Вінницький національний технічний університет
(повне найменування вищого навчального закладу)
Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту)
Кафедра обчислювальної техніки
(повна назва кафедри)

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Високопродуктивна система управління потоками даних на основі MySQL і MongoDB»

Виконав: студент 2-го курсу, групи 1КІ-20м
спеціальності 123 – Комп'ютерна інженерія
(шифр і назва напряму підготовки, спеціальності)
Шевчук М. О.
(прізвище та ініціали)
Керівник: к.т.н., доцент каф. ОТ
Ткаченко О. М.
(прізвище та ініціали)
«____» _____ 2021 р.
Опонент: к.т.н., професор каф. ЗІ
Кондратенко Н. Р.
(прізвище та ініціали)
«____» _____ 2021 р.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
(прізвище та ініціали)
«____» _____ 2021 р

Вінниця ВНТУ 2021

АНОТАЦІЯ

УДК 004.042

Шевчук М. О. Високопродуктивна система управління потоками даних на основі MySQL і MongoDB. Магістерська кваліфікаційна робота зі спеціальності 123 — комп’ютерна інженерія, освітня програма — комп'ютерна інженерія. Вінниця: ВНТУ, 2021. 115 с.

На укр. мові. Бібліогр.: 21 назв; рис.: 16; 21 табл.

В магістерській дипломній роботі були досліджені та проаналізовані підходи до організації архітектури мікросервісної програми з двома базами даних. Вибрані оптимальні методи та засоби для реалізації цієї архітектури. Були проведені дослідження продуктивності роботи баз даних з різними структурами та зроблені певні висновки про те як краще їх організовувати. Була розроблена тестова програма яка реалізує вищеописані принципи та використовує нові технології для відправки даних з бази даних по HTTP протоколу. Було проведено тестування та розроблено інструкцію користувача.

Високопродуктивна система управління потоками даних на основі MySQL і MongoDB розроблена для використання в будь-якій системі яка має JVM за допомогою середовища розробки IntellijIDEA Ultimate та мови програмування Java.

Ключові слова: Java, Spring Framework, Spring Boot, MySQL, MongoDB, база даних, поєднання баз даних, мікросервісна архітектура, мікросервіси, потоки даних, оптимізація.

ABSTRACT

Shevchuk M. O. High-performance data flow management system based on MySQL and MongoDB. Master's thesis in specialty 123 - computer engineering, educational program - computer engineering. Vinnytsia: VNTU, 2021. 115 p.

In Ukrainian language. Bibliography: 21 titles; fig.: 16; 21 tab.

In the master's thesis there were researches and analyzed approaches to the organization of microservice program architecture with two databases. The best methods and tools for implementing this architecture are selected. Research has been conducted on the work of the database with performance structures and some conclusions have been drawn on how best to organize them. A test program has been developed that implements the principles described above and uses new technologies to send data from a database via HTTP. Testing and user manual were developed.

The high-performance MySQL and MongoDB-based data flow management system is designed for use on any system that has a JVM through IntellijIDEA Ultimate development and Java programming.

Keywords: Java, Spring Framework, Spring Boot, MySQL, MongoDB, database, database connection, microservice architecture, microservices, data flows, optimization.

Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

Рівень вищої освіти ІІ-й (магістерський)

Галузь знань — _____

Спеціальність — 123 — Комп'ютерна інженерія

Освітньо-професійна програма — Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри
обчислювальної техніки
проф., д.т.н. О. Д. Азаров

_____

«___»_____2021 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФАКАЦІЙНУ РОБОТУ СТУДЕНТУ

Шевчуку Максиму Олеговичу
(прізвище, ім'я, по-батькові)

1 Тема роботи «Високопродуктивна система управління потоками даних на основі MySQL і MongoDB» Керівник роботи Ткаченко Олександр Миколайович, к.т.н., доцент. Затверджені наказом вищого навчального закладу від "06" березня 2021 року № 75.

2 Строк подання студентом роботи 21.12.2021.

3 Вихідні дані до роботи: проаналізувати існуючі технології для створення високопродуктивної системи з двома базами даних. Розробити систему з мікросервісною архітектурою на базі MySQL та MongoDB.

4 Зміст текстової частини: аналіз підходів до створення мікросервісних систем з двома базами даних, математичне обґрунтування продуктивності роботи баз даних з різними структурами даних, опис рекомендацій для підвищення продуктивності баз даних та створення програмного додатку який

реалізує вищеописані принципи, тестування додатку та написання інструкції користувача, обґрунтування економічної доцільності розробки.

5 Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): архітектура системи, гістограма росту продуктивності, UML-діаграма відношень в MySQL, UML-діаграма документів в MongoDB.

6 Консультанти розділів проекту (роботи):

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1,2,3,4	Ткаченко О. М., к.т.н., доцент каф. ОТ		
5	Кавецький В. В., к.е.н., доцент каф. ЕПВМ		

7 Дата видачі завдання 15.09.2021.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі роботи	21.09.21	
2	Аналіз підходів до створення мікросервісних систем з двома базами даних	15.10.21	
3	Математичне обґрунтування продуктивності роботи баз даних з різними структурами даних	01.11.21	
4	Опис рекомендацій для підвищення продуктивності баз даних та створення програмного додатку який реалізує систему з двома базами даних	28.11.21	
5	Тестування додатку та написання інструкції користувача	05.12.21	
6	Обґрунтування економічної доцільності розробки	11.12.21	
7	Оформлення пояснювальної записки	21.12.21	

Студент _____ Шевчук М. О.

(підпис) (прізвище та ініціали)

Керівник роботи _____ Ткаченко О. М.

(підпис) (прізвище та ініціали)

ЗМІСТ

ВСТУП

У сучасному світі, де час є одним із найважливіших ресурсів, люди потребують високого рівню комфорту в користуванні різноманітними цифровими системами. Зазвичай ніхто не хоче довго чекати, тому для забезпечення цього комфорту потрібно створювати швидкі, оптимізовані програми. Для цього було створено велику кількість різноманітних мов програмування, які ефективні кожна в своїх сферах інформаційного простору.

Однак не тільки вірний вибір мови програмування сприяє покращенню швидкодії системи, також важливі технології, бібліотеки, фреймворки, бази даних та архітектура. Кожна з цих речей в значній мірі впливає на кінцевий результат, тому до їх вибору потрібно віднестися з максимальною відповідальністю.

Також важливим фактором в комфорті користування системою є відмовостійкість та безпека.

В багатьох проектах де є клієнт та сервер використовується база даних. Як правило використовують реляційні бази даних але з появою NoSQL з’явились нові способи організації інформації. У кожної з цих методик є свої плюси та мінуси. Тому з метою покращення продуктивності обробки даних було б доцільно об’єднати і виділити переваги кожної з баз даних.

Для коректної та продуктивної роботи баз доцільно було б використати підходи мікросервісної архітектури і винести їх в окремі сервіси, що підвищить відмовостійкість і безпеку системи[1].

Метою дослідження магістерської роботи є підвищення продуктивності серверних застосунків за допомогою мікросервісної архітектури та розподілених баз даних.

Задачі дослідження магістерської роботи:

— здійснити аналіз способів побудови систем з мікросервісною архітектурою та розподіленими базами даних;

— запропонувати кращий підхід до організації роботи з даними в мікросервісних системах;

— створити алгоритм та по якому організовуються потоки даних в додатку.

Об’єкт дослідження магістерської роботи — процес обміну та збереження даних в мікросервісних системах.

Предмет дослідження магістерської роботи — методи організації даних для максимально можливої продуктивності в системі з двох баз.

Методи дослідження магістерської роботи: у роботі використано принципи об’єктно-орієнтованого програмування, мікросервісна архітектура та дві бази даних з різними підходами до зберігання інформації для реалізації запропонованого підходу.

Наукова новизна отриманих результатів магістерської роботи полягає у вдосконаленні технології побудови мікросервісних систем, в якій, на відміну від існуючих, реалізовано взаємодію з документальною та реляційною базою даних, що дозволило підвищити продуктивність роботи систем.

Практичне значення одержаних результатів магістерської роботи:

— розроблена архітектура додатку для забезпечення максимальної продуктивності в використанні реляційних та документальних баз;

— створений додаток який реалізує вищеописані принципи.

Апробація. Основні результати роботи повідомлено та затверджено на Всеукраїнській науково-практичній онлайн-конференції «Молодь у науці: дослідження, проблеми, перспективи» (МН-2021) (Вінниця, 05.01.2021 - 14.05.2021)

Публікації. За результатами дослідження опубліковано тези доповіді: Високопродуктивна система управління потоками даних на основі MySQL і MongoDB [Текст] М. О. Шевчук // Молодь в науці: дослідження, проблеми, перспективи (МН-2021) Тез. доп. — Вінниця, 2021. — Режим доступу https://conferences.vntu.edu.ua/index.php/mn/mn2021/paper/view/13144/11053 [2].

Рекомендації по покращенню роботи з даними з використанням MySQL і MongoDB [Текст] М. О. Шевчук // Молодь в науці: дослідження, проблеми, перспективи (МН-2022) Тез. доп — Вінниця, 2021. — Режим доступу https://conferences.vntu.edu.ua/index.php/mn/mn2022/paper/view/13144/11053 [3].

1 АНАЛІЗ СУЧАСНИХ ТЕХНОЛОГІЙ ДЛЯ ПОБУДОВИ МІКРОСЕРВІСНИХ СИСТЕМ З ДВОМА БАЗАМИ ДАНИХ

1.1 Вибір архітектури додатку

1.1.1 Аналіз концепцій контролю сесії

Перед початком написання програми потрібно вирішити яких архітектурний шаблон слід застосувати.

Архітектурний шаблон — це загальноприйняте рішення проблеми архітектури додатків в межах певного контексту. Архітектурні шаблони мають багато спільного з шаблонами проектування, однак вони мають більш широке охоплення.

Для повного розуміння коротко опишемо основні шаблони.

Багаторівневий шаблон використовується для структурування програм, які можна розбити на групи під задач, що находяться на певних рівнях абстракції. Кожний рівень надає служби для наступного, більш високого рівня.

Найчастіше виділяють такі 4 рівня:

— рівень представлення (користувацький інтерфейс);

— рівень додатку (сервісний рівень);

— рівень бізнес-логіки (рівень характерний для предметної області);

— рівень доступу до даних (рівень зберігання даних).

Зазвичай даний шаблон використовується при створенні десктопних додатків та веб-додатків e-commerce.

З плюсів даного шаблону можна виділити стандартизацію рівнів та їх відносну незалежність від змін. Однак бувають ситуації в яких потрібно пропустити деякі рівні, тому він не є універсальним.

Клієнт-серверний шаблон складається з двох частин: сервера та великої кількості клієнтів. Сервер надає послуги клієнтським службам. Вони ж в свою чергу дають запити на послуги сервера. Сервер знаходиться у стані постійного прослуховування клієнтських запитів.

Як правило клієнт-серверну архітектуру використовують онлайн додатки,

так як: електронна пошта, програми спільного доступу до даних, банківські послуги та інші.

Даний шаблон підходить для моделювання набору послуг, які можуть запитувати клієнти. Однак можна виділити те, що запити виконуються на сервері в різних потоках, що може викликати затримки при доступі до спільних ресурсів різними користувачами.

Подібний сервіс, як правило поділяється на серверну та клієнтську частини, де серверна частина містить бізнес логіку та оперує над даними, а клієнтська частина створює представлення для зручного використання сервісу кінцевим користувачем. Такий підхід дає можливість чітко розмежувати відповідальність між компонентами, що робить їх максимально незалежними один від одного. Комунікація між компонентами відбувається за допомогою мережевого з'єднання. Зазвичай веб-сервіси поверх інтернет протоколу HTTP або протоколів вищого рівня[4].

У контексті розробки веб-сервісів існують дві основні концепції: з збереженням стану та без збереження стану. Stateful — підхід зі збереженням стану. Це означає, що він залежить від стану об'єкта з часом і змінює результат залежно від об'єкта та конкретних вхідних даних. Як приклад можна використовувати традиційний FTP-сервер. Усі зміни статусу користувача, наприклад записи Active Directory, зберігаються на сервері як стан клієнта. Кожна зміна, зроблена на сервері, реєструється як зміна стану, і коли користувач відключається, стан користувача змінюється на стан відключення. Такий підхід може викликати деякі проблеми. Перш за все, є багато незавершених сесій і транзакцій. У деяких ситуаціях ви можете не знати, як довго служба буде тримати сеанс відкритим або як перевірити, чи відключений клієнт. Для всіх цих проблем існують обхідні шляхи, але в більшості випадків підтримка стану корисна лише в тому випадку, якщо сама функція залежить від стану. Більшість користувачів можуть взаємодіяти з веб-сервісами різними способами. Тому збереження стану сервера не залежить від клієнтської програми. Це тому, що якщо клієнт не може реалізувати функціональні можливості служби, йому не потрібно зберігати стан.

Stateless — це зовсім інше поняття. Вона ґрунтується на відсутності будь-якого стану. Це дозволяє обробляти запити незалежно від стану системи, виходячи виключно з вхідних даних і набору інструкцій для обробки. Концепція stateless є фундаментальним аспектом сучасного Інтернету. Наприклад, якщо ви хочете переглянути новини, скористайтеся протоколом HTTP для підключення без статусу. Повідомлення аналізуються та обробляються взаємно та окремо від статусу.

На основі цих концепцій було створено дві реалізації, SOAP і REST. SOAP — це стандартизований протокол для надсилання повідомлень за допомогою HTTP, UDP, TCP, SMTP тощо. Специфікація SOAP є офіційним веб-стандартом, який підтримується та розробляється Консорціумом World Wide Web. Перевагами використання SOAP є чіткі правила використання, розширені функції безпеки, такі як принципи ACID та дозволи. Але, з іншого боку, SOAP робить архітектуру дуже складною і немасштабованою. REST створено для вирішення проблем SOAP, відмовившись від суворих специфікацій, дозволяючи розробникам впроваджувати низькорівневі частини або використовувати рішення, створені кимось іншим, що робить сервіс більш гнучким. Можна сказати, що REST спеціально розроблений як stateless. Загальна концепція передачі типового стану спирається на ідею відправки всіх даних для обробки запиту, тому всі необхідні дані вже містяться в самому запиті.

Вибираючи технологію створення веб-сервісу, необхідно визначитися з кількома питаннями: обсягом сервісу та його цільовою аудиторією. Якщо ви хочете використовувати службу як менеджер для зберігання великих обсягів даних від необмеженої кількості користувачів, вам слід використовувати REST. Це збільшує пропускну здатність сервера і може витримувати більші навантаження. Якщо сервіс призначений для використання в корпоративній мережі, включає складну бізнес-логіку або ресурсомісткі обчислення, або якщо вам потрібна можливість використовувати різні протоколи, варто використовувати SOAP. Є. У більшості випадків функціональність SOAP зайва, і вам слід використовувати більш гнучку архітектуру REST[5].

1.1.2 Аналіз структурної архітектури

Розглянемо та порівняємо дві архітектури: монолітну та мікросервісну.

Монолітний додаток — це програма, яка представляється у вигляді одного додатку та містить одну точку входу.

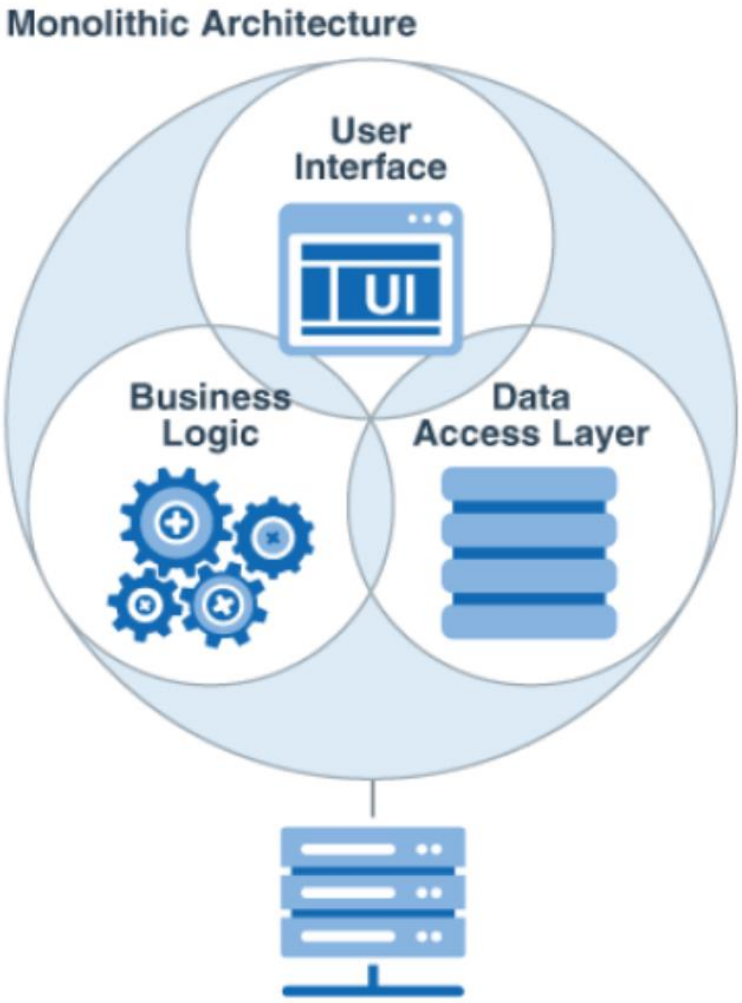

Рисунок 1.1 — Монолітна архітектура.

Велика перевага монолітів в тому, що їх легко реалізувати. Завдяки монолітній архітектурі ви можете відразу почати впроваджувати свою бізнес-логіку, не витрачаючи час на роздуми про міжпроцесний зв’язок.

Також перевагою є наскрізний тестування (E2E). Завдяки монолітній архітектурі такі тести легко створювати.

Коли справа доходить до експлуатації, важливо сказати, що моноліти легко розгортати та масштабувати. Ви можете використовувати сценарій для

завантаження модуля та запуску програми для розгортання. Масштабування досягається шляхом розміщення балансира навантаження перед кількома екземплярами програми.

Але і він не без недоліків. Зазвичай, моноліт переходить з чистого стану в величезний заплутаний клубок. Це виникає через те, що компонент порушує архітектурне правило і з часом збільшується.

Ці зміни уповільнює процес розвитку. Розробка кожної функції в майбутньому буде складною і часозатратною. Компоненти ростуть разом і їх потрібно міняти разом. Створення нової функції означає торкання п’яти різних місць. 5 місць, де потрібно створити тест. 5 місць, де наявні функції можуть мати небажані побічні ефекти.

Вище вказувалося, що масштабування за допомогою моноліту — це легко. Це вірно до тих пір, поки воно не переросте у клубок, як згадувалося вище. Масштабування може бути проблемою, якщо лише частини системи вимагають додаткових ресурсів, оскільки монолітні архітектури не дозволяють масштабувати окремі частини системи.

Моноліт практично не має ізоляції. Проблеми та помилки модуля можуть уповільнити або знищити всю програму.

Іншою архітектурою додатків є мікросервісна архітектура. В ній слабо пов’язані служби взаємодіють для виконання завдань, пов’язаних з бізнес-функціями.

Мікросервіси називаються насамперед через те, що в них менше служб, ніж у монолітних середовищах. Однак, мікро — це бізнес-можливості, а не про розмір.

У порівнянні з монолітами мікросервісів, існує кілька екземплярів.

Мікросервіси легко підтримувати модульними. Технічно це гарантується жорсткими межами між окремими службами.

