Міністерство освіти і науки України

Національний технічний університет України „КПІ”

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки

інформації та управління

ЗВІТ

з лабораторної роботи № 5

дисципліни

“ТЕХНОЛОГІЇ ПАРАЛЕЛЬНОГО ПРОГРАМУВАННЯ В УМОВАХ

ВЕЛИКИХ ДАНИХ”

на тему:

„Контейнеризація рішення”

Виконали:

студенти групи IT-01мн

Панасюк Станіслав

Лесогорський Кирило

Перевірив:

доц. Жереб К. А.

Зміст

[1. Постановка задачі 3](#__RefHeading___Toc305_686243043)

[2. Обрані інструменти 3](#__RefHeading___Toc307_686243043)

[3. Опис контейнеризації 4](#__RefHeading___Toc311_686243043)

[4. Опис додаткового функціоналу 5](#__RefHeading___Toc970_145050628)

[5. Отримані результати 6](#__RefHeading___Toc313_686243043)

[6. Висновки 6](#__RefHeading___Toc321_686243043)

# **1. Постановка задачі**

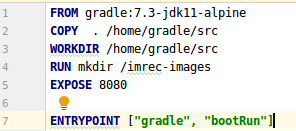
Для обраної задачі необхідно реалізувати послідовну (однопоточну) реалізацію, а також мультипоточну реалізацію зі спільною пам’яттю. У якості задачі було обрано побудову системи пошуку схожих зображень. У ядрі системи лежатиме використання D-hash для знаходження хешу зображення. D-hash дозволяє точно та швидко шукати схожі зображення. Він стійкий до скейлінгу зображеня, але погано справляються з обрізаними та повернутими під кутом зображеннями. Тому цю техніку аугментовано за допомогою наступного прийому: при завантаженні зображення воно буде аугментовано за допомогою декількох філтрів, при цьому для кожного фільтру буде згенеровано хеш і збережено у базу даних. При пошуку зображення буде використовуватись оператор XOR для знаходження зображень зі схожими хешами. Метою цієї лабораторної роботи є контейнеризація додатку та доробка функціоналу пошуку вже існуючих зображень.

# 2. Обрані інструменти

Окрім вже використанних технологій, ми скористаємось Docker та Docker Compose. З такою конфігурацією додаток буде легко розгорнути у клауд провайдері, наприклад AWS з сервісом Elastic Beanstalk. Також буде використано специфіку JPA – DTO Projection для мапінгу результатів нативного запиту у базу даних у інтерфейс для подальшої обробки у додатку.

# 3. Опис контейнеризації

Для контейнеризації було створено докер зображення для java сервісу та docker compose файл для оркестрацыъ БД, брокеру черги та двох сервісів. Докер зображення базується на зображені з Gradle версії 7.3, яке в свою чергу базується на OpenJDK 11 та Alpine linux. Файл можно переглянути у [репозиторії](https://github.com/klesogor/imrec/blob/feature/microservice-approach/injestor/Dockerfile), але для наочності додамо скріншот:

Рис. 3.1 – Dockerfile

Одразу видно що цей файл неоптимальний і його можна покращити. Для цього слід додати .dockerignore для запобігання зайвого копіювання та використати multi-stage build, де на першому етапі ми компілюємо сервіс у jar файл, а на другому етапі використовуємо отриманий файл на більш легковісному базовому зображені без інструментів необхідних для компіляції сервісу.

Далі слід розглянути [docker-compose](https://github.com/klesogor/imrec/blob/feature/microservice-approach/injestor/docker-compose.yml). У ньому ми визначаємо 4 сервіси: інжестор, воркер, брокер черги та СУБД.

Для СУБД ми використовуємо Postgres 14. До ньогу монтується вольюм для персистетного збереження даних між рестартами контейнера. Також ми використовуємо змінну оточення щоб вказати адміністраторський пароль та дерективу ports для того, щоб отримати доступ до СУБД ззовні docker.

Для брокера черги ми використовуємо RabbitMQ. До нього також монтуєтся вольюм для персистентного збереження даних. Для зовньошнього доступу відкривається порт 15672, на якому розгортається менеджер брокеру для налаштувань.

Далі розгортається два сервіси. Вони використовують однакове базове зображення докер та до них монтується спільний вольюм для завантаження та обробки зображень. Режим запуску контролюється за допомогою змінної оточення SPRING\_ACTIVE\_PROFILE, що дозволяє вказати у якому профілі слід запускати додаток.

