSend(exchange, routingkey, message);
```

```
}
```

Як ми бачимо в методі компоненти rabbitTemplate, окрім повідомлення, містяться параметри exchange та routingkey, які вказують певний тип обміну та ключ маршрутизації, що визначають як повідомлення буде відправлятись.

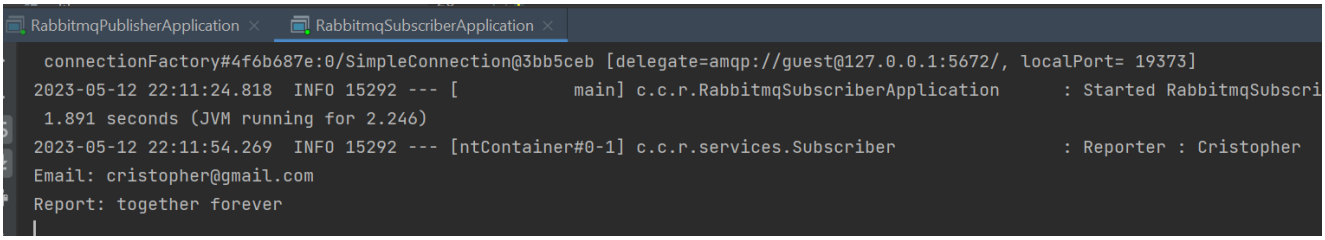
За отримання повідомлення у споживачі відповідає метод receivedMessage анотований @RabbitListener, що позначає метод як слухача RabbitMQ. Також анотація містить атрибут queues, що визначає черги, з яких слухач отримуватиме повідомлення.

```
@RabbitListener(queues = "${spring.rabbitmq.queue}")
public void receivedMessage(Message message) {
    log.info(String.format("Reporter : %s\nEmail: %s\nReport: %s", message.getName(),
            message.getEmail(), message.getReport()));
}
```

В проектах також є клас Message, що визначає модель повідомлення та контролер, який містить POST ендпоінт, за допомогою якого можна надсилати повідомлення.

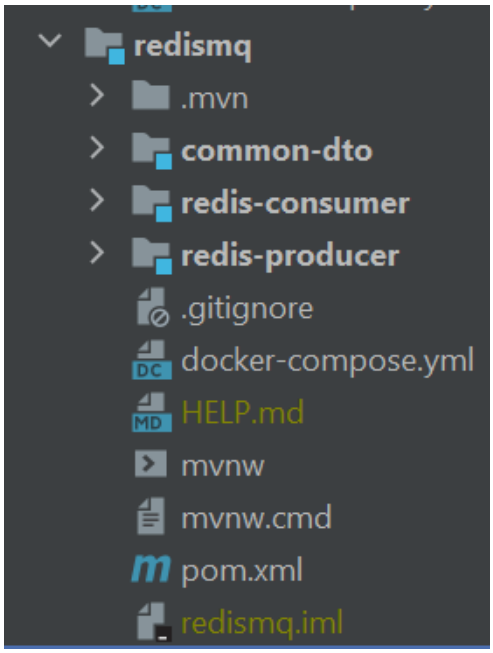
Для тестування програми необхідно запустити брокер повідомлень в докері, розпочати проекти споживача/ів та виробника.

Аналогічно до попереднього тестування надсилаємо POST запит на відповідний ендпоінт та бачимо ортимане повідомлення у споживача



(Рисунок 3.12 – результат отримання повідомлення)

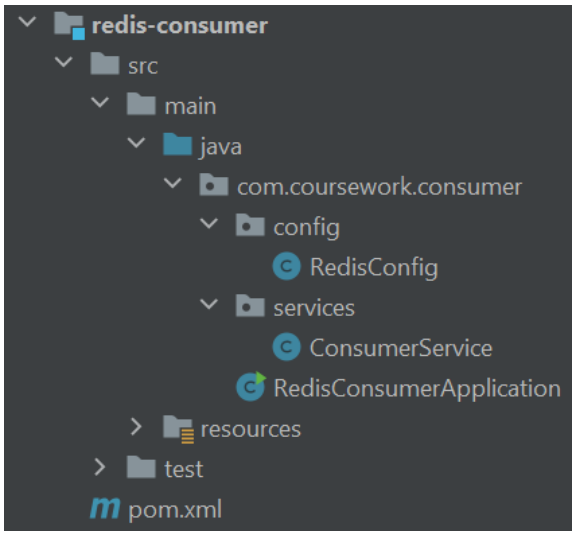
3.4 Redis

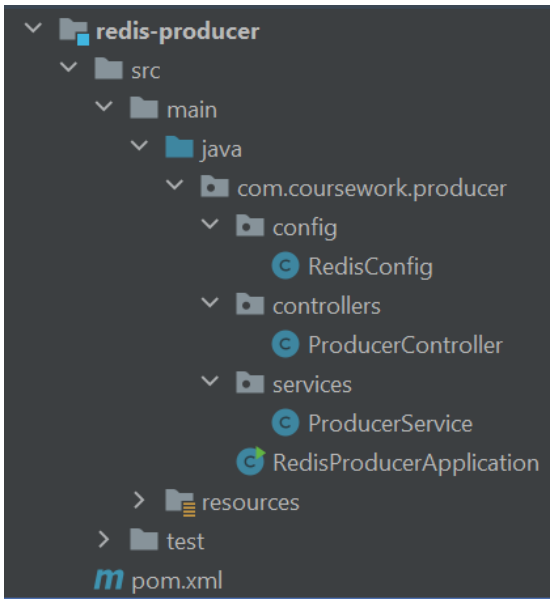


Для тестування Redis було реалізовано мультимодульний Spring Boot проект, що складається з виробника, споживача та спільного dto (data transport object), в якому міститься модель повідомлення, що слугує як основна структура даних повідомлень в спілкуванні двох сторін. Окрім того, проект містить файл docker-compose.yml, за допомогою якого можна розпочати сервер Kafka в докер контейнері.

(Рисунок 3.13 – структура програми для брокера повідомлення Redis)

Програми споживача та виробника мають наступну структуру:





(Рисунок 3.14 – Структура проектів Redis споживача та виробника)

В конфігураційних файлах визнаено bean-компоненти для з’єднання з брокером повідомлень та серіалізації JSON об’єктів.

У виробнику за відправку повіломлень відопвідає метод publish. Розглянемо його детальніше