У великих компаніях різними службами можуть володіти різні команди. Послугу можна повторно використовувати по всій компанії. Також команди можуть працювати над сервісом практично самостійно. Немає необхідності координувати розгортання між командами. Розвивати мікросервіси вигідніше

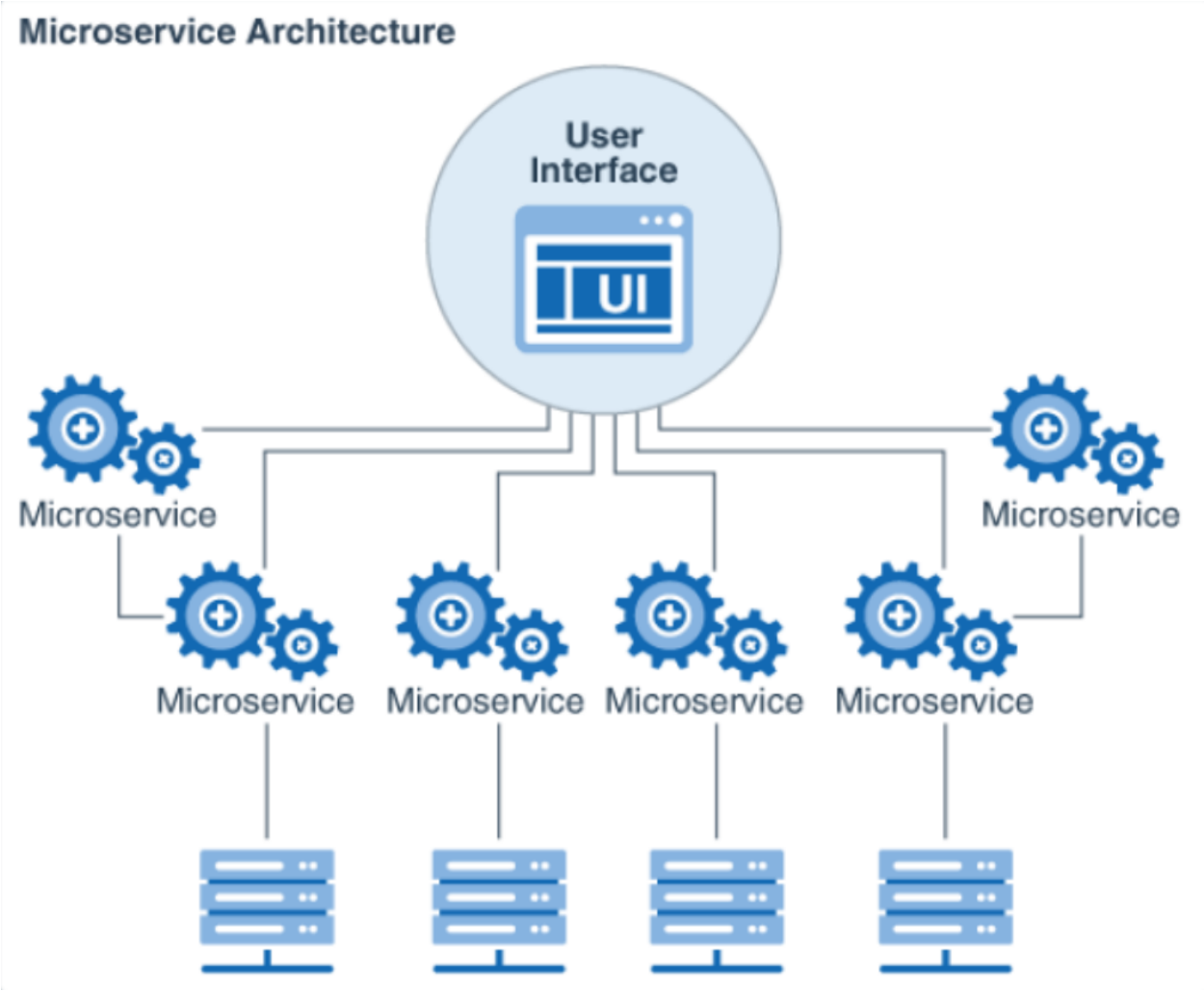

Рисунок 1.2 — Мікросервісна архітектура.

збільшенням кількості команд. Мікросервіси невеликі, їх легко зрозуміти та перевірити.

Зменшення розміру допомагає зменшити час компіляції, час виконання та час, необхідний для виконання тесту. Усі ці фактори впливають на продуктивність розробників, зменшуючи кількість часу, який вам доводиться чекати на кожному етапі розробки.

Можливість скорочувати час завантаження та самостійно розгортати мікросервіси дуже вигідна для CI/CD. Цей процес іде набагато простіше і гладше, ніж звичайний моноліт.

Мікросервіси не прив’язані до технологій, які використовуються іншими сервісами. Це означає, що ви можете використовувати найкращі підходи. Старі служби можна швидко переписати, щоб використовувати новіші технології.

Виходячи з цих положень було обрану мікросервісну архітектуру для підтримання stateless підходу та для забезпечення рівнів абстракції та ізоляції служб які оперують з базами даних для їх ізоляції і поліпшення швидкодії[6].

1.2 Огляд різновидів баз даних

Для початку розглянемо які є типи баз даних.

У централізованій базі даних зберігається у централізованому місці, і користувачі з різних місць мають можливість отримати доступ до цих даних. Цей тип баз даних має засоби, які допомагають користувачам встановлювати зв’язок з базою даних з віддаленого місця.

Для перевірки кінцевих користувачів застосовуються різноманітні види аутентифікації, також номер надається процедурами програми, яка веде облік даних про їх використання.

У розподіленій база даних є інша концепція яка зворотна до концепції централізованої бази даних — це розподілена база даних. Вона має внески як із загальної бази, так і із інформації, захопленої локальними комп'ютерами. Дані є не в одному місці і поширюються на варіативних сайтах організації. Ці сайти поєднані один з одним за допомогою посилань комунікації, що дає їм змогу легко отримувати доступ до розподілених даних.

Розподілену базу даних можна розглядати як базу даних, в якій різні частини бази даних зберігаються в кількох різних місцях, а також прикладні процедури, які реплікуються та розподіляються в різних точках мережі.

Існують два типи розподілених баз даних, однорідні та гетерогенні. База даних, яка має те саме обладнання та керує тими ж операційною системою та програмними процедурами, називається алогенною розподіленою базою даних.

У особистій базі даних дані збираються та зберігаються на персональному комп’ютері, який легко налаштувати. Дані зазвичай використовуються одним і тим же відділом організації і доступні для невеликої кількості людей.

База даних кінцевих користувачів — це база в якій кінцевих користувачів зазвичай не турбують транзакції або транзакції, які відбуваються на різних рівнях,

і відображають лише продукти, які можуть бути програмним забезпеченням або додатками. Таким чином, це спільна база даних, спеціально розроблена для кінцевих користувачів як менеджера іншого рівня. Ця база даних містить загальну інформацію про всю інформацію.

Комерційна база даних — це платна версії величезної бази даних, створеної спеціально для користувачів, які хочуть підтримувати інформацію про підтримку.

NoSQL база даних використовуються для великих наборів розподілених даних. Існують деякі проблеми з продуктивністю під час роботи з великими обсягами даних, які виникають у реляційних базах даних. Бази даних NoSQL можуть легко вирішити ці проблеми. Ви можете дуже ефективно аналізувати неструктуровані великомасштабні дані, які можна зберігати на кількох віртуальних хмарних серверах.

В операційній базі даних інформація, що стосується діяльності компанії, зберігається в цій базі даних. Це бази даних для маркетингу, взаємин із співробітниками, обслуговування клієнтів тощо.

Реляційні бази даних класифікуються за набором таблиць, а дані класифікуються за попередньо визначеними категоріями. Таблиця складається з рядків і стовпців, де стовпці містять записи даних для певної категорії, а рядки містять екземпляри цих даних, визначені категорією. Мова структурованих запитів (SQL) — це стандартний інтерфейс користувача та програми для реляційних баз даних.

Існують різні прості операції, які можна застосувати до таблиць, які полегшують розширення цих баз даних, підключення до загальних відносин між двома базами даних та змінення всіх існуючих програм.

Наразі дані зберігаються в хмарних сервісах, також відомих як віртуальні середовища, або в загальнодоступних або приватних гібридних хмарах. Хмарна база даних — це база даних, оптимізована або створена для такого віртуалізованого середовища. Хмарні бази даних мають багато переваг. Серед них – можливість платити за пропускну здатність і пропускну здатність на основі кожного користувача, що забезпечує масштабованість за вимогою та високу доступність.

Об’єктно-орієнтована база даних — це набір об’єктно-орієнтованого програмування та реляційна база даних. Існують різні елементи, створені за допомогою об’єктно-орієнтованих мов програмування, таких як C++ і Java, які можна зберігати в реляційних базах даних, і для цих елементів підходять об’єктно-орієнтовані бази даних.

Об’єктно-орієнтовані бази даних організовані навколо об’єктів, а не логіки, дій, даних. Наприклад, мультимедійний запис реляційної бази даних може бути об’єктом даних замість буквено-цифрового значення.

Граф — це набір вузлів і ребер, де кожен вузол використовується для представлення сутності, а кожне ребро представляє зв’язок між сутностями. Графічна або графічна база даних — це тип бази даних NoSQL, яка використовує теорію графів для зберігання, відображення та запиту зв’язків.

Графічні бази даних в основному використовуються для аналізу відносин. Наприклад, компанія може використовувати базу даних діаграм для вилучення даних клієнтів із соціальних мереж[7].

Є два типи баз даних, які вирішують цю проблему: реляційні та об’єктно-орієнтовані бази даних.

MySQL є хорошим прикладом реляційної бази даних. Його розробка зосереджена на простоті використання, гнучкості, низькій вартості володіння (порівняно з платною СУБД), а також масштабованості та продуктивності[8].

Об’єктно-орієнтовані бази даних включають MongoDB. Дані зберігаються у вигляді колекції об’єктів у форматі BSON (бінарний JSON), закодованих у двійковому форматі JSON. Це дозволяє швидко працювати з великими даними. Однак у нього є один серйозний недолік. Під час зміни наявних даних спостерігається значна затримка[9].

1.3 Вибір мови програмування

Перш за все потрібно вирішити які мови програмування підходять для реалізації даної задачі.

Для цього розглянемо схему класифікації мов програмування (Рис. 1.3)

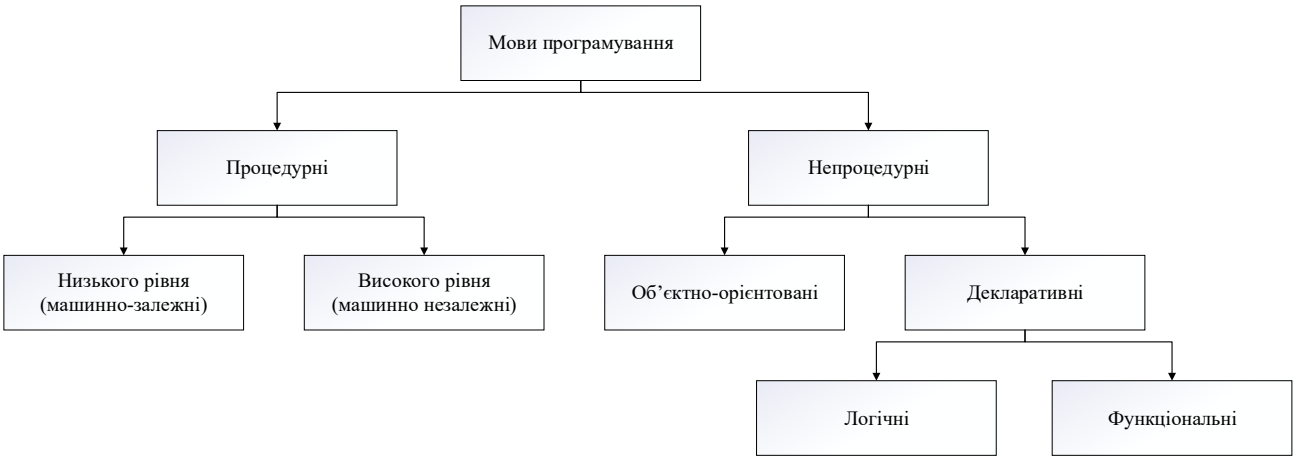

Рисунок 1.3 — Класифікація мов програмування

Згідно до поставленої задачі, а саме для покращення продуктивності роботи системи, логічно було б вибрати мову низького рівня, наприклад C, однак, архітектурні рішення які будуть застосовуватися в подальшому неможливі без використання об’єктно-орієнтованого підходу, тому потрібно вибирати серед мов даного типу. Насамперед через те, що в них є досить легкі методи для підключення різноманітних бібліотек та встановлення фреймворків. Це допоможе пришвидшити процес розробки[10].

1.3.1 Вибір об’єктно-орієнтованої мови програмування

Серед об’єктно-орієнтованих мов програмування найпопулярнішими на даний момент є чотири: Java, C#, Kotlin та C++.

C++ — потужна мова, успадкована від C багаті можливості для роботи з пам'яттю. Тому C++ часто знаходить своє застосування в системному програмуванні, зокрема, при створенні операційних систем, драйверів, різних утиліт, антивірусів і т. д. Але використання цієї мови не обмежується лише системним програмуванням. C++ можна використовувати в програмах будь-якого рівня, де важливі швидкість і продуктивність. Часто використовується для створення графічних додатків, різноманітних прикладних програм. Також він особливо часто використовується для створення ігор з багатою візуалізацією. Крім того, останнім часом набирає обертів мобільний напрямок, де також знайшов своє застосування C++. І навіть у веб-розробці також можна використовувати C++ для

створення веб-додатків або якихось допоміжних сервісів, які обслуговують веб-додатки. Загалом, C++ — це мова, на якій можна створювати практично будь-які програми.

C++ є компільованою мовою, що означає, що компілятор перекладає вихідний код C++ у виконуваний файл, який містить набір машинних інструкцій. Але різні платформи мають свої особливості, тому скомпільовані програми не можна просто переносити з однієї платформи на іншу і запускати там. Однак на рівні вихідного коду програми на C++ в основному є переносними, якщо не використовуються деякі функції, характерні для поточної ОС. А наявність компіляторів, бібліотек і засобів розробки практично для всіх поширених платформ дозволяє компілювати один і той же вихідний код C++ в програми для цих платформ.

На відміну від C, мова C++ дозволяє писати програми в об'єктно-орієнтованому стилі, представляючи програму у вигляді набору взаємодіючих класів і об'єктів. Це дозволяє легко створювати великі програми.

Однак потрібно також враховувати витрати на розробку, в яких C++ є одним із лідерів. Також є проблеми з контролем пам’яті та з кількістю і різноманіттям інструментів розробки. Також можливості для реалізації потрібного архітектурного шаблону залишають бажати кращого[11].

Kotlin з'явився, коли розробці Android знадобилася більш сучасна мова. JetBrains, який створив Intellij, інтегроване середовище розробки, також створив Kotlin. Це статична мова з відкритим вихідним кодом, заснована на віртуальній машині Java (JVM). Перевага Kotlin полягає в тому, що ви можете компілювати його в JavaScript і взаємодіяти з Java. Це не тільки дозволяє розробникам легко оновлювати свої старі програми Java до Kotlin, але також дозволяє їм переносити свою стару роботу на Java на Kotlin.

Плюси Kotlin:

— на Kotlin, безумовно, швидше пишеться і тому він дуже подобається розробниками, якщо в Java потрібні 40 рядків коду, в Kotlin потрібно всього 1-3 рядки;

— Kotlin допомагає в створенні нових чистих API;

— завдяки байт-коду Java можна використовувати бібліотеки та фреймворки Java в Kotlin, що робить перехід з Java на Kotlin плавним;

— система типів Kotlin має дуже необхідний null-safety, якого в Java дуже бракує, Kotlin може скористатися цією проблемою і використовувати null.

Мінуси Kotlin:

— Kotlin, безумовно, має круту криву навчання. Його дуже стислий синтаксис є великою перевагою, але він вимагає попереднього дослідження;

— Kotlin у більшості випадків компілює повільніше, ніж Java, але може навіть перевищувати Java;

— спільнота Kotlin ще молода, а навчальні ресурси обмежені, тому знайти відповідь на ваше запитання може бути трохи складно, однак у міру зростання його популярності ресурси та спільноти з часом будуть зростати.

— Kotlin все ще новачок, тому знайти досвідченого розробника, який може стати наставником для вашої команди, може бути трохи складно;

— деякі функції Android Studio, такі як автозаповнення та компіляція, працюють повільніше в Kotlin, ніж у Java[12].

Не зважаючи на те, що Kotlin більш нова та зручна мова, її повільність незначні архітектурні та платформні відмінності від Java, а також пріоритет розробки в сторону мобільних платформ не роблять її фаворитом.

C# належить до сімейства мов, які мають C-подібний синтаксис, синтаксис який найбільш близький до C++ і Java. Мова має потужну статичну типізацію та підтримує поліморфізм, перевантаження операторів, покажчики на функції-члени класу, атрибути, події, властивості, виключення та коментарі у стилі XML. Скориставшись багатьма від своїх попередників (C ++, Java, Delphi, Modul, Smalltalk), C# виявився проблематичним у розробці програмних систем на основі їхнього використання. Виключає модель. C# не підтримує множинну спадковість класів (на відміну від C++).

Коли люди кажуть C#, вони часто мають на увазі технологію платформи .NET (Windows Forms, WPF, ASP.NET, Xamarin). І навпаки, коли люди кажуть

.NET, вони часто мають на увазі C#. Однак, незважаючи на те, що ці поняття пов’язані, ідентифікувати їх помилково. Мова C# була спеціально створена для роботи з .NET Framework, але сама концепція .NET досить широка[13].

Однак навіть з виходом .NET Core 3 сама платформа все ще залишається орієнтованою на Windows. Тому незважаючи на швидкодію платформна орієнтованість а також трішки інші архітектурні підходи не дають нам зробити вибір на користь цієї мови.

Java — це об'єктно-орієнтована мова програмування, розроблена Sun Microsystems, який зараз належить Oracle.

Плюси Java:

— легко вчитися і розуміти;

— добре працює як для нативних, так і для кроссплатформенних додатків;

— Java також забезпечує більш швидкий процес виконання, дозволяючи писати більше коду за менший час;

— має величезну кількість фреймворків та бібліотек;

— має зручні механізми зборки такі як Maven та Gradle.

Мінуси Java.

— Java є мовою з великою кількістю типів, що означає, що необхідно писати набагато більше коду, збільшуючи ймовірність помилок;

— Java вимагає більше оперативної пам’яті в порівнянні з іншими мовами[14].

Незважаючи на мінуси, Java надає надзвичайно звучний механізм для роботи даними на сервері а також сучасний Spring Framework.

Тому в даній роботі я буду використовувати дану мову.

1.3.2 Стек технологій

В реаліях написання серверних систем на мові Java фігурує два основних фреймворка: Java EE та Spring Framework.

Java EE є офіційним стандартом для повнофункціонального стека корпоративних програм. Включає об'єктно-реляційне відображення, безпеку, веб-

додатки, підключення до бази даних, транзакції тощо...

На додаток до специфікації Java EE існують сервери реалізації/додатків Java EE, такі як JBoss, Glassfish, WebSphere і Weblogic.

Переваги Java EE:

— вважається, що програми, написані на Java EE, є більш надійними, безпечними та масштабованими;

— якщо ви використовуєте J2EE, вивчити JEE буде легше;

— він може працювати на кількох платформах;

— підтримує міжплатформну переносимість;

— JEE дуже добре обробляє складні, інтенсивні транзакції;

— Java EE має солідний досвід у тисячах успішних проектів, створених розробниками Java по всьому світу.

Недоліки Java EE:

— середовище розробки додатків дещо складне і важко зрозуміти новачкам;

— кінцева вартість проекту, включаючи розробку, розгортання та розробку додатків, може бути високою[15].

Spring, з іншого боку, є фреймворком, який багато робить відповідно до специфікації Java EE, але у своїй власній формі. Для цього він не відповідає специфікації Java EE та API. Однак він включає веб-фреймворки, управління транзакціями, безпеку та деякі інші рішення, надані Java EE.

Переваги Spring Framework:

— він дозволяє використовувати старі об’єкти Java, розробникам не потрібен корпоративний контейнер, як сервер додатків;

— він забезпечує розробникам Java чудовий рівень модульності;

— він забезпечує добре розвинутий веб-фреймворк;

— його можна використовувати для ефективної організації об’єктів середнього рівня;

— код програми Spring, як правило, дуже легко тестується.

