Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки Кафедра

інформаційних систем та технологій

ЛАБОРАТОРНА РОБОТА №2

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Технології розроблення програмного забезпечення»

Виконав
Перевірив:

студент групи ІА–33:
Мягкий М.Ю.

Якименко Ярослав

Київ 2025

Зміст

Вступ

У цій лабораторній роботі розглядаються аспекти архітектурного проєктування розподілених програмних систем на прикладі веб-сервісу "Онлайн-радіостанція". Основна увага приділяється візуалізації фізичного розміщення компонентів, їхній модульній структурі та динаміці взаємодії за допомогою уніфікованої мови моделювання UML. Окрім того, робота охоплює практичну реалізацію спроєктованої архітектури з використанням сучасних технологій контейнеризації (Docker) та асинхронної обробки даних через черги повідомлень (RabbitMQ), що є стандартом для сучасних високонавантажених систем.

Мета роботи:

Метою роботи є набуття практичних навичок у проєктуванні архітектури складного програмного забезпечення з використанням діаграм розгортання, компонентів та послідовностей. Також метою є вивчення принципів побудови мікросервісної або модульної архітектури з використанням Docker-контейнерів.

Завдання роботи:

1. Опрацювати теоретичний матеріал щодо побудови UML-діаграм розгортання, компонентів та послідовностей.
2. Спроєктувати діаграму розгортання (Deployment Diagram), що демонструє фізичну архітектуру сервісу. Схема повинна включати клієнтську частину, сервер додатків (Backend), базу даних (MongoDB), брокер повідомлень (RabbitMQ) та середовище контейнеризації.
3. Розробити діаграму компонентів (Component Diagram), яка ілюструє логічну розбивку системи на модулі (наприклад, Auth Module, Track Processing, Playlist Manager) та їхні залежності.
4. Створити діаграми послідовностей (Sequence Diagrams) для ключових бізнес-процесів:
    - Авторизація користувача (Login).
    - Завантаження музичного треку (Upload Track).
    - Асинхронна обробка треку та збереження метаданих (Process Track).

A. Реалізувати програмну частину системи згідно з розробленими діаграмами, забезпечивши повний цикл обробки даних: від завантаження файлу через API до його обробки у фоновому режимі та збереження результату в БД.

B. Оформити звіт з обґрунтуванням прийнятих архітектурних рішень.

Теоретичні відомості:

Діаграма розгортання (Deployment Diagram) Цей тип діаграм у UML використовується для моделювання фізичної інфраструктури системи. Вона відповідає на питання "Де і як працює код?". Основними елементами є:

- Вузли (Nodes): Фізичні або віртуальні обчислювальні ресурси (сервери, робочі станції).
- Артефакти (Artifacts): Конкретні файли, архіви (JAR, WAR) або скрипти, що розгортаються на вузлах.
- Середовища виконання: Специфічні контейнери або платформи всередині вузлів (наприклад, Docker Engine, JVM, Web Browser), де

безпосередньо виконуються компоненти.

- Зв'язки: Комунікаційні шляхи між вузлами, що часто супроводжуються вказівкою протоколу (HTTP, TCP/IP, AMQP).

Діаграма компонентів (Component Diagram) Відображає організацію програмної системи у вигляді набору модулів (компонентів) та залежностей між ними. Компонент — це автономна частина системи, яка приховує свою реалізацію за інтерфейсом. Діаграма дозволяє оцінити зв'язність системи та виділити функціональні блоки (наприклад, модуль доступу до даних, модуль бізнес-логіки, UI-компонент).

Діаграма послідовностей (Sequence Diagram) Динамічна модель, що показує взаємодію об'єктів у часі. Вона акцентує увагу на порядку передачі повідомлень. Ключові елементи:

- Лінія життя (Lifeline): Вертикальна лінія, що позначає існування об'єкта протягом часу.
- Повідомлення (Messages): Стрілки, що відображають виклики методів або передачу даних (синхронні, асинхронні, відповіді).
- Фрагменти: Блоки для позначення умов (alt), циклів (loop) або паралельного виконання.

