

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

**Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії**

«На правах рукопису»
УДК 004.021

«До захисту допущено»
В.о. завідувача кафедри
_____ Едуард ЖАРІКОВ
«__» _____ 2021 р.

**Магістерська дисертація
на здобуття ступеня магістра
за освітньо-професійною програмою «Інженерія програмного забезпечення
комп'ютерних систем»
зі спеціальності 121 «Інженерія програмного забезпечення»
на тему: «Метод та програмні засоби планування та управління виконанням
мікросервісних задач»**

Виконав (-ла):
студент (-ка) II курсу, групи ІТ-03мп
Смірнов Олександр Михайлович _____

Керівник:
доцент кафедри ІПІ, к.т.н., доц.
Ліхоузова Тетяна Анатоліївна _____

Рецензент:
доцент кафедри ІСТ, к.т.н.
Букасов Максим Михайлович _____

Засвідчую, що у цій магістерській дисертації
немає запозичень з праць інших авторів без
відповідних посилань.

Студент (-ка) _____

Київ – 2021 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних систем»

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

_____ Едуард ЖАРІКОВ

« ____ » _____ 2021р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Смірнову Олександру Михайловичу

1. Тема дисертації «Метод та програмні засоби планування та управління виконанням мікросервісних задач», науковий керівник дисертації Ліхоузова Тетяна Анатоліївна, к.т.н., доцент, затверджені наказом по університету від «27» жовтня 2021 р. № 3587-с

2. Термін подання студентом дисертації «6» грудня 2021 р.

3. Об'єкт дослідження – програмне забезпечення системи моніторингу та управління мікросервісними задачами.

4. Предмет дослідження – сервіси, що виконують запуск простих та періодичних мікросервісних задач.

5. Перелік завдань, які потрібно розробити – аналіз існуючих рішень; створення методів конфігурацій для планування запуску задач; побудова системи керування запусками задач; створення системи моніторингу; оцінка ефективності та тестування запропонованого рішення.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу – 4 плакати.

7. Орієнтовний перелік публікацій – одна публікація

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання «30» вересня 2020 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання	Примітка
1	Огляд літератури	30.09.2020	
2	Аналіз існуючих рішень	02.02.2021	
3	Створення архітектури для системи	05.04.2021	
4	Розробка програмного забезпечення	08.09.2021	
5	Оцінка ефективності	20.10.2021	
6	Виконання експериментальних досліджень	25.10.2021	
7	Розробка стартап-проекту	08.11.2021	
8	Оформлення пояснювальної записки	10.11.2021	
9	Подання дисертації на попередній захист	22.11.2021	
10	Подання дисертації на захист	06.12.2021	

Студент

Олександр СМІРНОВ

Науковий керівник

Тетяна ЛІХОУЗОВА

РЕФЕРАТ

Розмір пояснювальної записки – 74 аркуші, містить 13 ілюстрацій, 27 таблиць, 7 додатків.

Актуальність теми. У роботі розглянуто проблему в області проектування мікросервісної архітектури з програмним забезпеченням для запуску простих та періодичних мікросервісних задач, показано основні особливості існуючих рішень проблеми, їх переваги та недоліки. Виявлено потребу в удосконаленні методу побудови мікросервісної архітектури з урахуванням виконання та моніторингу розподілених задач.

Мета дослідження. Підвищити зручність та розширити функціонал сервісу для моніторингу та управління виконанням задач в мікросервісному середовищі.

Об'єкт дослідження: програмне забезпечення системи моніторингу та управління мікросервісними задачами.

Предмет дослідження: сервіси, що виконують запуск простих та періодичних мікросервісних задач.

Для реалізації поставленої мети **сформульовані наступні завдання:**

- аналіз існуючих рішень;
- створення методів конфігурацій для планування запуску задач;
- побудова системи керування запусками задач;
- створення системи моніторингу;
- оцінка ефективності розробленої системи.

Наукова новизна результатів магістерської дисертації полягає в тому, що удосконалено метод побудови мікросервісної архітектури з урахуванням виконання та моніторингу розподілених задач та запропоновано відповідне програмне забезпечення. Результат досягнутий шляхом розробки сервісу для моніторингу та управління виконанням задач в мікросервісному середовищі.

Практичне значення отриманих результатів полягає в тому, що створено автономний сервіс для управління виконанням задач у інших сервісах.

Зв'язок з науковими програмами, планами, темами. Робота виконувалась на кафедрі інформатики та програмної інженерії Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського".

Апробація. Наукові положення дисертації пройшли апробацію на X Міжнародній науково-практичній конференції «SCIENCE FOUNDATIONS OF MODERN SCIENCE AND PRACTICE» – м. Афіни, Греція.

Публікації. Наукові положення дисертації опубліковані в:

Смірнов О.М. Методи та програмні засоби планування і управління виконанням мікросервісних задач / О.М. Смірнов // Матеріали X Міжнародної науково-практичної конференції «SCIENCE FOUNDATIONS OF MODERN SCIENCE AND PRACTICE» – Афіни, Греція, 23-26 листопада 2021р.

Ключові слова: ПЛАНУВАННЯ, ПЛАНУВАЛЬНИК, ВИКОНАННЯ ЗАДАЧ, МІКРОСЕРВІСНА АРХІТЕКТУРА, АЛГОРИТМИ.

ABSTRACT

Explanatory note size – 74 pages, contains 13 illustrations, 27 tables, 7 applications.

Topicality. The paper considers the problem in the field of microservice architecture design with software for running simple and periodic microservice tasks, shows the main features of existing solutions to the problem, their advantages and disadvantages. The need to improve the method of building a microservice architecture, taking into account the implementation and monitoring of distributed tasks.

The aim of the study. The main target is to increase convenience and expand the functionality of the service for monitoring and managing the performance of tasks in a microservice environment.

Object of research: the software for monitoring and management of microservice tasks.

Subject of research: services that perform simple and periodic microservice tasks.

To achieve this goal, the **following tasks** were formulated:

- analysis of existing solutions;
- creating a configuration methods to plan the start of tasks;
- building a task start management system;
- creation of a monitoring system;
- evaluation of the effectiveness of the developed system.

The scientific novelty of the results is that the method of building a microservice architecture is improved, taking into account the implementation and monitoring of distributed tasks, and the appropriate software is proposed. The result was achieved by developing a service for monitoring and managing the implementation of tasks in a microservice environment.

The practical value of the obtained results is that an autonomous service has been created to manage the execution of tasks in other services.

Relationship with working with scientific programs, plans, topics. Work was performed at the Department of Informatics and Software Engineering of the National Technical University of Ukraine «Kyiv Polytechnic Institute. Igor Sikorsky».

Approbation. The scientific provisions of the dissertation were tested at the X International Scientific and Practical Conference "SCIENCE FOUNDATIONS OF MODERN SCIENCE AND PRACTICE" - Athens.

Publications. The scientific provisions of the dissertation published in:

Smirnov O. M. Methods and software for planning and managing the performance of microservice tasks / O. M. Smirnov // Proceedings of the X International Scientific and Practical Conference "SCIENCE FOUNDATIONS OF MODERN SCIENCE AND PRACTICE" - Athens, Greece, November 23-26, 2021.

Keywords: PLANNING, SCHEDULER, TASKS, MICROSERVICE ARCHITECTURE, ALGORITHMS.

ЗМІСТ

ВСТУП.....	11
1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ.....	13
1.1 Види планування задач.....	13
1.2 Алгоритми планування задач.....	14
1.3 Аналіз існуючих рішень	16
Висновки по розділу	18
2 ОПИС ТА АНАЛІЗ ВИМОГ ДО СИСТЕМИ.....	19
2.1 Нефункціональні вимоги до системи.....	19
2.2 Функціональні вимоги до системи	21
Висновки до розділу	25
3 ОБРАНІ МЕТОДИ, АЛГОРИТМИ ТА ІНСТРУМЕНТИ.....	26
3.1 Визначення парадигми програмування.....	27
3.2 Збереження даних	29
3.3 Реєстрація задач.....	31
3.4 Планування та запуск	32
3.5 Моніторинг результатів виконання задач	33
3.6 Проектування програмних інтерфейсів.....	33
3.7 Комунікація з клієнтськими сервісами	34
Висновки до розділу	36
4 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНИХ ЗАСОБІВ.....	37
4.1 Збереження даних	37
4.2 Реєстрація задач.....	40
4.3 Планування та запуск	42

4.4	Моніторинг та управління виконанням задач.....	48
4.5	Комунікація з клієнтськими сервісами	50
4.6	Оцінка ефективності запропонованого рішення.....	53
	Висновки до розділу	56
5	МАРКЕТИНГОВИЙ АНАЛІЗ СТАРТАП-ПРОЄКТУ.....	57
5.1	Опис ідеї проєкту.....	57
5.2	Технологічний аудит ідеї проєкту	58
5.3	Аналіз ринкових можливостей запуску стартап-проєкту	59
5.4	Розроблення ринкової стратегії проєкту	65
5.5	Розроблення маркетингової програми стартап-проєкту.....	68
	Висновки по розділу	70
	ВИСНОВКИ.....	71
	СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	73
	ДОДАТОК А.....	75
	ДОДАТОК Б.....	76
	ДОДАТОК В.....	77
	ДОДАТОК Г.....	78
	ДОДАТОК Д.....	79
	ДОДАТОК Е.....	80
	ДОДАТОК Є.....	85

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – програмний інтерфейс

HTTP – протокол передачі гіпертекстових документів

JPQL – стійка мова запитів Java

JSON – запис об'єктів JavaScript

ORM – об'єктно реляційне відображення

REST – підхід до архітектури мережових протоколів «передача репрезентативного стану»

SQL – мова структурованих запитів

XML – розширена мова розмітки

YAML – мова розмітки, назва це рекурсивний акронім «YAML не мова розмітки»

ООП – об'єктно орієнтоване програмування

СУБД – система управління базами даних

ВСТУП

Актуальність теми. Сьогодні є актуальною проблема запуску відкладених задач у сучасних мікросервісних середовищах. Високонавантажені системи потребують періодичного чи поодинокого проведення комплексних та простих робіт. У якості об'єктів проведення робіт виступають мікросервіси, що набули популярності у побудові сучасної масштабованої системи. Одним з головних принципів проектування такої архітектури є розбиття однієї цілісної системи на малі незалежні компоненти. Ці компоненти оперують власними наборами даних та мають відповідну роль, проте у таких компонент часто виникає проблема запуску задач, один раз або ж періодично, як наприклад очищення застарілих даних чи очищення кешу. Зазвичай її вирішують додаванням бібліотек до таких компонент, даний підхід вирішує проблему точково, і не дає можливості створювати моніторинг та управління виконанням усіх таких задач.

Об'єкт та предмет дослідження. Об'єктом дослідження є програмне забезпечення для запуску простих та періодичних мікросервісних задач. Предметом дослідження є сервіси, що виконують запуск простих та періодичних мікросервісних задач.

Мета. Підвищити зручність та розширити функціонал сервісу для моніторингу та управління виконанням задач в мікросервісному середовищі.

Завдання. Планується на базі механізму запуску задач та фреймворку для розробки автономних програм створити автономний сервіс для управління виконанням задач у інших сервісах (створення, корегування та скасування) та моніторинг статусу запущених задач. Обрати або розробити найбільш оптимальний варіант синхронізації даного сервісу при масштабуванні. Побудувати модель комунікації з іншими сервісами.

Задачі, що вирішуються:

- аналіз існуючих рішень
- побудова моделі комунікації з іншими сервісами
- побудова системи керування

- створення системи моніторингу
- оцінка ефективності розробленої системи

Наукова новизна. Удосконалено метод побудови мікросервісної архітектури з урахуванням виконання та моніторингу розподілених задач. Запропоновано відповідне програмне забезпечення.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

Мікросервісна архітектура – це підхід, у якому єдина система визначена як набір слабо пов'язаних невеликих сервісів, підтримуючих незалежного розгортання. Зазвичай мікросервіси мають власний стек технологій, взаємодіють між собою за допомогою REST API, потоків подій та агентів повідомлень [1]. Ключовими перевагами мікросервісної архітектури є зменшення затрат людських ресурсів на оновлення коду, використання різних технологій в залежності від потреб, незалежне масштабування компонентів.

Кількість мікросервісів у хмарних середовищах збільшується і актуальність проблеми запуску відкладених задач у сучасних мікросервісних середовищах теж зростає. Високонавантажені системи потребують періодичного чи поодинокого проведення комплексних та простих робіт, таких як створення звітів, очищення застарілих даних і т. д. В залежності від вимог користувачі потребують різних методів та алгоритмів якісного планування задач виконання цих робіт. В даному розділі ми розглянемо існуючі методи та алгоритми планування задач та розглянемо програмні засоби для виконання їх в мікросервісній архітектурі.

1.1 Види планування задач

Планування в мікросервісній архітектурі класифікується за різними категоріями. Першою категорією є конфігурація задачі. За цією категорією планування задач поділяється на статичне та динамічне [2].

При статичному плануванні задача надходить до ядра, що несе відповідальність за її запуск, після запуску задача направляється до вільного ресурсу, що має її виконати. Після закінчення виконання задачі час обробки оновлюється. У такому випадку рішення про виконання задачі приймається до планування, і система планування та виконувач є чітко визначені.

У разі динамічного планування ці дані розподілу цих ресурсів не фіксуються. Час прибуття задач не відомий до подачі. Динамічне планування поділяється на два режими: онлайн та комплексний. Онлайн режим передбачає планування нових задач, коли вони надходять до системи планування. У комплексному режимі, задачі зберігаються та упорядковуються в чергу, збираються в набір та запускаються системою в фіксований період часу.

Наступна категорія базується на визначенні моментів для запуску. За цією категорією планування поділяється на два типи: одноразовий запуск та періодичний запуск. Одноразовий запуск передбачає чітко визначений час виконання задачі, що задається за допомогою мітки часу з визначенням часової зони. Періодичний запуск визначається за допомогою виразів. Серед існуючих рішень найбільш популярні вирази часових інтервалів та cron вираз. Часові інтервали мають різний формат запису в залежності від системи, що запускає задачу, проте усі визначаються безпосередньою вказівкою на проміжок часу в мілісекундах, секундах, хвилинах, годинах тощо. Приклади символічних позначень cron виразу наведені у таблиці 1.1. Вираз cron має вигляд виразу наведеного вигляду:

0/5 14,18,3-39,52 * ? JAN,MAR,SEP MON-FRI 2002-2010
 "секунди" "хвилини" "години" "дні місяця" "місяці" "дні тижня" "роки"

1.2 Алгоритми планування задач

Алгоритми планування відіграють важливу роль, а підбір оптимального допомагає зменшити навантаження на системні ресурси та забезпечити паралельне виконання декількох задач. Метою планування є визначення порядку паралельних завдань за допомогою їх оцінки та призначення паралельних задач процесорам, що будуть виконувати задачі у конкретні моменти часу. Існує кілька поширених типів алгоритмів планування:

- 1) Rate monotonic;
- 2) Deadline monotonic;
- 3) Earliest Deadline First (EDF).

Таблиця 1.1 — Символи стоп виразу

Символи	Значення	Опис
*	Всі	Вказує, що подія має відбутися для кожної одиниці часу. Наприклад, у полі "хвилини" має значення кожної хвилини
?	Будь який	Використовується у полях день місяця та "дні тижня" для позначення довільного значення. Наприклад, ми у полі "дні тижня" це означає, що ми хочемо запускати сценарій незалежно від того який день тижня припадає на цю дату
-	Діапазон	Визначає діапазон значень. Наприклад, "10-11" у полі "години" означає 10-ту та 11-ту години
,	Перелік значень	Визначає кілька значень. Наприклад, "MON, WED, FRI" у полі "дні тижня" означає дні понеділок, середа та п'ятниця.
/	Наростання	Визначає зростаючі значення. Наприклад, "5/15" у полі "хвилини" означає 5, 20, 35 та 50 хвилин години
L	Останній	Визначає останній момент величини часового проміжку. Наприклад для поля "дні місяця" означає останній день місяця
W	День тижня	Визначає робочі дні тижня з понеділка по п'ятницю. Наприклад якщо "10W" в полі "дні місяця", і 10-те число припадає на суботу, то умова буде виконуватися в п'ятницю

Rate monotonic надає усім задачам статичний пріоритет у відповідності до їх періоду. У даному алгоритмі задача з найменшим періодом отримує найвищий пріоритет. Такий алгоритм має оптимальну схему, як для статичного алгоритму. Проте даний підхід має ряд недоліків. Такий алгоритм виконує лише періодичні задачі, що не може задовольняти користувачів, які потребують виконання разових задач. Також даний алгоритм визначає для першочергової задачі найвищий пріоритет, що може спричинити некоректну поведінку задач з однаковою періодичністю.

Deadline monolithic чудово підходить, коли задачам з коротким кінцевим терміном варто надати більший пріоритет [3]. Кінцевий термін - це обмеження до якого задача має бути виконана. У даному алгоритмі використовується підхід, у якому завдання отримує пріоритет обернено пропорційний величині її кінцевого терміну. Deadline monolithic передбачає планування, як періодичних, так і одноразових задач.

EDF – це оптимальний алгоритм з динамічним визначення пріоритетів планування, що використовується в системах реального часу [4]. EDF визначає пріоритети завдання відповідно до абсолютного кінцевого терміну. Завдання з найближчим кінцевим терміном отримує найвищий пріоритет, подібно до того як це відбувається у Deadline monolithic. Проте в даному алгоритмі пріоритети розподіляються та змінюються динамічно. EDF є дуже ефективним у порівнянні з іншими алгоритмами планування в системах реального часу. Такий підхід може збільшити використання центрального процесора приблизно до 100%, при цьому гарантуючи терміни виконання всіх завдань.

Серед наведених алгоритмів найкраще для реалізації сервісу планування мікросервісних задач підходить EDF, адже він підтримує одноразове та періодичне виконання задач, що динамічно надходять до системи реального часу.

1.3 Аналіз існуючих рішень

Методи та програмні засоби планування мікросервісних задач за своєю сутністю не можуть існувати без мікросервісної архітектури. Саме тому важливо обрати правильний напрямок розвитку програмних засобів, що зможе забезпечити максимальну пропускну здатність виконання задач та масштабованість у використанні.

Сучасний підхід до розробки компонентів програмного забезпечення за мікросервісною архітектурою передбачає одиничну відповідальність кожного окремого мікросервісу. Адже одинична відповідальність надає беззаперечні

переваги у розробці системи, такі як тестування, масштабування, таку систему легше розробляти загалом [5]. Саме тому виконанням планових та періодичних задач має займатися окремий сервіс, що бере на себе функцію агрегації таких задач.

Найбільш популярними інструментами для запуску та управління мікросервісних задач є Quartz та Obsidian. Кожен з сервісів придатний для запуску одноразових задач та періодичних задач проте мають певні відмінності.

Quartz є досить відомим інструментом, що перевірений роками, адже його перша версія вийшла більше ніж десять років тому. Код даного інструменту є відкритим, тож кожен бажаючий може отримати інформацію з деталей його реалізації, а також запропонувати внесення своїх змін. За замовчуванням Quartz підтримує лише статичну конфігурацію задач, тому при бажанні змінити її під час роботи серверу немає можливості. Проте варто зауважити, що даний інструмент має високу точність та його точність планування досягає секунди. Також дана система створена для роботи з високими навантаження та має високу пропускну здатність. Цей інструмент має можливість працювати у двох режимах, таких як зберігання запланованих задач в пам'яті та в базі даних. У випадку роботи з базою даних підтримується кластеризація.