Недоліки Spring Framework:

— створення програм з використанням Spring є відносно складним через

відсутність чіткого фокусу;

— вивчення фреймворку Spring може бути складним для початківців розробників Java;

— у документації Spring немає чітких рекомендацій щодо деяких тем;

— початкове налаштування займає багато часу і зусиль.

Незважаючи на все Spring має низку значних переваг. Основна — це Spring Boot.

Творці Spring вирішили надати розробникам кілька утиліт, які автоматизують процес налаштування та прискорюють процес створення та розгортання додатків Spring. Разом вони називаються Spring Boot.

Spring Boot — це корисний проект, спрямований на спрощення створення додатків на основі Spring. Це дозволяє створити найпростіший спосіб створення веб-додатків і вимагає від розробників мінімальних зусиль для налаштування та написання коду.

Spring Boot має чудові можливості, але найважливішими з них є керування залежностями, автоматична конфігурація та вбудований контейнер сервлетів.

Щоб прискорити процес керування залежностями, Spring Boot неявно упаковує необхідні сторонні залежності для кожного типу додатків на основі Spring, так званий стартовий пакет (spring-boot-starter-web). Надається розробникам через spring- boot-starter-data-jpa тощо.

Starter пакети — це зручний набір дескрипторів залежностей, які ви можете включити у свою програму. Це надає універсальне рішення для всіх технологій, пов’язаних із Spring, усуваючи потребу програмістам невиправдано шукати приклади коду та завантажувати з них необхідні дескриптори залежностей.

Наприклад, якщо ви хочете отримати доступ до своєї бази даних за допомогою Spring Data JPA, все, що вам потрібно зробити, це включити залежність spring-boot-starter-data-jpa у свій проект (не потрібно шукати сумісну базу даних Hibernate і драйвер).

Створюючи веб-програму Spring, просто додайте залежність spring-boot-starter-web і всі бібліотеки, необхідні для розробки програми Spring MVC, такі як

spring-webmvc, jackson-json, validation-api та Tomcat будуть включені у ваш проект.

Тобто Spring Boot збирає всі загальні залежності та визначає їх в одному місці. Це дозволяє розробникам легко використовувати їх замість того, щоб винаходити колеса щоразу, коли вони створюють нову програму.

Тому під час використання Spring Boot у файлі pom.xml набагато менше рядків, ніж при використанні його у Spring-програмах.

Другою чудовою особливістю Spring Boot є автоматичне налаштування програм.

Після вибору відповідного стартового пакета Spring Boot автоматично спробує налаштувати вашу програму Spring на основі доданих вами jar-залежностей.

Наприклад, якщо ви додасте spring-boot-starter-web, Spring Boot автоматично налаштує зареєстровані bean-компоненти, такі як DispatcherServlet, ResourceHandlers і MessageSource.

При використанні spring-boot-starter-jdbc Spring Boot автоматично створює контейнери для DataSource, EntityManagerFactory і TransactionManager і зчитує інформацію з файлу application.properties для підключення до бази даних.

Автоматичну конфігурацію можна повністю скасувати в будь-який момент за допомогою спеціальних налаштувань.

Кожна веб-програма Spring Boot містить вбудований веб-сервер. Перегляньте список підтримуваних контейнерів сервлетів "з коробки".

Якщо вам потрібно використовувати інший HTTP-сервер, достатньо виключити залежності за замовчуванням. Spring Boot надає окремі стартові пакети для різних HTTP-серверів[16].

Були розглянуті архітектурні програмні рішення та обрана мікросервісна архітектура для написання додатку. Були проаналізовані різноманітні бази даних та обрані реляційна MySQL та документальна MongoDB. Були проаналізовані мови програмування та обрана Java як найбільш коректна для поставленої задачі. Був розглянутий стек технологій Spring Framework 5 та Spring Boot.

2 ДОСЛІДЖЕННЯ ШВИДКОДІЇ БАЗ ДАНИХ

2.1 Обґрунтування переваг поєднання баз даних.

Часто конкретні бази даних застосовують для цілей в яких вони показують низьку продуктивність зумовлюючи це тим, що з одною базою даних простіше працювати. І це дійсно так, більшість мов програмування надають інструменти для роботи лише з одною базою даних в одному додатку, і конфігурація більше ніж одної бази викликає складнощі. Однак при використанні мікросервісної архітектури ця проблема зникає.

При такому підході можна розділити потоки даних для роботи з двома базами даних SQL та NoSQL. Кожна з них має свої переваги та недоліки які потрібно врахувати при проектуванні системи.

Всім відомі реляційні бази даних мають ряд переваг. Вони прості в розумінні, оскільки складаються з єдиної інформаційної конструкції таблиці. При проектуванні таких баз даних використовуються правила які базуються на математичному апараті, наприклад різноманітні нормальні форми. Реляційна модель забезпечує незалежність даних, при зміні структури бази зміни, які потрібно провести в прикладній програмі мінімальні. Реляційні бази даних досить швидко обробляють запити на оновлення даних.

Однак реляційні бази даних мають також і недоліки. При запиті даних з двох зв’язаних таблиць приходить операція з’єднання яка є досить повільною. Для організації зв’язків один-до-одного, багато-до-одного та одного-до-багатьох необхідні додаткові поля-ключі, а у випадку зв’язку багато-до-багатьох і ціла таблиця. Всі ці додаткові поля збільшують розмір бази даних на накопичувачі.

З приходом NoSQL баз даних, таких як MongoDB розпочав поширюватись документо-орієнтований підхід до проектування баз даних. Такі бази даних оперують поняттями колекцій і документів. Форма і поля документа не обмежується заданою структурою колекції, як це є в таблицях і їх полях. В MongoDB документи зберігаються в форматі BSON (Binary JSON), що є надзвичайно зручним при групуванні даних в складні структури. При такому

підході в базі даних відсутні операції з’єднання, що пришвидшує запити на отримання та збереження складних структур даних. Відсутність додаткових полів та таблиць зменшує розміри такої бази даних, за умови, що в ній дані мінімально дублюються.

До недоліків документо-орієнтованих баз даних можна віднести низьку швидкодію при оновленні полів в документах, а також те, що кожен документ містить в собі назву поля, що дещо збільшує розміри даних. Також відсутня пряма підтримка складних транзакцій.

Тому в складних системах, де дані можуть приймати різні форми в залежності від переваг різних підходів до організації баз даних слід розділити їх потоки в різні бази. При використанні мікросервісної архітектури це буде досить просто зробити, для цього потрібно створити сервіс з реляційною базою даних, сервіс з документо-орієнтованою базою даних та сервіс який буде регулювати запити до цих баз.

Основний принцип полягатиме в тому, щоб зберігати складні комплексні об’єкти та дані які часто записуються і не редагуються в MongoDB, а прості об’єкти які часто обновлюються в SQL базі. Також дані з полями які часто дублюються доцільніше буде зберігати в SQL базі для економії простору на накопичувачі[2].

2.2 Тестування баз даних

2.2.1 Технології для тестування

Для тестування продуктивності баз даних потрібно мати технологію для з’єднання з ними. Все це забезпечує Spring Boot. До його складу входять різноманітні залежності, в тому числі драйвери для баз даних, тестовий фреймворк JUnit5 та Testcontainers[17].

Анотація @SpringBootTest дозволяє підняти повний контекст додатку для рестування на реальних даних в реальній базі даних, але даний метод є недосконалим по ряду причин. Основна із них — це постійна необхідність видаляти базу даних і заново її створювати. Якщо цього не робити то можуть виникнути конфлікти даних і тестування буде проведене некоректно. Іншою проблемою є

невідома продуктивність роботи комп’ютера. Його можуть навантажувати різноманітні фонові процеси та інші відкриті програми, тому результати можуть відрізнятися.

Для вирішення цих проблем було вирішено використовувати технологію Testcontainers для SpringBootTest.

Testcontainers — це бібліотека для Java та JUnit5. Вона використовує Docker-образи для автоматичного створення тестових контейнерів з необхідними сервісами. В нашому випадку будуть створюватися контейнери баз даних для MySQL та MongoDB[18].

Для використання даної технології потрібно створити клас з конфігураціями.

Лістинг 2.1 — Клас конфігурації Testcontainers для MongoDB

```
@Testcontainers
public abstract class DatabaseTest {
   private static final String MONGO_IMAGE = "mongo:5.0.5";
   private static final int MONGO_PORT = 27017;
   public static final GenericContainer<?> MONGO = new GenericContainer<>(MONGO_IMAGE).withExposedPorts(MONGO_PORT);
   static {
      MONGO.start();
   }
   @DynamicPropertySource
   public static void databaseProperties(DynamicPropertyRegistry registry) {
      registry.add("spring.data.mongodb.host", MONGO::getContainerIpAddress);
      registry.add("spring.data.mongodb.port", () -> MONGO.getMappedPort(MONGO_PORT).toString());
   }
}
```

Лістинг 2.2 — Клас конфігурації Testcontainers для MySQL.

```
@Testcontainers
public abstract class DatabaseTest {
   private static final String MYSQL_IMAGE = "mysql:8.0.27";
   private static final int MYSQL_PORT = 3306;
   public static final MySQLContainer<?> MYSQL = new MySQLContainer<>(MYSQL_IMAGE)
         .withInitScript("startup.sql")
         .withExposedPorts(MYSQL_PORT)
         .withDatabaseName("master")
         .withUsername("root")
         .withPassword("11235813");
   static {
      MYSQL.start();
   }
   @DynamicPropertySource
   public static void databaseProperties(DynamicPropertyRegistry registry) {
      registry.add("spring.datasource.url", () -> MYSQL.getJdbcUrl() + "?cachePrepStmts=true&useServerPrepStmts=true&rewriteBatchedStatements=true");
   }
}
```

Після створення дані класи потрібно пронаслідувати класом в якому пишуться тести, після чого Testcontainers буде підміняти базу даних тою, що піднімається в контейнері. Оскільки на контейнері в Docker виділяється завжди однакові, завчасно зарезервовані ресурси, то проблем описаних вище бути не повинно, також при закінченні роботи тестів він автоматично вимикається і видаляється разом з базою даних і даними які є в ньому, що дозволяє уникнути низки проблем з видаленням даних та використанням лишнього місця на цифровому накопичувачі.

2.2.2 Порівняння продуктивності баз даних

Оскільки в реаліях написання програм використовуються різні ORM для MongoDB — це Spring Data MongoDB, а для MySQL — це Spring Data JPA разом з Hibernate.

Для цього проведемо низку тестів з використанням технологій описаних в підрозділі 2.2.1. Для кожного з випадків проводиться 5 тестів та заміряється час в мілісекундах. Після цього виводиться середнє значення по цим тестам, а також записується відношення між ними.

Для перших чотирьох тестів були створені базові класи Main та Included (Рис. 2.1).

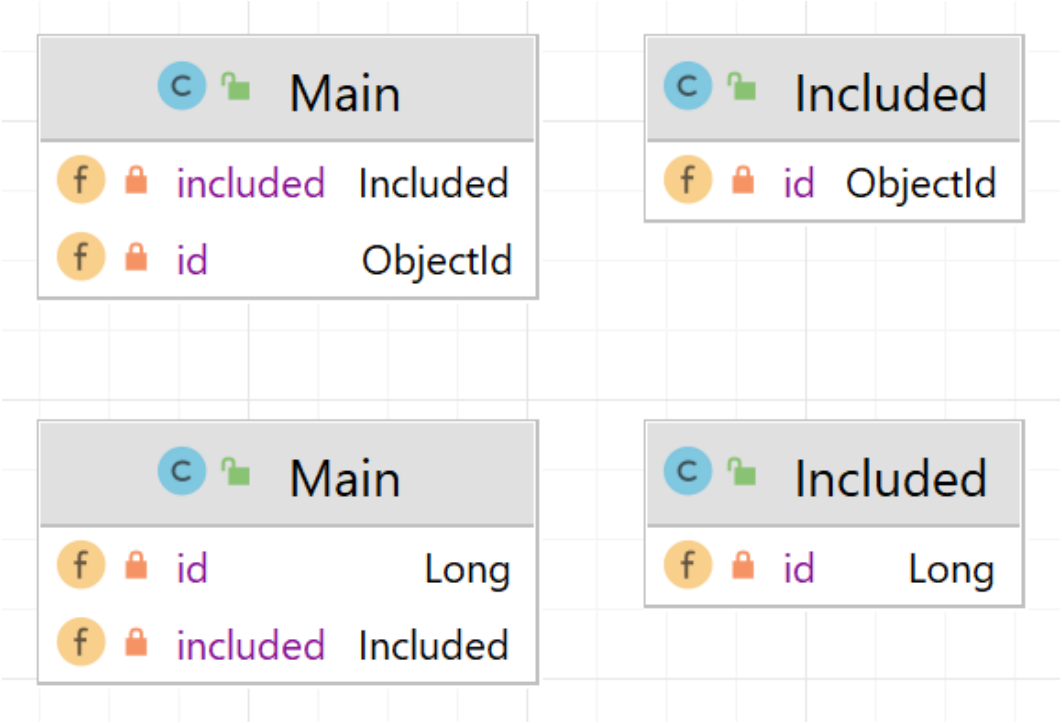

Рисунок 2.1 — UML-діаграма класів для тестування баз даних

Таблиця 2.1 — Збереження даних зі зв’язками

База даних	Тест 1	Тест 2	Тест 3	Тест 4	Тест 5	Середнє
MongoDB	4573	4163	3756	3578	3548	3923,6
MySQL	2777	2681	2785	2354	2260	2571,4

3923,6 / 2571,4 = 1,5259

Лістинг 2.1 — Тест а збереження даних.

```
@Test
   public void saveData() {
      List<Main> mains = new ArrayList<>();
      List<Included> inclusions = new ArrayList<>();
      for (int i = 0; i < 1000; i++) {
         Main main = new Main();
         Included included = new Included();
         included.setId(new ObjectId());
         inclusions.add(included);
         main.setIncluded(included);
         mains.add(main);
      }
      long pre = System.currentTimeMillis();
      includedRepository.saveAll(inclusions);
      mainRepository.saveAll(mains);
      long res = System.currentTimeMillis() - pre;
      log.info("saveData\n{}", res);
   }
```

З даного тесту можна зробити висновки, що методи зберігання даних в реляційній базі даних не ефективні в документальних базах, тому при необхідності прийдеться використовувати MySQL коли знадобляться зв’язки такого типу.

Таблиця 2.2 — Вичитка даних за зв’язками

База даних	Тест 1	Тест 2	Тест 3	Тест 4	Тест 5	Середнє
MongoDB	2306	2371	2372	2351	2226	2325,2
MySQL	1231	1235	1201	1297	1261	1245

2325,2 / 1245 = 1,8676

Лістинг 2.2 — Тест на вичитку даних через посилання.

```
@Test
   public void getData() {
      List<Main> mains = new ArrayList<>();
      List<Included> inclusions = new ArrayList<>();

      for (int i = 0; i < 10000; i++) {
         Main main = new Main();
         Included included = new Included();
         included.setId(new ObjectId());
         inclusions.add(included);
         main.setIncluded(included);
         mains.add(main);
      }
      includedRepository.saveAll(inclusions);
      mainRepository.saveAll(mains);
      for (int i = 0; i < 6; i++) {
         long pre = System.currentTimeMillis();
         List<Main> getedMains = mainRepository.findAll();
         long res = System.currentTimeMillis() - pre;
         log.info("getData: {}", res);
      }
   }
```

Цей тест дає розуміння того, що аналог JOIN в MongoDB працює повільно.

Далі проведемо аналогічні тести, але замінимо зв’язки за ключами на вкладені об’єкти для MongoDB та Embedded для MySQL.

Embedded клас — це такий клас, поля якого розкладаються на поля в таблиці батьківського класу. Він не зв’язаний через первинний ключ. Тому він виглядає як вкладений об’єкт.

Таблиця 2.3 — Збереження даних з внутрішнім об’єктом

База даних	Тест 1	Тест 2	Тест 3	Тест 4	Тест 5	Середнє
MongoDB	1823	1441	1488	1445	1454	1530,2
MySQL	396	266	282	210	176	266

1530,2 / 266 = 5,7526

З даного тесту видно, що SQL швидше зберігає в одну таблицю чим MongoDB це зв’язано з тим, що створення Id та індексів відбувається швидше за рахунок внутрішніх систем бази даних.

Таблиця 2.4 — Вичитка даних з внутрішнім об’єктом

База даних	Тест 1	Тест 2	Тест 3	Тест 4	Тест 5	Середнє
MongoDB	21	20	20	17	19	19,4
MySQL	13	12	13	13	14	13

19,4 / 13 = 1,4923

З даного тесту видно, що SQL швидше зберігає в одну таблицю чим MongoDB. Але потрібно зауважити, що таке використання таблиць порушує принципи організації даних в реляційних базах даних, тому це є поганою практикою і краще так не робити.

Для подальшого тестування потрібно створити іншу структуру даних. Клас Container містить список із 10 об’єктів класу FirstIn який в свою чергу містить список із 10 об’єктів класу SecondIn та одного об’єкту класу ThirdIn. Вся ця структура зображена на рисунку 2.2.

В цілому створюється 1000 Container, 10000 FirstIn, 100000 SecondIn та 10000 ThirdIn, що є досить великою кількістю даних і несе вагоме навантаження на базу даних. Вони викликаються за типом поєднання EAGER. Це означає, що одночасно буде завантажуватися вся ієрархія об’єктів за запитом JOIN. Це операція аналогічна простому запиту на складний об’єкт в MongoDB, але займає значно більше ресурсів часу поки шукає по таблицях дані.

Рисунок 2.2 — UML-діаграма класів для тестування баз даних

Таблиця 2.5 — Збереження об’єктів з багатьма вкладеними об’єктами

База даних	Тест 1	Тест 2	Тест 3	Тест 4	Тест 5	Середнє
MongoDB	1790	1757	1763	1719	1719	1749,6
MySQL	52454	496227	47721	47991	48333	49224

49224 / 1749,6 = 28,135

Лістинг 2.3 — Тест на створення даних з багатьма вкладеними об’єктами.

```
@Test
    public void saveContainerData() {
        List<Container> containers = new ArrayList<>();
        for (int i = 0; i < 10; i++) {
            Container container = new Container();
            container.setData("data");
            List<FirstIn> firstIns = new ArrayList<>();
            for (int j = 0; j < 10; j++) {
```

```
            FirstIn firstIn = new FirstIn();
            firstIn.setData("data");
            List<SecondIn> secondIns = new ArrayList<>();
            for (int k = 0; k < 10; k++) {
                SecondIn secondIn = new SecondIn();
                secondIn.setData("data");
                secondIns.add(secondIn);
            }
            firstIn.setSecondIns(secondIns);
            ThirdIn thirdIn = new ThirdIn();
            thirdIn.setData("data");
            firstIn.setThirdIn(thirdIn);
            firstIns.add(firstIn);
        }
        container.setFirstIns(firstIns);
        containers.add(container);
    }

    for (int i = 0; i < 6; i++) {
        long pre = System.currentTimeMillis();
        containerRepository.saveAll(containers);
        long res = System.currentTimeMillis() - pre;
        log.info("saveContainerData: {}", res);
    }
}
```

В результаті виконання даного тесту було виявлено колосальну різницю в швидкості збереження об’єктів даного типу. MongoDB та всі інші документальні бази розраховані на такого роду операції.

Таблиця 2.6 — Збереження об’єктів з багатьма вкладеними об’єктами

База даних	Тест 1	Тест 2	Тест 3	Тест 4	Тест 5	Середнє
MongoDB	228	171	184	134	139	171,2
MySQL	3144	3177	3065	3144	3135	3133

3133/ 171,2 = 18,3

Лістинг 2.4 — Тест на вичитку даних з багатьма вкладеними об’єктами.

```
@Test
   public void getContainerData() {
      List<Container> containers = new ArrayList<>();
      for (int i = 0; i < 1000; i++) {
         Container container = new Container();
         container.setData("data");
         List<FirstIn> firstIns = new ArrayList<>();
         for (int j = 0; j < 10; j++) {
            FirstIn firstIn = new FirstIn();
            firstIn.setData("data");
            List<SecondIn> secondIns = new ArrayList<>();
            for (int k = 0; k < 10; k++) {
               SecondIn secondIn = new SecondIn();
               secondIn.setData("data");
               secondIns.add(secondIn);
            }
            firstIn.setSecondIns(secondIns);
            ThirdIn thirdIn = new ThirdIn();
            thirdIn.setData("data");
            firstIn.setThirdIn(thirdIn);
            firstIns.add(firstIn);
         }
         container.setFirstIns(firstIns);
```

```
            containers.add(container);
        }
        containerRepository.saveAll(containers);
        for (int i = 0; i < 6; i++) {
            long pre = System.currentTimeMillis();
            containerRepository.findAll();
            long res = System.currentTimeMillis() - pre;
            log.info("getContainerData: {}", res);
        }
    }
```

Як видно з результату, дані зчитуються швидше в MongoDB, це зумовлено тим, що не потрібно робили величезну кількість операцій JOIN.