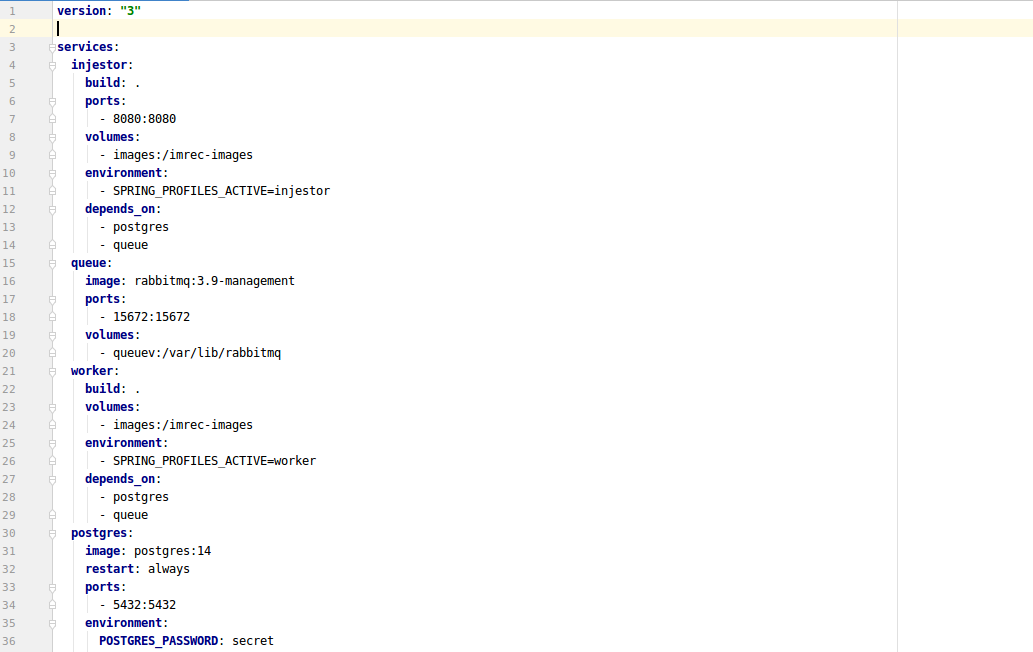
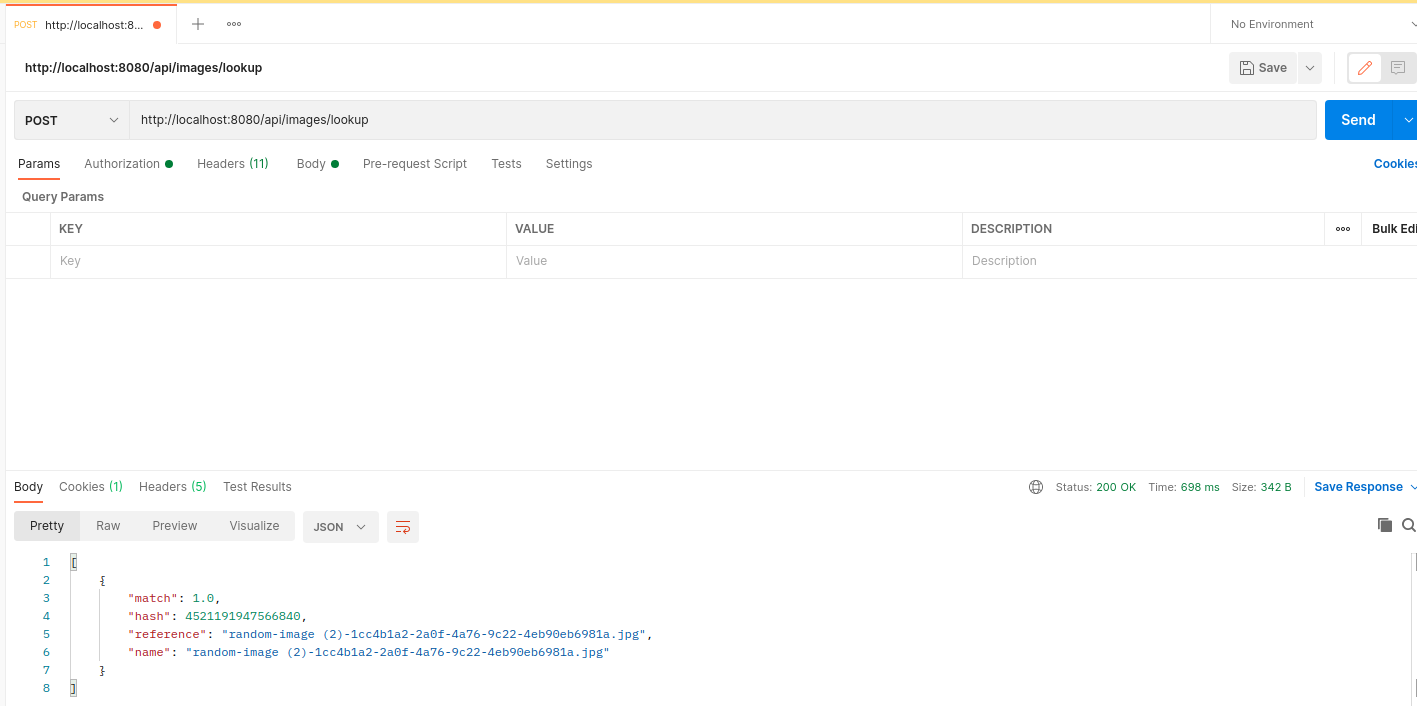