Хід роботи

1. Опис архітектури та Діаграма розгортання

Спроєктована система "Онлайн-радіостанція" базується на клієнт-серверній архітектурі та використовує контейнеризацію для ізоляції сервісів. Система складається з двох основних вузлів:

1. Клієнтський вузол (Client Device): Персональний комп'ютер або мобільний пристрій користувача. У середовищі веб-браузера виконується клієнтський SPA-застосунок (Single Page Application), написаний на React. Він взаємодіє з сервером через REST API, забезпечуючи відображення інтерфейсу плеєра та панелі керування.
2. Серверний вузол (Docker Host): Виділений сервер під управлінням Linux, на якому розгорнуто середовище Docker. Всі серверні компоненти запущені як окремі Docker-контейнери, об'єднані у

внутрішню мережу:

- Radio App (Backend): Основний додаток на Spring Boot, що реалізує бізнес-логіку.
- MongoDB: Документо-орієнтована база даних для зберігання користувачів, плейлистів та метаданих треків. Використовує Docker-том (Volume) для надійного зберігання даних.
- RabbitMQ: Брокер повідомлень для організації черги завдань (асинхронна обробка завантажених файлів).
- Web Server (Frontend Container): Контейнер (наприклад, Nginx або Node.js), що роздає статику клієнтського додатку.

Взаємодія між клієнтом та сервером відбувається за протоколом HTTP/HTTPS. Всередині Docker-мережі сервіси спілкуються за власними протоколами (MongoDB Wire Protocol, AMQP).

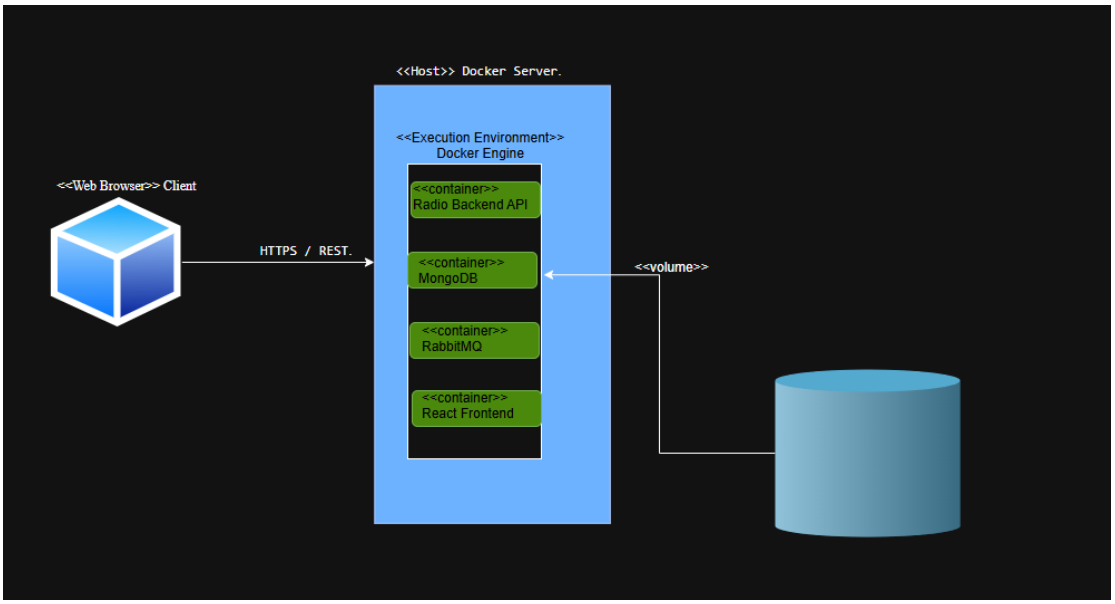

Рисунок 3.1 – Діаграма розгортання системи

Діаграма компонентів системи

Діаграма компонентів відображає логічну структуру веб-платформи «Онлайн-радіостанція» та залежності між її модулями (рис. 3.2). Архітектура

системи побудована за багатошаровим принципом та поділена на три основні частини .