Для останніх тестів створимо простий клас ToUpdate з трьома полями: id, data, num (Рис 2.3). Тести будуть проводитися на 1000 та на одному об’єкті.

Таблиця 2.7 — Редагування тисячі об’єктів

База даних	Тест 1	Тест 2	Тест 3	Тест 4	Тест 5	Середнє
MongoDB	1507	1604	1550	1501	1512	1534,8
MySQL	6066	6148	6064	5952	6072	6060,4

6060,4/ 1534,8 = 3,9487

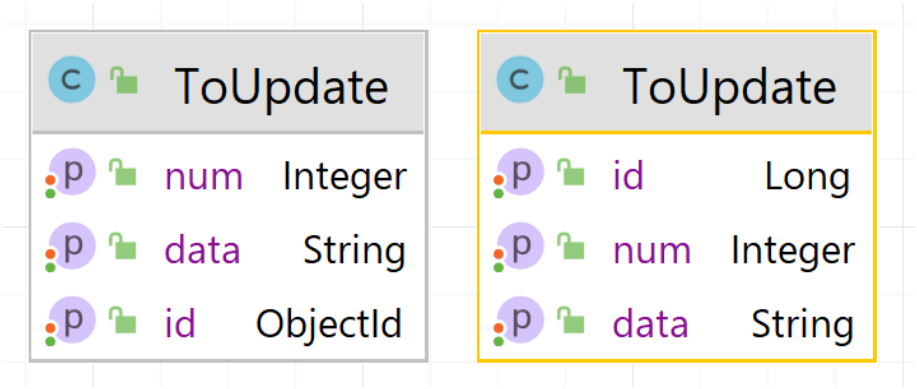

Рисунок 2.3 — UML-діаграма класів для тестування баз даних.

Лістинг 2.5 — Тест на редагування тисячі об’єктів.

```
@Test
    public void update() {
        List<ToUpdate> toUpdates = new ArrayList<>();
        for (int i = 0; i < 1000; i++) {
            ToUpdate toUpdate = new ToUpdate();
            toUpdate.setData("d" + i);
            toUpdate.setNum(i);
            toUpdates.add(toUpdate);
        }

        toUpdates = toUpdateRepository.saveAll(toUpdates);
        for (int i = 0; i < 6; i++) {
            for (ToUpdate toUpdate : toUpdates) {
                toUpdate.setData(toUpdate.getData() + i);
                toUpdate.setNum(toUpdate.getNum() + toUpdate.getNum());
            }

            long pre = System.currentTimeMillis();
            toUpdateRepository.saveAll(toUpdates);
            long res = System.currentTimeMillis() - pre;
            log.info("getContainerData: {}", res);
        }
    }
```

Таблиця 2.8 — Редагування одного об’єкта

База даних	Тест 1	Тест 2	Тест 3	Тест 4	Тест 5	Середнє
MongoDB	3	3	2	3	2	2.6
MySQL	18	17	20	17	16	17.6

17.6 / 2.6 = 6,769

Лістинг 2.6 — Тест на редагування одного об’єкта.

```
@Test
   public void updateOne() {
      ToUpdate toUpdate = new ToUpdate();
      toUpdate.setData("data");
      toUpdate.setNum(1);
      toUpdate = toUpdateRepository.save(toUpdate);
      for (int i = 0; i < 6; i++) {
         toUpdate.setData("new data" + i);
         toUpdate.setNum(2 + i);
         long pre = System.currentTimeMillis();
         toUpdateRepository.save(toUpdate);
         long res = System.currentTimeMillis() - pre;
         log.info("getContainerData: {}", res);
      }
   }
```

Можна зробити висновок, що MongoDB редагує дані швидше. Це дійсно таке, але тільки тоді коли вони це невеликі об’єкти. Якщо скористатися дослідженнями які зроблені вище, то можна зробити висновки які кажуть про те, що якщо об’єкт буде мати велику вкладеність, і сам буде мати великий розмір, то редагування вкладених класів може відбуватися швидше якщо вони зв’язані через відношення. Це є перевагою реляційної бази даних в даному випадку і потрібно це розуміти і використовувати.

В цьому розділі було наведено переваги підходу з використанням двох різних баз даних. Проведений огляд технологій для тестування продуктивності без даних. Розглянуті Docker та Testcontainers. Проведено тестування продуктивності баз даних на восьми сценаріях. Зібрані статистичні дані про продуктивність. Проведений аналіз даних та побудовані характерні графіки.

3 РОЗРОБКА ВИСОКОПРОДУКТИВНОЇ СИСТЕМИ

3.1 Архітектура та рекомендації по розподіленню даних

Для розробки програми було вирішено обрати мікросервісну архітектуру. Це дозволяє нам створити розподілену ізольовану систему в якій програмні додатки які відповідають за бази даних є окремими сервісами і не залежать від стану інших систем. Це дозволяє підвищити надійність а також дозволяє реалізувати нашу задумку з двома базами даних, оскільки в Spring Boot в одному додатку виникають конфлікти якщо є більше одної бази даних.

Для прикладу була розроблена архітектура зображена на рисунку 3.1.

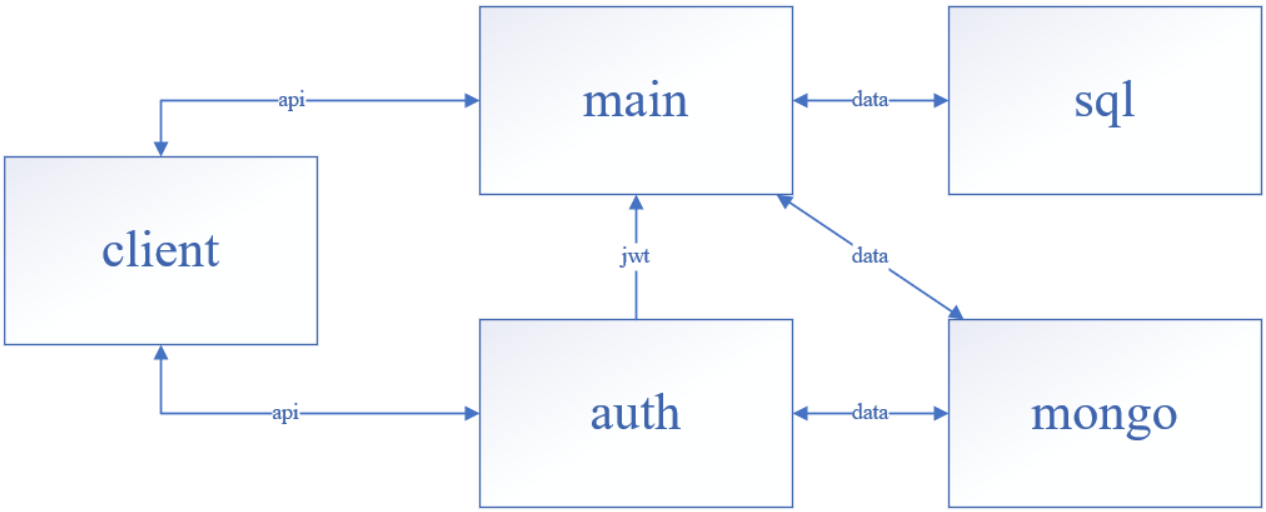

Рисунок 3.1 — Мікросервісна архітектура з кількома базами даних.

Дані між мікросервісами передаються по протоколу HTTP в форматі JSON. Це є невеликим мінусом в продуктивності, однак це дозволяє зробити реалізацію з двома базами даних, а також ORM для MongoDB швидко серіалізує та десеріалізує JSON в BJSON, оскільки ці формати досить схожі. Також важливо розуміти, що всі запити до бази даних проводяться да допомогою ORM і в реаліях їх реалізації, тому деякі з них можуть виконувати запити повільніше чим чисті запити до бази даних. Також є проблема серіалізації та десеріалізації даних.

Основною ідеєю є розподіл даних та операцій над ними по різним базам

даних, тому на базі досліджень описаних в розділі 2 я приведу список рекомендацій по яким це потрібно робити.

Використовувати MySQL потрібно коли:

— об’єкти добре зв’язуються за відношеннями, причому звертання до цих об’єктів відбувається відносно нечасто, тоді можна використати lazy ініціалізацію і не робити постійно операцію JOIN, це дозволить скоротити час мапінгу даних в об’єкт, а також кількість часу яку витрачає база даних для запиту;

— при необхідності часто додавати елементи відношення багато-до-багатьох та багато-до-одного, в цьому випадку потрібно зробити INSERT в таблицю, без необхідності шукати та вивантажувати весь об’єкт з бази даних так як це потрібно робити в MongoDB;

— при необхідності редагувати елемент вкладеної таблиці, зв’язаної відношенням;

— при необхідності робити різноманітні складні запити по кільком таблицям, MySQL це робить краще;

— при запитах на один або кілька невеликих об’єктів без відношень і складних структур.

Використовувати MongoDB потрібно коли:

— коли потрібно часто звертатися до складних структур даних в яких завжди потрібно отримувати всі дані. Аналог eager JOIN;

— потрібно зберігати складні незмінні структури даних. Тобто при реалізації архіву;

— при необхідності кардинально редагувати весь документ з всіма його вкладеними об’єктами;

— при необхідності зберігати списки та мапи.

На основі цих ідей буде базуватися тестовий додаток наведений нижче. В ньому буде два сервіси з базами даних MySQL та MongoDB та один основний — main. Все це буде реалізовано за принципами мікросервісної архітектури[3].

3.2 Реалізація високопродуктивної системи управління потоками даних на основі MySQL і MongoDB

Для написання системи використовується maven як інструмент контроля залежностей та організації проектів. Створюється основний проект master і в ньому створюється чотири модуля: main, mongo, sql та entity (Рис. 3.2).

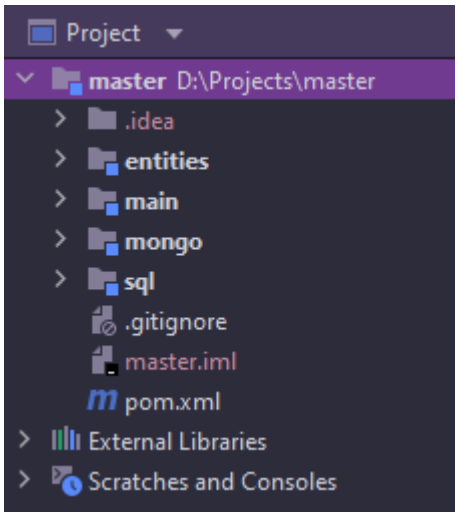

Рисунок 3.2 — Структура проекту.

Модуль entities містить в собі класи сутностей для баз даних. Даний модуль необхідний для того, щоб можна було працювати з цими сутностями в різних сервісах без необхідності їх копіювання.

Сутності для MongoDB позначаються анотацією @Document, а для MySQL — @Entity.

Було створено декілька класів сутностей для демонстрації основного принципу системи. Їх діаграми зображені на рисунках 3.3 та 3.4.

Для MySQL було створено чотири класи: Country, Department, Region та City. Кожен з них зв’язаних з наступним відношенням один-до-багатьох і з всіма попередніми відношенням багато-до-одного. Це дозволяє отримати повну структуру даних по бажанню. Всі ці відношення мають властивість LAZY що означає що вони не будуть під’єднуватись до тих пір, поки не викликається гетер на це поле. Це дозволяє пришвидшити процеси роботи з різними варіантами запитів до цих даних.

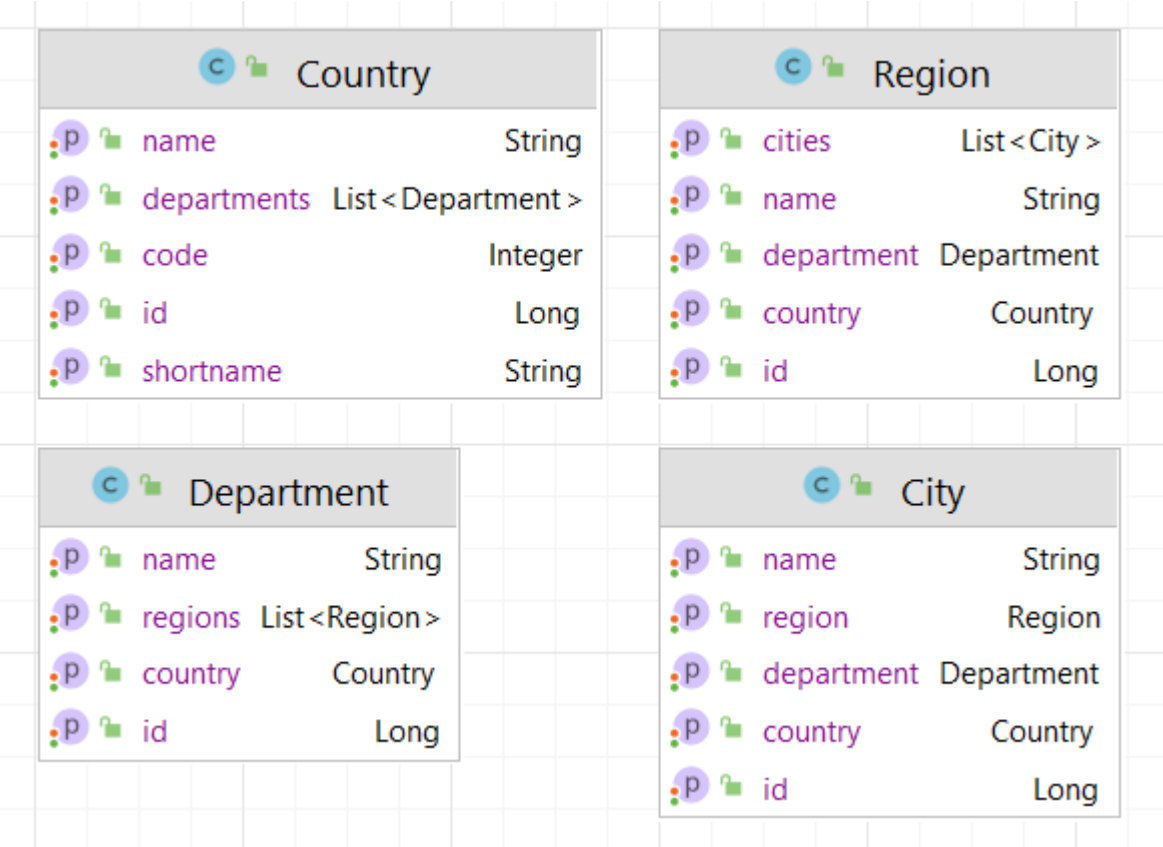

Рисунок 3.3 — Сутності MySQL

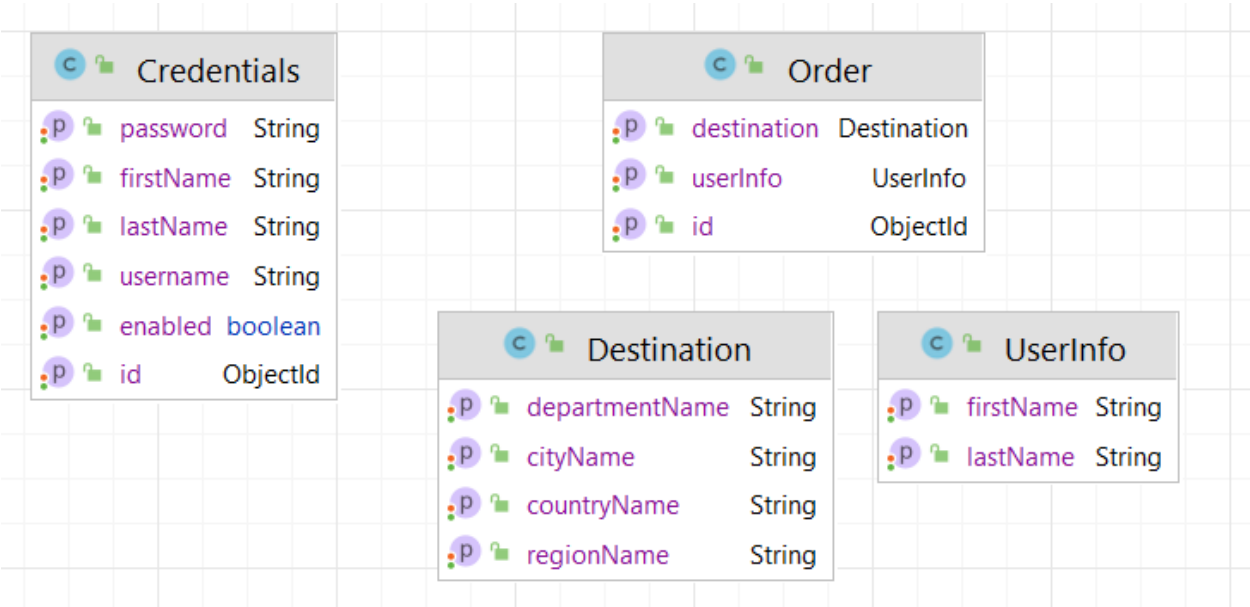

Рисунок 3.4 — Сутності MongoDB

Для MongoDB було створено 2 документа: Credentials та Order. В Order є 2 вкладених класи — це Destination та UserInfo, вони не є сутностями а лиш описують структуру даних.

Сервіси sql та mongo схожі за своєю структурою оскільки обидва є сервісами доступу до бази даних. В них містяться репозиторії, це такі спеціальні інтерфейси які за замовчуванням містять в собі всі базові методи для роботи з базою.

Для зменшення рівнів абстракції і кількості класів було вирішено не писати окремі контролери для доступу до методів репозиторіїв, а використати нову технологію під назвою Spring Data REST Repositories[19].

Дана технологія автоматично створює API для базових методів репозиторію. Для активації даної функції над інтерфейсом репозиторію потрібно вказати анотацію @RepositoryRestResource(collectionResourceRel = "name", path = "name"). Після чого можна буде звертатися до бази даних за допомогою HTTP протоколу. Це дозволяє скоротити кількість серіалізацій та десеріалізацій даних. Приклад репозиторію для Order наведений в лістингу 3.1.

Лістинг 3.1 — репозиторій для Order.

```
@Repository
@RepositoryRestResource(collectionResourceRel = "order", path = "order")
public interface OrderRepository extends MongoRepository<Order, ObjectId> {
}
```

Як видно з лістингу виходить досить лаконічний елемент який реалізує в собі все що нам потрібно.

З MongoDB все працює чудово, однак для MySQL потрібно зробити додаткові дії. Для того щоб не втрачати lazy відношення, і за рахунок них продуктивність роботи, потрібно зробити можливість реалізовувати цю технологію. По своїй суті lazy поля під’єднуються по запиту JOIN тільки тоді коли викликається гетер цього поля. Тобто нам потрібно зробити так, щоб він викликався за допомогою Spring Data REST Repositories.

Цього можна досягти за допомогою інтерфейсу з анотацією @Projection.

Даний інтерфейс являє собою набір методів з іменами полів, які за допомогою рефлексії розпізнають поля сутностей і додають в HTTP відповідь лише їх. Це дозволяє викликати гетери для lazy відношення в будь-якій комбінації,

оскільки ім’я проекції вказується в HTTP запиті як параметр. Приклад наведений в лістингу 3.2.