Рис. 3.2 – docker-compose

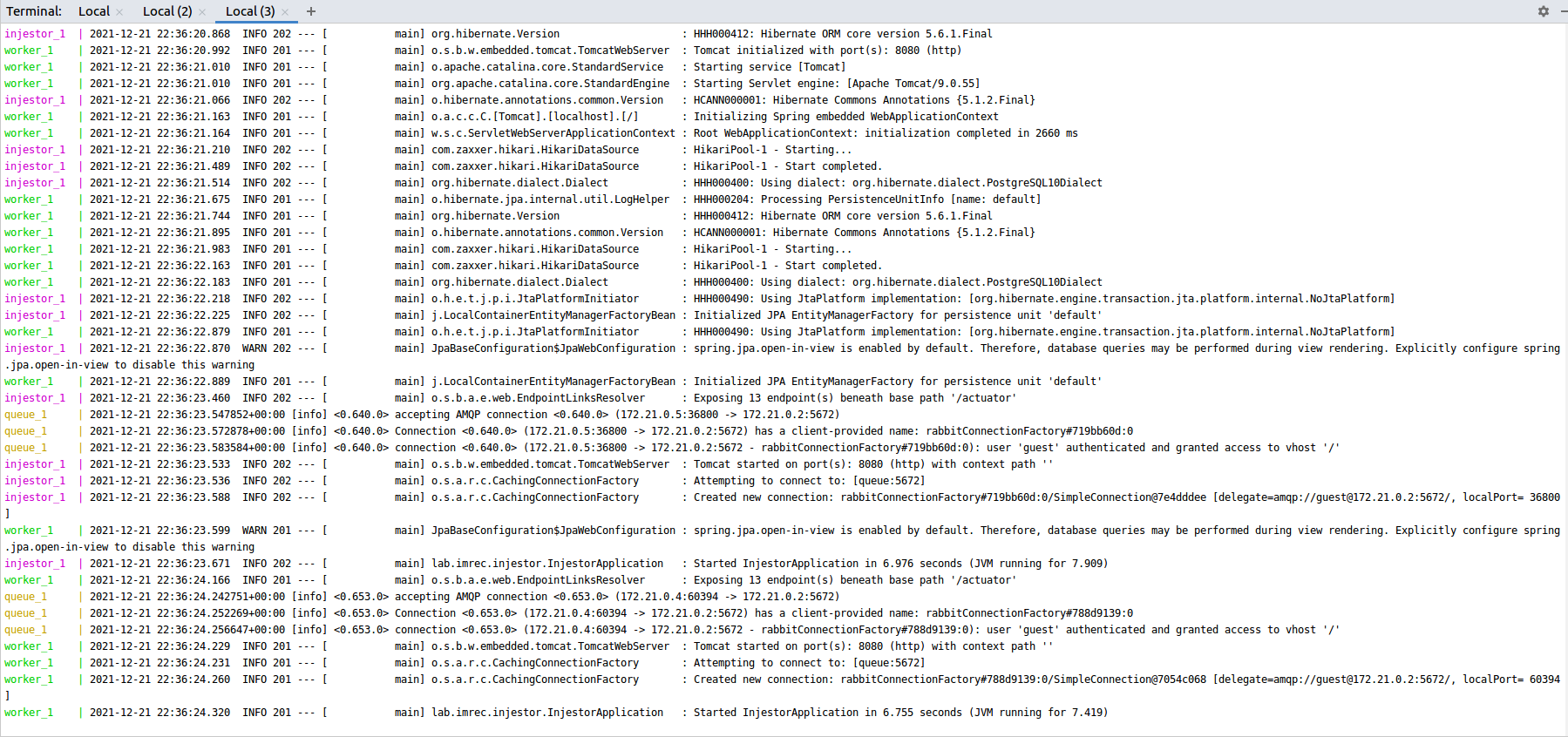
# **4. Опис додаткового функціоналу**

У ході виконання роботи було реалізовано додатковий функціонал пошуку зображень за хешем. Для цього було створено додатковий [запит у репозиторії](https://github.com/klesogor/imrec/blob/feature/microservice-approach/injestor/src/main/java/lab/imrec/injestor/images/repository/ImageRepository.java" \l "L16), відповідну [проекцію](https://github.com/klesogor/imrec/blob/feature/microservice-approach/injestor/src/main/java/lab/imrec/injestor/images/dto/ImageLookupResult.java) даних, [сервіс](https://github.com/klesogor/imrec/blob/feature/microservice-approach/injestor/src/main/java/lab/imrec/injestor/images/service/ImageLookupService.java) та метод [контроллера](https://github.com/klesogor/imrec/blob/feature/microservice-approach/injestor/src/main/java/lab/imrec/injestor/images/controllers/ImageUploadController.java" \l "L46).

Рис. 4.1 – Пошук схожих зображень

# 5. Отримані результати

В результаті виконання лабораторної роботи додаток було контейнерізовано та додано новий функціонал.

Рис. 5.1 – Приклад роботи контейнеризованого додатку

# 6. Висновки

У ході лабораторної роботи було розвинуто архітектуру попередніх лабораторних робіт, додано додатковий функціонал та контейнерізацію до додатку.