1. Client Component (Клієнтські компоненти): Містить модулі, що відповідають за візуалізацію та взаємодію з користувачем:

- UI Module — модуль користувацького інтерфейсу, реалізований з використанням бібліотеки React. Він включає:
    - LoginForm — форма автентифікації користувача.
    - TrackList — список доступних музичних треків та плейлистів.
    - UploadForm — інтерфейс для завантаження нових аудіофайлів діджеєм.
    - PlayerControl — компонент керування відтворенням (Play/Pause, гучність).
- Client Logic Module — відповідає за взаємодію з API-сервісами. Використовує бібліотеку Axios для HTTP-запитів та забезпечує збереження стану (JWT-токен).

2. Server Component (Серверні компоненти): Серверна частина реалізована як застосунок Spring Boot та складається з кількох шарів:

- Controller Module — обробляє HTTP-запити, надає REST API та виконує валідацію вхідних даних.
- Service Module — містить основну бізнес-логіку:
    - AuthService — автентифікація користувачів.
    - TrackService — управління метаданими треків.
    - PlaylistService — створення та редагування плейлистів.
    - AudioProcessingPipeline — компонент для асинхронної обробки завантажених файлів (перевірка формату, вилучення метаданих) через RabbitMQ.
- Repository Module — шар доступу до даних, що містить репозиторії для MongoDB (UserRepository, TrackRepository, PlaylistRepository).

3. Infrastructure Component (Інфраструктурні компоненти):

- API Gateway — виконує маршрутизацію запитів від клієнта до бекенду.
- RabbitMQ — брокер повідомлень для організації черги обробки аудіофайлів.
- MongoDB — база даних для зберігання інформації.

Залежності між компонентами: Клієнтський UI залежить від логіки API, яка через шлюз (Gateway) звертається до контролерів. Сервісний шар використовує репозиторії для роботи з БД, а для важких задач (обробка файлів) передає повідомлення у RabbitMQ .

