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 № 2

дисципліни

“ТЕХНОЛОГІЇ ПАРАЛЕЛЬНОГО ПРОГРАМУВАННЯ В УМОВАХ

ВЕЛИКИХ ДАНИХ”

на тему:

„Паралельні обчислення в моделі з декількома процесами”

Виконали:

студенти групи IT-01мн

Панасюк Станіслав

Лесогорський Кирило

Перевірив:

доц. Жереб К. А.

Київ – 2021

Зміст

[1. Постановка задачі 3](#__RefHeading___Toc305_686243043)

[2. Обрані інструменти 3](#__RefHeading___Toc307_686243043)

[3. Високорівнева архітектура системи 4](#__RefHeading___Toc309_686243043)

[4. Опис роботи програмного забезпечення 5](#__RefHeading___Toc311_686243043)

[5. Отримані результати 5](#__RefHeading___Toc313_686243043)

[5.1 Закон Амдала при збільшені кількості воркерів: 7](#__RefHeading___Toc315_686243043)

[5.2 Закон Амдала при збільшені кількості потоків у воркерах: 7](#__RefHeading___Toc317_686243043)

[5.3 Результати для інших оптимальних варіантів 7](#__RefHeading___Toc319_686243043)

[6. Висновки 8](#__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.

# 2. Обрані інструменти

Для виконання перших двох частин лабораторної роботи буде використано стандартні інструменти Java. З самого початку буде використано фреймворк *Spring* для створення веб-інтерфейсу у майбутньому. *Spring Data* буде використано для доступу до бази даних. *Lombok* буде використано для зменшення кількості бойлерплейту. *JUnit* буде використано для тестування. Вбудована бібліотека *AWT* буде використана для роботи з зображеннями. Для роботи з чергою буде використано *RabbitMQ* та відповідно інтеграцію зі Spring.

# 3. Високорівнева архітектура системи

На високому рівні система виглядає наступним чином:

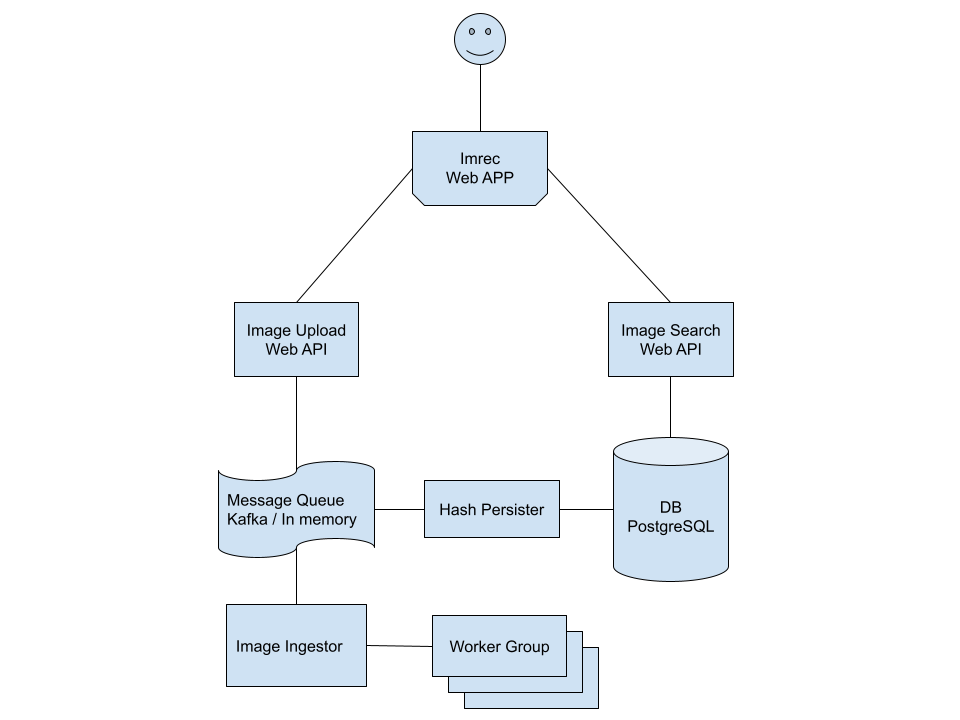


Рис. 3.1 – Високорівнева архітектура

1. **Imrec Web APP –** браузерний додаток, який надає інтерфейс для користувача. Підтримується додавання нового зображення і пошук у переліку уснуючих.
2. **Image Upload Web API** – веб сервер для завантаження зображень. Виконує валідацію запиту, зберігає файл у персистентне сховище і передає його на подальшу обробку у Image Ingestor через Kafka або InMemory чергу.
3. **Image Ingestor** – оркестратор процесу обробки зображень. Отримує повідомлення з черги і передає йх на обробку воркерам. Після обчислення хешу, передає інформацію назад у чергу
4. **Worker** – обчислює необхідні хеші для зображення.
5. **Hash Persister** – зберігає отриманні хеші у персистентне сховище.
6. **Image Search Web API** – веб сервер, який выдповідає за пошук серед вже існуючих зображень.

Така архітектура має низьку зчепність і дуже модульна. Слід зазначити, шо кожен воркер буде обчислювати хеш для зображеня з фільтрами, це дозволяє знизити затримку при обратній збірці результатів при розподілені. Також це дозволить винести усю роботу, яка потребує багато обчислювальних ресурсів на окремі машини, дозволяючи інджестору працювати у однопоточному режимі при умові використання асинхронних інтерфейсів вводу-виводу. Також такий підхід дозволить знизити навантаження на мережу, коли декілька воркерів завантажують досить великі фотографії (більше 10 мегабайт) з файлового сховища.

У порівнянні з попередньою лабораторною роботю тепер інджестор та воркер и комінікують за допомогою черги повідомлень. У якості черги було використано RabbitMQ. Також обов’язки інджестора та веб АПІ було поєднано, оскільки тоді робота виходить занадто гранулярною. Для простоти(спрощення у рамках лабораторної роботи) Hash Persister є частиною WebAPI.

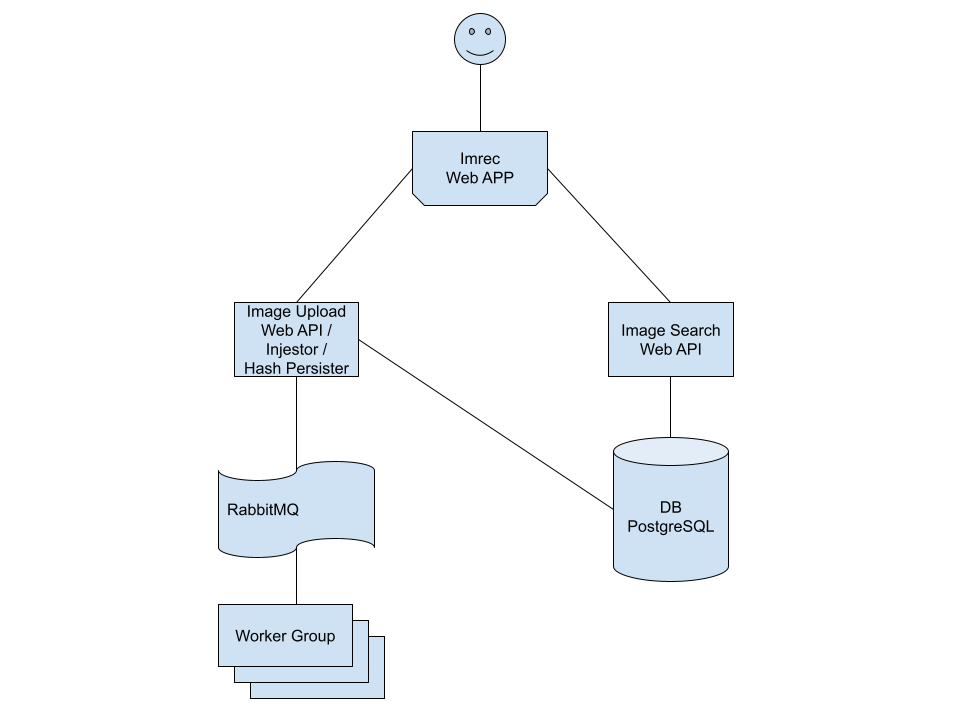


Рис. 3.2 – Оновлена схема архітектури

# 4. Опис роботи програмного забезпечення

Багато коду базується на результатах попередніх робіт, тому ми розглянемо лише зміни, які додались у зв’язку з багатопроцесною реалізацією.