Obsidian - це комерційний проект, що є платним для використання в комерційних цілях, відповідно його реалізація прихована від користувачів. Obsidian підтримує онлайн зміни в конфігураціях та перепланування задач. Цей інструмент надає зручний інтерфейс користувача, що беззаперечно є плюсом для адміністраторів систем. Проте одним з мінусів Obsidian є точність планування у цьому інструменті вона досягає хвилини. Як і у випадку з Quartz, Obsidian підтримує кластеризацію. Порівняння цих програмних засобів наведено в таблиці 1.2.

Провівши аналіз існуючих рішень варто визнати, що обидва інструменти є хорошими для використання у разі потреби їх цільового призначення. Obsidian пасує системам, що не потребують високої точності та готові витратити на нього

кошти. А Quartz для систем для систем, що не потребують динамічної конфігурації, але натомість отримують високу точність виконання задач. Проте

Таблиця 1.2 — Порівняння програмних засобів планування мікросервісних задач

Функціональність	Quartz	Obsidian
Динамічні конфігурації	-	+
Відкритий код	+	-
Точність планування	секунди	хвилини
Витримка високих навантажень	+	+
База даних є не обов'язковою	+	-
Підтримка кластеризації	+	+

жодний програмний засіб не задовольняє умову, коли потрібний висока точність виконання в секундах та підтримку динамічної зміни конфігурації задач при цьому забезпечити масштабованість даної системи [6].

Висновки по розділу

В даному розділі було розглянуто предметну область та існуючі рішення методів та програмних засобів планування та управління виконанням мікросервісних задач. Було проведено дослідження та виявлено поширені алгоритми планування задач, а саме: Rate monotonic, Deadline monotonic, Earliest Deadline First.

Також було детально проаналізовано існуючі рішення, серед яких визначено велику кількість позитивних моментів та недоліків. Подібні типи аналізу корисні для створення конкурентоспроможного програмного засобу та виняткових рис призначених для створення кращого продукту. В результаті дослідження було визначено низку вимог до створення та проектування кінцевої системи.

2 ОПИС ТА АНАЛІЗ ВИМОГ ДО СИСТЕМИ

2.1 Нефункціональні вимоги до системи

Успішна реалізація інформаційних систем в першу чергу залежить від вдалого проектування на ранніх стадіях розвитку системи. Вимоги та призначення до кожного компоненту повинні чітко визначені та враховані. На даному етапі створення програмного забезпечення важливо продумати якомога більшу кількість дрібниць, що принесуть кінцевому користувачеві враження дійсно якісного продукту.

Методи та програмні засоби планування і управління виконанням мікросервісних задач за своєю сутністю не можуть існувати без мікросервісної архітектури. Саме тому важливо обрати правильний напрямок розвитку програмних засобів, що зможе забезпечити максимальну пропускну здатність виконання задач та масштабованість у використанні.

На даний момент існує багато варіантів запуску планових задач, які працюють для специфічних цілей окремих задач. Проте сучасна розробка мікросервісної архітектури потребує підхід з агрегацією таких задач в окремий сервіс [7]. Адже одинична відповідальність на сьогодні є ключовим фактором трансформації старих монолітних систем. Такий підхід надає особливу гнучкість системі та переваги у швидких та легких змінах, простоті тестування окремих одиниць системи та зменшує вірогідність перенавантаження системи в цілому і окремих сервісів через простоту їх масштабування [8].

Кожен сервіс створений для обробки та обміну інформацією, тому важливо детально спроектувати програмний інтерфейс для обміну даними. Відповідно треба обрати протокол за яким буде зручно проводити цей обмін. При розвитку інформаційних систем часто відбувається трансформація через зміну вимог до конкретних інтерфейсів, саме через постає вимога підтримки версій програмних інтерфейсів. Також важливо правильно спроектувати уніфікатори ресурсів для підсвідомого розуміння про операцію, що буде відбуватися за відповідним локатором.

Мікросервіси - це розподілені частинки великої системи, що спілкуються між собою за раніше визначеними контрактами, тому важливо дотримуватися контракту програмного інтерфейсу. Контракт встановлюється розробниками конкретної системи, проте є також загальноприйняті контракти, як наприклад обмінюватися даними з назвами атрибутів у стилі “camelCase”. Ще один поширений контракт це не додавати атрибути у структуру обміну даними по мірі їх необхідності зайвими атрибутами та атрибути мають бути названі відповідно до їх призначення.

Одним з важливих аспектів проектування мікросервісів є балансування навантажень між вузлами. Для балансування навантажень існує безліч алгоритмів, проте під кожен систему підбирається оптимальний в залежності від вимог до неї. Основною ідеєю балансування навантаження є розподіл виконання задач між вузлами мікросервісів.

Іншим моментом, що потребує окремої уваги є доступність вузлів. Концепції балансування навантаження та доступності вузлів пересікаються, оскільки вузли повинні бути ізольованими і на них потрібно направляти трафік, щоб забезпечити стійкість. Якщо нехтувати цим аспектом, то в результаті може відбутися затримка або збій системи, через недоступність окремих її компонент.

Для оптимізації роботи систем використовують кешування, воно допомагає зменшити час відповіді та уникнути зайвих запитів до джерела даних. Кешування може бути реалізовано за допомогою зовнішніх та внутрішніх механізмів, але при цьому потрібно дотримуватися принципів масштабованості системи, щоб усі вузли працювали з актуальними даними.

Також потрібно використовувати кешування або набір з'єднань з сторонніми системами, наприклад набір з'єднань з базою даних та з сторонніми сервісами, тому що відкриття нового з'єднання потребує багато ресурсів.

Іноді виконання задач іншими сервісами є складним процесом та не відповідає критеріям часових обмежень, тому варто підтримувати асинхронну обробку задач, що включає у себе збереження статусу у сховище і ці статуси повинні оновлюватися за допомогою кінцевих точок сервісу.

Складність рішення в проектуванні сервісів заключається у виборі оптимальної системи зберігання даних. Цей крок слід оцінювати на ранніх етапах проектування, адже він впливає на складність внесення змін до мікросервісів у майбутньому. На даному етапі потрібно розуміти, що в майбутньому потрібно буде редагувати початкову модель, задля цього використовують версіонування. Іншим темою для роздумів є накопичення великої кількості даних, зазвичай для цього передбачають видалення застарілих даних або їх архівація. Саме таких політик збереження даних потрібно дотримуватися для розумного використання ресурсів та легкого внесення змін.

Залежно від вимог до даних і пропускної здатності залежить дизайн бази даних мікросервісів, а у розробку програмного застосунку планування та управління виконанням мікросервісних задач потрібно врахувати високу пропускну здатність. При великих навантаженнях краще застосовувати індексування даних у базі даних, такий підхід зменшує складність пошуку потрібних даних та покращує час відповіді, але потрібно зберігати баланс, адже при великій кількості індексів редагування даних буде займати велику частку часу.

Загалом розроблений програмний продукт має відповідати нормам масштабування та відображати високі показники продуктивності та витримувати навантаження розмірами у сотні задач на секунду та при цьому бути зручним інструментом в проектуванні мікросервісної архітектури.

2.2 Функціональні вимоги до системи

Сервіс запуску та планування мікросервісних задач надасть змогу зібрати інформацію про всі задачі в межах однієї невеликої частини великої системи на базі мікросервісної архітектури. Такий підхід надає додаткову гнучкість системі, як у змозі його перевикористання для задач різних цілей, так і для єдиного структурованого підходу у запуску задач та відображення прогресу їх виконання. Важливо, щоб даний сервіс підтримував загально прийняті правила комунікації

з іншими частинками системи та мав високу стійкість, що надаватиме гарантію надійності користувачам.

Система розроблена в межах даної роботи повинна мати можливість для зберігання даних щодо мікросервісних задач та їх виконання. Дана проблема вирішується за допомогою використання системи бази даних. У базі даних буде розміщена інформація про задачі, їх конфігурації, дії, а також екземпляри їх виконання. Для сервісу планування та запуску мікросервісних задач важливо правильно спроектувати систему зберігання даних, саме вона буде відігравати одну з ключових ролей у масштабуванні та пропускній здатності цього сервісу. В рамках даної дисертації розроблено сервіс на основі наступних ключових сутностей:

- задача;
- конфігурації тригерів;
- дії;
- виконання.

Між переліченими сутностями звичайно мають бути зв'язки, саме вони забезпечать цілісність даних у програмному забезпеченні з урахуванням характеристик перелічених сутностей.

Головною сутністю системи є задача, саме для неї розроблений даний сервіс і їй притаманні такі властивості, як назва, опис та назва мікросервісу (на який вона посилається).

Найскладнішою сутністю цієї системи є конфігурація тригерів вона поділяється за видом запуску та за налаштуванням конфігурації. Види запуску бувають двох типів: ручного та планованого. Ручний запуск виконується користувачем для запуску задачі у даний момент часу або у майбутньому у чітко визначений час. Планований запуск виконується завдяки загально прийнятому виразу «cron», що використовується, як формат визначення періодичності задач. Також використовуються дві конфігурації за налаштуваннями користувача та системи. Потрібно врахувати, що системна конфігурація отримується при зміні конфігурацій самого сервісу, а налаштування користувача вказуються під час

роботи сервісу та перевизначають системні конфігурації, вони можуть бути як тимчасовими так і постійними. Конфігурація містить параметри запуску, вираз періодичності (для періодичних задач), час запуску для одноразових задач, тип планування та ким налаштовано.

Під час процесу виконання задач, окремі їх конфігурації можуть підтримувати виконання відповідних дій, таких як отримання статусу прямо зараз або зупинка виконання задачі, отже це потрібно врахувати у проектуванні схеми бази даних.

Остання сутність це виконання, вона створюється та модифікується в процесі виконання задач. Дана сутність є важливою для швидкої роботи системи, адже якщо вона буде невірно спроектована, то оновлення статусу виконання задач може стати дорогою та потребувати багато ресурсів. Сутність має наступні характеристики: статус запуску задачі в мікросервісі, помилки запуску задачі, час старту, час закінчення, результат та статус задачі.

Планування задач та їх запуск є основною метою створення сервісу, тож варто визначити чіткі вимоги щодо коректної роботи цієї частини системи. При запуску сервісу потрібно отримати дані про існуючі мікросервісні задачі. На даному етапі потрібно підтримати інтеграцію з зовнішніми системами, а саме використати зовнішнє сховище, бажано таке, що підтримує інформування про зміну конфігурацій без перезавантаження сервісу. Після отримання актуальних даних про задачі сервіс має оновити їх у базі даних та запланувати їх виконання, якщо це періодичні задачі. Алгоритм планування задач має підтримувати масштабування, якщо існує два і більше вузли сервісу планування задач, то заплановані задачі у них мають бути ідентичні.

Особливу цінність у розробці даного мікросервісу несе перевизначення конфігурацій запуску задач. Не завжди у адміністраторів є доступ до загальної конфігурації вузлів, тому одразу варто закласти це у вимоги до такого функціоналу та інтерфейсу, за допомогою якого можна це зробити. Такий підхід буде використовуватися з метою зміни конфігурацій без перезавантаження

мікросервісу і тим самим зробити кінцевий програмний засіб більш зручним у користуванні.

Під час запуску задачі потрібно передбачити механізм єдиного виконання, тобто щоб усі вузли сервісу планування задач вирішили між собою єдиного виконавця. Даний сервіс повинен підтримувати, як синхронне так і асинхронне виконання задач, тому варто додати можливість відслідковувати статус на етапах запуску, початку виконання на клієнтському сервісі та завершення.

У сервісі планування задач передбачений ручний запуск задач завдяки визначеному для цього програмного інтерфейсу. Такі задачі можуть виконуватися негайно або на вказаний час.

Запущена задача може виконуватися тривалий проміжок часу, за цей час користувач повинен мати можливість отримати статус прямо на даний момент з сервісу що виконує задачу або зупинити виконання задачі, якщо це можливо.

Зв'язок між сервісом планування задач та клієнтським сервісом передбачає використання чітко визначеного контракту обміну даними, тобто протоколу передачі даних, відомого для обох сторін локатора ресурсів та формат і структура обміну інформацією. З точки зору клієнта, конфігурація має задаватися легко та за вже відомими методами, адже при створенні нових методів майбутнім користувачам доведеться витратити свій час на вивчення та застосування його на практиці, що буде надавати привід для використання альтернативних систем.

Іншим важливим компонентом є система моніторингу. Моніторинг дозволяє відслідковувати статус виконання задачі, отримати інформацію про успішний запуск, успішне виконання або помилку та її причину.

Сьогодні невід'ємним підходом до розробки розподілених систем є їх платформи незалежність і він має бути передбачений у даній роботі, адже саме цей підхід дає можливість встановлювати необхідні компоненти програмного забезпечення безболісно на будь які системи, а ядро функціональності цих компонент абстрагувати від нативних вимог для конкретної платформи.

В даній дисертації також буде розглянуто механізми отримання інформації про розташування клієнтських сервісів та балансування навантажень при виконанні мікросервісних задач.

Висновки до розділу

Визначено загальні вимоги та описано методи та програмні засоби планування та управління виконанням мікросервісних задач. Враховано багато дрібниць, що неодмінно мають вплинути на якісні характеристики системи. Вирішено спроектувати та реалізувати програмний засіб планування та управління виконанням задач для побудови мікросервісної архітектури. Також проведено аналіз характеристик, яким має відповідати даний сервіс. В результаті проведеного аналізу сформульована постановка задачі, а саме створити масштабований сервіс планування та управління виконанням задач, що буде підтримувати виконання відкладених та періодичних задач на клієнтських мікросервісах, також динамічну зміну конфігурації задач та моніторинг виконання мікросервісних задач.

3 ОБРАНІ МЕТОДИ, АЛГОРИТМИ ТА ІНСТРУМЕНТИ

Важливу роль вдалої розробки програмного забезпечення відіграє вибір для вирішення конкретних задач оптимальних методів та використання вдало підібраних інструментів. Дана система буде виконана у вигляді мікросервісу, варто розділити вибір методів, алгоритмів та інструментів для наступних частин системи:

- збереження даних;
- реєстрація задач;
- планування та запуск;
- моніторинг результатів виконання задач;
- проектування програмних інтерфейсів;
- комунікація з клієнтськими сервісами.

Рівень збереження даних відповідає за коректне зберігання інформації у базі даних та коректність збереженої інформації, що передбачає тип та обмеження. Також на даному кроці потрібно визначити інструменти та методи взаємодії з базою даних та програмним застосунком.

Реєстрація задач є ключовим етапом для отримання інформації щодо існуючих мікросервісних задач. На цьому рівні варто спроектувати гнучке рішення для отримання цієї інформації, адже конфігурації можуть змінитися в будь який момент і це потрібно підтримувати.

Етап планування та запуску задач вміщує механізм планування, що має забезпечити синхронізацію запланованих задач на усіх вузлах сервісу планування мікросервісних задач та єдиний запуск.

Розроблений в програмний засіб також повинен відтворювати інформацію щодо запущених задач та зберігати дані про них для можливості подальшого моніторингу результату виконання задачі.

Проектування програмних інтерфейсів передбачає визначення протоколів для кінцевих точок, що надаватиме сервіс планування та управління виконанням

задач. На даному етапі потрібно розробити підтримку версій у кінцевих точках та створити лаконічний контракт програмному інтерфейсу.

Іншою невід’ємною частиною розробки даного програмного забезпечення є комунікація з клієнтськими сервісами на даному етапі потрібно визначити протоколи за якими це буде відбуватися, також обмін інформацією про адреси розташування сервісів та розподіл навантаження при створенні запитів.

Завдяки розбиттю системи на перераховані вище частини створюється раціональна розмежованість відповідальності відповідних рівнів системи за певний набір обов’язків. Для більш детального огляду обраних технологій розглянемо кожен частину системи у розділах 3.2 -3.7.

3.1 Визначення парадигми програмування

При виборі парадигми програмування було розглянуто об’єктно-орієнтовану, функціональну та процедурну. Серед раніше названих варіантів вирішено обрати об’єктно-орієнтовану парадигму програмування. Такий підхід надає переваги, серед яких:

- декомпозиція програмних компонентів;
- легка масштабованість;
- прості внесення змін;
- повторне використання коду.

Проте даний підхід має також свої недоліки, а саме:

- витрати часу на аналіз та проектування системи;
- збільшений час на виконання програмних команд;
- розміри коду.

Об’єктно-орієнтований підхід підтримує приховування даних, а саме інкапсуляцію. Наявність інкапсуляції одна із характеристик для об’єктної мови програмування, але не підтверджує об’єктної орієнтованості — для цього також потрібна підтримка успадкування. Але навіть наявність інкапсуляції і спадкування не підтверджує об’єктної орієнтованості з точки зору ООП. Основні

переваги ООП проявляються тільки в тому випадку, коли в мові програмування реалізований ще й поліморфізм, тобто можливість об'єктів з однаковим інтерфейсом мати різну реалізацію [9].

Абстрагування — це властивість системи виділяти набір значимих характеристик об'єкта на вищий рівень, виключаючи з діапазону видимості користувача приховану складність.

Інкапсуляція — це властивість системи, що передбачає об'єднання даних разом з методами, що працюють з ними в класі, та приховання деталей реалізації від користувача.

Спадкування — це властивість системи, що дозволяє створення нового класу на основі вже існуючого з частковим або повним запозиченням функціональності. Клас, у якого запозичують функціональність, називається суперкласом, батьківським або базовим. Новий клас — похідним класом, спадкоємцем або нащадком.

Поліморфізм — це властивість системи, що забезпечує використання об'єктів з однаковим інтерфейсом, що не мають інформації про тип і внутрішню структуру об'єкта.

Мова програмування Java є відомим представником об'єктно-орієнтованої парадигми [10]. Дана мова програмування є перевірена часом та за інформацією, що надають відомі джерела у світі інформаційних технологій, на даний момент є найпопулярнішою мовою програмування.

Величезна кількість інструментів є великою перевагою Java у порівнянні з іншими мовами програмування. Ці інструменти сприяють розв'язку найбільш поширених задач у корпоративному та науковому програмному забезпеченні. Також варто відзначити підтримку від хостинг-провайдерів цієї технології.

Важливою складовою будь яких систем у наш час є безпека даних, Java має підтримку визначення правил доступу в політиці безпеки та запобігає несанкціонованому доступу до пам'яті.

Віртуальна машина Java дозволяє не турбуватися розробнику про операційну систему, на якій буде запущено програмне забезпечення, адже вона є абстракцією між додатком та обладнанням [11].

3.2 Збереження даних

В основі даного сервісу є збереження даних задач та результати їх виконання. Після огляду існуючих СУБД було обрано PostgreSQL. PostgreSQL є потужним інструментом з відкритим кодом, що надає підтримку мову запитів SQL разом з спеціально налаштованими функціями адаптованими під специфіку даної СУБД [12]. Ця база даних є популярною у високонавантажених та масштабованих системах світу.

Репутацію даної СУБД у світі технологій потребує особливої уваги, цей показник відзначається у збереженні цілісності даних, постійного оновлення ефективних та інноваційних рішень, а також високою стійкістю до відмов у системах.

PostgreSQL є реляційною базою даних. Реляційна модель визначає зберігання даних у вигляді таблиць, кожна з яких складається з стовпців і рядків. Кожен об'єкт є окремим рядком, а атрибути цього об'єкта представлені у вигляді стовпчиків.

Реляційним базам даних притаманна нормалізація. На даний момент відомо більше 6 нормальних форм баз даних. У системі планування та управління виконанням мікросервісних задач схему бази даних приведено лише до першої нормальної форми, через збитковість у продуктивності системи при приведенні у вищі нормальні форми. Перша нормальна форма свідчить про те, що використовуються лише скалярні значень у таблиці бази даних. У другій нормальній формі схема бази даних передбачене приведення до першої нормальної форми, а також безпосередність у залежності кожного атрибуту від первинного ключа.