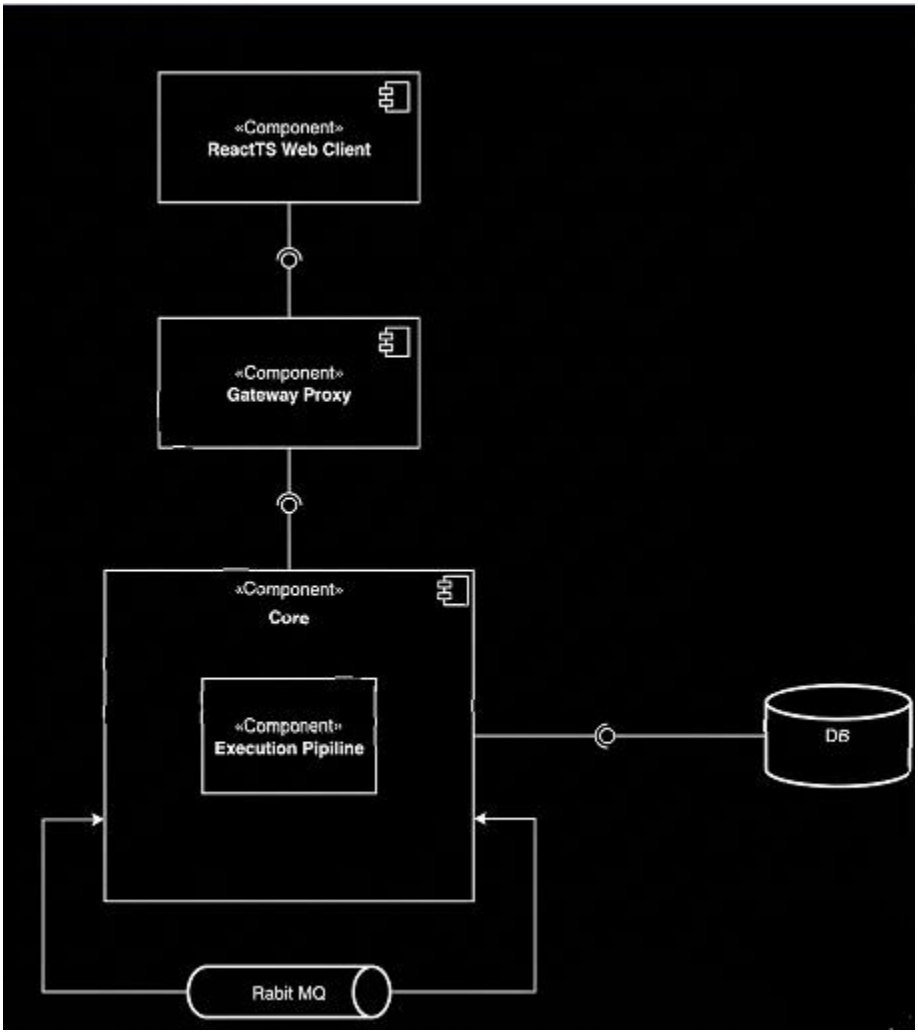

Рисунок 3.2 – Діаграма компонентів системи

Діаграма послідовності «Авторизація користувача в системі»

Ця діаграма моделює процес входу користувача (Діджея або Слухача) у систему

Учасники:

- User — користувач, який вводить облікові дані.
- Frontend — клієнтський інтерфейс (форма логіну на React).
- Service Proxy — API Gateway, що маршрутизує запити.
- Service Backend — серверна бізнес-логіка (AuthService).
- DB — база даних MongoDB з інформацією про користувачів .

Основний сценарій:

1. Користувач вводить логін/пароль і натискає кнопку «Login».
2. Frontend виконує локальну валідацію (наприклад, чи не порожні поля).
3. Frontend надсилає POST-запит /api/auth/login на Service Proxy.
4. Service Proxy перенаправляє запит до Service Backend.
5. Backend шукає користувача в базі даних (DB).
6. DB повертає дані користувача (хеш пароля).
7. Backend перевіряє пароль. Якщо все вірно — генерує JWT-токен.
8. Backend повертає токен успіху (HTTP 200).

Альтернативи:

- Невдача: Якщо пароль невірний, Backend повертає помилку 401 Unauthorized. Користувач бачить повідомлення про помилку.