По-перше, з’явився [контролер](https://github.com/klesogor/imrec/blob/feature/microservice-approach/injestor/src/main/java/lab/imrec/injestor/images/controllers/ImageUploadController.java) та [сервіс](https://github.com/klesogor/imrec/blob/feature/microservice-approach/injestor/src/main/java/lab/imrec/injestor/images/service/ImageUploadService.java) для завантаження зображень та його додавання у чергу обробки. У цьому сервісі відбувається валідація зображень, збереження у [файлове сховище](https://github.com/klesogor/imrec/blob/feature/microservice-approach/injestor/src/main/java/lab/imrec/injestor/common/LocalFsStorage.java) та делегація роботи до воркерів. У якості основи воркеру було обрано [однопоточну реалізацію](https://github.com/klesogor/imrec/blob/feature/microservice-approach/injestor/src/main/java/lab/imrec/injestor/processor/SingleThreadedWorker.java) з попередньої роботи, щоб запобігти неефективному використанню ресурсів при синхронізації. Час обробки одного зображення є достатнім і не вимагає багатопоточної реалізації. Паралелізм може бути доданий на рівні [лістенеру черги](https://github.com/klesogor/imrec/blob/feature/microservice-approach/injestor/src/main/java/lab/imrec/injestor/processor/WorkerQueueAdaptor.java). Також було додано необхідну для зберігання результатів роботи інфраструктуру: [репозиторій](https://github.com/klesogor/imrec/blob/feature/microservice-approach/injestor/src/main/java/lab/imrec/injestor/images/repository/ImageRepository.java), модель [зображення](https://github.com/klesogor/imrec/blob/feature/microservice-approach/injestor/src/main/java/lab/imrec/injestor/images/entity/Image.java) та [хешу](https://github.com/klesogor/imrec/blob/feature/microservice-approach/injestor/src/main/java/lab/imrec/injestor/images/entity/ImageFeature.java), [лістенер](https://github.com/klesogor/imrec/blob/feature/microservice-approach/injestor/src/main/java/lab/imrec/injestor/images/listeners/ImageSaver.java) черги результатів.

# 5. Отримані результати

В результаті виконання роботи було отримано наступні результати з процесінгу:

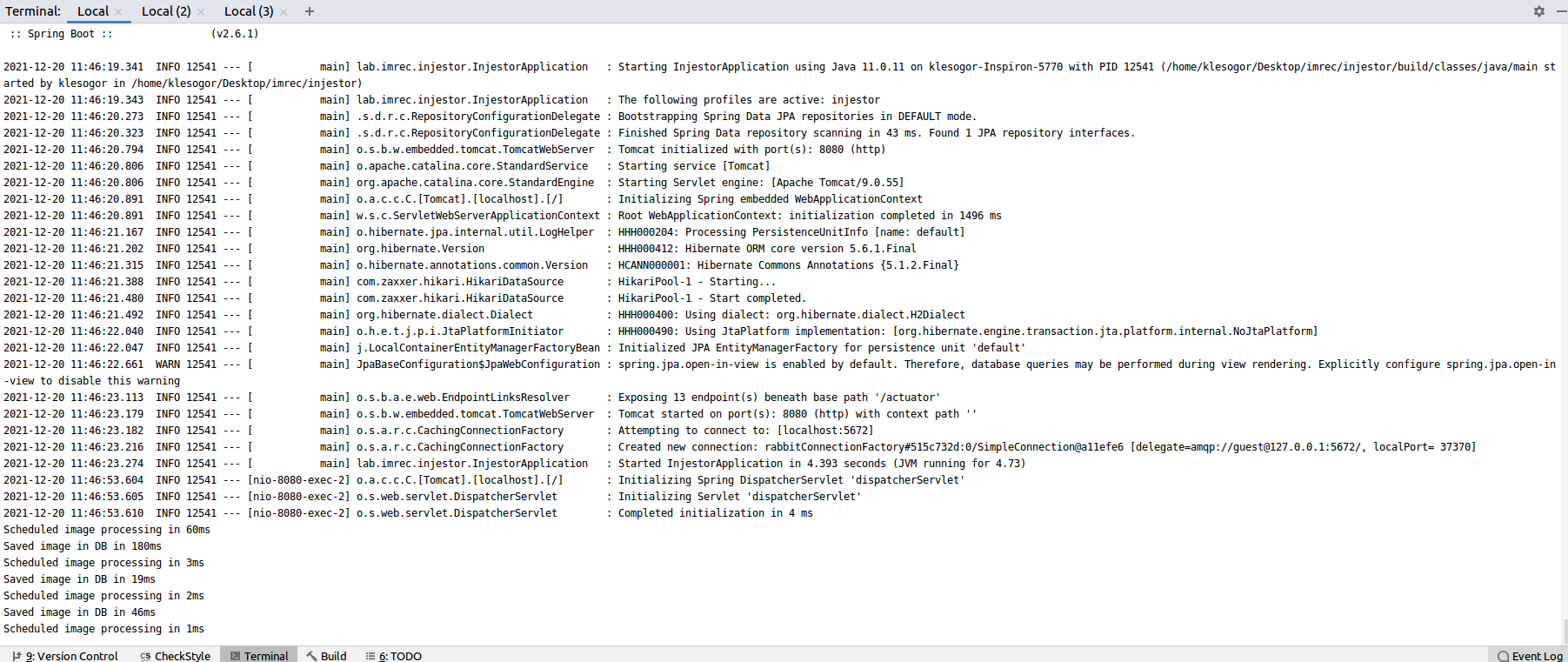
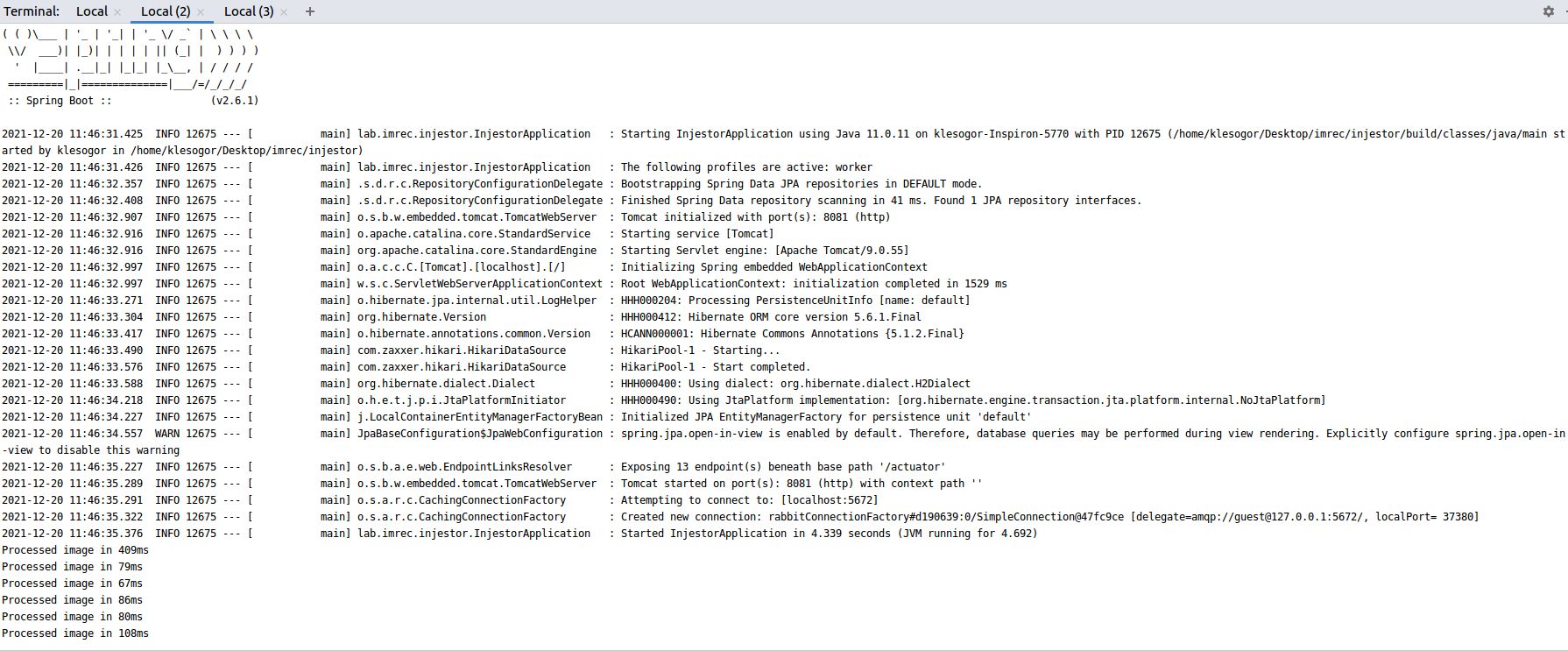
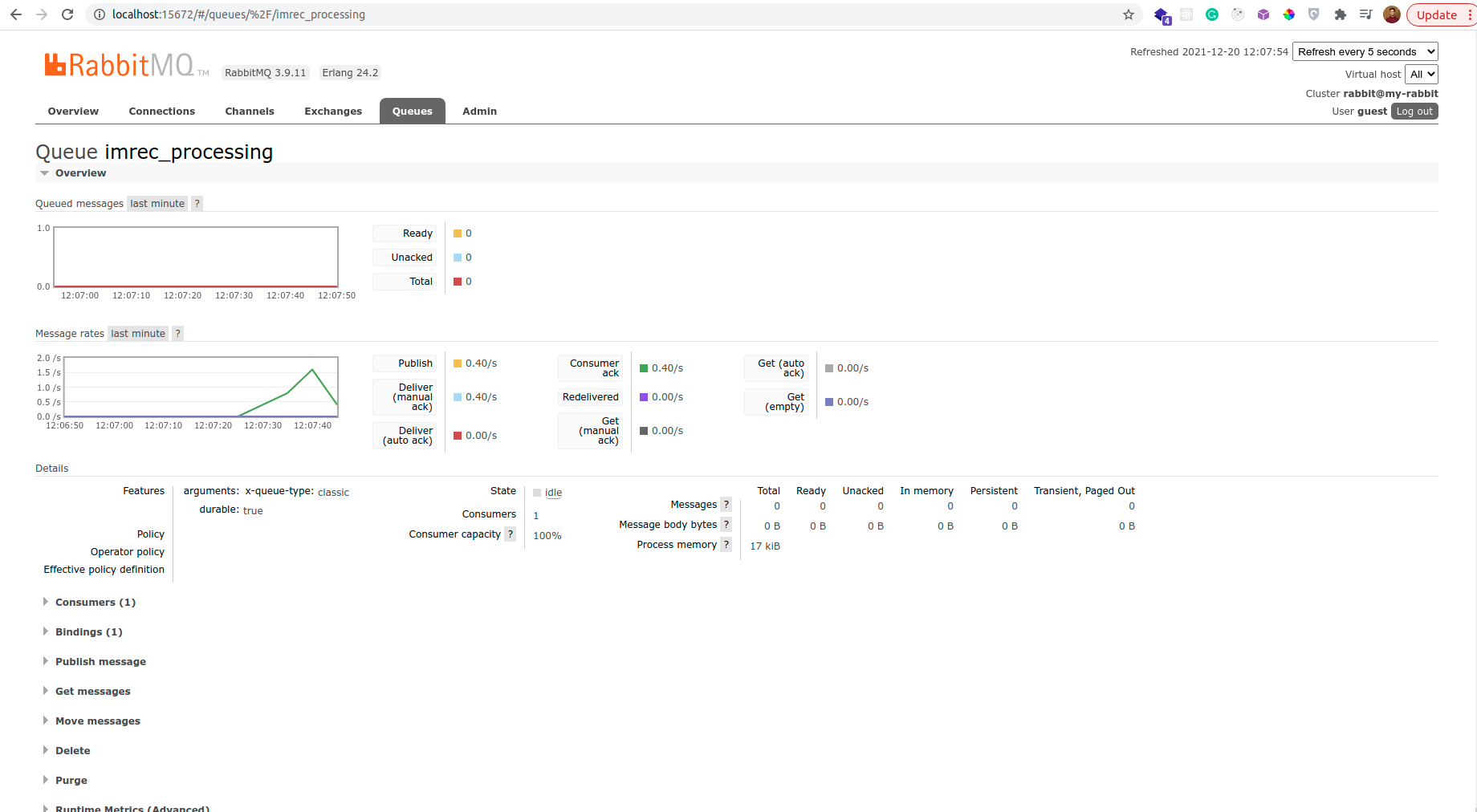
****

Рис. 5.1 – Результат процесінгу зображення у черзі та збереження у БД

****Рис. 5.2 – Результат обробки зображень

****Рис. 5.3 – Навантаження на чергу

# 6. Висновки

У ході лабораторної роботи було розвинуто архітектуру попередньої лабораторної роботи, створено мультипроцесну реалізацію.