```
public Mono<String> publish(Message message) {

return redisTemplate.convertAndSend(topicName, message)

    .map(r -> "Message was successfully sent");

}
```

Тип даних, що повертає метод Mono<String> . Mono — це тип реактивних даних, який видає одне значення (у цьому випадку повідомлення про успішне завершення) після завершення операції публікації.

Слід зауважити, що для того, щоб виконувати роль брокера повідомлень, Redis має працювати в реактивному режимі. За патерном pub/sub споживачі підписуються на один або більше каналів, і отримують сповіщення асинхронно щоразу, коли нове повідомлення публікується в цих каналах. Ця асинхронна комунікація добре узгоджується з парадигмою реактивного програмування.

Відправку повідомлень виконує метод компоненти redisTemplate convertAndSend, який, окрім самого повідомлення, приймає також параметр topicName, що визначає тему, в яку надсилатиметься повідомлення.

Далі розглянуто метод в споживачі, за допомогою якого відбувається підписка на тему каналу Redis. В методі використано redisTemplate, який перетворює отримані повідомлення з теми каналу в об’єкти Message та підписується на потік перетворених даних. Це довзоляє реактивно прослуховувати повідомлення та виконувати дії після їх отримання.

```
@PostConstruct
 private void subscribe() {
    redisTemplate
        .listenTo(ChannelTopic.of(topicName))
        .map(ReactiveSubscription.Message::getMessage)
        .subscribe(message -> log.info(
            String.format("Reporter : %s\nEmail: %s\nReport: %s", message.getName(),
            message.getEmail(), message.getReport())));
 }
```

Також в модулі виробника знаходить клас контролера, який містить ендпоінт для відправки повідомлень.

Подібно до тестування попередніх проектів, за допомогою команди docker compose up запускаємо брокер повідомлень, розпочинаємо програми виробника та споживача.

Використовуючи Postman надсилаємо повідомлення та бачимо його у споживача.

(Рисунок 3.15 – результат отримання повідомлення)

3.5 Підсумки реалізації проектів

Після реалізації проектів можна підвести підсумки щодо зручності використання кожного з брокерів повідомлень.

ActiveMQ Artemis із багатим набором функцій забезпечує зручний інтерфейс, особливо для тих, хто добре розбирається в JMS. API, які він надає для роботи з повідомленнями, прості та добре задокументовані. Як частина набору Apache, ActiveMQ має досить велику спільноту, у розмірі 200 контрібютерів на Github-і [3] та загалом 400 тисяч завантажень на DockerHub різних образів брокеру повідомлень [4]. Цифри не маленькі, проте не порівнювані з відповідними RabbitMQ або Kafka.

RabbitMQ є зручним у користуванні. Він підтримує кілька протоколів обміну повідомленнями та пропонує клієнтські бібліотеки для широкого діапазону мов програмування. Його початкове налаштування нескладне, а в консолі керування досить легко орієнтуватися. Спільнота RabbitMQ велика та активна, на проектах RabbitMQ Github-у на сьогодні налічується близько 700 котнрібютерів [3], а кількість завантажень офіційного образу на DockerHub

сягнула 1 біліона [4]. Підсумовуючи, без проблем можна знайти рішення для виникаючих проблем.