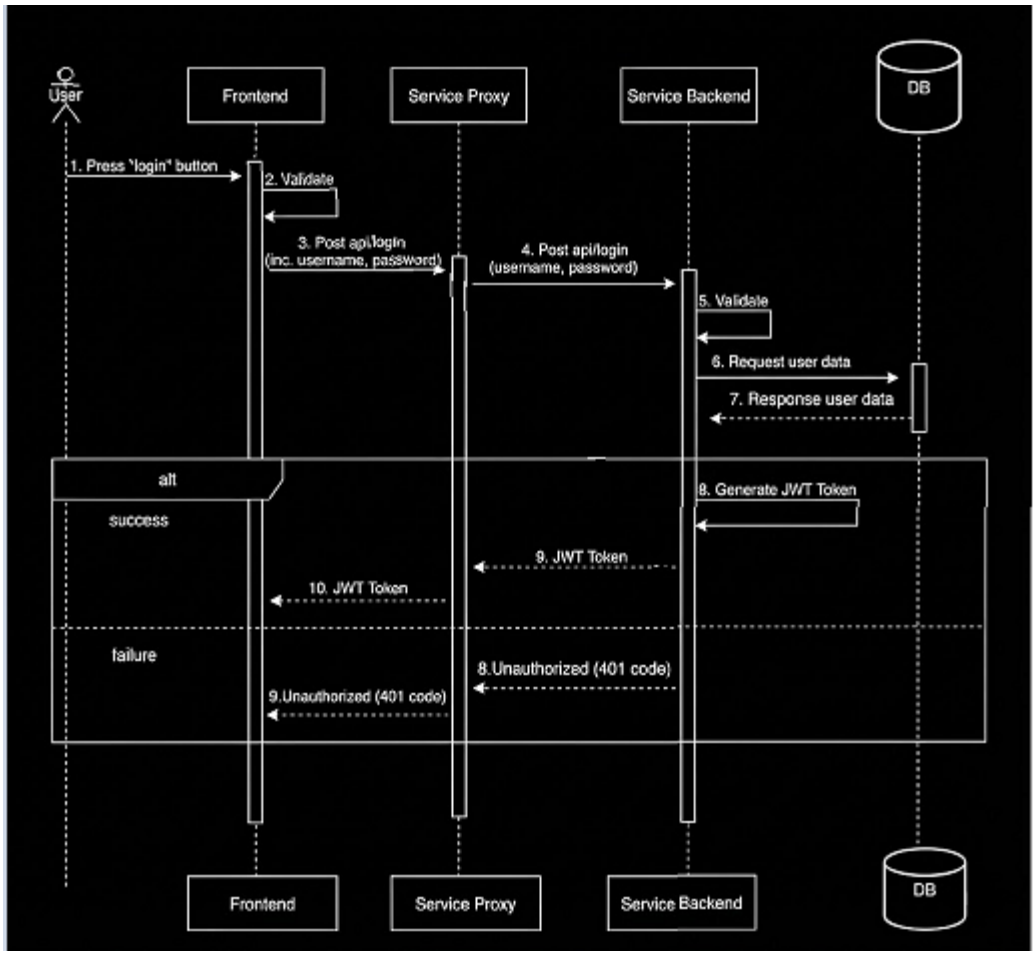

Рисунок 3.3 – Діаграма послідовності "авторизація користувача в системі"

1. Діаграма послідовності «Завантаження музичного треку»

Учасники:

- User — Діджей, який ініціює завантаження.
- Frontend — веб-інтерфейс (форма UploadForm).
- Service Proxy — API Gateway.
- Service Backend — сервіс, що приймає файл.
- DB — база даних (MongoDB + GridFS) .

Основний сценарій:

- User натискає кнопку «Upload» та обирає MP3-файл.

- Frontend перевіряє формат файлу (чи це .mp3/.wav) та розмір.
- Frontend надсилає POST-запит /api/tracks/upload (Content-Type: multipart/form-data) на Service Proxy.
- Service Proxy пересилає запит на Service Backend.
- Service Backend зберігає бінарні дані файлу в GridFS (DB) та створює запис метаданих.
- DB підтверджує збереження.
- Backend повертає статус 200 OK .

Альтернативи:

Помилка: Якщо формат файлу не підтримується (наприклад, .exe), Backend повертає 415 Unsupported Media Type. Frontend показує помилку