При проектуванні бази даних потрібно долучити підтримку версій. Серед відомих інструментів підтримки версій у реляційних базах даних існує два найбільш популярних варіанти, такі як Flyway та Liquibase. Якщо порівняти принципи роботи цих інструментів, одразу зрозуміло, що Flyway є більш простим та підтримує, як SQL так і Java міграції, натомість Liquibase дозволяє абстрагуватися від SQL та створити міграцію на основі XML, JSON, YAML форматів, що дає можливість не прив'язуватися до конкретної реалізації бази даних [13]. У рамках розробки програмних засобів цієї системи було обрано Flyway, так як зберігання даних буде відбуватися у раніше визначеній реалізації бази даних.

Наступним ключовим етапом проектування частини збереження даних є налаштування зв'язку між сутностями та базою даних. Сутності у даній системі відображені у вигляді об'єктів.

ORM є базовою технологією, що допомагає описати зв'язки інформації у базі даних та сутностях. Опис даних за допомогою об'єктно-орієнтованої парадигми та можливість проведення операцій з даними у вигляді об'єктів надає беззаперечні переваги у відображенні даних в системі.

Підтримку концепцію ORM, асортименті інструментів Java, забезпечують чимало бібліотек, проте Hibernate є раціональним вибором. Фреймворк підтримує водночас декілька способів взаємодії з реальними даними, такі як JPQL, власна бібліотека Criteria API та нативні запити. Також даний фреймворк відзначається особливою продуктивністю і гнучкістю у використанні та перевіреним часом.

Проте цього не достатньо, проблему зв'язку об'єктів із даними вирішує JPA, даний програмний інтерфейс надає інструменти опису структури даних, що будуть вкладатися у об'єкти. JPA складається з набору анотацій та інтерфейсів, які є стандартом ORM фреймворків та допомагають створити опис структури об'єктів на зрозумілій для них мові.

Для більш зручного доступу до даних у системі було вирішено застосувати Spring Data бібліотеку [14]. Використовуючи даний інструмент реалізовує

шаблон DAO, що є шаром об'єктів які забезпечують доступ до даних, натомість розробник має доступ до зручного інтерфейсу користування переліченими вище технологіями.

3.3 Реєстрація задач

Невирішеною проблемою є зберігання конфігурацій самих задач, а саме інформацію про сервіс якому належить дана задача, назву задачі, тип запуску і конфігурація запуску. Серед загально прийнятих варіантів це можна зробити за допомогою серверу конфігурацій, такий підхід використовують для централізованого зберігання конфігурацій для усіх сервісів. Проте в сучасних системах відходять від такого підходу, адже він потребує також використання конфігурацій специфічних для платформи на якій розгортаються мікросервіси, натомість більшість платформ також підтримує налаштування конфігурацій під розгортання кожного мікросервісу.

Натомість іншим інструментом, що є менш поширеним, проте надає додаткову гнучкість в налаштуванні конфігурацій є ZooKeeper [15]. ZooKeeper не призначений конкретно для налаштування конфігурацій він є мультиінструментом для різних випадків забезпечуючи ієрархічний простір імен для зберігання інформації. Крім того, Zookeeper підтримує додавання наглядців для повідомлення про зміну конфігурацій.

Тож алгоритм реєстрації задач буде наступним, коли піднімається новий вузол сервісу клієнта, він перевіряє актуальність своїх задач у ZooKeeper, якщо інформація про задачі не актуальна, корегує її. Після цього всі вузли сервісу планування задач отримують цю інформацію, оновлюють її в базі даних та переналаштовують тригери цих задач.

3.4 Планування та запуск

Планувати виконання задач є непростю задачею. Існують різні алгоритми вирішення даної проблеми в рамках цієї дипломної дисертації використовується наступний: спершу створюється тригер з інформацією про задачу часом її виконання або виразом стоп; на основі часу та виразів вираховується час найближчого спрацювання тригеру та створюються слухачі для усіх тригерів; запускається потік з циклом, що працює до моменту поки системний час не є більшим за слухачі отримують повідомлення про спрацювання, а далі запускається задача що мала спрацювати у відповідний час; після цього знову вираховується час спрацювання наступного тригеру і т. д. Інструмент, що працює за вище описаним алгоритмом вже існує це Quartz [16]. Він створений для планування бізнес задач та надає для цього зручний програмний інтерфейс.

При реалізації монолітної архітектури цього було б достатньо, проте у даній дисертації розглядається саме мікросервісна архітектура, тому варто розробити алгоритм горизонтального масштабування даного підходу. Проблема полягає у синхронізації запланованих задач на усіх вузлах при ручній реєстрації задачі на запланований або зміні конфігурації час через кінцеві точки вузлів.

У мікросервісному світі цю проблему вирішують Message Broker-и. Завдяки ним можна обмінюватися повідомленнями між сервісами швидше ніж по протоколу HTTP, тому що повідомлення є меншими. Серед розглянутих варіантів Message Broker-ів є RabbitMQ та Kafka. В просторах інтернету існує багато статей з порівняннями цих двох інструментів. Проте основними для даної системи є те, що у RabbitMQ не зберігаються повідомлення та вона надає швидкий обмін повідомленнями при невеликому навантаженні черг, а це важливо у даному випадку [17].

Інша проблема створити умови для єдиного запуску задачі, адже якщо декілька вузлів і усі мають ідентичні задачі, то на усіх вузлах вони запустяться в один момент. Тому для вирішення даної проблеми було розроблено наступний алгоритм: у всіх запланованих тригерів існує значення lock; після запуску задачі

значення збільшується на одиницю та зберігається в базі даних; потім перевіряється, якщо база даних повернула кількість оновлених рядків, що рівна одиниці, задача запускається, інакше ігнорується.

3.5 Моніторинг результатів виконання задач

Сервіс планування та управління виконанням задач надає можливість користувачеві проводити моніторинг статусу виконання задачі. Для цього в процесі запуску задачі одразу створюється об'єкт запуску та зберігається до бази даних. Після запуску задачі на стороні сервісу клієнта статус оновлюється. І якщо задача виконується асинхронно, то після завершення виконання задачі, сервіс клієнту надсилає на кінцеву точку сервісу планування та управління виконанням задач фінальний статус.

Користувачу надається можливість наглядати за статусом за допомогою кінцевих точок сервісу та отримувати статус з сервісу клієнта або зупинити задачу, якщо задача не у фінальному статусі.

3.6 Проектування програмних інтерфейсів

Наступною частиною сервісного програмного забезпечення є API. На цьому рівні в якості архітектурного підходу обрано REST. Отримувати, ту чи іншу інформацію або редагувати її можна за відповідними URL шляхами. Обмін даними з проводиться у форматі JSON. Також на цьому рівні данні відбувається валідація, що приходять у систему. Найважливішим у подібних системах є коректне реагування на будь який запит, особливо невірний, даний рівень вміщує це та надає за допомогою протоколу HTTP відповідь відповідно помилки у системі [18].

В даній системі використовується прикладний протокол обміну даними в комп'ютерних мережах HTTP. Особливістю даного протоколу є відсутність зберігання стану. Він діє за принципом «запит-відповідь», як і більшість схожих

за призначенням протоколів. Запити та відповіді у такому протоколі складаються з наступних частин:

- заголовки;
- рядок запиту;
- тіло повідомлення.

Рядок запиту у свою чергу вміщує один із визначених у протоколі HTTP методів, а саме: CONNECT, HEAD, OPTIONS, GET, POST, PATCH, PUT, DELETE, TRACE, та шлях за яким відбувається запит повідомлення.

Запит або відповідь описують заголовки, що також призначені для розпізнавання запиту браузером чи сервером.

Данна система використовує архітектурний підхід REST, що описує набір вимог для забезпечення доступу до сервісних ресурсів. Підхід має наступні архітектурні обмеження:

- однорідний інтерфейс;
- клієнт-серверна архітектура;
- кешування даних;
- відсутність збереження стану;
- шари абстракції.

Обмін повідомленнями у сервісі з іншими мікросервісами ведеться у форматі JSON. Такий формат надає можливість структуровано передавати інформацію у мережі інтернет. У стандарті JSON визначено наступні форми даних: масив, об'єкт, рядок та значення.

Усі локатори доступу до ресурсів матимуть версію, що дозволить створювати міграцію системи з розширенням функціоналу у майбутньому та зберігати контракт обміну даними.

3.7 Комунікація з клієнтськими сервісами

У мікросервісній архітектурі, на відмінно від монолітної, потрібно вирішувати проблему комунікації між сервісами. Якщо сервіс клієнта який

звертається до цільового сервісу має в конфігурації пряму адресу, то нам для масштабування доведеться змінювати конфігурації усіх сервісів, що залучені до цього процесу та розгортати усі сервіс заново. У вирішенні цього питання нам допоможе Service Discovery. Існує два поширених види Service Discovery:

- Server-Side Service Discovery;
- Client-Side Service Discovery.

Server-Side Service Discovery передбачає, що сервіс клієнта буде взаємодіяти не напряму з цільовим сервісом, а через сервіс вирівнювач навантаження. У такому випадку потрібен ще додатковий реєстр сервісів. Сервіс вирівнювач навантажень бере всі данні з реєстру сервісів. Таким чином, завдання вирівнювача навантажень брати всю інформацію про адреси сервісів та перенаправляти запит на один з них. А завдання реєстру сервісів зберігати данні про сервіси, що він може робити різними методами: опитувати сервіси сам, брати дані з зовнішнього конфігурацій і т. д.

Client-Side Service Discovery не має в своїх реалізаціях сервісу вирівнювання навантажень і сервіс клієнт звертається напряму до реєстру сервісів, звідки бере адресу потрібного сервісу. Переваги такого підходу очевидні в ланцюжку на один сервіс менше і працює все відповідно швидше. Цей підхід є кращим, але лише в разі, якщо сервіс клієнт є внутрішнім.

В розробці даного програмного засобу буде використано Client-Side Service Discovery, тому що сервіс планування та управління виконанням задач створюється, для внутрішнього використання у мікросервісній архітектурі.

Реєстр сервісів також може бути використано, як окремий вузол системи, проте в випадку вибору попередніх інструментів для розробки зручніше використати ZooKeeper у якості реєстру сервісів. Даний підхід є дуже зручний і головне, що екосистема Spring Cloud підтримує такий підхід до розробки [19].

Проте варто потурбуватися про вирівнювач навантаження, тому що без нього всі виконання задачі будуть направлятися на один вузол сервісу клієнта. Було вирішено реалізувати власний вирівнювач навантажень, що буде визначати

адресу вузла сервісу клієнта на основі ділення випадково створеного великого числа на кількість існуючих вузлів сервісу клієнта конкретної задачі.

Висновки до розділу

В результаті огляду методів та інструментів, а також врахування їх переваг та недоліків, для розробки веб-застосунку було обрано мову програмування Java, що підтримує об'єктно-орієнтовану парадигму програмування, платформу Spring Boot для створення сервісу [20]. Для взаємодії з базою даних обрано технологію Spring Data, що містить ORM фреймворк Hibernate та підтримує принципи ACID. У якості системи управління базою даних використовується PostgreSQL з міграцією бази даних Flyway. Реєстрація задач відбуватиметься з використанням ZooKeeper. У якості механізму планування задач буде використовуватися бібліотека Quartz, масштабування системи з динамічним плануванням задач відбуватиметься через брокер повідомлень RabbitMQ. Комунікація між сервісом планування та управління виконанням задач та клієнтськими мікросервісами відбуватиметься з використанням методу Client-Side Service Discovery та реєстром мікросервісів у ZooKeeper. В даному розділі розглянуто і проаналізовано усі перелічені технології, наведено опис і характеристики, їх основні переваги та недоліки.

4 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНИХ ЗАСОБІВ

Проектування та реалізацію було розділено на декілька частин, що відтворюють архітектурні рішення та основні процеси реалізації програмних засобів. До таких частин входять:

- збереження даних;
- реєстрація задач;
- планування та запуск;
- моніторинг виконання задач;
- комунікація з клієнтськими сервісами.

4.1 Збереження даних

Проектування системи на ранніх стадіях реалізації є вагомою частиною успішного результату. Даний етап притаманний для всіх різновидів програмного забезпечення. Під час виконання цього кроку важливо врахувати об'ємну кількість деталей, що мають безпосередній вплив на швидкість та якість реалізації системи.

В основі всіх програмних засобів лежить обробка даних, які також доводиться зберігати. У проектуванні програмного інтерфейсу важливо надавати бажану та лаконічну інформацію. Проте у програмному застосунку інформація повинна зберігатися та оброблятися оптимальними шляхами. Здебільшого це впливає на швидкість відгуку системи на запити з зовні, але також має прямий вплив на час розробки системи та її масштабованість у майбутньому.

Першим етапом проектування сервісу планування та управління виконанням мікросервісних задач є побудова якісної моделі «сутність – зв'язок». Даний вид моделювання був розроблений Пітером Ченом для проектування схеми бази даних і опубліковане 1976 року. Основою цієї моделі є типи сутностей, що представляють собою інформаційні об'єкти за спільними характеристиками, та зв'язки, які можуть існувати між даними сутностями.

Серед сутностей, що визначено у сервісі планування та управління виконанням задач є задача, конфігурація триггеру задачі, плановий тригер, ручний тригер, запуск та дії, що можуть виконуватися над задачею. Діаграму класів сутностей зображено на рисунку 4.1.

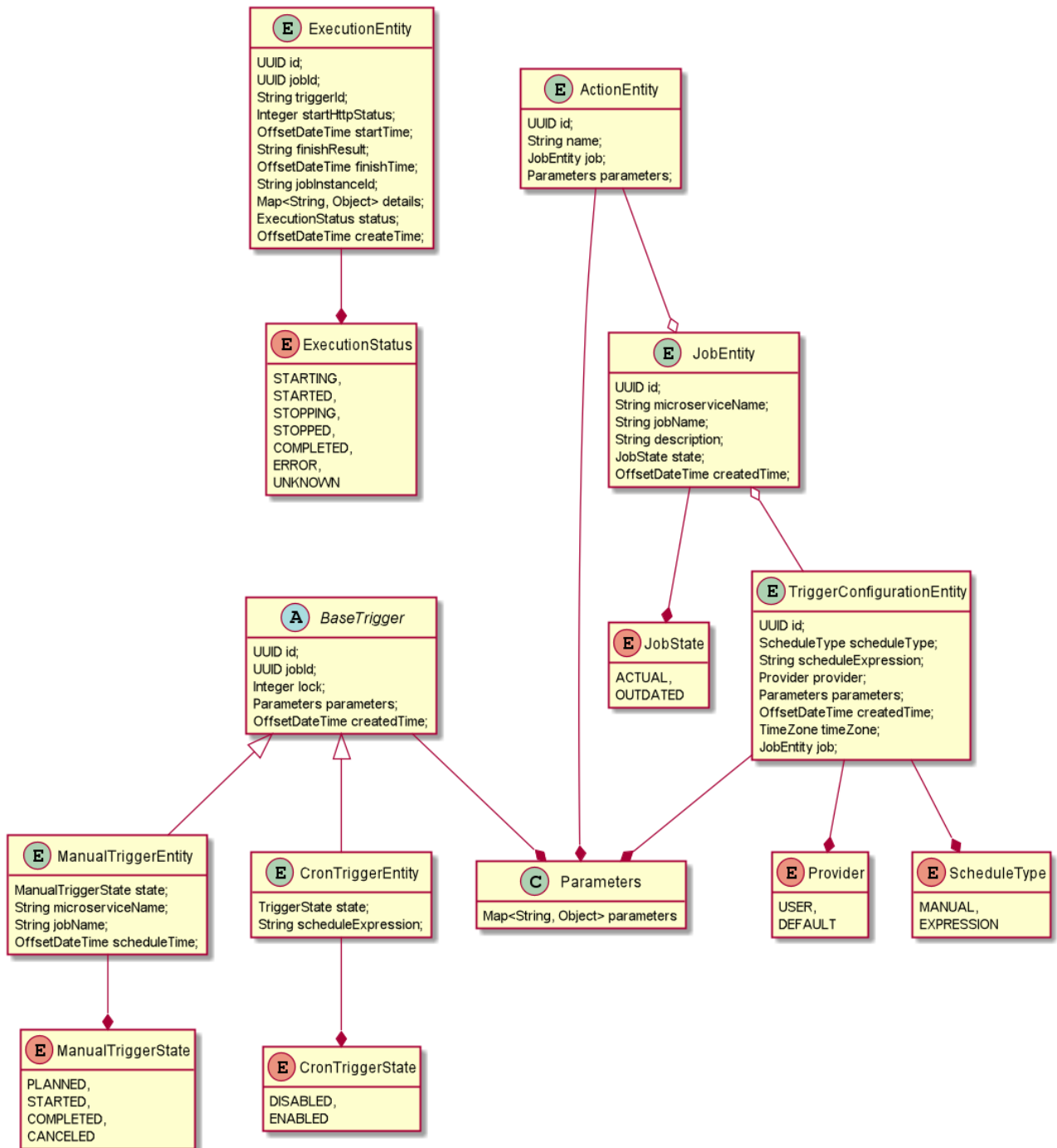


Рисунок 4.1 — Діаграма класів сутностей

Схема бази даних знаходиться у першій нормальній формі, але не у другій. Дані зміни були свідомо виконані задля ефективного виконання ручних задач. Схема таблиць бази даних описана в таблиці 4.1.

Таблиця 4.1 — Опис таблиць схеми бази даних

Назва таблиці	Опис
JOB	Загальна інформація про задачу, що містить назву мікросервісу на якому виконується задача, назву задачі, опис, стан та час створення задачі
TRIGGER_CONFIGURATION	Конфігурація тригерів задач в якій вказується ідентифікатор задачі, тип планування (ручний або періодичний), вираз сгоп для періодичних задач, параметри виконання та інформація ким встановлену дану конфігурацію (користувачем чи сервісом клієнта)
MANUAL_TRIGGER	Таблиця тригерів для задач, що виконуються в заданий момент часу. Дана таблиця знаходиться у першій нормальній формі у цілях збільшення продуктивності виконання задач та зменшення кількості запитів до бази даних. В ній зберігається ідентифікатор задачі, назва мікросервісу, назва задачі, параметри запуску, лічильник блокування, статус, запланований час запуску та час створення тригеру
CRON_TRIGGER	Таблиця тригерів для задач, що виконується за заданим сгоп виразом. В ній зберігається ідентифікатор задачі, параметри запуску, лічильник блокування, статус, вираз сгоп та час створення тригеру
ACTION	Таблиця дій, які можна виконувати у процесі виконання задачі. Містить ідентифікатор задачі, назву дії та параметри для їх виконання
EXECUTION	Таблиця запусків задач. Містить ідентифікатор задачі, статус запуску задачі на мікросервісі клієнта, помилки під час запуску, час запуску, кінцевий результат, час закінчення виконання, ідентифікатор екземпляру виконання, отриманий з клієнтського мікросервісу, ідентифікатор тригеру та деталі виконання

4.2 Реєстрація задач

Ключовою властивістю сервісу планування та управління виконанням задач є саме планування задач. Проте для успішного планування задачі потрібно отримувати їх. У рамках даної дисертації було спроектовано рішення, у якому сервісом реєстрації задач слугує ZooKeeper, а сервіс планування та управління виконанням задач читає оновлення у файловій системі цього програмного засобу.

Кожну з перерахованих нижче способів реєстрації та зчитування задач буде зображено на рисунках, що матимуть наступні умовні позначення: «Scheduler» - сервісу планування та управління виконанням задач; «ZooKeeper» - сервіс конфігурацій ZooKeeper; «PostgreSQL» - база даних PostgreSQL; «Client Service» - екземпляр клієнтського сервісу.