Налаштування та робота з Redis відносно прості, але як імпровізований брокер повідомлень він не має розширених функцій для більш комлпексних випадків. Він підтримує модель pub/sub і базові черги через структури списків, але може бути недостатнім для повноцінної роботи в якості брокера повідмлень. Redis має величезну спільноту з понад 2000 контрибюторів [3] на GitHub і мільйонами завантажень на місяць [4]. Однак, більшість цих завантажень пов'язані з його використанням як бази даних у пам'яті, а не брокера повідомлень. Інформації про нього у ролі брокера повідомлень може бути недостатньо.

Kafka потужний інструмент для роботи з великими обсягами потоків даних, який в той же час є досить складним в налаштуванні та використанні. Його складність компенсується процвітаючою спільнотою, з показником контрібютерів на Github-і в майже 1000 осіб на момент 2023 [3], 5 мільйонами локальних завантажень за життя проекту та більше 100 мільйонів одного з образів Kafka на DockerHub-і [4], що результує у велику кілкьість ресурсів та активні онлайн-розмови.

В контексті використання згаданих брокерів повідомлень разом з Spring Boot і Java, усі вони пропонують чудову інтеграцію та підтримку, безліч модулей та бібліотек. RabbitMQ і Kafka можуть мати невелику перевагу в підтримці спільноти та доступності ресурсів через їх широке промислове використання.

Висновок

На основі проведеного дослідження та порівняння брокерів повідомлень Kafka, RabbitMQ, ActiveMQ Artemis та Redis, я прийшла до наступного висновку: вибір брокера повідомлень необхідно засновувати на конкретних вимогах до проекту, його масштабі, цілях та специфікаціях.

Kafka, з своєю високою пропускною здатністю (мільйон повідомлень за секунду), ідеально підходить для ситуацій, де необхідно зберігати історію повідомлень для подальшого відтворення, обробки великих потоків даних, а також для відслідкування активності. Хоч цей інстурмент є досить складним для опанування та використання, він має потужну спільноту, де одне число контрібюторів проекту сягає 1000 осіб. Через широке використання в індустрії Kafka весь час розвивається та покращується, що робить його дуже поргресивним фреймворком.

RabbitMQ, хоча й вимагає більше ресурсів для досягнення такої ж пропускної здатності як Kafka, найкраще підходить для випадків, де необхідна комплексна маршрутизація, гарантія отримання повідомлення від споживача, або в системах, де кожне повідомлення вимагає особливої обробки чи послідовності. Цей фреймворк легко налаштовувати та використовувати, а про його популярність свідчить 1 біліон скачувань офіційного образу на DockerHub-i.

ActiveMQ Artemis, ефективний для випадків невеликої кількості даних, що вимагають високого рівня транзакційності та надійності, а також для

трансформації даних on-the-fly та імплементації ETL процесів. Він має достатню спільноту для того, щоб знайти необхідну інформацію для вирішення виникаючих проблем, а також є досить легким та зручним у користуванні.

Redis, хоча й обмежений у обробці великої кількості даних та більш комплексної маршрутизації, корисний у випадках, коли потрібна миттєва відправка повідомлень в невеликих обсягах та з можливими втратами. Хоч спільнота Redis-у дуже велика, переважка кількість користувачів вибирають його як базу даних у пам’яті, тому інформації інколи буває недостатньо. Загалом, цей інструмент не підходить для повноцінної роботи в ролі брокера повідомлень, та в багатьох випадках є радше допоміжним.