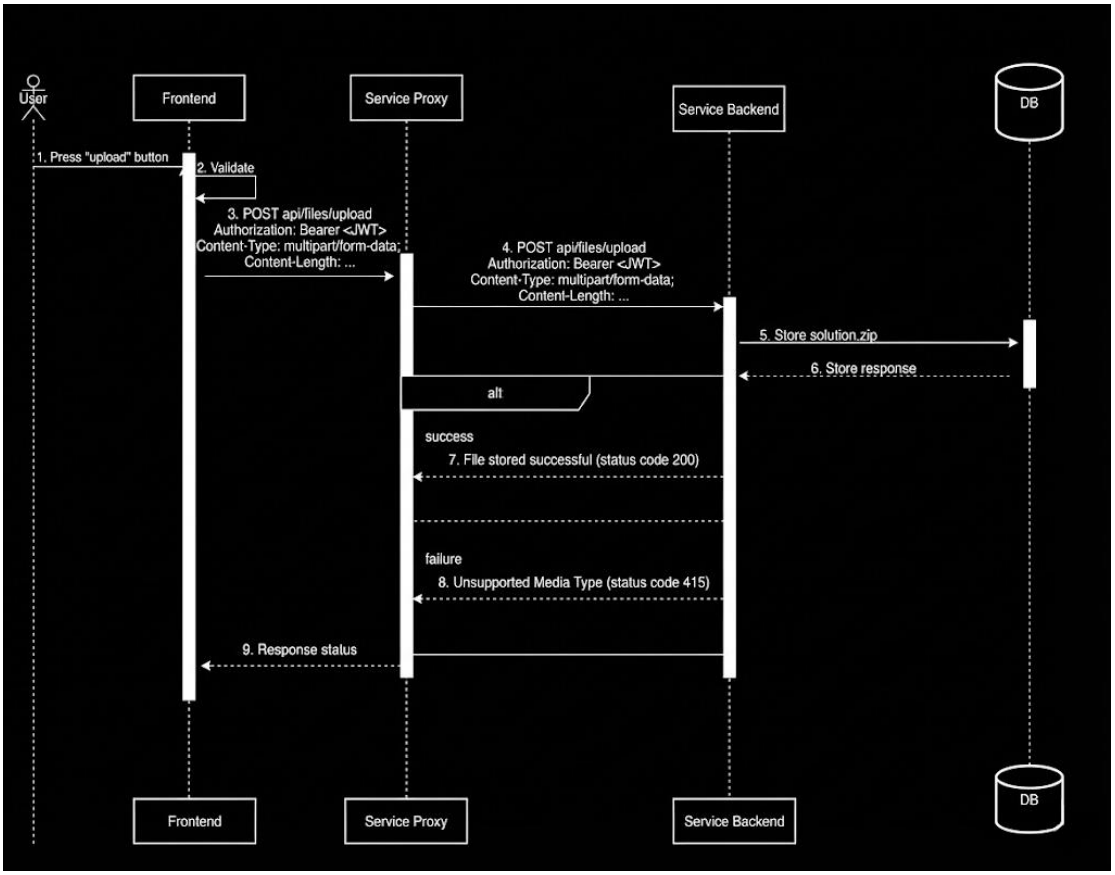

Рисунок 3.4 – Діаграма послідовності «Завантаження музичного треку»

Діаграма послідовності «Асинхронна обробка аудіофайлу»

Ця діаграма ілюструє процес фонової обробки завантаженого треку, який відбувається без прямої участі користувача.

Учасники:

- Service Backend — серверна частина, ініціатор події.
- RabbitMQ — черга повідомлень для розв'язки процесів.
- Audio Processor — окремий сервіс (воркер), що виконує "важку" роботу з аудіофайлами.
- DB — база даних MongoDB .

Основний сценарій:

1. Після успішного завантаження файлу Service Backend відправляє повідомлення (ID треку) у чергу RabbitMQ.
2. RabbitMQ зберігає повідомлення та передає його вільному воркеру (Audio Processor).
3. Audio Processor робить запит до DB, щоб отримати метадані треку та посилання на файл.
4. DB повертає дані.
5. Processor виконує аналіз файлу (визначення тривалості, бітрейту, формату).
6. (Опціонально) Processor витягує обкладинку альбому з метатегів MP3.
7. Після завершення обробки Processor оновлює статус треку в DB на READY та зберігає отримані характеристики (тривалість).
8. Backend зчитує оновлений статус (при наступному запиті клієнта).