Лістинг 3.2 — репозиторій для Order.

```
@Projection(name = "withAll", types = { City.class })
public interface CityWithAllProjection {
   Long getId();
   String getName();
   Country getCountry();
   Department getDepartment();
   Region getRegion();
}
```

Для того щоб Spring Data REST Repositories використовував проекції, потрібно додати їх в клас конфігурації для даної технології. Також в ньому можна вказати щоб в JSON відповіді додавався Id для сутності, бо по замовчуванню він ігнорується. Для сервісу mongo створюється ідентичний клас, але без налаштувань проекцій, там вони поки не потрібні, але їх завжди можна додати якщо така потреба появиться.

Лістинг 3.3 — Файл конфігурації MySQL.

```
@Configuration
public class CustomRestMvcConfiguration {
   @Bean
   public RepositoryRestConfigurer repositoryRestConfigurer() {
      return new RepositoryRestConfigurer() {
         @Override
         public void configureRepositoryRestConfiguration(RepositoryRestConfiguration
config, CorsRegistry cors) {
            config.exposeIdsFor(Country.class);
            config.exposeIdsFor(Department.class);
```

```
            config.exposeIdsFor(Region.class);
            config.exposeIdsFor(City.class);
            config.getProjectionConfiguration()
                  .addProjection(CityWithAllProjection.class);
         }
      };
   }
}
```

В сервісі main стандартна архітектура репозиторій-сервіс-контролер була модифікована до клієнт-сервіс-контролер. Клієнт в даному випадку — це HTTP клієнт який надається технологією OpenFeign. Він заміняє роль репозиторію, оскільки звертається до них через HTTP запити.

Щоб створити Feign Client потрібно проанотувати інтерфейс анотацією @FeignClient, а також додати анотацію @EnableFeignClients в основний клас Spring Boot додатку. Цей клієнт автоматично серіалізує і десеріалізує об’єкти та передає їх через HTTP протокол, що робить його використання надзвичайно зручним. Також головною його перевагою перед аналогами є дуже лаконічний синтаксис у вигляді сигнатур методів в інтерфейсі, з невеликою кількістю анотацій. Приклад наведений в лістингу 3.4.

Лістинг 3.4 — Feign Client для City.

```
@FeignClient(name = "city", url = "${sql.url}/city/")
public interface CityClient {
   @GetMapping("/{id}")
   City getCityById(@PathVariable Long id, @RequestParam(required = false) String projection);
}
```

Як видно з лістингу є можливість досить просто вказувати навіть імена проекцій, що робить цей інструмент дуже потужним.

Інша частина сервісу main являє собою звичайні сервіси для бізнес логіки та REST контролери для API. Вони були створені для демонстрації роботи програми, їй функціонал буде показано в розділі 4, а лістинг коду буде у додатку Е.

Як висновок даного підрозділу можна підсумувати, що була розроблена система за принципами мікросервісної архітектури та за рекомендаціями до розподілу даних по різним базах даних. Було створено чотири мікросервіси: main, sql, mogno та entity. Були описані принципи роботи кожного з цих сервісів. Створені сутності за вказівками для оптимальної роботи баз даних. Описані технології Spring Data REST Repositories та OpenFeign. В результаті була спроектована та створена високопродуктивна та надійна система з двома базами даних.

4 ОЦІНЮВАННЯ ПРОДУКТИВНОСТІ СИСТЕМИ ТА РОЗРОБКА ДОКУМЕНТАЦІЇ

4.1 Результати тестування

Візьмемо результати тестування отримані в підрозділі 2.2 та переведемо їх у відсотки. Також вкажемо базу даних яка має перевагу в продуктивності та є оптимальною базою для операцій над даними та занесемо ці дані в таблицю 4.1.

Таблиця 4.1 — Ріст продуктивності та оптимальна база даних

№ тесту	Ріст продуктивності відносно менш ефективної бази, %	Оптимальна база даних
Тест 1	153%	MySQL
Тест 2	187%	MySQL
Тест 3	575%	MySQL
Тест 4	149%	MySQL
Тест 5	2814%	MongoDB
Тест 6	1830%	MongoDB
Тест 7	395%	MongoDB
Тест 8	677%	MongoDB

На основі цих даних побудуємо графік, де еталоном 100% (Рис 4.1).

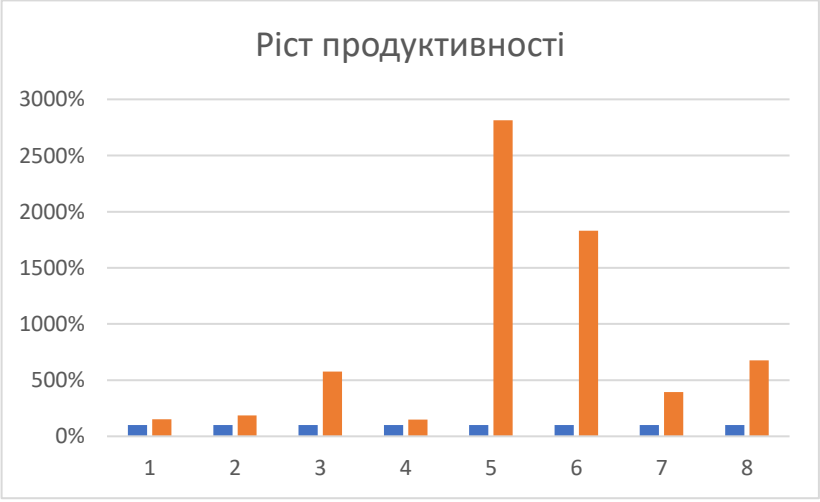

Рисунок 4.1 — Гістограма росту продуктивності

4.2 Інструкція користувача

Для використання даної системи потрібно звертатися до неї за допомогою HTTP запитів.

Для цього був доданий Swagger для кожного з сервісів.

Swagger — це структура специфікацій RESTful API. Його краса полягає в так званому інтерфейсі користувача Swagger, який не тільки дозволяє переглядати специфікації в інтерактивному режимі, але й дозволяє надсилати запити.

Ви також можете створювати клієнтів або сервери безпосередньо відповідно до специфікації API Swagger. Для цього потрібен Swagger Codegen.

Документація базується на вашому коді.

Такий підхід позиціонується як «дуже простий». Досить додати деякі залежності в проект і додати конфігурацію. Необхідна документація вже є, але необхідних пояснень немає.

Вся документація вигравірувана в нашому коді (всі контролери та моделі конвертовані у свого роду код Java Swagger).

Також Swagger автоматично створює документацію для Spring Data REST Repositories, що допоможе в розробці сервісів.

В документації містяться шляхи до кінцевих точок, а також вказівки для необхідних параметрів та JSON формат тіла запиту та тіла відповіді (Рис. 4.2).

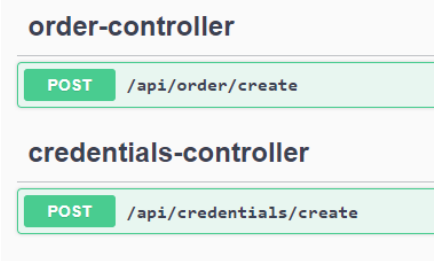

Рисунок 4.2 — Посилання кінцевих точок.

Spring Data REST Repositories генерує складну структуру схем. Вони містять не лише сутності, а ще й метаінформацію, яку можна використовувати в своїх програмах (Рис. 4.3).

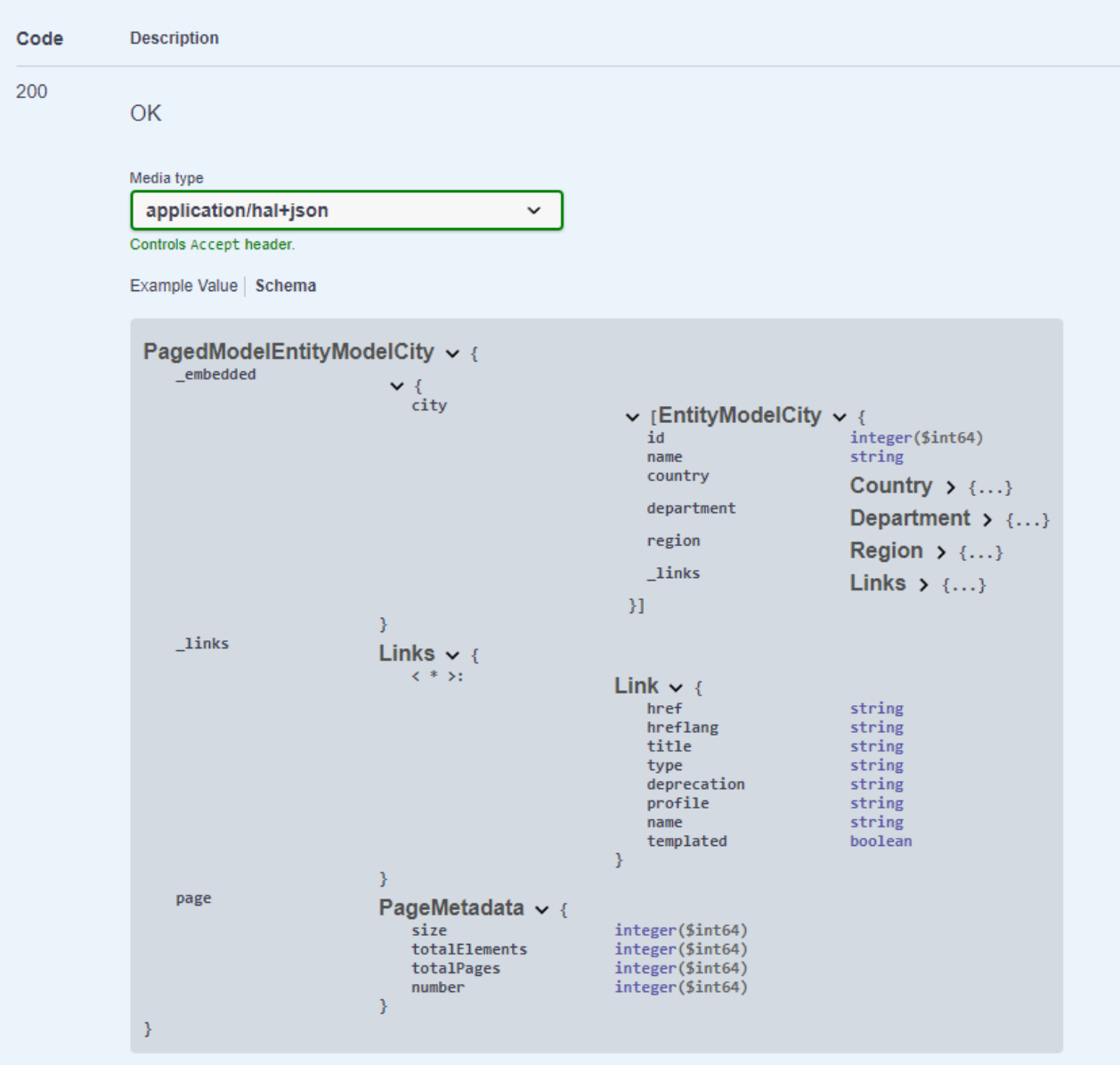

Рисунок 4.3 — Схема для City створена в Spring Data REST Repositories

Всі інші схеми побудовані за подібним принципом.

За допомогою програми Postman виконаємо кілька запитів до основного сервісу в якості демонстрації роботи програми.

Перший запит створює нового користувача в через через сервіс main. В результаті чого в MongoDB створюється документ з цим користувачем, а об’єкт з Id повертається у відповіді (Рис 4.4).

Другий запит є запитом до API створеного Spring Data REST Repositories і вертає в відповідь користувача за заданим Id. Окрім цього він містить також гіперпосилання на самого себе по двох ключах. (Рис 4.5).

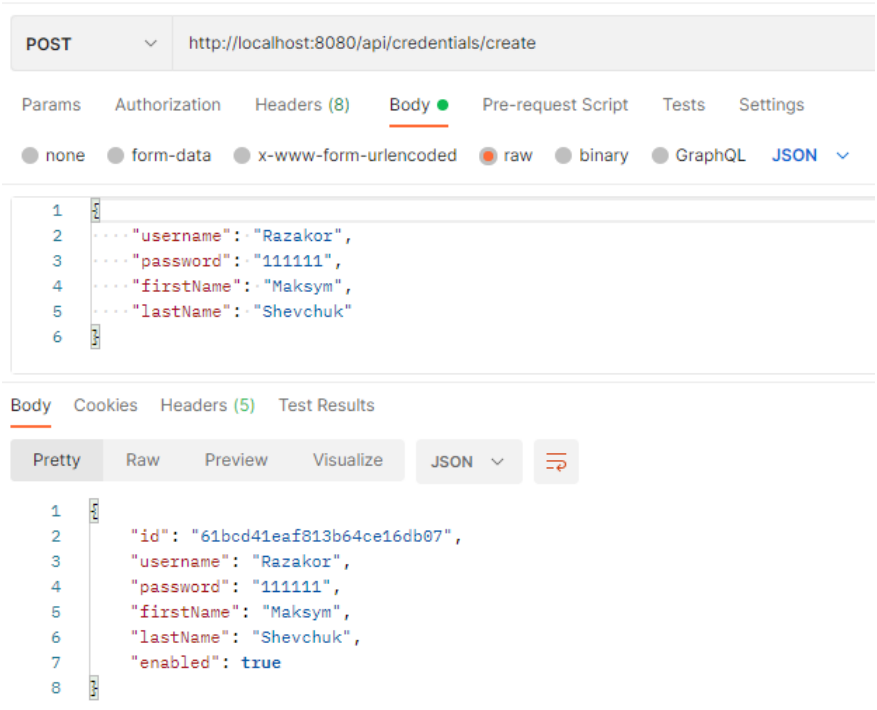

Рисунок 4.4 — Запит на створення користувача

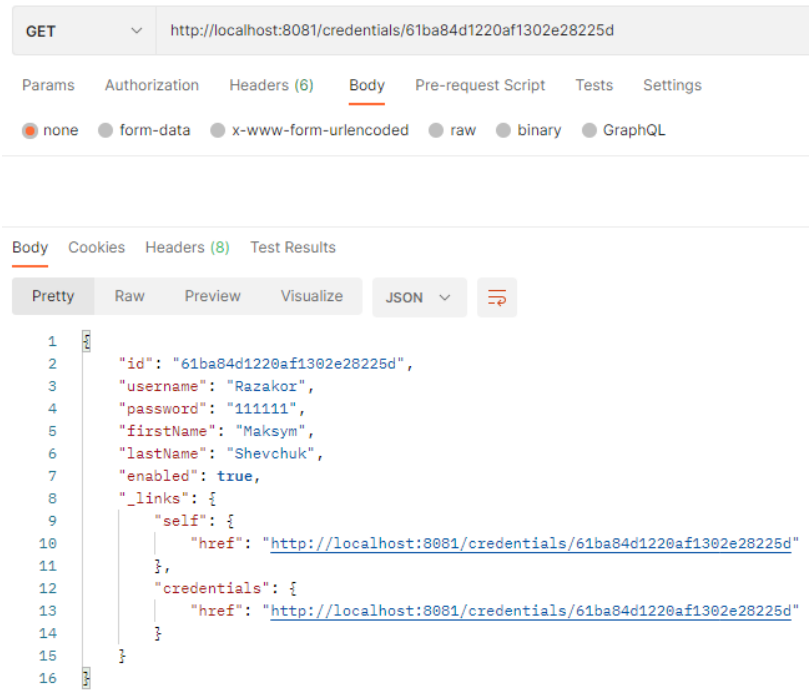

Рисунок 4.5 — Запит на отримання користувача з бази даних

Наступний запит — це запит до MySQL бази на отримання міста по Id. Основна його відмінність в тому, що цей запит містить параметр projection. Це означає що використовується проекція для того щоб отримати lazy поля з бази даних в відповіді.

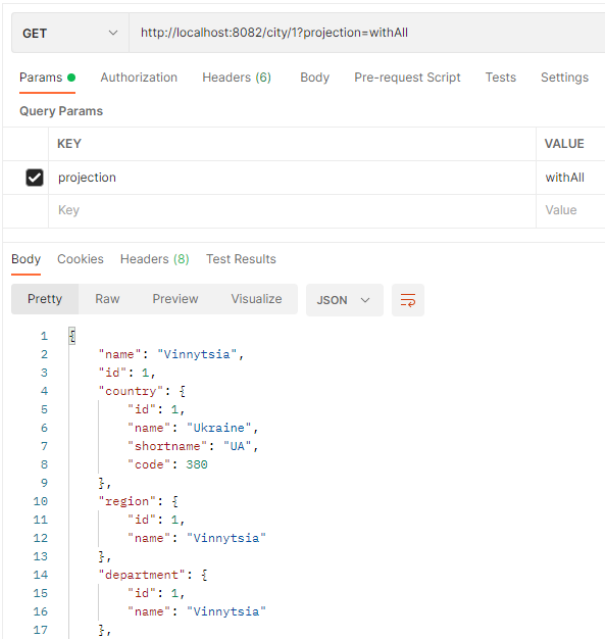

Рисунок 4.5 — Запит на отримання міста з бази даних

Останній запит — це запит на створення замовлення. В ньому поєднується основна ідея двох баз даних. UserId — це id MongoDB, а cityId — це id MySQL. В процесі обробки запиту сервіс main завертається за даними до mongo та sql через FeignClient. В результаті чого компонує дані з двох баз в один об’єкт який і повертає.

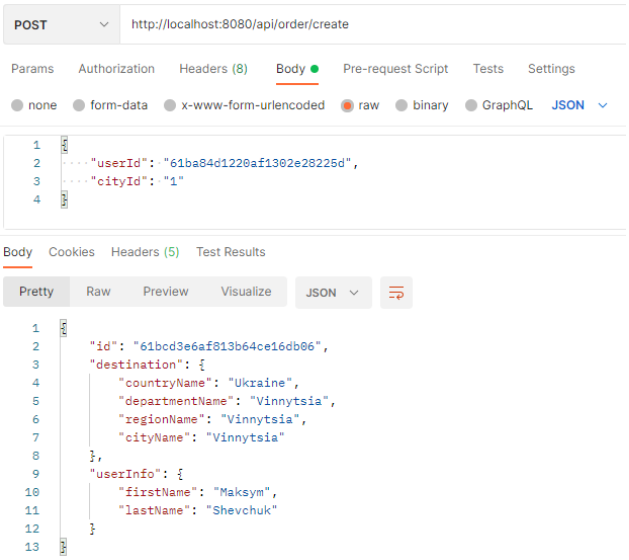

Рисунок 4.6 — Запит на створення замовлення

В даному розділі було розроблено інструкцію користувача і проведено оцінювання продуктивності роботи системи.

5 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка має право на існування та впровадження, якщо вона відповідає вимогам часу, як в напрямку науково-технічного прогресу та і в плані економіки. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

Магістерська кваліфікаційна робота з розробки та дослідження «Високопродуктивна система управління потоками даних на основі MySQL і MongoDB» відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи), тобто коли відбувається так звана комерціалізація науково-технічної розробки. Цей напрямок є пріоритетним, оскільки результатами розробки можуть користуватися інші споживачі, отримуючи при цьому певний економічний ефект. Але для цього потрібно знайти потенційного інвестора, який би взявся за реалізацію цього проекту і переконати його в економічній доцільності такого кроку.

Для наведеного випадку нами мають бути виконані такі етапи робіт:

1) проведено комерційний аудит науково-технічної розробки, тобто встановлення її науково-технічного рівня та комерційного потенціалу;

2) розраховано витрати на здійснення науково-технічної розробки;

3) розрахована економічна ефективність науково-технічної розробки у випадку її впровадження і комерціалізації потенційним інвестором і проведено обґрунтування економічної доцільності комерціалізації потенційним інвестором.

5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Високопродуктивна система управління потоками даних на основі MySQL і MongoDB» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в табл. 5.1[20].

Таблиця 5.1 — Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в	Технічні та споживчі влас-тивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в	Технічні та споживчі властивості продукту значно кращі, ніж в
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає

Закінчення таблиці 5.1

Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь- які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно звести до таблиці.