Зчитування актуальних задач зображено на рисунку 4.2 та відбувається наступним чином:

- при старті сервіс планування та управління виконання задач підписується на зміни конфігурацій задач в ZooKeeper;
- зчитує всі конфігурації існуючих там задач;
- отримує існуючі задачі з актуальним статусом з бази даних;
- задачі з бази даних перевіряються, отримують оновлений статус та конфігурації, після перевірки на зміни серед задач ZooKeeper;
- задачі зберігаються до бази даних та плануються до запуску.

У випадку запуску нового клієнтського сервісу відбувається оновлення конфігурацій задач в ZooKeeper, зображені на рисунку 4.3, за наступними кроками:

- клієнтський сервіс оновлює свої існуючі задачі у ZooKeeper;
- сервіс планування та управління виконанням задач отримує оновлення про зміни у існуючих задачах, якщо вони присутні;

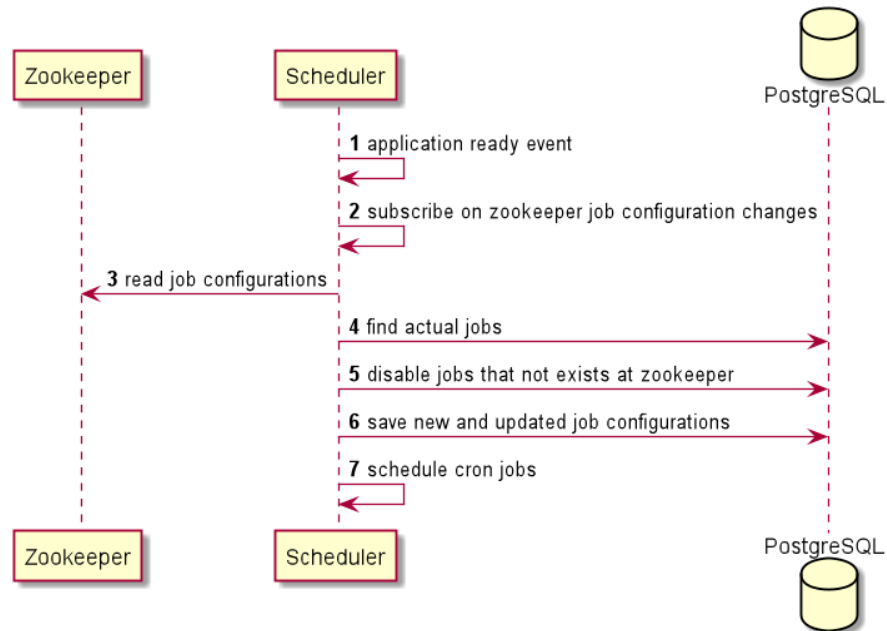


Рисунок 4.2 — Діаграма послідовності зчитування та планування задач

- після отриманого повідомлення знаходить задачі з актуальним статусом у базі даних;
- оновлює статус даним задачам;
- видаляє з планування не активні задачі;
- переплановує оновлені задачі та планує нові.

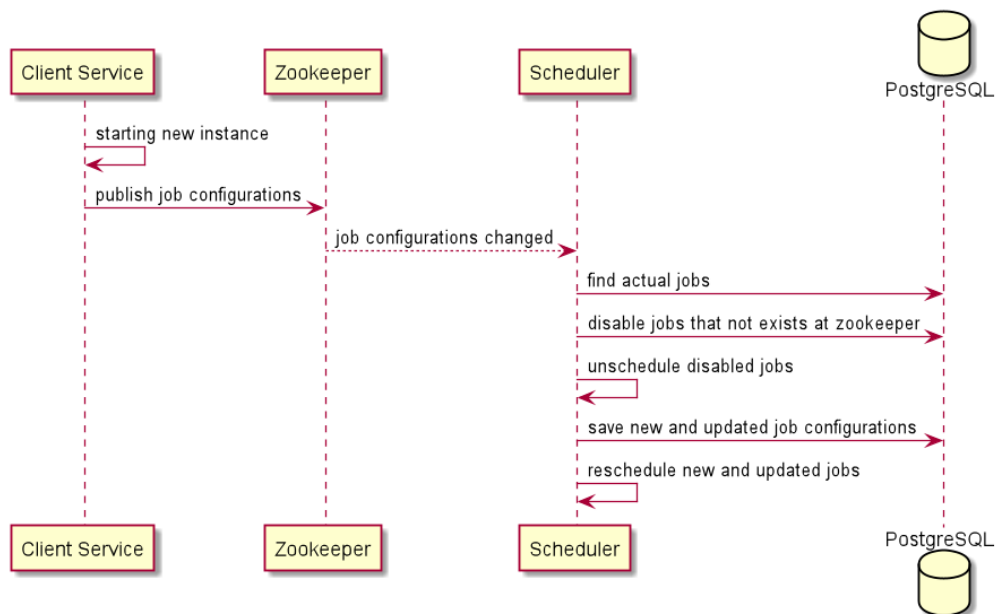


Рисунок 4.3 — Діаграма послідовності оновлення задач при старті нового сервісу клієнта

Таким чином ZooKeeper допомагає підтримати задачі з динамічним типом планування та масштабуватися системі, тому що події про оновлення задач отримують усі екземпляри сервісу планування та управління виконанням задач.

4.3 Планування та запуск

Планування та запуск варто розділити на дві окремі частини: ручних та планових задач. Кожну з перерахованих нижче дій над плануванням мікросервісних задач буде зображено на діаграмі послідовностей, що матимуть наступні умовні позначення: «Admin» - це адміністратор системи; «Scheduler» - екземпляр сервісу планування та управління виконанням задач; «Other instances of Scheduler» - інші екземпляри сервісу планування та управління виконанням задач; «RabbitMQ» - брокер повідомлень RabbitMQ; «PostgreSQL» - база даних PostgreSQL; «Client Service» - екземпляр клієнтського сервісу.

Ручні задачі плануються за допомогою програмного інтерфейсу та можуть використовуватися, як зовнішніми сервісами, так і адміністраторами системи. Послідовність планування та виконання ручної задачі зображено на рисунку 4.4.

Планування та виконання ручної задачі виконується таким чином:

- надходить HTTP запит ззовні на створення тригера такої задачі;
- задача перевіряється на існування в базі даних, якщо вона відсутня надсилається відповідь з інформацією про некоректний запит, інакше продовжується процес планування;
- з запиту отримується інформація про задачу та час, у який буде виконана задача та зберігається до бази даних;
- після закриття транзакції надсилається повідомлення до RabbitMQ про нову задачу;
- задача планується до виконання;
- інші екземпляри сервісу планування та управління виконанням задач зчитують повідомлення про новий тригер задачі;

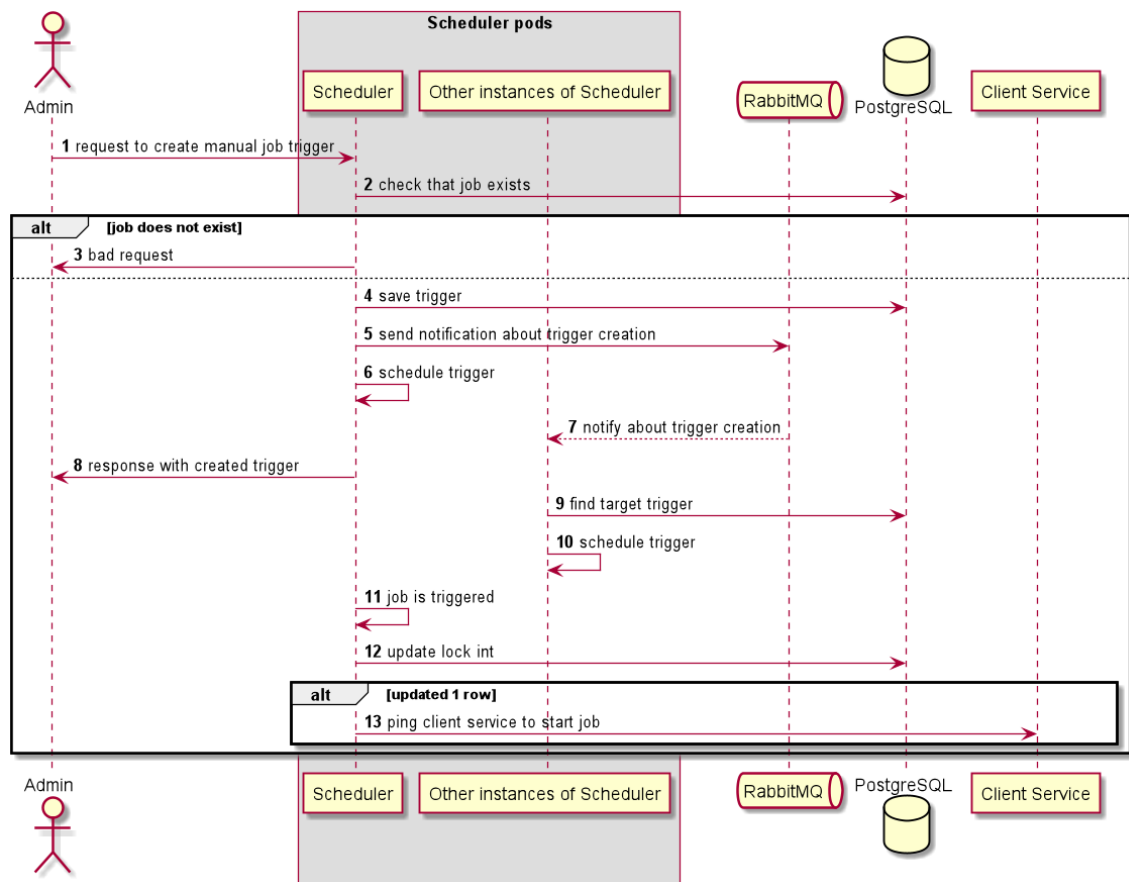


Рисунок 4.4 — Діаграма послідовності планування та виконання ручної задачі

- перевіряють наявність тригера в базі даних;
- планують спрацювання тригера задачі;
- після спрацювання тригера задачі, у визначений час, відбувається збільшення lock на записі тригера, за допомогою функції в базі даних;
- зберігається та перевіряється кількість кортежів у базі даних, що було оновлено, якщо не було оновлено, отже задача запущена іншим екземпляром сервісу, інакше виконується запуск задачі на клієнтському сервісі.

При проектуванні системи також передбачено оновлення тригерів ручної задачі, під час якого можна змінити час виконання та параметри запуску ручної задачі. Послідовність оновлення планування ручної задачі зображено на рисунку 4.5. При надходженні HTTP запиту на оновлення тригера задачі відбувається наступне:

- перевіряється тригер на наявність в базі даних та статус тригеру, якщо задача відсутня або тригер має статус виконано чи відмінено, повертається відповідь про некоректний запит;
- тригер оновлюється в базі даних;
- надсилається повідомлення до RabbitMQ про зміну тригера;
- перепланується спрацювання тригеру;
- інші екземпляри сервісу планування та управління виконанням задач зчитують повідомлення про оновлення тригеру;
- перевіряють його наявність в базі даних;
- перепланують спрацювання тригеру задачі.

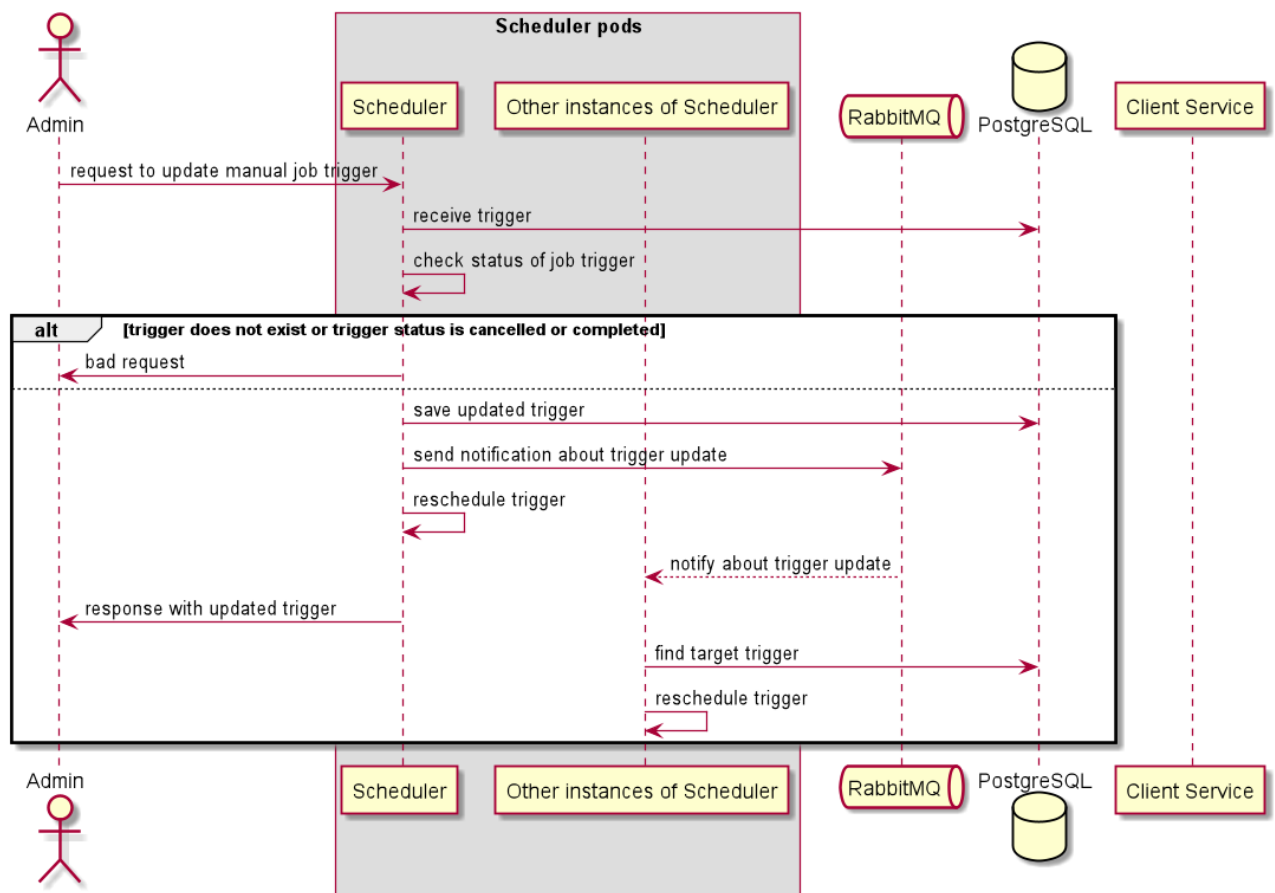


Рисунок 4.5 — Діаграма послідовності оновлення тригеру ручної задачі

Останнім, але не менш важливим пунктом є функція відміни запланованого тригеру ручної задачі. Послідовність відміни планування ручної

задачі зображено на рисунку 4.6. Відміна запуску ручної задачі за програмним інтерфейсом, що відбувається наступним чином:

- отримується HTTP запит на відміну спрацювання тригера задачі;
- перевіряється тригер на наявність в базі даних та статус тригера, якщо задача відсутня або тригер має статус виконано чи відмінено, повертається відповідь про некоректний запит;
- тригер оновлюється в базі даних з статусом відмінено;
- надсилається повідомлення до RabbitMQ про відміну спрацювання тригера задачі;
- переплановується спрацювання тригера;
- інші екземпляри сервісу планування та управління виконанням задач зчитують повідомлення про оновлення тригера;
- перевіряють його наявність в базі даних;
- відмінюється спрацювання тригера задачі.

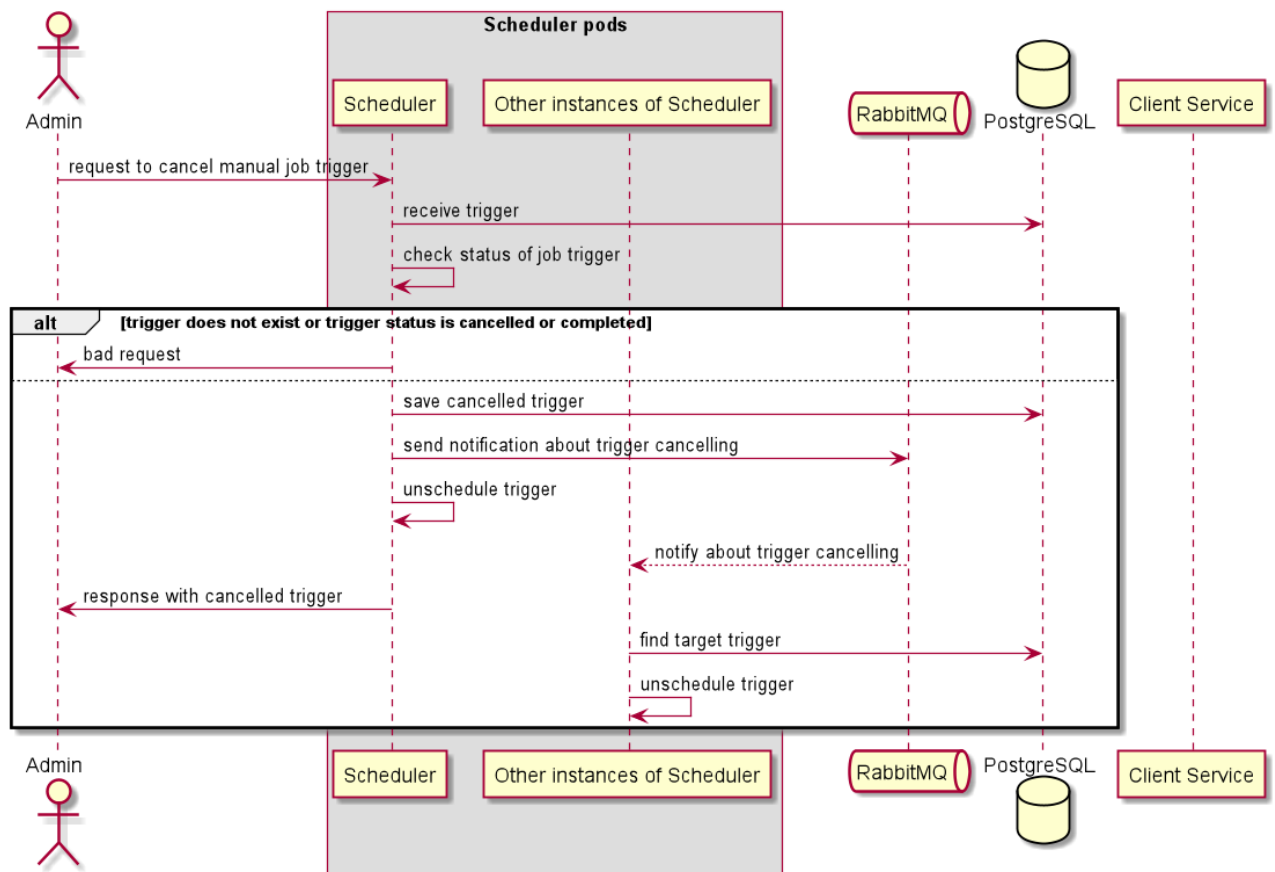


Рисунок 4.6 — Діаграма послідовності відміни тригера ручної задачі

Періодичні задачі використовують cron вираз для визначання періодичності запуску. Послідовність планування та запуску задачі зображено на рисунку 4.7. Такі задачі планування та виконання таких задач дещо відрізняється:

- тригери задач плануються на старті сервісу, одразу після зчитування актуальних задач з ZooKeeper;
- тригер спрацьовує;
- після спрацювання тригера задачі, у визначений час, відбувається збільшення lock на записі тригера, за допомогою функції в базі даних;
- зберігається та перевіряється кількість кортежів у базі даних, що було оновлено, якщо не було оновлено, отже задача запущена іншою реплікою, інакше виконується запуск задачі на клієнтському сервісі.