Кожен з цих фреймворків має свої особливості, тому вибір конкретного має основуватись на вимогах відповідного проекту. Ця робота допомогла краще зрозуміти, які переваги та недоліки в кожного з досліджуваних брокерів повідомлень та в яких ситуаціях кожен з них може стати найкращим вибором.

Список використаної літератури

1. Apache ActiveMQ vs. Kafka [Електронний ресурс] // Baeldung. – 2022. – Режим доступу до ресурсу: https://www.baeldung.com/apache-activemq-vs-kafka.
2. Barmin A. Introduction to message brokers. Part 1: ApacheKafka VS RabbitMQ [Електронний ресурс] / Artem Barmin // Freshcodeit – Режим доступу до ресурсу: https://freshcodeit.com/blog-introduction-to-message-brokers-part-1-apache-kafka-vs-rabbitmq.
3. Contributions [Електронний ресурс] // Github. – 2023. – Режим доступу до ресурсу: https://github.com/.
4. Downloads [Електронний ресурс] // DockerHub. – 2023. – Режим доступу до ресурсу: https://hub.docker.com/.
5. Hegde R. G. Low Latency Message Brokers / R. G. Hegde, N. G. S. // International Research Journal of Engineering and Technology. – 2020. – №7.
6. Jena M. Message Brokers: Key Models, Use Cases & Tools Simplified 101 [Електронний ресурс] / Manisha Jena // Hevo. – 2022. – Режим доступу до ресурсу: https://hevodata.com/learn/message-brokers/.
7. Joao D. A. Network abstractions and emulation for distributed systems : CS / Joao Daniel Almeida – NOVA University Lisbon, 2023.
8. Levy E. Kafka vs. RabbitMQ: Architecture, Performance & Use Cases [Електронний ресурс] / Eran Levy // Upsolver. – 2022. – Режим доступу до ресурсу: https://www.upsolver.com/blog/kafka-versus-rabbitmq-architecture-performance-use-case.

9. Łuczak B. 5 use cases of message brokers. When you should consider adopting a message broker in your system? [Електронний ресурс] / Bartosz Łuczak // The Software House. – 2021. – Режим доступу до ресурсу: https://tsh.io/blog/message-broker/.
10. Maarek S. Comparing Apache Kafka, ActiveMQ, and RabbitMQ [Електронний ресурс] / Stéphane Maarek // Conductor. – 2022. – Режим доступу до ресурсу: https://www.conduktor.io/blog/comparing-apache-kafka-activemq-and-rabbitmq/.
11. Magnoni L. Modern Messaging for Distributed Sytems / Magnoni. // Journal of Physics. – 2014. – №608.
12. Malhotra R. Java Messaging / Raj Malhotra // Rapid Java Persistence and Microservices / Raj Malhotra., 2019.
13. Mcclain B. Understanding the Differences Between RabbitMQ vs Kafka [Електронний ресурс] / Brian Mcclain // VMware. – 2020. – Режим доступу до ресурсу: https://tanzu.vmware.com/developer/blog/understanding-the-differences-between-rabbitmq-vs-kafka/.
14. Queue Orchestration Using an In-Memory Broker / Ganesh Shreyas – IEEE 2nd International Conference on Mobile Networ, 2022.
15. Raje S. N. Performance Comparison of Message Queue Methods / Raje Sanika N. – University of Nevada, Las Vegas.
16. What is a message broker? [Електронний ресурс] // IBM – Режим доступу до ресурсу: https://www.ibm.com/topics/message-brokers.
17. When to use RabbitMQ or Apache Kafka [Електронний ресурс] // CloudAMQP. – 2022. – Режим доступу до ресурсу:

https://www.cloudamqp.com/blog/when-to-use-rabbitmq-or-apache-kafka.html.

18. Benchmarking Apache Kafka, Apache Pulsar, and RabbitMQ: Which is the Fastest? [Електронний ресурс] // Confluent. – 2022. – Режим доступу до ресурсу: https://www.confluent.io/blog/kafka-fastest-messaging-system/.