Таблиця 5.2 — Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	5	5	5
2. Ринкові переваги (наявність аналогів)	4	4	4
3. Ринкові переваги (ціна продукту)	3	4	4
4. Ринкові переваги (технічні властивості)	4	4	4
5. Ринкові переваги (експлуатаційні витрати)	3	4	4
6. Ринкові перспективи (розмір ринку)	4	4	3
7. Ринкові перспективи (конкуренція)	3	3	4
8. Практична здійсненність (наявність фахівців)	3	3	3
9. Практична здійсненність (наявність фінансів)	3	4	4
10. Практична здійсненність (необхідність нових матеріалів)	4	4	4
11. Практична здійсненність (термін реалізації)	4	4	4
12. Практична здійсненність (розробка документів)	4	4	3
Сума балів	44	47	46
Середньоарифметична сума балів $СБ_с$	45,7		

За результатами розрахунків, наведених в таблиці 5.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 5.3[20].

Таблиця 5.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ , розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Високопродуктивна система управління потоками даних на основі MySQL і MongoDB» становить 45,7 бала, що, відповідно до таблиці 4.3, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки високий).

5.2 Визначення рівня конкурентоспроможності розробки

В процесі визначення економічної ефективності науково-технічної розробки також доцільно провести прогноз рівня її конкурентоспроможності за сукупністю параметрів, що підлягають оцінюванню.

Одиничний параметричний індекс розраховуємо за формулою[20]:

$$q_i = \frac{P_i}{P_{баз\,i}}. \qquad (5.1)$$

де q_i — одиничний параметричний індекс, розрахований за i-м параметром;

P_i — значення i-го параметра виробу;

$P_{базі}$ — аналогічний параметр базового виробу-аналога, з яким проводиться порівняння.

Загальні технічні та економічні характеристики розробки представлено в таблиці 5.4.

Таблиця 5.4 — Основні техніко-економічні показники аналога та розробки, що проектується

Показники (параметри)	Одиниця вимірю-вання	Аналог	Проектоване програмне забезпечення	Відношення параметрів нової розробки до аналога	Питома вага показника
Продуктивність при збережені даних зі зв'язками	%	100	153	1,53	0,3
Продуктивність при вичитці даних зі зв'язками	%	100	187	1,87	0,4
Продуктивність при збережені даних з багатьма вкладеними об'єктами	%	100	2814	28,14	0,05
Продуктивність при вичитці даних з вкладеними об'єктами	%	100	1830	18,3	0,07
Продуктивність при редагуванні багатьох об'єктів	%	100	395	3,95	0,18
Експлуатаційні витрати	грн	210	200	0,95	0,45
Ціна	грн	3000	2500	0,83	0,55

Нормативні параметри оцінюємо показником, який отримує одне з двох значень: 1 — пристрій відповідає нормам і стандартам; 0 — не відповідає.

Груповий показник конкурентоспроможності за нормативними параметрами розраховуємо як добуток частинних показників за кожним параметром за формулою[20]:

$$I_{НП} = \prod_{i=1}^{n} q_i \ , \tag{5.2}$$

де $I_{нп}$ — загальний показник конкурентоспроможності за нормативними параметрами;

q_i — одиничний (частинний) показник за i-м нормативним параметром;

n — кількість нормативних параметрів, які підлягають оцінюванню.

За нормативними параметрами розроблюваний пристрій відповідає вимогам ДСТУ, тому $I_{нп} = 1$.

Значення групового параметричного індексу за технічними параметрами визначаємо з урахуванням вагомості (частки) кожного параметра[20]:

$$I_{ТП} = \sum_{i=1}^{n} q_i \cdot \alpha_i \ , \tag{5.3}$$

де $I_{тп}$ — груповий параметричний індекс за технічними показниками (порівняно з виробом-аналогом);

q_i — одиничний параметричний показник i-го параметра;

α_i — вагомість i-го параметричного показника, $\sum_{i=1}^{n} \alpha_i = 1$;

n —— кількість технічних параметрів, за якими оцінюється конкурентоспроможність.

Проведемо аналіз параметрів згідно даних таблиці 4.4.

$$I_{mn} = 1{,}53 \cdot 0{,}3 + 1{,}87 \cdot 0{,}4 + 28{,}14 \cdot 0{,}05 + 18{,}3 \cdot 0{,}07 + 3{,}95 \cdot 0{,}18 = 4{,}61.$$

Груповий параметричний індекс за економічними параметрами

розраховуємо за формулою[20]:

$$I_{ЕП} = \sum_{i=1}^{m} q_i \cdot \beta_i \,, \tag{5.4}$$

де $I_{ЕП}$ — груповий параметричний індекс за економічними показниками;

q_i — економічний параметр i-го виду;

β_i — частка i-го економічного параметра, $\sum_{i=1}^{m} \beta_i = 1$;

m — кількість економічних параметрів, за якими здійснюється оцінювання.

Проведемо аналіз параметрів згідно даних таблиці .

$$I_{ЕП} = 0{,}95 \cdot 0{,}45 + 0{,}83 \cdot 0{,}55 = 0{,}88.$$

На основі групових параметричних індексів за нормативними, технічними та економічними показниками розрахуємо інтегральний показник конкурентоспроможності за формулою[20]:

$$K_{ІНТ} = I_{НП} \cdot \frac{I_{ТП}}{I_{ЕП}} \,, \tag{5.5}$$

$$K_{ІНТ} = 1 \cdot 4{,}61 / 0{,}88 = 5{,}21.$$

Інтегральний показник конкурентоспроможності $К_{ІНТ} > 1$, отже розробка переважає відомі аналоги за своїми техніко-економічними показниками.

5.3 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Високопродуктивна система управління потоками даних на основі MySQL і MongoDB», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

5.3.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою[20]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \quad (5.6)$$

де k — кількість посад дослідників залучених до процесу досліджень;

M_{ni} — місячний посадовий оклад конкретного дослідника, грн;

t_i — число днів роботи конкретного дослідника, дн.;

T_p — середнє число робочих днів в місяці, T_p=22 дні.

$$З_о = 13170{,}00 \cdot 22 / 22 = 13170{,}00 \; грн.$$

Проведені розрахунки зведемо до таблиці.

Таблиця 5.5 — Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	13170,00	598,64	22	13170,00
Інженер-розробник програмного забезпечення	12150,00	552,27	17	9388,64

Закінчення таблиці 5.5

Науковий співробітник	12400,00	563,64	5	2818,18
Технік	7350,00	334,09	5	1670,45
Всього				27047,27

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР на тему «Високопродуктивна система управління потоками даних на основі MySQL і MongoDB» розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i , \tag{5.7}$$

де C_i — погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i — час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{М_М \cdot К_i \cdot К_с}{Т_р \cdot t_{зм}} , \tag{5.8}$$

де $М_М$ — розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо $М_М$=2379,00 грн;

$К_i$ — коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду[20];

$К_с$ — мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об’єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

$Т_р$ — середнє число робочих днів в місяці, приблизно $Т_р$ = 22 дн;

$t_{зм}$ — тривалість зміни, год.

$C_1 = 2379{,}00 \cdot 1{,}10 \cdot 1{,}65 / (22 \cdot 8) = 24{,}53$ *грн.*

$З_{р1} = 24{,}53 \cdot 7{,}10 = 174{,}19$ *грн.*

Таблиця 5.6 — Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Установка електронно-обчислювального обладнання	7,10	2	1,10	24,53	174,19
Підготовка робочого місця розробника програмного забезпечення	5,50	2	1,10	24,53	134,93
Інсталяція програмного забезпечення	8,20	5	1,70	37,92	310,91
Формування дослідної бази даних (2 бази)	24,00	2	1,10	24,53	588,80
Налагодження програмних модулів	5,40	5	1,70	37,92	204,74
Підготовка програмного забезпечення серверного обладнання	11,00	5	1,70	37,92	417,07

Закінчення таблиці 5.6

Тестування програмного забезпечення	11,00	4	1,50	33,45	368,00
Всього					2198,64

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (5.9)$$

де $Н_{дод}$ — норма нарахування додаткової заробітної плати. Приймемо 11%.

$З_{дод}$ = (27047,27 + 2198,64) · 11 / 100% = 3217,05 грн.

5.3.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (5.10)$$

де $Н_{зп}$ — норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (27047,27 + 2198,64 + 3217,05) · 22 / 100% = 7141,85 грн.

5.3.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення

досліджень за темою «Високопродуктивна система управління потоками даних на основі MySQL і MongoDB».

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$M = \sum_{j=1}^{n} H_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} B_j \cdot Ц_{вj} , \quad (5.11)$$

де H_j — норма витрат матеріалу *j*-го найменування, кг;

n — кількість видів матеріалів;

$Ц_j$ — вартість матеріалу *j*-го найменування, грн/кг;

$К_j$ — коефіцієнт транспортних витрат, ($К_j$ = *1,1 … 1,15*);

B_j — маса відходів *j*-го найменування, кг;

$Ц_{вj}$ — вартість відходів *j*-го найменування, грн/кг.

M_1 = *4,00 · 100,00 · 1,1 - 0,000 · 0,00 = 440,00 грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 5.7 — Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, кг	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Офісний папір 500-80	100,00	4,00	0,000	0,00	440,00
Папір для записів А5	50,00	4,00	0,000	0,00	220,00
Органайзер офісний Office	210,00	3,00	0,000	0,00	693,00
Канцелярське приладдя (набір офісного працівника)	180,00	3,00	0,000	0,00	594,00

Закінчення таблиці 5.7

Картридж для принтера Canon LBP	900,00	1,00	0,000	0,00	990,00
Диск оптичний Optivisio CD-RW	15,00	3,00	0,000	0,00	49,50
Flesh-пам'ять DATA 16 GB	120,00	1,00	0,000	0,00	132,00
Тека для паперів	95,00	3,00	0,000	0,00	313,50
Всього					3432,00

5.3.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Високопродуктивна система управління потоками даних на основі MySQL і MongoDB» відсутні.

5.3.5 Спецустаткування для наукових робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення.

Балансову вартість спецустаткування розраховуємо за формулою:

$$B_{спец} = \sum_{i=1}^{k} Ц_i \cdot C_{пр.i} \cdot K_i \ , \qquad (5.12)$$

де $Ц_i$ — ціна придбання одиниці спецустаткування даного виду, марки, грн;

$C_{пр.i}$ — кількість одиниць устаткування відповідного найменування, які придбані для проведення досліджень, шт.;

K_i — коефіцієнт, що враховує доставку, монтаж, налагодження устаткування тощо, (K_i = *1,10…1,12);*

k — кількість найменувань устаткування.

$B_{спец}$ = *27394,00 · 1 · 1,11 = 30407,34 грн.*

Отримані результати зведемо до таблиці:

Таблиця 5.8 — Витрати на придбання спецустаткування по кожному виду

Найменування устаткування	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Ноутбук Asus TUF Gaming A17	1	27394,00	30407,34
Всього			30407,34

5.3.6 Програмне забезпечення для наукових робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (5.13)$$

де $Ц_{iпрг}$ — ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ — кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i — коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, (K_i = *1,10…1,12);*

k — кількість найменувань програмних засобів.

$B_{прг}$ = *9325,00 · 1 · 1,11 = 10350,75 грн.*

Отримані результати зведемо до таблиці:

Таблиця 5.9 — Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Середовище розробки Intellij IDEA Ultimate	1	9325,00	10350,75
База даних MySQL Server	1	4672,00	5185,92
База даних MongoDB Server	1	3975,00	4412,25
Всього			19948,92

5.3.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (5.14)$$

де $Ц_б$ — балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ — термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ — строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл} = (21750{,}00 \cdot 1) / (3 \cdot 12) = 604{,}17$ *грн.*

Проведені розрахунки зведемо до таблиці.

Витрати на силову електроенергію ($B_е$) розраховуємо за формулою:

$$B_е = \sum_{i=1}^{n} \frac{W_{уі} \cdot t_i \cdot Ц_е \cdot K_{вті}}{\eta_i}, \quad (5.15)$$

де $W_{уі}$ — встановлена потужність обладнання на певному визначеному етапі розробки, кВт;

Таблиця 5.10 — Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Персональний комп'ютер	21750,00	3	1	604,17
Робоче місце розробника програмного забезпечення	7980,00	5	1	133,00
Пристрої виводу інформації	6840,00	4	1	142,50
Оргтехніка	8395,00	4	1	174,90
Приміщення лабораторії	197000,00	25	1	656,67
ОС Windows 10	5459,00	2	1	227,46
Прикладний пакет Microsoft Office 2016	3786,00	2	1	157,75
Засоби передачі даних	5265,00	2	1	219,38
Всього				2315,81

5.3.8 Паливо та енергія для науково-виробничих цілей

t_i — тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ — вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_e = 4,50$ грн;

$K_{вп i}$ — коефіцієнт, що враховує використання потужності, $K_{вп i} < 1$;

η_i — коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 0,45 \cdot 176,0 \cdot 4,50 \cdot 0,95 / 0,97 = 356,40$ грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 5.11 — Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Персональний комп'ютер	0,45	176,0	356,40
Робоче місце розробника програмного забезпечення	0,15	160,0	108,00
Пристрої виводу інформації	0,03	24,0	3,24
Оргтехніка	0,70	6,0	18,90
Ноутбук Asus TUF Gaming A17	0,05	100,0	22,50
Засоби передачі даних	0,05	160,0	36,00
Всього			545,04

5.3.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи на тему «Високопродуктивна система управління потоками даних на основі MySQL і MongoDB» належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру,

аспірантів, зайнятих розробленням досліджень, відрядження, пов'язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \quad (5.16)$$

де $Н_{св}$ — норма нарахування за статтею «Службові відрядження», приймемо $Н_{св}$ = 20%.

$B_{св}$ = (27047,27 + 2198,64) · 20 / 100% = 5849,18 грн.

5.3.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуємо як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{сп} = (З_о + З_р) \cdot \frac{Н_{сп}}{100\%}, \quad (5.17)$$

де $Н_{сп}$ — норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо $Н_{сп}$= 30%.

$B_{сп}$ = (27047,27 + 2198,64) · 30 / 100% = 8773,77 грн.

5.3.11 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми

основної заробітної плати дослідників та робітників за формулою:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.18)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 60%.

$I_в$ = (27047,27 + 2198,64) · 60 / 100% = 17547,55 грн.

5.3.12 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.19)$$

де $Н_{нзв}$ — норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 122%.

$В_{нзв}$ = (27047,27 + 2198,64) · 122 / 100% = 35680,02 грн.

Витрати на проведення науково-дослідної роботи на тему «Високопродуктивна система управління потоками даних на основі MySQL і MongoDB» розраховуємо як суму всіх попередніх статей витрат за формулою:

$$В_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}. \quad (5.20)$$

$В_{заг}$= 27047,27 +2198,64 +3217,05 +7141,852392 +3432,00 +0,00 +30407,34 + 19948,92 + 2315,81 +545,04 +5849,18 +8773,77 +17547,55 +35680,02 = 164104,45 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{В_{заг}}{\eta}, \quad (5.21)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,9.

ЗВ = 164104,45 / 0,9 = 182338,28 грн.

5.4 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження проведені за темою «Високопродуктивна система управління потоками даних на основі MySQL і MongoDB» передбачають комерціалізацію протягом 4-х років реалізації на ринку.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

ΔN — збільшення кількості споживачів продукту, у періоди часу, що аналізуються, від покращення його певних характеристик;

Таблиця 5.12 — Показник збільшення кількості споживачів

Показник	1-й рік	2-й рік	3-й рік	4-й рік
Збільшення кількості споживачів, осіб	500	2000	4000	1000

N — кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, приймемо 17000 осіб;

$Ц_б$ — вартість програмного продукту у році до впровадження результатів

розробки, приймемо 1800,00 грн;

$\pm\Delta Ц_o$ — зміна вартості програмного продукту від впровадження результатів науково-технічної розробки, приймемо 700,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 4-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою [20]:

$$\Delta П_i = (\pm\Delta Ц_o \cdot N + Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (5.22)$$

де λ — коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2021 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ — коефіцієнт, який враховує рентабельність інноваційного продукту). Приймемо $\rho = 25\%$;

ϑ — ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2021 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1 = (700,00 \cdot 17000,00 + 2500,00 \cdot 500) \cdot 0,83 \cdot 0,25 \cdot (1\text{-}0,18/100\%) = 2237472,50$ грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2 = (700,00 \cdot 17000,00 + 2500,00 \cdot 2500) \cdot 0,83 \cdot 0,25 \cdot (1\text{-}0,18/100\%) = 3088222,50$ грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3 = (700,00 \cdot 17000,00 + 2500,00 \cdot 6500) \cdot 0,83 \cdot 0,25 \cdot (1\text{-}0,18/100\%) = 4789722,50$ грн.

Збільшення чистого прибутку 4-го року:

$\Delta П_4 = (700,00 \cdot 17000,00 + 2500,00 \cdot 7500) \cdot 0,83 \cdot 0,25 \cdot (1\text{-}0,18/100\%) = 5215097,50$ грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.23)$$

де $\Delta П_i$ — збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T — період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ — ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,1;

t — період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = 2237472{,}50/(1+0{,}1)^1 + 3088222{,}50/(1+0{,}1)^2 + 4789722{,}50/(1+0{,}1)^3 +$$
$$+ 5215097{,}50/(1+0{,}1)^4 = 2034065{,}91 + 2552250{,}00 + 3598589{,}41 + 3561981{,}76 =$$
$$= 11746887{,}08 \text{ грн.}$$

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \quad (5.24)$$

де $k_{інв}$ — коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$ =2;

$ЗВ$ — загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 182338,28 грн.

$PV = k_{інв} \cdot ЗВ = 2 \cdot 182338{,}28 = 364676{,}56$ грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \quad (5.25)$$

де $ПП$ — приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 11746887,08 грн;

PV — теперішня вартість початкових інвестицій, 364676,56 грн.

$E_{абс} = ПП - PV$ = 11746887,08 - 364676,56 = 11382210,52 грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (5.26)$$

де $E_{абс}$ — абсолютний економічний ефект вкладених інвестицій, 11382210,52 грн;

PV — теперішня вартість початкових інвестицій, 364676,56 грн;

$T_ж$ — життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 4 роки.

$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1$ = $(1+11382210,52/364676,56)^{1/4}$= 1,38.

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \quad (5.27)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2021 році в Україні d =0,11;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,16.

$\tau_{мін}$ = 0,11+0,16 = 0,27 < 1,38 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою

«Високопродуктивна система управління потоками даних на основі MySQL і MongoDB» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \tag{5.28}$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 1,38 = 0,72 р.

$T_{ок} <$ 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок[21].

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Високопродуктивна система управління потоками даних на основі MySQL і MongoDB» становить 45,7 бала, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки високий). При оцінюванні рівня конкурентоспроможності, згідно узагальненого коефіцієнту конкурентоспроможності розробки, науково-технічна розробка переважає існуючі аналоги приблизно в 5,21 рази. Також термін окупності становить 0,72 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

ВИСНОВКИ

В першому розділі були розглянуті архітектурні програмні рішення та обрана мікросервісна архітектура для написання додатку. Були проаналізовані різноманітні бази даних та обрані реляційна MySQL та документальна MongoDB. Були проаналізовані мови програмування та обрана Java як найбільш коректна для поставленої задачі. Був розглянутий стек технологій Spring Framework 5 та Spring Boot.

В другому розділі було наведено переваги підходу з використанням двох різних баз даних. Проведений огляд технологій для тестування продуктивності без даних. Розглянуті Docker та Testcontainers. Проведено тестування продуктивності баз даних на восьми сценаріях. Зібрані статистичні дані про продуктивність. Проведений аналіз даних та побудовані характерні графіки.

В третьому розділі була розроблена система за принципами мікросервісної архітектури та за рекомендаціями до розподілу даних по різним базах даних. Було створено чотири мікросервіси: main, sql, mogno та entity. Були описані принципи роботи кожного з цих сервісів. Створені сутності за вказівками для оптимальної роботи баз даних. Описані технології Spring Data REST Repositories та OpenFeign. В результаті була спроектована та створена високопродуктивна та надійна система з двома базами даних.