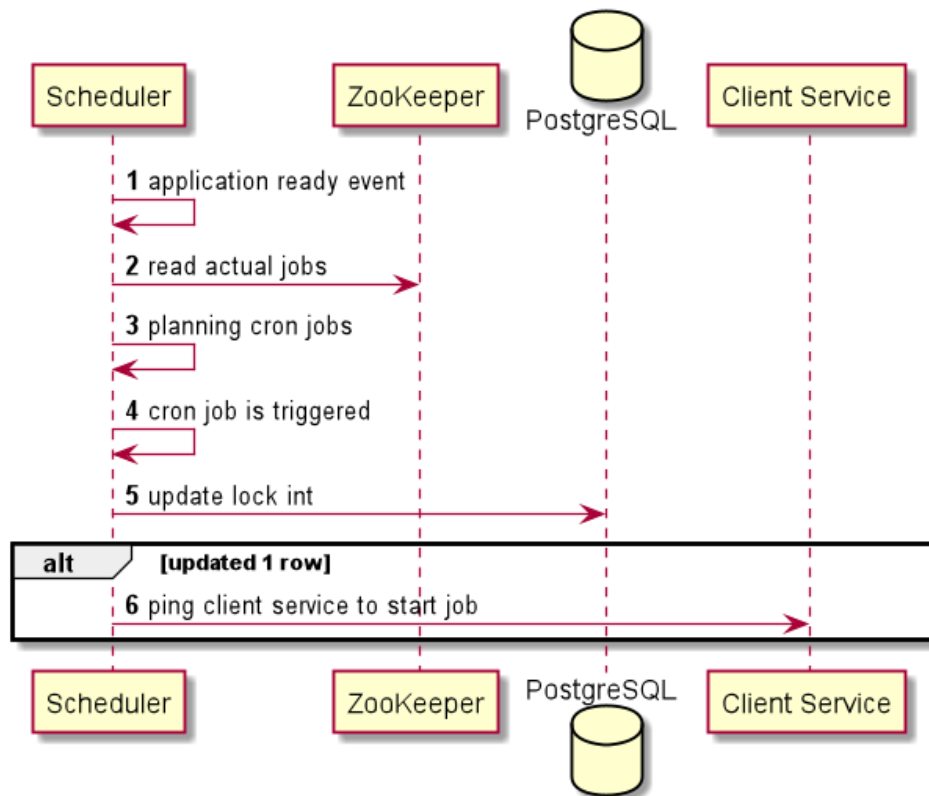


Рисунок 4.7 — Діаграма послідовності планування та запуску періодичної задачі

Конфігурацію таких задач можна змінити за допомогою HTTP запитів, що створено для підтримки динамічної зміни конфігурації. Діаграму послідовності зміни конфігурації адміністратором зображено на рисунку 4.8. Розглянемо послідовність дій:

- отримується HTTP запит на створення конфігурації тригерів визначеної користувачем;
- конфігурація зберігається до бази даних з поміткою, що це конфігурація користувача, що надає вищий пріоритет ніж конфігурації визначеної конфігурацією задачі за замовчуванням;
- після закриття транзакції надсилається повідомлення у RabbitMQ для інших екземплярів даного сервісу про зміну конфігурації;
- усі екземпляри є повідомлені про зміну конфігурації для тригерів;
- виконується перевірка конфігурації в базі даних та переплановуються тригери виконання задач.

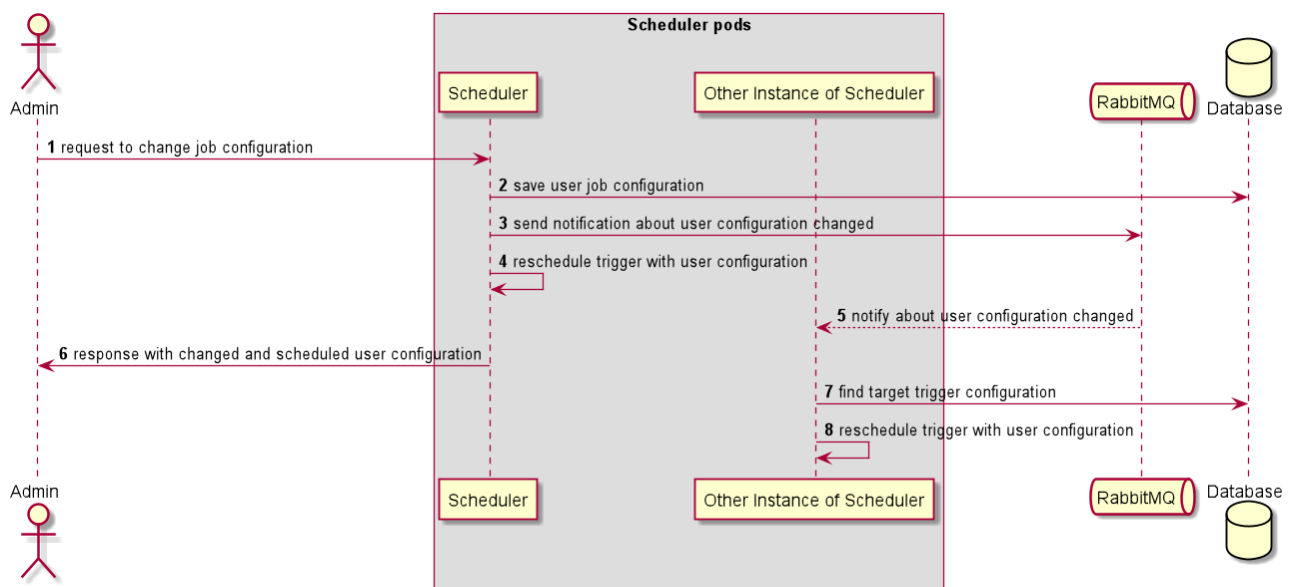


Рисунок 4.8 — Діаграма послідовності зміни конфігурації користувача для задачі

Також сервісом передбачено можливість видаляти конфігурацію визначену користувачем, відображено на рисунку 4.9, що відбувається наступним чином:

- отримується HTTP запит на видалення конфігурації тригерів визначеної користувачем;
- конфігурація видаляється з бази даних;
- після закриття транзакції надсилається повідомлення у RabbitMQ для інших екземплярів даного сервісу про видалення конфігурації користувача;
- усі екземпляри є повідомлені про зміну конфігурації для тригерів цієї задачі;
- виконується перевірка відсутності конфігурації в базі даних та переплановуються тригери виконання задач.

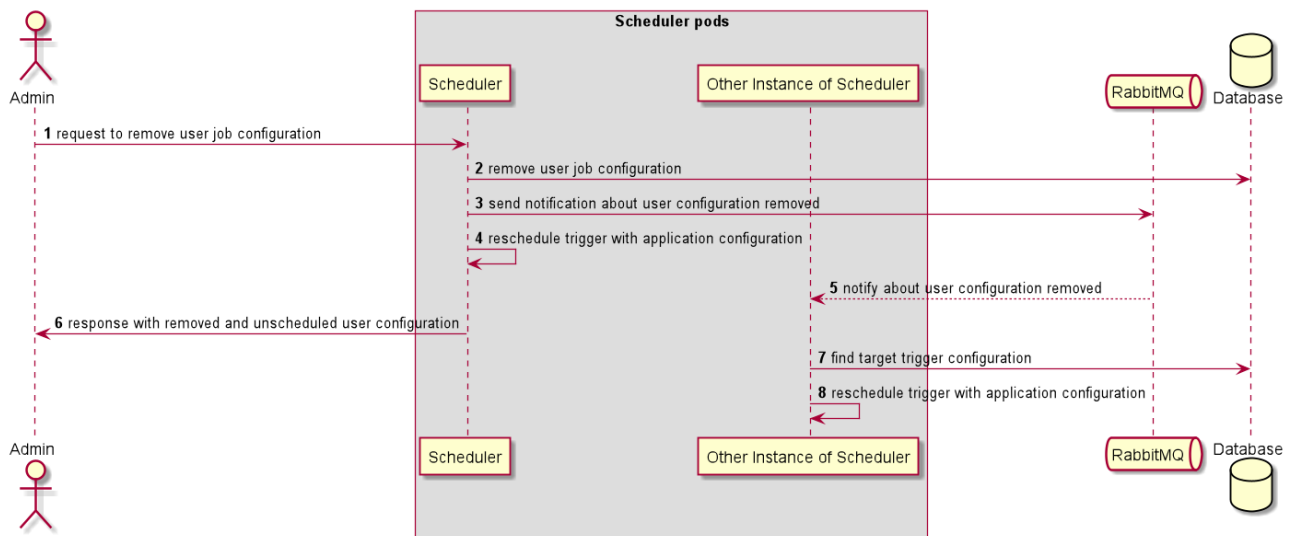


Рисунок 4.9 — Діаграма послідовності видалення конфігурації користувача для задачі

4.4 Моніторинг та управління виконанням задач

Присутність моніторингів є все більш актуальною в сучасних системах. Завдяки моніторингам можна відслідковувати послідовність виконання різного виду задач, спостерігати за продуктивністю та пропускнуою здатністю систем, побудова статистичних даних, збір інформації для звітів тощо. У сервісі планування та управління виконанням задач ці дані збираються на різних стадіях

виконання задачі. Також відбувається зміна стану виконання задач, які наведено в таблиці 4.2, стани можуть переходити у інші, що вказано на рисунку 4.10.

Таблиця 4.2 — Стани виконання задач

Стан	Кінцевий стан	Опис
STARTING	-	Даний статус означає, що спрацював тригер виконання задачі, проте виконання на клієнтському сервісі ще не розпочато.
STARTED	-	Цей статус створено для підтримки виконання асинхронних задач, при отриманні такого статусу задача вже виконується на мікросервісі клієнта
STOPPING	-	Оброблено запит на припинення в виконання задачі, але виконання ще триває.
STOPPED	+	Виконання задачі зупинено на мікросервісі клієнта
COMPLETED	+	Задача успішно виконана та отримано результати виконання задачі
UNKNOWN	-	Статус передбачає, що невідомо на якій стадії знаходиться виконання задачі, виникає при відсутності відповіді з мікросервісу клієнта
ERROR	+	На одному з етапів виконання задачі виникла помилка

Перші данні про початок виконання задачі створюються на етапі спрацювання тригера, у той момент виконання задачі отримує стан «STARTING», що означає підготовку для виконання задачі. У цей момент виконання отримує інформацію про ідентифікатор задачі, час початку виконання та ідентифікатор тригера, що спрацював.

Наступним кроком є запуск задачі на мікросервісі клієнта. В результаті виконання цього етапу повертається відповідь з сервісу, якщо відповідь відсутня виконання отримує стан «UNKNOWN», інакше встановлюється стан у відповідності до відповіді клієнтського сервісу, також при поверненні кінцевої відповіді, тобто виконання завершено успішно чи з помилкою, встановлюються час завершення та результати виконання або причини помилки.

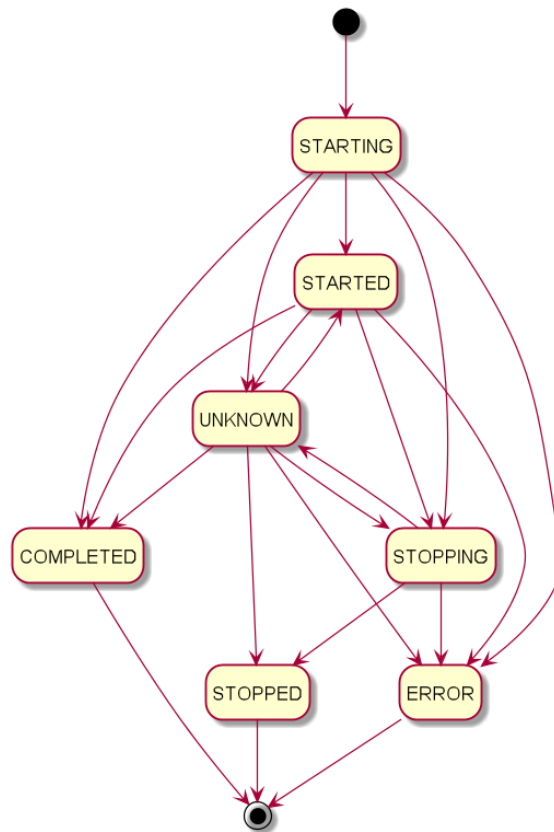


Рисунок 4.10 — Діаграма станів виконання задачі

Після початку виконання задачі на сервісі клієнта у разі повернення не кінцевого статусу можливе призупинення або отримання актуальної інформації для задач, що мають такі дії у конфігурації. В результаті виконання цих дій отримується актуальний стан з мікросервісу клієнта та оновлюється в базі даних.

Розроблено підтримку виконання асинхронних задач, що дозволяють клієнтським мікросервісам надсилати результати виконання через програмний інтерфейс до сервісу планування та управління виконанням задач.

4.5 Комунікація з клієнтськими сервісами

У попередньому розділі було досліджено та описано різні види комунікацій в мікросервісній архітектурі та було обрано використовувати архітектурний підхід REST, а для комунікаціями з клієнтськими мікросервісами Client-Side Service Discovery та Zookeeper в якості реєстру сервісів, тож було

спроектовано та реалізовано комунікацію між сервісом планування та управління виконанням задач, що зображено на рисунку 4.11.

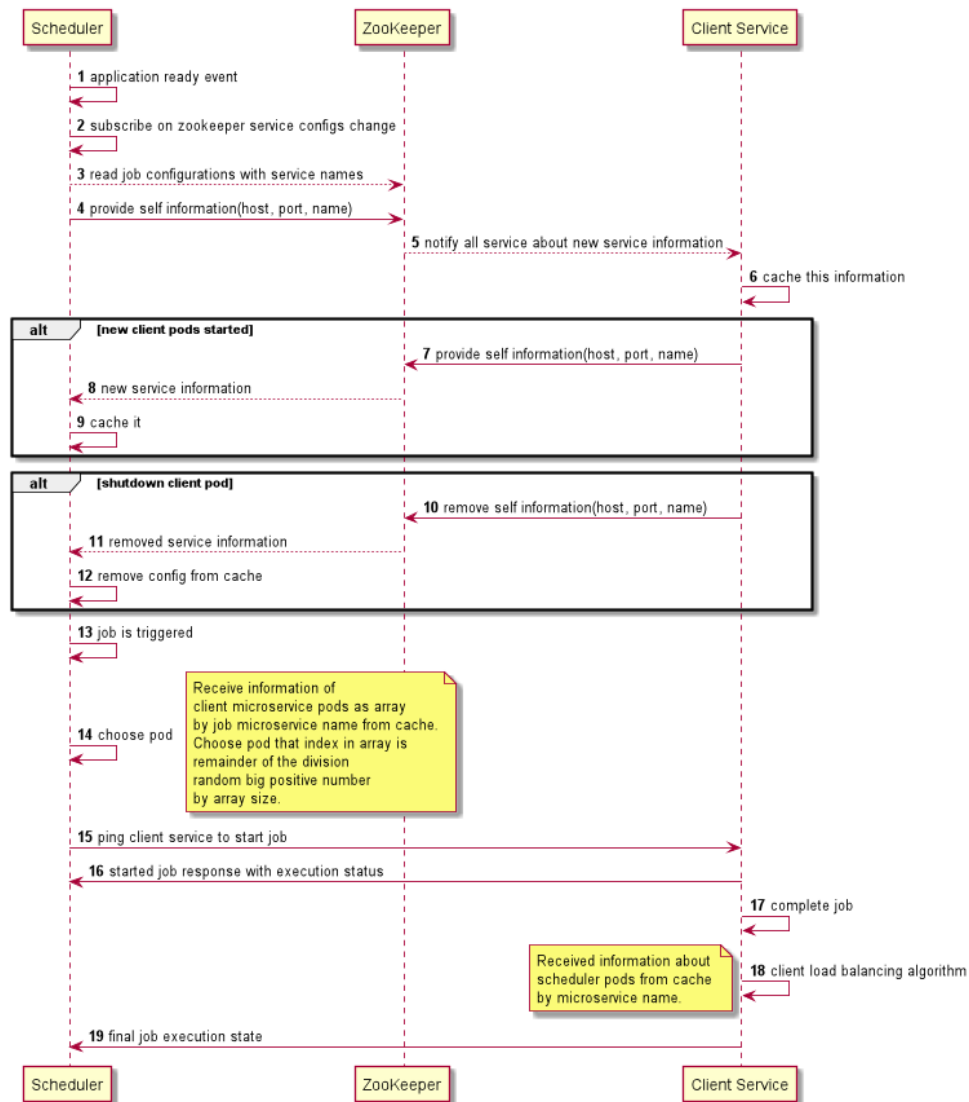


Рисунок 4.11 — Діаграма послідовності комунікацій з клієнтськими мікросервісами

Після старту сервісу планування та управління виконанням мікросервісних задач він підписується на оновлення в ZooKeeper змін щодо робочих клієнтських мікросервісів та вичитує всі данні про них собі в кеш, натомість публікує інформацію про себе. В інформацію, що зберігається у ZooKeeper входять IP, порт, назва мікросервісу та згенерований унікальний ідентифікатор.

ZooKeeper надає можливість підписки та надсилає інформацію про оновлення у відповідних конфігураціях, тож клієнтські мікросервіси мають також підписатися на ці оновлення та заносити де кешу.

Якщо мікросервіс припиняє свою роботу у ZooKeeper очищаються дані про нього та надсилає повідомлення про видалення конфігурації, що допомагає підтримувати конфігурацію в актуальному стані.

Коли сервісу планування та управління виконання задач необхідно запустити її на мікросервісі клієнта він отримує інформацію з кешу про всі існуючі екземпляри за назвою мікросервісу та обирає серед них визначивши адресу екземпляру сервісу клієнта на основі ділення випадково створеного великого числа на кількість існуючих вузлів сервісу клієнта конкретної задачі. Аналогічно спроектовано звернення мікросервісу клієнта для повернення інформації.

Комунікація з клієнтськими мікросервісами також передбачає створення ними відповідного API, що являє собою кінцеві точки для HTTP викликів представлених у таблиці 4.3. Серед наведених кінцевих точок є обов'язкові та не обов'язкові для реалізації, якщо у їх конфігурації відсутні відповідні дії над процесом виконання задачі.

Таблиця 4.3 — API клієнтських мікросервісів

Призначення	Шлях	HTTP метод	Опис
Запуск задачі	/v1/scheduler/jobs/{jobName}	POST	Кінцева точка для запитів на запуск задач, що передають параметри в тілі HTTP запиту. Визначення даного шляху є обов'язковим

Закінчення таблиці 4.3

Призначення	Шлях	HTTP метод	Опис
Отримання інформації про виконання	/v1/scheduler/jobs/{jobName}/instances/{instanceId}/get	POST	Кінцева точка для запитів на отримання інформації про виконання задач, що передають параметри в тілі HTTP запиту. Визначення даного шляху є обов'язковим, лише якщо конфігурація задачі передбачає зупинку
Зупинка виконання задачі	/v1/scheduler/jobs/{jobName}/instances/{instanceId}/stop	POST	Кінцева точка для запитів на зупинку виконання задач, що передають параметри в тілі HTTP запиту. Визначення даного шляху є обов'язковим, лише якщо конфігурація задачі передбачає зупинку

4.6 Оцінка ефективності запропонованого рішення

Оцінка ефективності обраних методів та алгоритмів було проведено на заздалегідь підготовленому тестовому середовищі. До складу тестового середовища входив тестовий клієнтський мікросервіс, що виконував задачу, ZooKeeper, RabbitMQ, PostgreSQL та безпосередньо сервіс планування мікросервісних задач. Тестові дані були сформовані для перевірки:

- пропускну здатності;
- ефективність масштабування;
- відмовостійкість процесів.

В таблиці 4.4 наведено результати тестування задач з відкладеним виконанням, що планувалися на один момент часу та графічно відображено на рисунку 4.12.

Таблиця 4.4 — Тестування задач з відкладеним виконанням

Кількість запущених задач	Різниця виконання першої та останньої задачі з одним екземпляром, с	Середній час виконання задачі з одним екземпляром, с	Максимальний час запуску задачі з одним екземпляром, с	Різниця виконання першої та останньої задачі з двома екземплярами, с	Середній час виконання задачі з двома екземплярами, с	Максимальний час запуску задачі з двома екземплярами, с
10	0,014	0,0072	0,0453	0,002	0,0007	0,048
100	0,667	0,395	0,6927	0,547	0,3237	0,568
1000	4,124	2,3019	4,1586	3,3982	1,6958	3,413

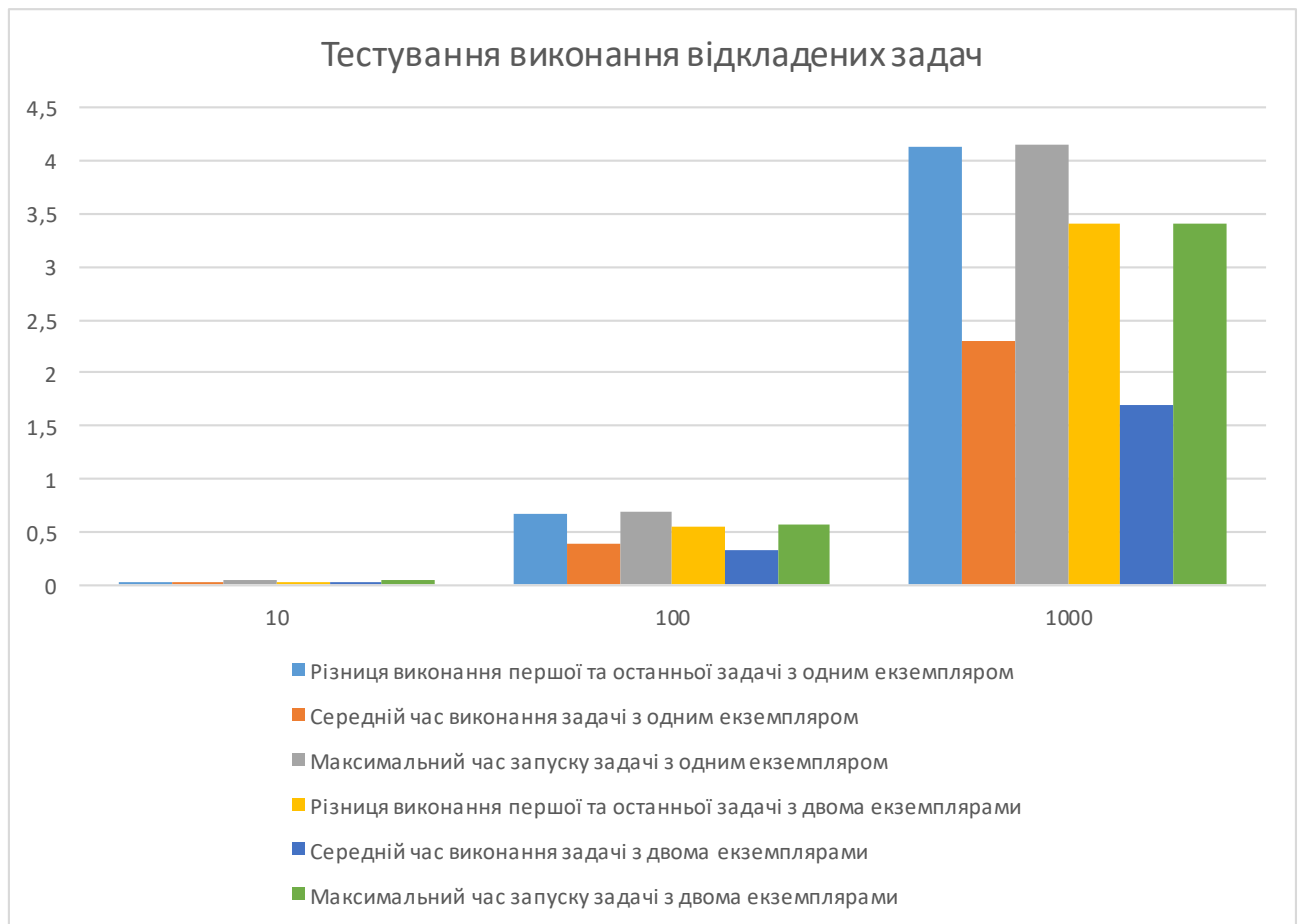


Рисунок 4.12 — Гістограма тестування задач з відкладеним виконанням

З проведеного експерименту очевидно, що 10 запущених відкладених задач не несуть особливого навантаження на систему, так як їх похибка

вимірюється в десятках мілісекунд, проте похибка значно виросла з кількістю запусчених задач і це зрозуміло, адже під кожен запусну задачу доводиться виділяти програмний потік, кількість яких не є безмежна, масштабування частково поліпшило ситуацію, але не набагато.

За такими ж принципами виконувалась перевірка виконання періодичних задач, але у даному випадку задачі проходили по 10 циклів виконання для точності експерименту. Результати тестування відображені в таблиці 4.5 та графічно на рисунку 4.13.

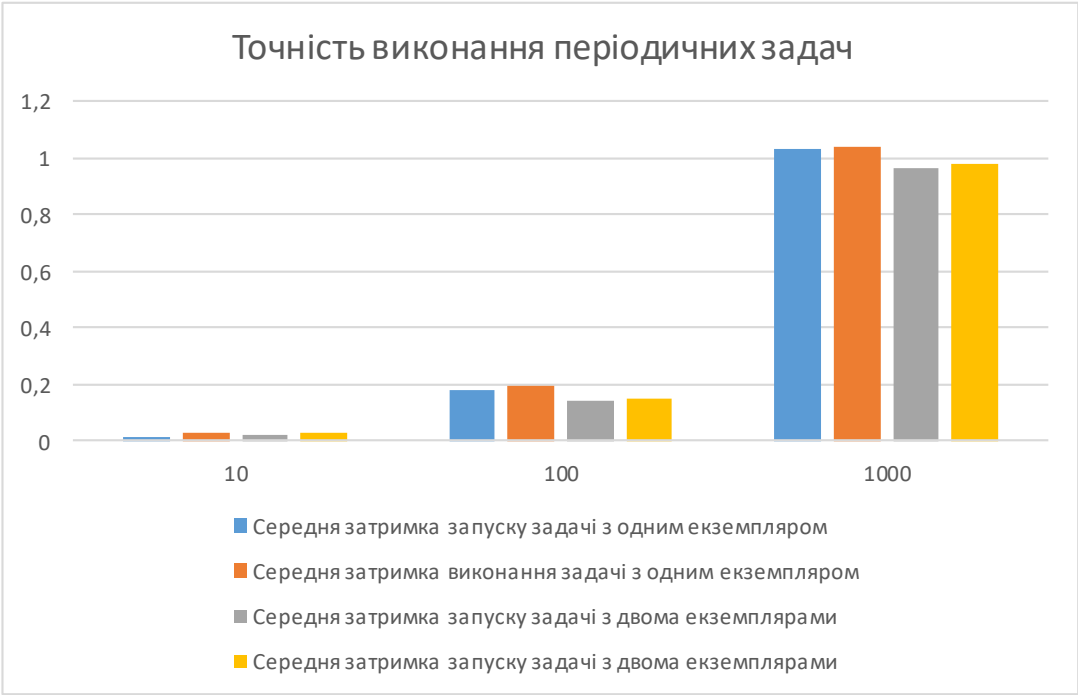


Рисунок 4.13 — Гістограма тестування періодичних задач

Таблиця 4.5 — Тестування задач з періодичним виконанням

Кількість запусчених задач	Час середньої затримки запуску задачі з одним екземпляром, с	Час середньої затримки виконання задачі з одним екземпляром, с	Час середньої затримки запуску задачі з двома екземплярами, с	Час середньої затримки виконання задачі з двома екземплярами, с
10	0,0177	0,03	0,02132	0,03
100	0,1814	0,1972	0,1421	0,1518
1000	1,0311	1,041	0,9645	0,9725

В контексті аналізу ефективності виконання періодичних задач очевидно пропорційний ріст середньої затримки виконання задач відповідно до кількості у конкретний момент часу. Також наглядно видно, що періодичні задачі виконуються вдвічі швидше за відкладені.

В результаті дослідження розроблених методів та програмних засобів було визначено пропускну здатність системи та відносну похибку у виконанні задач пов'язаних з затримками виконання. Причиною цих затримок є системний ресурс, мікросервіси отримали б набагато більший приріст продуктивності у виконаннях задач у ізольованих контейнерах та збільшенням виділення процесорного ресурсу, але такі навантаження мають дуже рідкісні випадки, тож напевно користувачі зустрінуться з даною проблемою. Також потрібно відзначити відмовостійкість системи, адже жодна задача не було проігнорована чи завершена з помилкою. Визначено роль та ефективність масштабування системи. Підводячи підсумок розробки та тестування даної системи, визначені переваги та недоліки дозволяють визнати систему ефективною.

Висновки до розділу

В розділі описано етап розробки алгоритмічного та програмного забезпечення системи. Наведено архітектуру програмного продукту та технічні вимоги, описано структуру бази даних та стани виконання мікросервісних задач. Відтворено реалізацію управління виконанням та плануванням мікросервісних задач, як відкладеного запуску так і періодичних. Розроблено методи комунікації сервісу планування та управління виконанням задач та клієнтських мікросервісів.

5 МАРКЕТИНГОВИЙ АНАЛІЗ СТАРТАП-ПРОЄКТУ

5.1 Опис ідеї проєкту

Таблиця 5.1 — Опис ідеї стартап-проєкту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Планування відкладених та періодичних мікросервісних задач і моніторинг їх виконання	Системи, що потребують регулярного створення звітів	Системи, що працюють з великими обсягами даних, і мають розподілену архітектуру, зможуть ефективно застосовувати свої потужності і не втрачати дані, що будуть оброблюватись планувальником на різних мікросервісах
	Очищення застарілих даних	

Таблиця 5.2 — Опис ідеї стартап-проєкту

№	Техніко-економічні характеристики ідеї	Продукція конкурентів			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проєкт	Quartz	Obsidian			
1	Динамічні конфігурації	+	-	+			Наявна підтримка динамічних конфігурацій задач, що забезпечує економію на витрати ресурсів
2	Точність планування	секунди	секунди	хвилини			Точність у плануванні є вагомим критерієм для вибору продукту, що дозволяє системам бути більш пунктуальними
3	Визнання в спільноті	-	+	+	Потрібно працювати над створенням власної спільноти для вирішення типових проблем системи		

Закінчення таблиці 5.2

№	Техніко-економічні характеристики ідеї	Мій проєкт	Quartz	Obsidian	W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
4	Початкові вкладення	-	-	+			Не потребує початкових вкладе
5	Підтримка масштабування	+	+	+		Усі програмні засоби підтримують масштабування	

5.2 Технологічний аудит ідеї проєкту

Таблиця 5.3 — Технологічна здійсненність ідеї проєкту

№	Ідея проєкту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Планування задач	Планування відкладених задач за вказаним часом та періодичних задач за cron виразом	Наявна	Бібліотеки з відкритим кодом
2	Динамічна зміна конфігурації задач	Отримання HTTP запиту та перепланування задач за допомогою RabbitMQ	Наявна	Технології доступні
3	Комунікація з клієнтським мікросервісом	Використовується інструмент ZooKeeper	Наявна	Відкрита для використання
4	Налаштування зручного програмного інтерфейсу	Реалізовується за архітектурним підходом REST	Наявна	Фреймворки
Обрана технологія реалізації ідеї проєкту: 1				

Висновок: технологічна реалізація продукту – можлива, вибрана технологія №1

5.3 Аналіз ринкових можливостей запуску стартап-проєкту

Таблиця 5.4 — Попередня характеристика потенційного ринку

№	Показники стану ринку	Характеристика
1	Кількість головних гравців, од	5-7
2	Загальний обсяг продаж, грн./ум.од	10-15 тис. / місяць
3	Динаміка ринку	Зростає
4	Наявність обмежень для входу	Наявність конкурентів
5	Специфічні вимоги до стандартизації та сертифікації	Відсутні
6	Середня норма рентабельності в галузі або по ринку, %	65-70%

Висновок: враховуючи кількість головних гравців по ринку, зростаючу динаміку ринку, невелику кількість конкурентів та середню норму рентабельності можна зробити висновок, що на даний момент, ринок для входження стартап-продукту є привабливим [21].

Таблиця 5.5 — Характеристика потенційних клієнтів стартап-проєкту

№	Потреба, що формує ринок	Цільова аудиторія	Відмінності у поведінці цільових груп клієнтів	Вимоги споживачів до товару
1	Потреба виконання задач за розкладом	Компанії, що використовують різноманітні системи автоматизації своїх процесів	Кожна з груп має свої сценарії використання продукту та свої вимоги до швидкості і якості надання результатів виконання	Зручна конфігурація; Можливість використання у мікросервісній архітектурі; Висока точність

Таблиця 5.6 — Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1	Новий продукт	Вихід на ринок, Зменшення монополії, Надання нових рішень у сфері	Розробка нової функціональності; Вихід нової продукції на ринок; Надання різноманітних типів ліцензій в залежності від потреб користувача \ замовника.
2	Вихід аналогу	Надати продукт з певними характеристиками та можливостями що відсутні у компаній конкурентів	Аналіз ринку та користувачів зادля задоволення їх потреб та надання функціональності у найкоротші строки за ціну, котра є дешевшою ніж у продуктів- замінників.
3	Зворотній зв'язок від користувачів	Можливість отримання необхідної інформації для вдосконалення продукту	Наявність вхідних даних та реакція на них з боку команди розробників задля задоволення потреб та бажань кінцевих користувачів системи.
4	Грошова винагорода за рекламу	При достатньому попиту на систему планування та запуску мікросервісних задач комерціалізація продукту на основі реклами задля отримання грошової винагороди для подальшого розвитку продукту	Точкова комерціалізація продукту; Введення реклами; Ведення додаткових коштів у проект задля його подальшого розвитку.

Таблиця 5.7 — Фактори загроз

№	Фактор	Зміст загрози	Можлива реакція компанії
1	Конкуренти	Наявність конкурентів котрі надають схожі рішення	Зменшення ціни на поставлену послугу; Розробка унікальних характеристик товару; Надання ліцензій на обслуговування
2	Кошти на розробку та підтримку продукту	Закінчення грошей та недостатнє фінансування	Залучення додаткових інвесторів, мотивація роботи на перспективу; Ітеративна розробка продукту задля покрокового виведення продукту на ринок та отримання відповіді користувачів
3	Вихід аналогу	Вихід аналогу даного товару може призвести до знецінення та безідейності даного товару	Вихід товару на ринок в коротші строки з не повною, але достатньою, функціональністю для зацікавлення усіх цільових аудиторій; Проведення рекламної компанії

Таблиця 5.8 — Ступеневий аналіз конкуренції на ринку

№	Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1	Тип конкуренції: монополістична	Товар від кожної компанії на ринку, являється недосконалим замінником товару, реалізованого іншими фірмами; На ринку є умови для входу та виходу; Ціна корелює між суперниками;	Розробка продукту з характеристиками, які покривають сфери вживання що не покривають інші товари-замінники; Кореляція цін у відповідності до товарів замінників; Різні типи ліцензій.

Закінчення таблиці 5.8

№	Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
2	Рівень конкурентної боротьби: світовий	Всі продукти замітники розроблялись інтернаціональними командами з різних куточків світу, продукти не належать до певної держави, а належать команді розробників	Вихід на ринок збуту продукту з клієнто-необхідною функціональністю; Налагодження маркетингу на основних Інтернет ресурсах задля охоплення великої кількості потенційних користувачів; Надання бета-версій продукту.
3	Галузева ознака: внутрішньогалузева	Даний тип продукту може використовуватися тільки у сфері розробки ІТ додатків \ продуктів	Надання зручного, інтуїтивно зрозумілого інтерфейсу; Підтримка всім відомих методів взаємодії з середовищем розробки; Наявність документації та он-лайн підтримки.
4	Конкуренція за видами товарів: товарно-видова	Дана конкуренція – конкуренція між товарами одного виду.	Впровадження функціональності яка відсутня у товарів-замінників; Спрощення інтерфейсів; Надання підтримки.
5	Характер конкурентних переваг: цінова та не цінова	Цінові переваги – точкова комерціалізація; Не цінова – надання функціональності, що відсутня у товарах-замінниках.	Надання платних ліцензій лише на критично важливу функціональність для клієнта з певним строком підтримки, що зазначена у відповідній ліцензії; Впровадження унікальної функціональності.
6	За інтенсивністю: марочна	Наявність унікального знаку що відрізняє даний продукт від продуктів-замінників	Впровадження власної назви та власного знаку.

Проаналізувавши можливості роботи на ринку з огляду на конкурентну ситуацію можна зробити висновок: оскільки кожний з існуючих продуктів не впливає у великій мірі на поточну ситуацію на ринку в цілому, кожний з існуючих продуктів має свою специфічну сферу використання та свої позитивні та негативні сторони щодо рішення певних типів задач, то робота та вихід на даний ринок є можливою і реалізованою задачею.

Для виходу на ринок продукт повинен мати функціонал що відсутній у продуктів-аналогів, повинен задовольняти потреби користувачів, мати необхідний та достатній функціонал з конфігурування, підтримку зі сторони розробників та можливість розробки спеціального функціоналу за відповідною ліцензією.

Таблиця 5.9 — Обґрунтування факторів конкурентоспроможності

№	Фактор конкурентоспроможності	Обґрунтування
1	Ціна	Продукт позиціонується як open-source, що надає переваги перед конкурентами з точки зору цінового фактору вибору
2	Зручність адміністрування	Технічна перевага продукту так як конкуренти мають складні механізми розгортання та складно адмініструються
3	Зручність налаштування	Технічна перевага продукту так як конкуренти мають складні механізми налаштування та потребують залучення спеціалістів з налаштування ПЗ зазначеного типу
4	Висока точність, пропускна здатність та масштабованість	Перевага продукту для клієнтів бізнес сегменту, які потребують точності з виконанням задач до секунд в умовах виконання сотень задач в один момент часу

Скорочені назви конкурентів, що містяться в таблиці: Q-Quartz, O-Obsidian.