В четвертому розділі було розроблено інструкцію користувача і проведено оцінювання продуктивності роботи системи.

В п’ятому розділі згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Високопродуктивна система управління потоками даних на основі MySQL і MongoDB» становить 45,7 бала, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки високий). При оцінюванні рівня конкурентоспроможності, згідно узагальненого коефіцієнту конкурентоспроможності розробки, науково-технічна розробка переважає існуючі аналоги приблизно в 5,21 рази. Також термін окупності становить 0,72 р., що менше 3-х років, що свідчить про комерційну

привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Високопродуктивна система управління потоками даних на основі MySQL і MongoDB».

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Nadareishvili I., Mitra R., McLarty, M., Amundsen M., Microservice Architecture: Aligning Principles, Practices, and Culture. — O'Reilly Media 2016. — 146 c.

2. М. О. Шевчук. Високопродуктивна система управління потоками даних на основі MySQL і MongoDB, 2021. — 2 с..

3. М. О. Шевчук. Рекомендації по покращенню роботи з даними з використанням MySQL і MongoDB, 2021. — 2 с..

4. Краткий обзор 10 популярных архитектурных шаблонов приложений [Електронний ресурс]. Режим доступу: https://medium.com/nuances-of-programming/краткий-обзор-10-популярных-архитектурных-шаблонов-приложений-81647be5c46f . Дата звернення: Грудень 19, 2021.

5. Г. О. Горбачов, М. О. Шевчук, О. М. Ткаченко. Аналіз підходів до проектуванняклієнт-серверних сервісів, 2020. — 2 с..

6. Монолитная vs Микросервисная архитектура [Електронний ресурс]. Режим доступу: https://proglib.io/p/monolitnaya-vs-mikroservisnaya-arhitektura-2019-09-16. Дата звернення: Грудень 19, 2021.

7. Types of databases [Електронний ресурс]. Режим доступу: https://www.tutorialspoint.com/Types-of-databases. Дата звернення: Грудень 19, 2021.

8. Garcia-Molina H., Ullman J. D., Widom J., Database Systems The Complete Book Second Edition. — Department Of Computer Science Stanford University, 2009. — 1240 c.

9. Banker K., MongoDB in Action. — Manning Publications 2011.— 312 c.

10. Классификация языков программирования [Електронний ресурс]. Режим доступу: https://spravochnick.ru/programmirovanie/yazyki_programmirovaniya/klassifikaciya_yazykov_programmirovaniya/. Дата звернення: Грудень 19, 2021.

11. C++ | Введение [Електронний ресурс]. Режим доступу: https://metanit.com/cpp/tutorial/1.1.php. Дата звернення: Грудень 19, 2021

12. Jemerov D., Isakov S., Kotlin in Action. — Manning; 1st edition, 2017. — 360 с.

13. C# и .NET | Введение [Електронний ресурс]. Режим доступу: https://metanit.com/sharp/tutorial/1.1.php. Дата звернення: Грудень 19, 2021

14. Urma R., Fusco M., Mycroft A., — Modern Java in Action. — Manning; 2nd edition, 2018. — 592 с.

15. Java EE Vs Spring: Which One Is More Popular Among The Developers? [Електронний ресурс]. Режим доступу: https://xperti.io/blogs/java-ee-vs-spring/. Дата звернення: Грудень 19, 2021

16. Введение в Spring Boot: создание простого REST API на Java / Хабр [Електронний ресурс]. Режим доступу: https://habr.com/ru/post/435144/. Дата звернення: Грудень 19, 2021

17. Testcontainers [Електронний ресурс]. Режим доступу: https://www.testcontainers.org/. Дата звернення: Грудень 19, 2021

18. Что такое Docker? | AWS [Електронний ресурс]. Режим доступу: https://aws.amazon.com/ru/docker/. Дата звернення: Грудень 19, 2021

19. Spring Data REST [Електронний ресурс]. Режим доступу: https://spring.io/projects/spring-data-rest. Дата звернення: Грудень 19, 2021

20. В. О. Козловський, О. Й. Лесько, В. В. Кавецький., — Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. — Вінниця: ВНТУ, 2021. – 42 с.

21. В. В. Кавецький, В. О. Козловський, І. В. Причепа, — Економічне обґрунтування інноваційних рішень: практикум. — Вінниця: ВНТУ, 2016. – 42 с.

ДОДАТОК А

Міністерство освіти та науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп’ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ ВНТУ
д.т.н., проф. О. Д. Азаров
“__” _____ 2021 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи
«Методи виявлення передавання прихованої інформації в службових заголовках протокольних блоків даних комп'ютерних мереж»
08-23.МКР.014.00.000 ПЗ

Науковий керівник к.т.н., доц. каф. ОТ
_____ Ткаченко О. М.
Студент групи 1КІ-20м
_____ Шевчук М. О.

Вінниця 2021

1 Підстава для використання МКР:

— актуальність розробки полягає у вдосконаленні технології побудови мікросервісних систем, в якій, на відміну від існуючих, реалізовано взаємодію з документальною та реляційною базою даних, що дозволило підвищити продуктивність роботи систем;

— наказ про затвердження теми дипломної роботи від “06” березня 2021 року №75.

2 Мета і призначення МКР:

— мета роботи — підвищення продуктивності серверних застосунків за допомогою мікросервісної архітектури та розподілених баз даних;

— призначення розробки — здійснити аналіз способів побудови систем з мікросервісною архітектурою та розподіленими базами даних та запропонувати кращий підхід до організації роботи з даними в мікросервісних системах.

3 Джерела розробки магістерської кваліфікаційної роботи

Інтегроване середовище розробки IntellijIDEA Ultimate для платформи JVM та мови програмування Java з використанням Spring Framework та Spring Boot.

4 Технічні вимоги до виконання МКР:

— наявність встановленої JVM;

— наявність програми, яка робить HTTP запити.

5 Етапи МКР та очікувані результати

Робота виконується за сім етапів, таблиця А.1.

6 Матеріали, що подаються до захисту МКР

Пояснювальна записка ДР, графічні і ілюстративні матеріали, протокол попереднього захисту ДР на кафедрі, відзив наукового керівника, відзив опонента, протоколи проходження перевірки на плагіат, анотації до МКР українською та іноземною мовами, нормоконтроль про відповідність оформлення ДР діючим вимогам.

Таблиця А.1 — Етапи виконання роботи.

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Постановка задачі роботи	12.09.22	21.09.21	Вступ
2	Аналіз підходів до створення мікросервісних систем з двома базами даних	22.09.21	15.10.21	Розділ 1
3	Математичне обґрунтування продуктивності роботи баз даних з різними структурами даних	16.10.21	01.11.21	Розділ 2
4	Опис рекомендацій для підвищення продуктивності баз даних та створення програмного додатку який реалізує систему з двома базами даних	02.11.21	28.11.21	Розділ 3
5	Тестування додатку та написання інструкції користувача	29.11.21	05.12.21	Розділ 4, додатки
6	Обґрунтування економічної доцільності розробки	06.12.21	11.12.21	Розділ 5
7	Оформлення пояснювальної записки	12.12.21	21.12.21	ПЗ

7 Порядок контролю виконання та захисту МКР

Виконання етапів графічної та розрахункової документації ДР контролюється науковим керівником згідно зі встановленими термінами. Захист ДР відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

8 Вимоги до оформлення МКР

Вимоги викладені ДСТУ 3008-2015, ДСТУ 3974-2000 «Правила виконання дослідно-конструкторських робіт. Загальні положення» та діючого ГОСТ 2.114-95 ЕСКД.

Технічне завдання до виконання отримав _____ Шевчук М. О.

ДОДАТОК Б

Архітектура системи

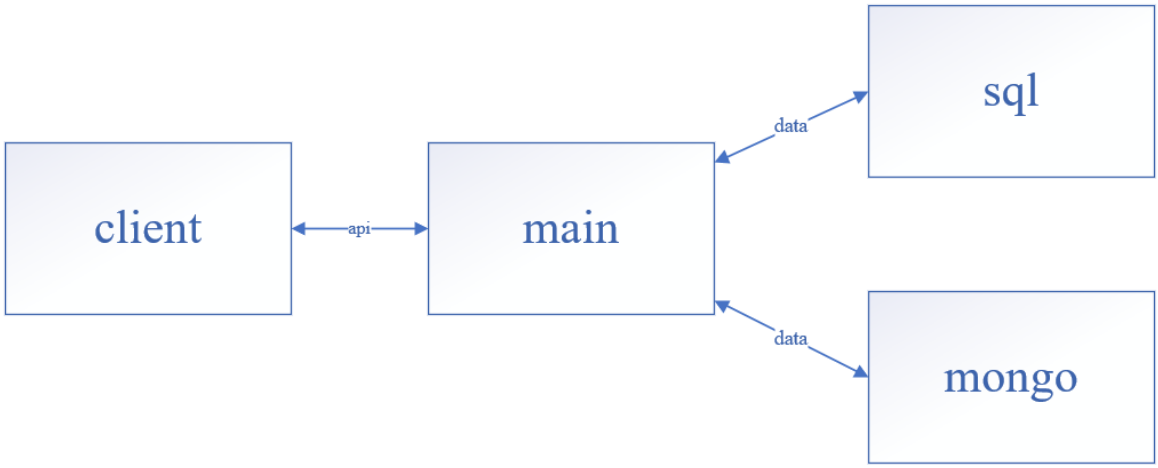

Рисунок Б.1 — Архітектура системи

ДОДАТОК В

Гістограма росту продуктивності

Рисунок В.1 — Гістограма росту продуктивності

ДОДАТОК Г

UML-діаграма відношень в MySQL

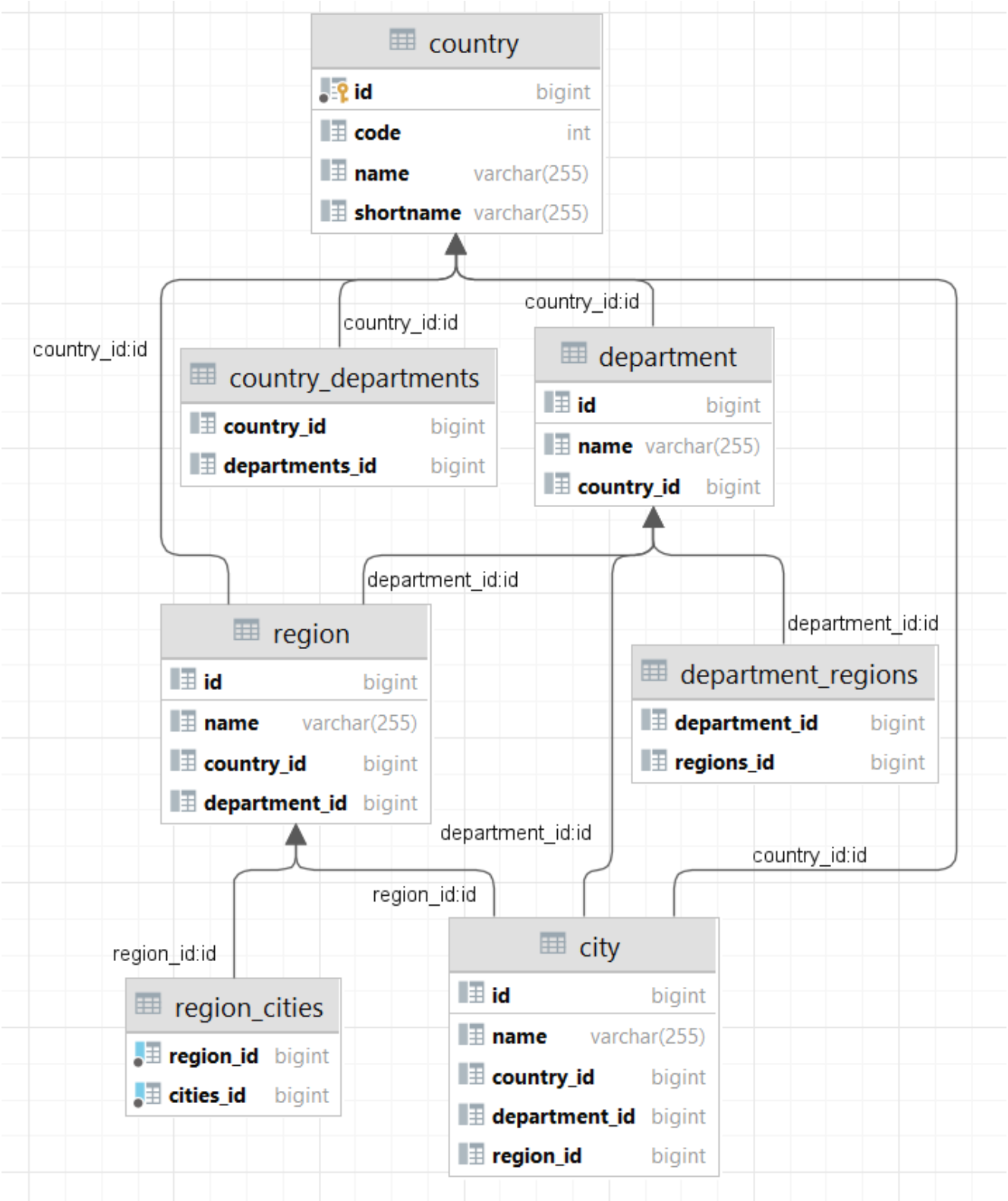

Рисунок Г.1 — UML-діаграма відношень в MySQL

ДОДАТОК Д

UML-діаграма документів в MongoDB

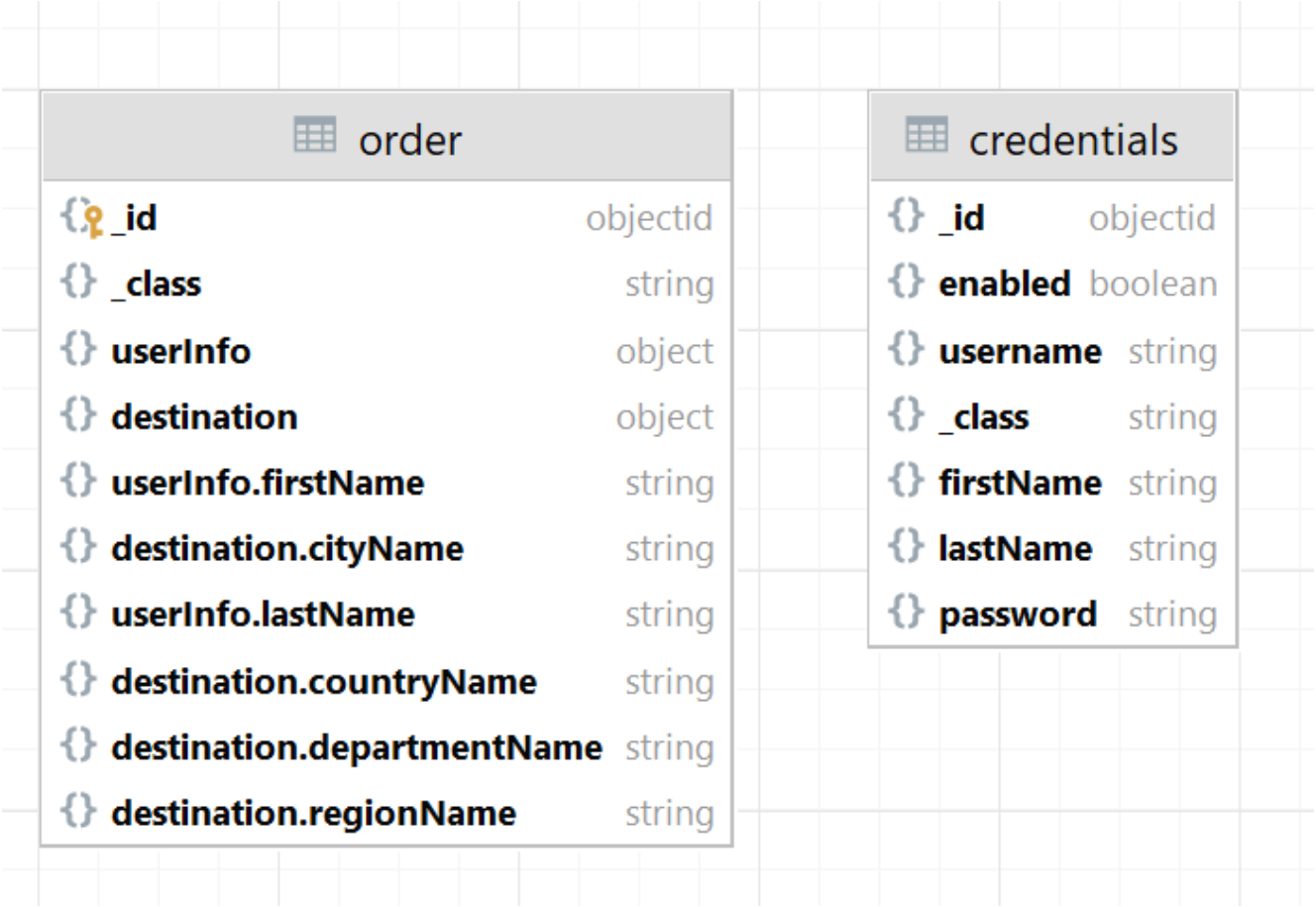

Рисунок Д.1 — UML-діаграма документів в MongoDB

ДОДАТОК Е

Лістинг коду програми

Сервіс entity

Клас Destination

```
package com.razakor.entities.mongo.order;

import lombok.Data;

@Data
public class Destination {
    private String countryName;
    private String departmentName;
    private String regionName;
    private String cityName;
}
```

Клас Order

```
package com.razakыor.entities.mongo.order;

import lombok.Data;
import org.bson.types.ObjectId;
import org.springframework.data.mongodb.core.mapping.Document;

@Data
@Document
public class Order {
    private ObjectId id;
    private Destination destination;
    private UserInfo userInfo;
}
```

Клас UserInfo

```
package com.razakor.entities.mongo.order;
```

```
import lombok.Data;

@Data
public class UserInfo {
    private String firstName;
    private String lastName;
}
```

Клас Credentials

```
package com.razakor.entities.mongo;

import lombok.Data;
import org.bson.types.ObjectId;
import org.springframework.data.mongodb.core.mapping.Document;

import javax.persistence.Id;

@Data
@Document
public class Credentials {
    @Id
    private ObjectId id;
    private String username;
    private String password;
    private String firstName;
    private String lastName;
    private boolean enabled;
}
```

Клас City

```
package com.razakor.entities.sql;

import lombok.*;
import org.hibernate.Hibernate;
```

```
import javax.persistence.*;
import java.util.Objects;

@Getter
@Setter
@ToString
@RequiredArgsConstructor
@Entity
public class City {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    @ManyToOne(targetEntity = Country.class, cascade = CascadeType.ALL)
    private Country country;
    @ManyToOne(targetEntity = Department.class, cascade = CascadeType.ALL)
    private Department department;
    @ManyToOne(targetEntity = Region.class, cascade = CascadeType.ALL)
    private Region region;

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || Hibernate.getClass(this) != Hibernate.getClass(o)) return false;
        City city = (City) o;
        return id != null && Objects.equals(id, city.id);
    }

    @Override
    public int hashCode() {
        return getClass().hashCode();
    }
}
```

Клас Country

```
package com.razakor.entities.sql;

import lombok.*;
import org.hibernate.Hibernate;

import javax.persistence.*;
import java.util.List;
import java.util.Objects;

@Getter
@Setter
@ToString
@RequiredArgsConstructor
@Entity
public class Country {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String shortname;
    private Integer code;
    @OneToMany(targetEntity = Department.class, cascade = CascadeType.ALL)
    @ToString.Exclude
    private List<Department> departments;

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || Hibernate.getClass(this) != Hibernate.getClass(o)) return false;
        Country country = (Country) o;
        return id != null && Objects.equals(id, country.id);
    }

    @Override
    public int hashCode() {
```

```
        return getClass().hashCode();
    }
}
```

Клас Department

```
package com.razakor.entities.sql;

import lombok.*;
import org.hibernate.Hibernate;

import javax.persistence.*;
import java.util.List;
import java.util.Objects;

@Getter
@Setter
@ToString
@RequiredArgsConstructor
@Entity
public class Department {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    @ManyToOne(targetEntity = Country.class, cascade = CascadeType.ALL)
    private Country country;
    @OneToMany(targetEntity = Region.class, cascade = CascadeType.ALL)
    @ToString.Exclude
    private List<Region> regions;

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || Hibernate.getClass(this) != Hibernate.getClass(o)) return false;
```

```
        Department that = (Department) o;
        return id != null && Objects.equals(id, that.id);
    }

    @Override
    public int hashCode() {
        return getClass().hashCode();
    }
}
```

Клас Region

```
package com.razakor.entities.sql;

import lombok.*;
import org.hibernate.Hibernate;

import javax.persistence.*;
import java.util.List;
import java.util.Objects;

@Getter
@Setter
@ToString
@RequiredArgsConstructor
@Entity
public class Region {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    @ManyToOne(targetEntity = Country.class, cascade = CascadeType.ALL)
    private Country country;
    @ManyToOne(targetEntity = Department.class, cascade = CascadeType.ALL)
    private Department department;
    @OneToMany(targetEntity = City.class, cascade = CascadeType.ALL)
```

```
    @ToString.Exclude
    private List<City> cities;

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || Hibernate.getClass(this) != Hibernate.getClass(o)) return false;
        Region region = (Region) o;
        return id != null && Objects.equals(id, region.id);
    }

    @Override
    public int hashCode() {
        return getClass().hashCode();
    }
}
```

Сервіс main

Клас ObjectIdDeserializer

```
package com.razakor.main.config.json;

import com.fasterxml.jackson.core.JsonParser;
import com.fasterxml.jackson.databind.DeserializationContext;
import com.fasterxml.jackson.databind.JsonDeserializer;
import com.fasterxml.jackson.databind.JsonNode;
import org.bson.types.ObjectId;

import java.io.IOException;

public class ObjectIdDeserializer extends JsonDeserializer<ObjectId> {

        @Override
        public ObjectId deserialize(JsonParser jp, DeserializationContext context) throws IOException
{
```

```
                JsonNode tree = jp.readValueAsTree();
                return new ObjectId(tree.asText());
        }
}
```

Клас ObjectIdSerializer

```
package com.razakor.main.config.json;

import com.fasterxml.jackson.core.JsonGenerator;
import com.fasterxml.jackson.databind.JsonSerializer;
import com.fasterxml.jackson.databind.SerializerProvider;
import org.bson.types.ObjectId;

import java.io.IOException;

public class ObjectIdSerializer extends JsonSerializer<ObjectId> {

   @Override
   public void serialize(ObjectId value, JsonGenerator jsonGenerator, SerializerProvider provider) throws IOException {
      jsonGenerator.writeString(value.toString());
   }
}
```

Клас MappingConfig

```
package com.razakor.main.config;

import com.fasterxml.jackson.core.JsonParser;
import com.fasterxml.jackson.core.Version;
import com.fasterxml.jackson.databind.DeserializationFeature;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.module.SimpleModule;
import com.fasterxml.jackson.datatype.jsr310.JavaTimeModule;

import com.razakor.main.config.json.ObjectIdDeserializer;
```

```
import com.razakor.main.config.json.ObjectIdSerializer;
import org.bson.types.ObjectId;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class MappingConfig {

    @Bean
    public ObjectMapper objectMapper() {
        ObjectMapper mapper = new ObjectMapper();
        mapper.configure(JsonParser.Feature.ALLOW_UNQUOTED_FIELD_NAMES, true);
        mapper.configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES, false);
        SimpleModule objectIdModule = new SimpleModule("CustomObjectIdMapper", new Version(1,
0, 0, null, null, null));
        objectIdModule.addSerializer(ObjectId.class, new ObjectIdSerializer());
        objectIdModule.addDeserializer(ObjectId.class, new ObjectIdDeserializer());
        mapper.registerModule(objectIdModule);
        mapper.registerModule(new JavaTimeModule());
        return mapper;
    }
}
```

Клас CredentialsController

```
package com.razakor.main.controller;

import com.razakor.entities.mongo.Credentials;
import com.razakor.main.dto.CreateCredentialsDto;
import com.razakor.main.service.CredentialsService;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
```

```
@RequestMapping("/api/credentials")
public class CredentialsController {

    private final CredentialsService credentialsService;

    public CredentialsController(CredentialsService credentialsService) {
        this.credentialsService = credentialsService;
    }

    @PostMapping("/create")
    Credentials create(@RequestBody CreateCredentialsDto dto) {
        Credentials credentials = credentialsService.create(dto);
        return credentials;
    }
}
```

Клас OrderController

```
package com.razakor.main.controller;

import com.razakor.entities.mongo.order.Order;
import com.razakor.main.dto.CreateOrderDto;
import com.razakor.main.service.OrderService;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api/order")
public class OrderController {

    private final OrderService orderService;

    public OrderController(OrderService orderService) {
        this.orderService = orderService;
```

```
    }

    @PostMapping("/create")
    public Order create(@RequestBody CreateOrderDto dto) {
        return orderService.create(dto);
    }
}
```

Клас CreateCredentialsDto

```
package com.razakor.main.dto;

import lombok.Data;

@Data
public class CreateCredentialsDto {
    private String username;
    private String password;
    private String firstName;
    private String lastName;
}
```

Клас CreateOrderDto

```
package com.razakor.main.dto;

import lombok.Data;
import org.bson.types.ObjectId;

@Data
public class CreateOrderDto {
    private ObjectId userId;
    private Long cityId;
}
```

Клас CredentialsClient

```
package com.razakor.main.feign.mongo;
```

```
import com.razakor.entities.mongo.Credentials;
import org.bson.types.ObjectId;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;

@FeignClient(name = "credentials", url = "${mongo.url}/credentials/")
public interface CredentialsClient {

    @PostMapping
    Credentials save(@RequestBody Credentials credentials);

    @GetMapping
    Credentials getAll();

    @GetMapping("/{id}")
    Credentials getById(@PathVariable ObjectId id);

}
```

Клас OrderClient

```
package com.razakor.main.feign.mongo;

import com.razakor.entities.mongo.order.Order;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;

import java.util.List;
```

```
@FeignClient(name = "order", url = "${mongo.url}/order/")
public interface OrderClient {

    @PostMapping
    Order save(@RequestBody Order order);

    @GetMapping
    List<Order> getAll();

    @GetMapping("/{id}")
    Order getById(@PathVariable String id);
}
```

Клас CityClient

```
package com.razakor.main.feign.sql;

import com.razakor.entities.sql.City;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestParam;

@FeignClient(name = "city", url = "${sql.url}/city/")
public interface CityClient {

    @GetMapping("/{id}")
    City getCityById(@PathVariable Long id, @RequestParam(required = false) String projection);

}
```

Клас CredentialsServiceImpl

```
package com.razakor.main.service.impl;

import com.razakor.entities.mongo.Credentials;
import com.razakor.main.dto.CreateCredentialsDto;
```

```
import com.razakor.main.feign.mongo.CredentialsClient;
import com.razakor.main.service.CredentialsService;
import org.springframework.stereotype.Service;

@Service
public class CredentialsServiceImpl implements CredentialsService {

    private final CredentialsClient credentialsClient;

    public CredentialsServiceImpl(CredentialsClient credentialsClient) {
        this.credentialsClient = credentialsClient;
    }


    @Override
    public Credentials create(CreateCredentialsDto dto) {
        Credentials credentials = new Credentials();
        credentials.setUsername(dto.getUsername());
        credentials.setPassword(dto.getPassword());
        credentials.setFirstName(dto.getFirstName());
        credentials.setLastName(dto.getLastName());
        credentials.setEnabled(true);

        return credentialsClient.save(credentials);
    }
}
```

Клас OrderServiceImpl

```
package com.razakor.main.service.impl;

import com.razakor.entities.mongo.Credentials;
import com.razakor.entities.mongo.order.Destination;
import com.razakor.entities.mongo.order.Order;
import com.razakor.entities.mongo.order.UserInfo;
import com.razakor.entities.sql.City;
```

```
import com.razakor.main.dto.CreateOrderDto;
import com.razakor.main.feign.mongo.CredentialsClient;
import com.razakor.main.feign.mongo.OrderClient;
import com.razakor.main.feign.sql.CityClient;
import com.razakor.main.service.OrderService;
import org.springframework.stereotype.Service;

@Service
public class OrderServiceImpl implements OrderService {

    private final OrderClient orderClient;
    private final CredentialsClient credentialsClient;
    private final CityClient cityClient;

    public OrderServiceImpl(OrderClient orderClient,
                            CredentialsClient credentialsClient,
                            CityClient cityClient) {
        this.orderClient = orderClient;
        this.credentialsClient = credentialsClient;
        this.cityClient = cityClient;
    }

    @Override
    public Order create(CreateOrderDto dto) {
        Order order = new Order();

        Credentials credentials = credentialsClient.getById(dto.getUserId());
        if (!credentials.isEnabled()) throw new RuntimeException();
        UserInfo userInfo = new UserInfo();
        userInfo.setFirstName(credentials.getFirstName());
        userInfo.setLastName(credentials.getLastName());
        order.setUserInfo(userInfo);

        City city = cityClient.getCityById(dto.getCityId(), "withAll");
        Destination destination = new Destination();
```

```
        destination.setCountryName(city.getCountry().getName());
        destination.setDepartmentName(city.getDepartment().getName());
        destination.setRegionName(city.getRegion().getName());
        destination.setCityName(city.getName());
        order.setDestination(destination);

        return orderClient.save(order);
    }
}
```

Клас CredentialsService

```
package com.razakor.main.service;

import com.razakor.entities.mongo.Credentials;
import com.razakor.main.dto.CreateCredentialsDto;

public interface CredentialsService {

    Credentials create(CreateCredentialsDto dto);
}
```

Клас OrderService

```
package com.razakor.main.service;

import com.razakor.entities.mongo.order.Order;
import com.razakor.main.dto.CreateOrderDto;

public interface OrderService {

    Order create(CreateOrderDto dto);
}
```

Клас MainApplication

```
package com.razakor.main;
```

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.autoconfigure.data.mongo.MongoDataAutoConfiguration;
import org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration;
import
org.springframework.boot.autoconfigure.jdbc.DataSourceTransactionManagerAutoConfiguration;
import org.springframework.boot.autoconfigure.mongo.MongoAutoConfiguration;
import org.springframework.boot.autoconfigure.orm.jpa.HibernateJpaAutoConfiguration;
import org.springframework.cloud.openfeign.EnableFeignClients;

@EnableFeignClients
@SpringBootApplication(exclude = {
        MongoAutoConfiguration.class,
        MongoDataAutoConfiguration.class,
        DataSourceAutoConfiguration.class,
        DataSourceTransactionManagerAutoConfiguration.class,
        HibernateJpaAutoConfiguration.class
})
public class MainApplication {

    public static void main(String[] args) {
        SpringApplication.run(MainApplication.class, args);
    }

}
```

Файл application.properties

```
server.port=8080

mongo.url=http://localhost:8081
sql.url=http://localhost:8082
```

Сервіс mongo

Клас ObjectIdDeserializer

```
package com.razakor.mongo.config.json;
```

```
import com.fasterxml.jackson.core.JsonParser;
import com.fasterxml.jackson.databind.DeserializationContext;
import com.fasterxml.jackson.databind.JsonDeserializer;
import com.fasterxml.jackson.databind.JsonNode;
import org.bson.types.ObjectId;

import java.io.IOException;

public class ObjectIdDeserializer extends JsonDeserializer<ObjectId> {

        @Override
        public ObjectId deserialize(JsonParser jp, DeserializationContext context) throws IOException
{
                JsonNode tree = jp.readValueAsTree();
                return new ObjectId(tree.asText());
        }
}
```

Клас ObjectIdSerializer

```
package com.razakor.mongo.config.json;

import com.fasterxml.jackson.core.JsonGenerator;
import com.fasterxml.jackson.databind.JsonSerializer;
import com.fasterxml.jackson.databind.SerializerProvider;
import org.bson.types.ObjectId;

import java.io.IOException;

public class ObjectIdSerializer extends JsonSerializer<ObjectId> {

   @Override
   public void serialize(ObjectId value, JsonGenerator jsonGenerator, SerializerProvider provider)
throws IOException {
      jsonGenerator.writeString(value.toString());
```

```
    }
}
```

Клас CustomRestMvcConfiguration

```
package com.razakor.mongo.config;

import com.razakor.entities.mongo.Credentials;
import com.razakor.entities.mongo.order.Order;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.rest.core.config.RepositoryRestConfiguration;
import org.springframework.data.rest.webmvc.config.RepositoryRestConfigurer;
import org.springframework.web.servlet.config.annotation.CorsRegistry;

@Configuration
public class CustomRestMvcConfiguration {

    @Bean
    public RepositoryRestConfigurer repositoryRestConfigurer() {

        return new RepositoryRestConfigurer() {
            @Override
            public void configureRepositoryRestConfiguration(RepositoryRestConfiguration config,
CorsRegistry cors) {
                config.exposeIdsFor(Credentials.class);
                config.exposeIdsFor(Order.class);
            }
        };
    }
}
```

Клас MappingConfig

```
package com.razakor.mongo.config;

import com.fasterxml.jackson.core.JsonParser;
```

```
import com.fasterxml.jackson.core.Version;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.module.SimpleModule;
import com.fasterxml.jackson.datatype.jsr310.JavaTimeModule;
import com.razakor.mongo.config.json.ObjectIdDeserializer;
import com.razakor.mongo.config.json.ObjectIdSerializer;
import org.bson.types.ObjectId;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class MappingConfig {

    @Bean
    public ObjectMapper objectMapper() {
        ObjectMapper mapper = new ObjectMapper();
        mapper.configure(JsonParser.Feature.ALLOW_UNQUOTED_FIELD_NAMES, true);
        SimpleModule objectIdModule = new SimpleModule("CustomObjectIdMapper", new Version(1,
0, 0, null, null, null));
        objectIdModule.addSerializer(ObjectId.class, new ObjectIdSerializer());
        objectIdModule.addDeserializer(ObjectId.class, new ObjectIdDeserializer());
        mapper.registerModule(objectIdModule);
        mapper.registerModule(new JavaTimeModule());
        return mapper;
    }
}
```

Клас CredentialsRepository

```
package com.razakor.mongo.repository;

import com.razakor.entities.mongo.Credentials;
import org.bson.types.ObjectId;
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import org.springframework.stereotype.Repository;
```

```
@Repository
@RepositoryRestResource(collectionResourceRel = "credentials", path = "credentials")
public interface CredentialsRepository extends MongoRepository<Credentials, ObjectId> {
}
```

Клас OrderRepository

```
package com.razakor.mongo.repository;

import com.razakor.entities.mongo.order.Order;
import org.bson.types.ObjectId;
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import org.springframework.stereotype.Repository;

@Repository
@RepositoryRestResource(collectionResourceRel = "order", path = "order")
public interface OrderRepository extends MongoRepository<Order, ObjectId> {
}
```

Клас MongoApplication

```
package com.razakor.mongo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MongoApplication {

    public static void main(String[] args) {
        SpringApplication.run(MongoApplication.class, args);
    }

}
```

Файл application.properties

```
server.port=8081

spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
spring.data.mongodb.database=Master
```

Сервіс sql

Клас CustomRestMvcConfiguration

```
package com.razakor.sql.config;

import com.razakor.entities.mongo.Credentials;
import com.razakor.entities.mongo.order.Order;
import com.razakor.entities.sql.City;
import com.razakor.entities.sql.Country;
import com.razakor.entities.sql.Department;
import com.razakor.entities.sql.Region;
import com.razakor.sql.projection.CityWithAllProjection;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.rest.core.config.RepositoryRestConfiguration;
import org.springframework.data.rest.webmvc.config.RepositoryRestConfigurer;
import org.springframework.web.servlet.config.annotation.CorsRegistry;

@Configuration
public class CustomRestMvcConfiguration {

    @Bean
    public RepositoryRestConfigurer repositoryRestConfigurer() {

        return new RepositoryRestConfigurer() {
            @Override
            public void configureRepositoryRestConfiguration(RepositoryRestConfiguration config,
CorsRegistry cors) {
                config.exposeIdsFor(Country.class);
```

```
            config.exposeIdsFor(Department.class);
            config.exposeIdsFor(Region.class);
            config.exposeIdsFor(City.class);
            config.getProjectionConfiguration()
                  .addProjection(CityWithAllProjection.class);
         }
      };
   }
}
```

Клас CityWithAllProjection

```
package com.razakor.sql.projection;

import com.razakor.entities.sql.City;
import com.razakor.entities.sql.Country;
import com.razakor.entities.sql.Department;
import com.razakor.entities.sql.Region;
import org.springframework.data.rest.core.config.Projection;

@Projection(name = "withAll", types = { City.class })
public interface CityWithAllProjection {
   Long getId();
   String getName();
   Country getCountry();
   Department getDepartment();
   Region getRegion();
}
```

Клас CityRepository

```
package com.razakor.sql.repository;

import com.razakor.entities.sql.City;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import org.springframework.stereotype.Repository;
```

```
@Repository
@RepositoryRestResource(collectionResourceRel = "city", path = "city")
public interface CityRepository extends JpaRepository<City, Long> {
}
```

Клас CountryRepository

```
package com.razakor.sql.repository;

import com.razakor.entities.sql.Country;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import org.springframework.stereotype.Repository;

@Repository
@RepositoryRestResource(collectionResourceRel = "country", path = "country")
public interface CountryRepository extends JpaRepository<Country, Long> {
}
```

Клас DepartmentRepository

```
package com.razakor.sql.repository;

import com.razakor.entities.sql.Department;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import org.springframework.stereotype.Repository;

@Repository
@RepositoryRestResource(collectionResourceRel = "department", path = "department")
public interface DepartmentRepository extends JpaRepository<Department, Long> {
}
```

Клас RegionRepository

```
package com.razakor.sql.repository;
```

```
import com.razakor.entities.sql.Region;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import org.springframework.stereotype.Repository;

@Repository
@RepositoryRestResource(collectionResourceRel = "region", path = "region")
public interface RegionRepository extends JpaRepository<Region, Long> {
}
```

Клас SqlApplication

```
package com.razakor.sql;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.autoconfigure.domain.EntityScan;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

@EntityScan(basePackages = "com.razakor.entities.sql.**")
@EnableJpaRepositories
@SpringBootApplication
public class SqlApplication {

    public static void main(String[] args) {
        SpringApplication.run(SqlApplication.class, args);
    }
}
```

Файл application.properties

```
server.port=8082

spring.datasource.url=jdbc:mysql://localhost:3306/master?cachePrepStmts=true&useServerPrepStmts
=true&rewriteBatchedStatements=true
spring.datasource.username=root
spring.datasource.password=11235813
```

```
spring.jpa.database-platform=org.hibernate.dialect.MySQL5Dialect
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.jdbc.batch_size=30
```

ДОДАТОК Ж

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: Високопродуктивна система управління потоками даних на основі MySQLi MongoDB

Тип роботи магістерська кваліфікаційна робота

(кваліфікаційна роботи, курсовий проект (робота), реферат, аналітичний огляд, інше (зазначити))

Підрозділ кафедра обчислювальної техніки

(кафедра, факультет (інститут), навчальна група)

Науковий керівник к.т.н. доц. Ткаченко О. М.

(прізвище, ініціали, посада)

Показники звіту подібності

Plagiat.pl (StrikePlagiarism)		Unicheck	
КП1			
КП2		Оригінальність	89,8
Тривога/Білі знаки	/	Схожість	10,2

Аналіз звіту подібності (відмінити подібне)

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності і відсутності самостійності її автора. Робот направити на доопрацювання.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений(-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор _____ Шевчук М. О.

(підпис) (прізвище, ініціали)

Опис прийнятого рішення

Ступінь оригінальності роботи відповідає вимогам, що висуваються до МКР

Особа, відповідальна за перевірку_____ Захарченко С. М.

(підпис) (прізвище, ініціали)

Експерт _____ _____

(за потреби) (підпис) (прізвище, ініціали)