Таблиця 5.10 — Порівняльний аналіз сильних та слабких сторін

№	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з запропонованим						
			-3	-2	-1	0	+1	+2	+3
1	Ціна	18	O		Q				
2	Зручність адміністрування	15	Q				O		
3	Зручність налаштування	15	Q			O			
4	Висока точність, пропускна здатність та масштабованість	20		O		Q			

Таблиця 5.11 — SWOT аналіз стартап-проекту

Сильні сторони (S): Ціна, зручність адміністрування, зручність налаштування, висока точність, пропускна здатність та масштабованість	Слабкі сторони (W): Можливість внесення власних виправлень в роботу системи
Можливості (O): Створення бібліотек інструментів адміністрування розподілених бізнес рішень в якості окремого продукту. Залучення користувачів до створення бази знань, що значно покращить інформаційне забезпечення. Створення статистичних даних та відтворення діаграм на основі планування та виконання задач.	Загрози (T): Поява широкої потреби в індивідуальних рішеннях, яка буде супроводжуватись наявністю великого переліку нових задач до розробки, які не будуть відноситись до цільової аудиторії загалом, а лише до конкретних клієнтів. Це приводить до того що немає чіткого бачення стратегії розвитку продукту, а отже перешкоджає якісному плануванню розвитку проекту в майбутньому.

Таблиця 5.12 — Альтернативи ринкового впровадження стартап-проекту

№	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Залучення фінансової та маркетингової допомоги від компаній зацікавлених в розробці даного продукту	Головний ресурс – партнери, даний ресурс - наявний	2-3 місяці
2	Реклама	Залучення власних коштів для реклами товару	1-2 місяці
3	Написання статей та опис товару на відомих ресурсах	Головний ресурс – час, даний ресурс - наявний	2-3 тижні
4	Презентація товару на хакатонах й інших ІТ заходах	Ресурс – час та гроші для участі, наявні	1-3 місяці

5.4 Розроблення ринкової стратегії проекту

Таблиця 5.13 — Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Компанії з системами автоматизації процесів	Низька	Низький	Висока	Висока
2	Організації, що використовують мікросервісну архітектуру	Висока	Низький	Низька	Низька
3	Бізнес, що потребує генерації регулярних звітів	Висока	Висока	Низька	Низька
Які цільові групи обрано: Бізнес, що потребує генерації регулярних звітів та організації, що проектують свої рішення за мікросервісною архітектурою					

Відповідно до проведеного аналізу можна зробити висновок, що підходящою цільовою групою для розповсюдження даного програмного продукту є працівники ІТ сфери, ІТ компанії в цілому та будь-які підприємства котрі використовують програмні продукти побудовані на мікросервісній архітектурі, та потребують виконання відкладених чи періодичних мікросервісних задач. Відповідно до стратегії охоплення ринку збуту товару обрано стратегію масового маркетингу, оскільки для підприємств, ІТ працівників та ІТ компаній у цілому надається стандартизований продукт з можливістю розширення функціональності за домовленістю (відповідно до ліцензії).

Таблиця 5.14 — Визначення базової стратегії розвитку

Обрана альтернатива розвитку проєкту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
Надання функціональності що відсутня у товарів-замінників, підтримка клієнтів	Проведення реклами, освітлення унікальної функціональності через інтернет ресурси та інші канали, контакт напряду з споживачами; формування лояльності і прихильності споживачів	Зниження ступеню замінності товару; Прихильність клієнтів; Відмітні властивості товару; Відмітні характеристики товару;	Стратегія диференціації

Таблиця 5.15 — Визначення базової стратегії конкурентної поведінки

Чи є проєкт «першопрохідцем» на ринку	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, які?	Стратегія конкурентної поведінки
Ні, оскільки є товари-замінники, але дані товари замінники не мають деякого необхідного функціоналу	Так, ціль компанії знайти нових споживачів та, частково, забрати існуючих у конкурентів задля задоволення потреб останніх	Компанія частково копіює характеристики товару конкурента, основна ціль компанії розробка нового унікального функціоналу, з підтримкою основного функціоналу конкурентів	Стратегія заняття конкурентної ніші

Таблиця 5.16 — Визначення стратегії позиціонування

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проєкту	Вибір асоціацій, які мають сформувати комплексну позицію власного проєкту
1	Лаконічна інформація по застосуванню	Створення документації з підказками для клієнтів	Система матиме вдалий інтерфейс	Зручність
2	Активна спільнота для вирішення проблем	Система відгуків про співпрацю	Створення форуму підтримки на вже існуючих платформах	Чуйність
3	Надійність програмного забезпечення	Впровадження різних типів тестувань	Добре протестований товар	Надійність

Відповідно до проведеного аналізу можна зробити висновок, що стартап-компанія вибирає як базову стратегію розвитку – стратегію диференціації, як базову стратегію конкурентної поведінки – стратегію заняття конкурентної ніші.

5.5 Розроблення маркетингової програми стартап-проєкту

Таблиця 5.17 — Визначення ключових переваг концепції потенційного товару

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1	Простота	Організація системи у зрозумілому для користувача вигляді	Відсутні надлишкові модулі та спрощена робота у системі
2	Адміністрування	Надання послуг для організацій	Безпечний та легкий для розуміння програмний засіб
3	Підтримка	Завжди можна звернутися за порадами в підтримку	Створити власний центр підтримки користувачів

Таблиця 5.18 — Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
1. Товар за задумом	Сервіс планування та управління виконанням мікросервісних задач		
2. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх/Тл/Е/Ор
	Висока пропускну здатність	Нм	Вр/Тх
	Швидкість роботи	М	Тх/Тл
	Масштабованість	М	Тх
	Легкість в налаштуванні	Нм	Е
	Динамічна зміна конфігурацій	М	Тх/Тл
	Підтримка каналів для адміністрування	М	Вр/Тх
3. Товар із підкріпленням	До продажу: наявна повна документація, акції на придбання декількох ліцензій, знижки для певних сегментів на покупку товару		
	Після продажу: додаткова підтримка спеціалістів налаштування, підтримка з боку розробника		
За рахунок чого потенційний товар буде захищено від копіювання: захист інтелектуальної власності, патент			

Таблиця 5.19 — Формування системи збуту

Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
Орієнтація на зручність використання продукту та обґрунтування його вартості. Також для більшості клієнтів важливі технічні міркування	1. Підтримка контактів із клієнтами 2. Сервісне обслуговування 3. Забезпечення зворотного зв'язку	0 - посередники відсутні	Вертикальна з прямою взаємодією між підтримкою та замовником

Таблиця 5.20 — Концепція маркетингових комунікацій

№	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Потреба у регулярному виконанні відкладених або періодичних задач	Форум та чат у меседжері	1. Легкість використання системи та її простота 2. Можливість удосконалення 3. Зворотний зв'язок з клієнтами	1. Формування у споживача знання про продукт 2. Інформування потенційних споживачів про продукт та його переваги 3. Створення умов до вступу в контакт з рекламодавцем	Система є легкою та зручною для використання

Як результат було створено ринкову (маркетингову) програму, що включає в себе визначення ключових переваг концепції потенційного товару, опис моделі

товару, визначення меж встановлення ціни, формування системи збуту та концепцію маркетингових комунікацій.

Висновки по розділу

В четвертому розділі описано стратегії та підходи з розроблення стартап-проєкту, визначено наявність попиту, динаміку та рентабельність роботи ринку, як висновок було вказано що існує можливість ринкової комерціалізації проєкту. Розглянувши потенційні групи клієнтів, бар'єри входження, стан конкуренції та конкурентоспроможність проєкту було встановлено що проєкт є перспективним. Розглянуто та вибрано альтернативу впровадження стартап-проєкту та доведено доцільність подальшої імплементації проєкту.

ВИСНОВКИ

В магістерській дисертації розглянуті питання, пов'язані з програмним забезпеченням для запуску відкладених та періодичних мікросервісних задач, а також їх планування та управління виконанням.

На основі даних, отриманих в процесі аналізу, сформульовано задачу спроектувати та реалізувати програмний засіб планування та управління виконанням задач для побудови мікросервісної архітектури.

Для розробки програмного засобу планування та управління виконанням мікросервісних задач використано платформу для створення додатків Spring Boot, бібліотеку планування задач Quartz, ORM фреймворк Hibernate, систему управління базами даних PostgreSQL, технологію міграції бази даних Flyway, ZooKeeper для реєстрації задач та даних про мікросервіси, брокер повідомлень RabbitMQ. Все перераховане є широко вживаними та безкоштовними інструментами, що дозволило створити високоточний та масштабований програмний засіб планування та управління виконанням мікросервісних задач.

Розроблена модель масштабування сервісу планування задач, з урахуванням динамічної зміни конфігурацій задач та їх виконання.

Розроблено програмне забезпечення для запуску простих та періодичних мікросервісних задач.

Проведений маркетинговий аналіз стартап-проєкту. Проведено опис ідеї проєкту, технологічний аудит ідеї проєкту, аналіз ринкових можливостей запуску стартап-проєкту. Розроблено ринкову стратегію проєкту та маркетингову програму стартап-проєкту.

Результати роботи над магістерською дисертацією опубліковано статті.

Наукова новизна одержаних результатів магістерської дисертації:

вперше:

- запропоновано модель масштабування сервісу планування задач, з урахуванням динамічної зміни конфігурацій задач.

удосконалено:

- метод побудови мікросервісної архітектури з урахуванням виконання та моніторингу розподілених задач;
- спрощено управління конфігураціями мікросервісних задач.

Практична значимість одержаних результатів полягає у підвищенні зручності та розширенні варіантів вибору програмних засобів для моніторингу та управління виконанням задач в мікросервісному середовищі

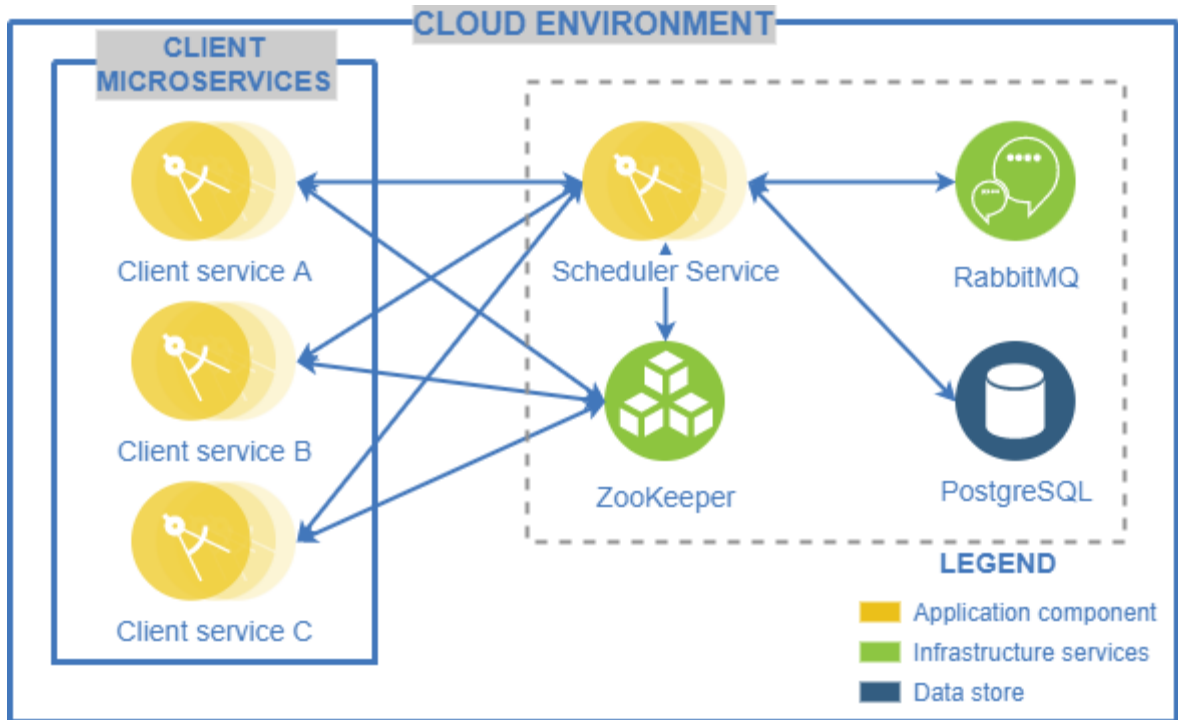
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- 1) Richardson's C. Microservice patterns. Manning, 2018. – 520 с.
- 2) Akilandeswari P., Srimathi H. Survey and Analysis on Task Scheduling in Cloud Environment. Indian Journal of Science and Technology, 2016.
- 3) Difference between rate monotonic and deadline monotonic scheduling [Електронний ресурс] – Режим доступу: <https://www.geeksforgeeks.org/difference-between-rate-monotonic-and-deadline-monotonic-scheduling/>
- 4) Earliest Deadline First [Електронний ресурс] – Режим доступу: <https://www.geeksforgeeks.org/earliest-deadline-first-edf-cpu-scheduling-algorithm/>
- 5) Maude L. Refactoring at Scale. O'REILLY, 2020. – 246 с.
- 6) Смірнов О.М. Методи та програмні засоби планування і управління виконанням мікросервісних задач / О.М. Смірнов // Матеріали X Міжнародної науково-практичної конференції «SCIENCE FOUNDATIONS OF MODERN SCIENCE AND PRACTICE» – Афіни, Греція, 23-26 листопада 2021р.
- 7) Ingeno J. Software Architect's Handbook. Packt, 2018. – 594 с.
- 8) Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. O'REILLY, 2020. – 426 с.
- 9) Основні принципи об'єктно-орієнтованого програмування [Електронний ресурс] – Режим доступу: http://uk.wikipedia.org/wiki/Об'єктно-орієнтоване_програмування
- 10) Б. Дж. Эванс, Д. Флэнаган Java. Справочник разработчика. O'REILLY, 2019. – 594 с.
- 11) Sciore E. Java Program Design. Apress, 2019. – 456 с.
- 12) Ганс-Юрген Шениг. PostgreSQL 11. Мастерство разработки. ДМК, 2019. – 352 с.

- 13) Огляд технологій для розробки на мові програмування Java [Електронний ресурс] – Режим доступу: <https://www.baeldung.com/>
- 14) Раджпут Д. Spring. Все паттерны проектирования. Packt, 2019. – 320 с.
- 15) Apache ZooKeeper [Електронний ресурс] – Режим доступу: <https://zookeeper.apache.org/>
- 16) Quartz Cookbook [Електронний ресурс] – Режим доступу: <http://www.quartz-scheduler.org/documentation/quartz-2.3.0/cookbook/>
- 17) RabbitMQ [Електронний ресурс] – Режим доступу: <https://www.rabbitmq.com/>
- 18) Таненбаум Э., Уэзеролл Д. Компьютерные сети. Питер, 2018. – 960 с.
- 19) Spring документація [Електронний ресурс] – Режим доступу: <https://spring.io>
- 20) Крейг У. Spring в действии. ДМК Пресс, 2015. – 752 с.
- 21) Розроблення стартап-проєкту: Методичні рекомендації до виконання розділу магістерських дисертацій для студентів інженерних спеціальностей / За заг. ред. О.А. Гавриша. – Київ : НТУУ «КПІ», 2016. – 28 с.

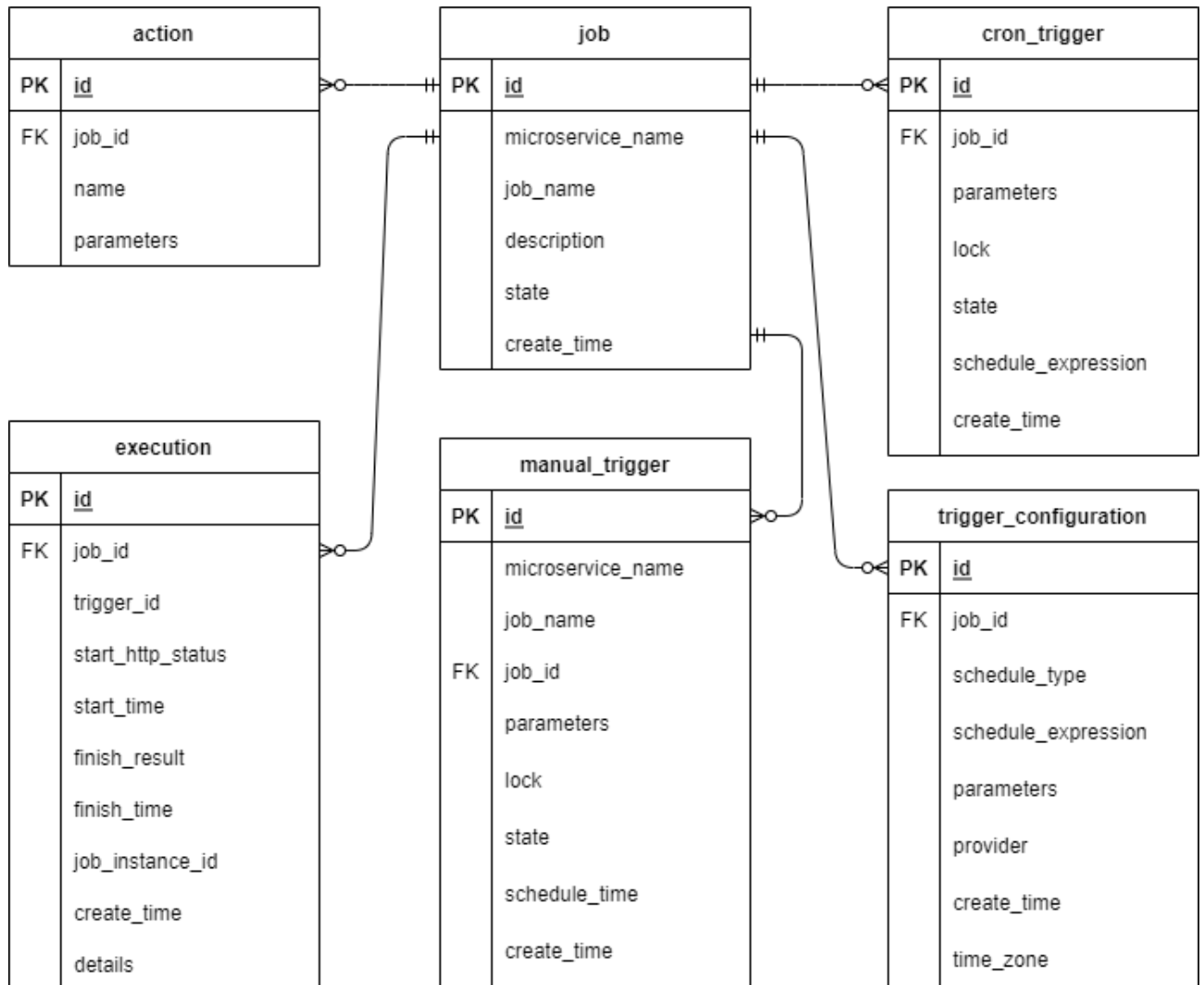
ДОДАТОК А

Архітектура системи



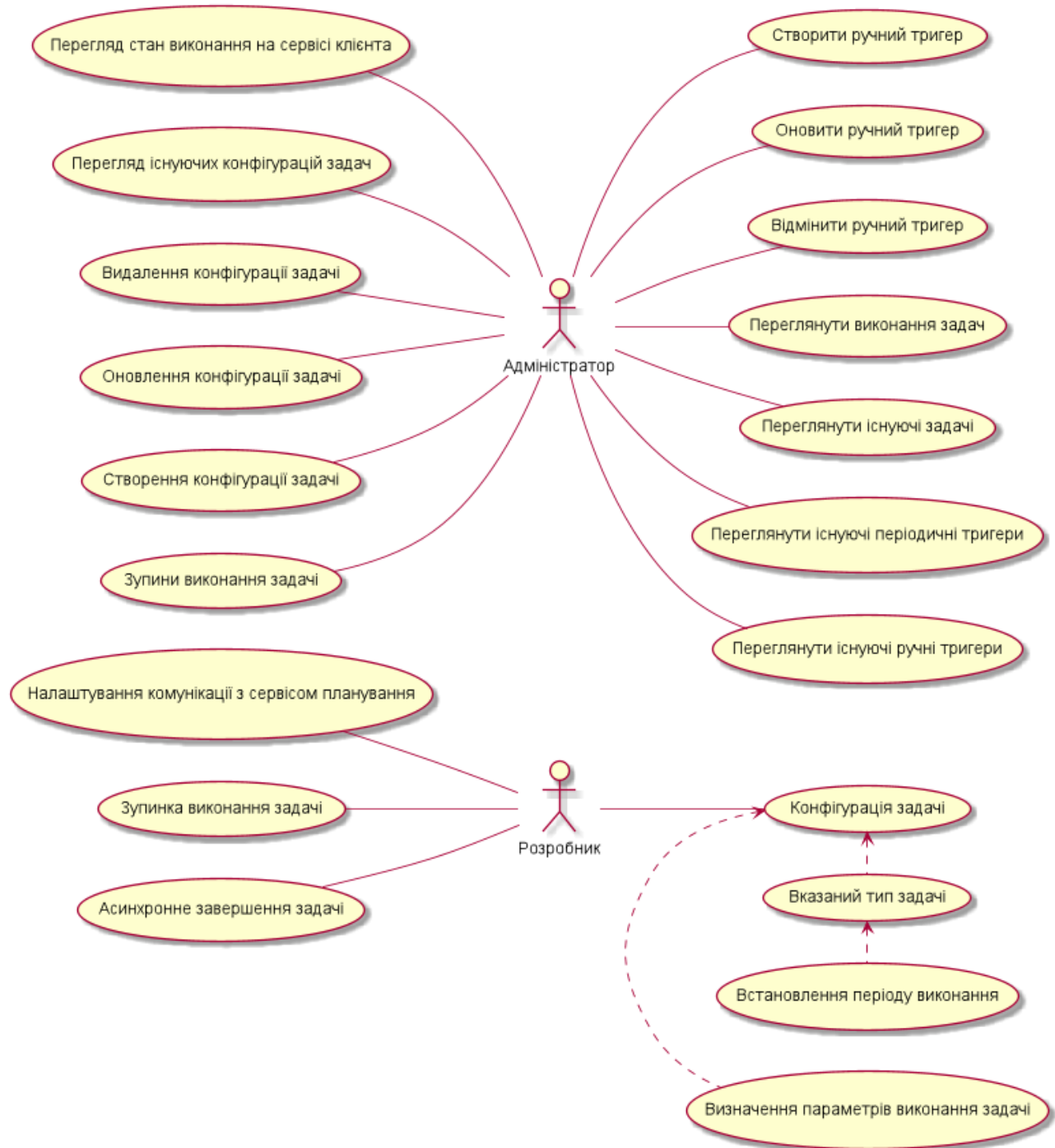
ДОДАТОК Б

Схема бази даних



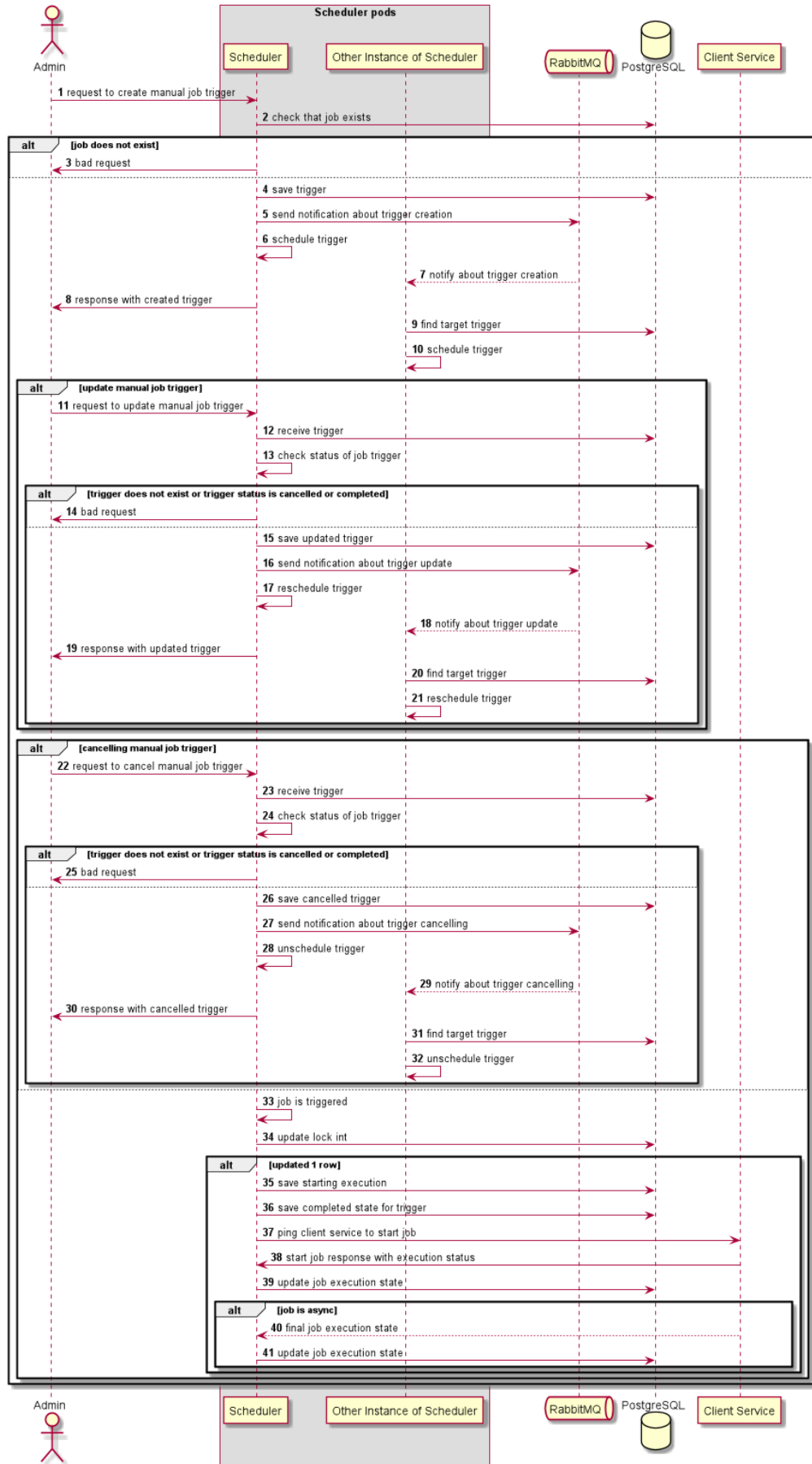
ДОДАТОК В

Діаграма прецедентів



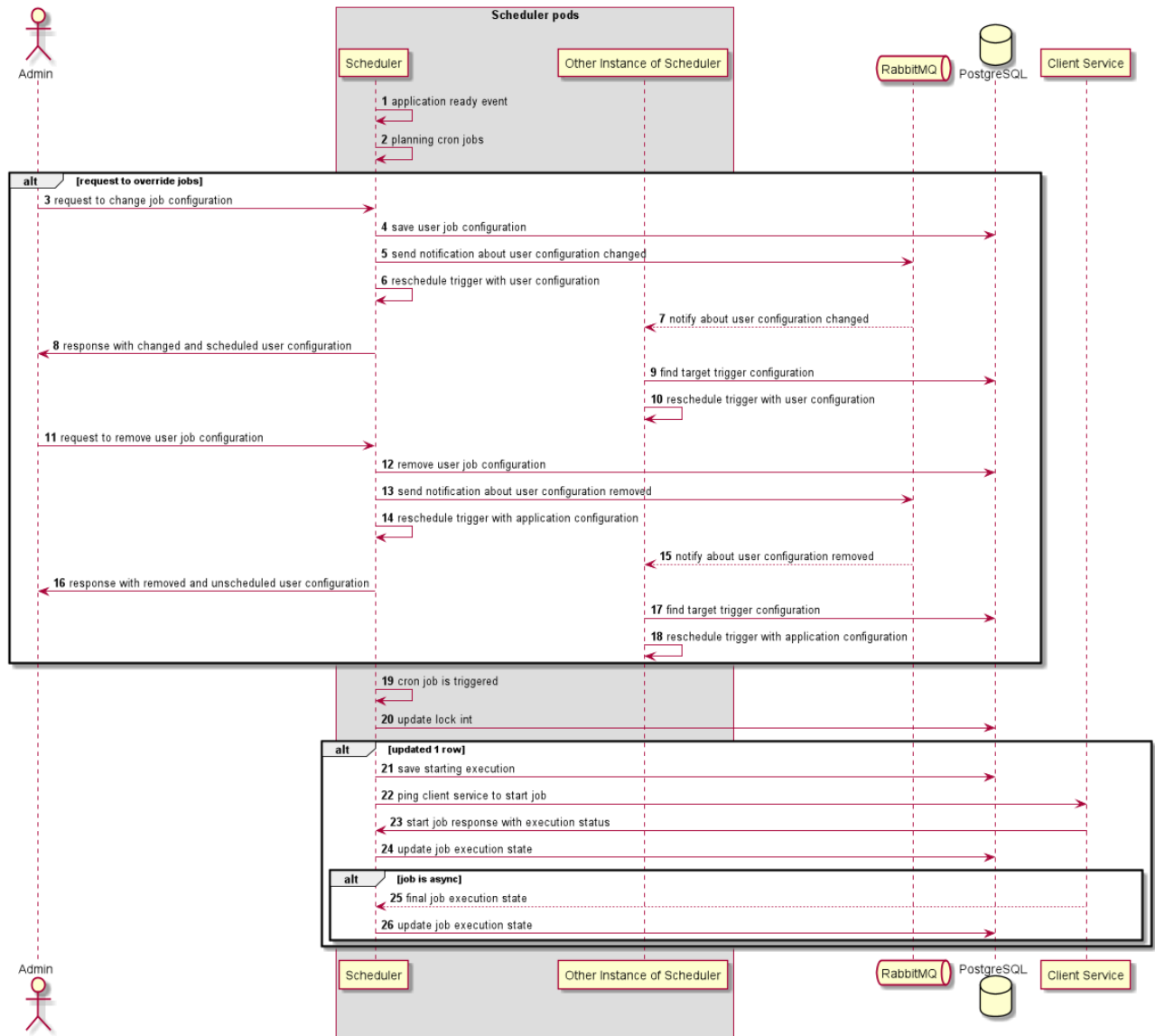
ДОДАТОК Г

Діаграма послідовності виконання відкладених задач



ДОДАТОК Д

Діаграма послідовності виконання періодичних задач за умови зміни конфігурації



ДОДАТОК Е

Лістинг програмного коду

ManualTriggerServiceImpl.java

```

@Service
@Slf4j
@RequiredArgsConstructor
public class ManualTriggerServiceImpl implements ManualTriggerService {

    private final JpaJobRepository jpaJobRepository;
    private final JpaManualTriggerRepository jpaManualTriggerRepository;

    private final ManualTriggerConverter manualTriggerConverter;
    private final QuartzManualTriggerService quartzManualTriggerService;
    private final NotificationSenderService notificationSenderService;

    @Override
    @Transactional
    public ManualTrigger createTrigger(CreateManualTrigger manualTrigger) {
        JobEntity jobEntity = getJob(manualTrigger.getJobId());

        if (jobEntity == null) {
            throw new BadRequestException();
        }

        ManualTriggerEntity manualTriggerEntity = saveAndNotifyOtherInstances(manualTrigger,
jobEntity);

        quartzManualTriggerService.scheduleTrigger(jobEntity, manualTriggerEntity);

        return manualTriggerConverter.toDto(manualTriggerEntity);
    }

    private JobEntity getJob(UUID jobId) {
        return jpaJobRepository.findById(jobId)
            .orElseThrow(() -> new RuntimeException());
    }

    private ManualTriggerEntity saveAndNotifyOtherInstances(CreateManualTrigger triggerDto,
JobEntity jobEntity) {
        ManualTriggerEntity manualTriggerEntity = manualTriggerConverter.createEntity(triggerDto,
jobEntity);

        saveAndNotifyOtherInstances(manualTriggerEntity);

        return manualTriggerEntity;
    }

```

```

private void saveAndNotifyOtherInstances(ManualTriggerEntity manualTriggerEntity) {
    jpaManualTriggerRepository.save(manualTriggerEntity);
    String triggerId = manualTriggerEntity.getId().toString();
    SchedulerNotificationMessage message =
        NotificationFactory.createOrUpdateManualTrigger(triggerId);
    notificationSenderService.sendNotificationAfterTransactionComplete(message);
}
}

```

NotificationSenderService.java

```

@Slf4j
@Service
@RequiredArgsConstructor
@Transactional(propagation = Propagation.MANDATORY)
public class NotificationSenderService {

    private final RabbitMqProducer rabbitMqProducer;

    public void sendNotificationAfterTransactionComplete(SchedulerNotificationMessage
schedulerNotificationMessage) {
        TransactionSynchronizationManager.registerSynchronization(new
TransactionSynchronizationAdapter() {
            public void afterCommit() {
                log.info("Send notification message after commit {})",
schedulerNotificationMessage.getSerialId());
                rabbitMqProducer.sendNotification(schedulerNotificationMessage);
            }
        });
    }
}

```

TriggerConfigurationServiceImpl.java

```

@Service
@RequiredArgsConstructor
public class TriggerConfigurationServiceImpl implements TriggerConfigurationService {

    private final JpaJobRepository jpaJobRepository;
    private final JpaJobTriggerConfigurationRepository jobTriggerConfigurationRepository;
    private final TriggerConfigurationConverter configurationConverter;
    private final QuartzCronTriggerService quartzCronTriggerService;
    private final CronTriggerService cronTriggerService;
    private final NotificationSenderService notificationSenderService;

    @Override
    @Transactional
    public TriggerConfiguration createTriggerConfiguration(CreateTriggerConfiguration
triggerConfiguration) {
        validateConfiguration(triggerConfiguration);
    }
}

```

```

    JobEntity jobEntity = jpaJobRepository.findById(triggerConfiguration.getJobId())
        .orElseThrow(() -> new NotFoundException());

    validateUserConfigurationExistence(jobEntity);

    TriggerConfigurationEntity configuration =
configurationConverter.createConfiguration(triggerConfiguration, jobEntity);
    configuration = jobTriggerConfigurationRepository.save(configuration);

    SchedulerNotificationMessage message =
        NotificationFactory.createOrUpdateTriggerConfiguration(
            configuration.getId().toString(), jobEntity.getId().toString());
    notificationSenderService.sendNotificationAfterTransactionComplete(message);
    overrideJobInQuartz(configuration);

    return configurationConverter.toDto(configuration);
}

private void overrideJobInQuartz(TriggerConfigurationEntity configuration) {
    JobEntity jobEntity = configuration.getJobEntity();
    List<CronTriggerEntity> triggers = cronTriggerService.generateCronTriggers(jobEntity);
    quartzCronTriggerService.scheduleTriggers(jobEntity, triggers, true);
}

private void validateUserConfigurationExistence(JobEntity jobEntity) {
    if (jobEntity.userTriggerConfiguration() != null) {
        throw new BadRequestException();
    }
}
}

```

ExecutionManagementServiceImpl.java

```

@Service
@Slf4j
@RequiredArgsConstructor
public class ExecutionManagementServiceImpl implements ExecutionManagementService {

    private final JpaManualTriggerRepository jpaManualTriggerRepository;
    private final JpaExecutionRepository jpaExecutionRepository;
    private final LockRepository lockRepository;
    private final ExecutionGenerator executionGenerator;
    private final ClientService clientService;

    @Override
    public ExecutionEntity execute(QuartzJobData quartzJobData,
                                   QuartzTrigger quartzTrigger) {
        if (!lockRepository.lockAndUpdateTrigger(quartzTrigger)) {
            return null;
        }
        Parameters parameters = quartzTrigger.getParameters();
        ExecutionEntity execution = beforeStart(quartzJobData, quartzTrigger);
    }
}

```

```

        if (execution == null) {
            return null;
        }

        try {
            Parameters executionParameters = executionGenerator.generateParameters(parameters,
execution);
            ResponseEntity<JobInstance> responseEntity =
                clientService.execute(
                    quartzJobData.getMicroserviceName(),
                    quartzJobData.getJobName(),
                    executionParameters);
            afterStart(execution, responseEntity);
        } catch (Exception e) {
            onError(execution, e);
            throw e;
        }

        return execution;
    }

    @Override
    @Transactional(propagation = Propagation.REQUIRES_NEW)
    public ExecutionEntity beforeStart(QuartzJobData quartzJobData,
        QuartzTrigger quartzTrigger) {
        ExecutionEntity execution =
            executionGenerator.generateExecution(quartzJobData, quartzTrigger);
        jpaExecutionRepository.save(execution);

        completeManualTrigger(quartzTrigger);

        return execution;
    }

    @Override
    @Transactional(propagation = Propagation.REQUIRES_NEW)
    public void afterStart(ExecutionEntity execution,
        ResponseEntity<JobInstance> responseEntity) {
        updateExecutionAfterResponse(execution, responseEntity);
        jpaExecutionRepository.save(execution);
    }

    @Override
    @Transactional(propagation = Propagation.REQUIRES_NEW)
    public void onError(ExecutionEntity execution,
        Exception e) {
        execution.setStatus(JobStatuses.ERROR);
        execution.setFinishTime(OffsetDateTime.now());
        execution.setFinishResult(e.getMessage());
        jpaExecutionRepository.save(execution);
    }
}

```

ClientServiceImpl.java

```

@Service
@RequiredArgsConstructor
public class ClientServiceImpl implements ClientService {

    private final DiscoveryClient discoveryClient;
    private final RestTemplate restTemplate;

    @Override
    public ResponseEntity<JobInstance> execute(String microserviceName, String jobName,
                                              Parameters parameters) {
        String url = resolveClientURL(microserviceName) + startPath();
        HttpEntity<Parameters> request = getHttpEntity(parameters);
        return restTemplate.postForEntity(url, request, JobInstance.class, jobName);
    }

    private String resolveClientURL(String microserviceName) {
        List<ServiceInstance> list = discoveryClient.getInstances(microserviceName);
        if (CollectionUtils.isEmpty(list)) {
            throw new UnsupportedOperationException("No clients url for this microservice");
        }
        ServiceInstance instance;
        if (list.size() == 1) {
            instance = list.get(0);
        } else {
            instance = list.get(Math.abs(UUID.randomUUID().hashCode() % list.size()));
        }
        return Optional.of(instance)
            .map(ServiceInstance::getUri)
            .map(URI::toString)
            .get();
    }

    private HttpEntity<Parameters> getHttpEntity(Parameters parameters) {
        HttpHeaders httpHeaders = new HttpHeaders();
        httpHeaders.setContentType(MediaType.APPLICATION_JSON);
        return new HttpEntity<>(parameters, httpHeaders);
    }
}

```

ДОДАТОК Є

Звіт перевірки на співпадіння



Имя пользователя:
Лісовиченко Олег Іванович

ID проверки:
1009331226

Дата проверки:
24.11.2021 14:04:41 EET

Тип проверки:
Doc vs Internet + Library

Дата отчета:
24.11.2021 14:05:23 EET

ID пользователя:
76913

Название файла: IT-03мп_Смірнов_ПЗ

Количество страниц: 51 Количество слов: 9514 Количество символов: 78207 Размер файла: 89.60 KB ID файла: 1009354889

8.44% Совпадения

Наибольшее совпадение: 4.28% с источником из Библиотеки (ID файла: 1004129200)

2.48% Источники из Интернета 82 Страница 53

8.27% Источники из Библиотеки 655 Страница 54

0% Цитат

Исключение цитат выключено

Исключение списка библиографических ссылок выключено

0% Исключений

Нет исключенных источников