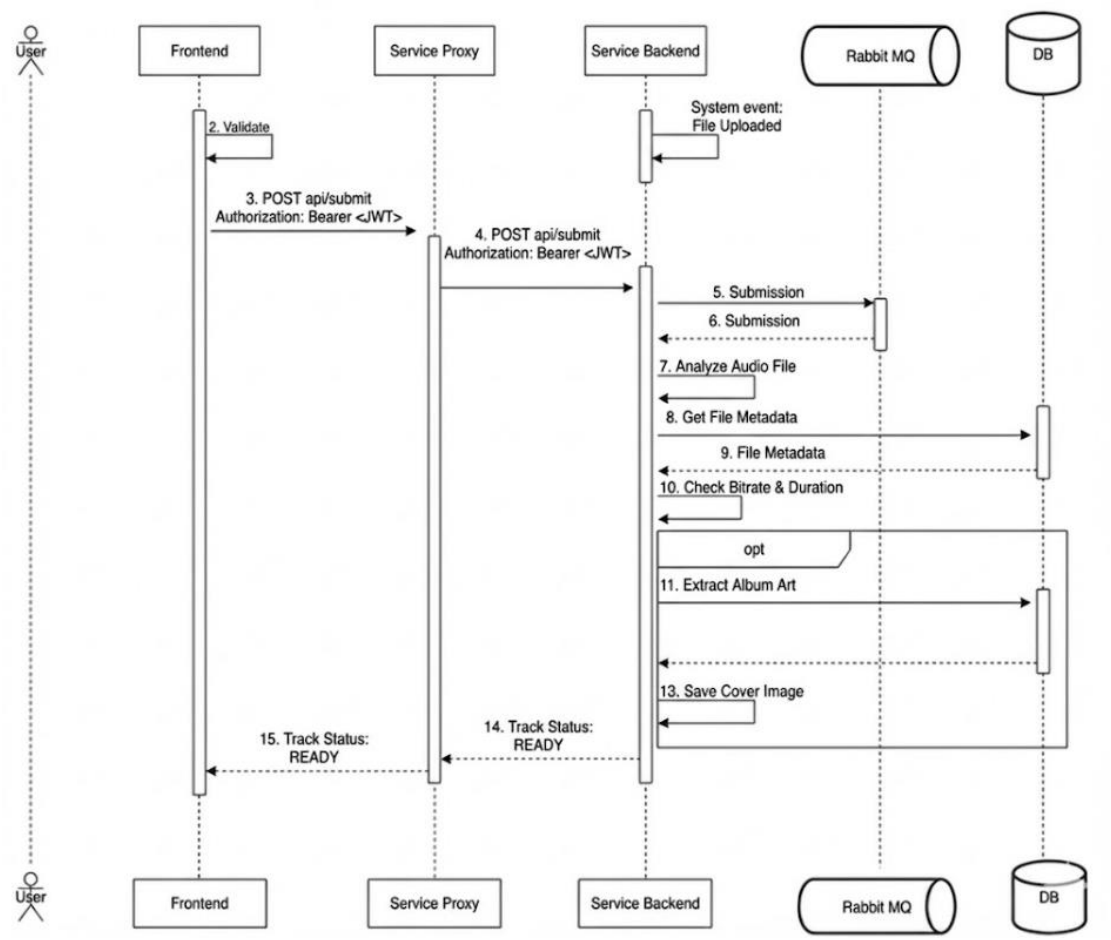

Рисунок 3.5 – Діаграма послідовності «Завантаження музичного треку»

Опис реалізації програмної частини

На основі спроєктованих діаграм розгортання та компонентів було реалізовано програмну частину веб-сервісу «Онлайн-радіостанція». Реалізація охоплює повний цикл роботи – від завантаження музичного треку діджеєм до його обробки та відображення у публічному плейлисті для слухачів.

Система побудована за багатошаровою архітектурою з чітким розподілом відповідальності:

- Presentation Layer (Шар представлення):
- React-клієнт – веб-інтерфейс для взаємодії з користувачем (SPA).
- Компоненти:
    - LoginPage.tsx – сторінка автентифікації користувачів;

        - TracksPage.tsx – сторінка зі списком треків та аудіоплеєром;
        - UploadPage.tsx – інтерфейс для завантаження нових композицій (доступний для ролі DJ).
- Business Logic Layer (Шар бізнес-логіки):
    - TrackService – управління метаданими треків та файлами;
    - AuthService – автентифікація та генерація JWT-токенів;
    - PlaylistService – створення та редагування списків відтворення;
    - AudioProcessingPipeline – асинхронна обробка завантажених файлів (аналіз формату, тривалості) через RabbitMQ.
- Data Access Layer (Шар доступу до даних):
    - Інтерфейси репозиторіїв: UserRepository, TrackRepository, PlaylistRepository;
    - Реалізація – MongoDB з використанням Spring Data MongoDB;
    - FileStorage – управління збереженням бінарних файлів (MP3) через GridFS.
- Domain Layer (Доменний шар):
    - Модельні класи: User, Track, Playlist, FileMetadata;
    - Енумерації: UserRole (LISTENER, DJ, ADMIN), TrackStatus (PROCESSING, READY, ERROR)

Опис інтерфейсу користувача

Форма №1: LoginPage (Сторінка входу)

Інтерфейс містить:

- Заголовок із навігацією: посилання «Radio» (Головна), «Login» (Вхід).
- Поле введення логіна з підказкою (placeholder) «Username».
- Поле введення пароля з підказкою «Password».
- Кнопку підтвердження з текстом «Sign In» (Увійти).

Основні можливості:

- Авторизація користувача – введення облікових даних для доступу до персональних функцій.
- Надсилання форми – після натискання кнопки дані передаються на сервер для валідації та отримання токена.
- Розподіл прав – залежно від ролі (Слухач або Діджей) інтерфейс адаптується (наприклад, з'являється кнопка завантаження).

Форма №2: UploadPage (Завантаження треку)

Інтерфейс відображає:

- Панель навігації з кнопкою «Logout».
- Зону для вибору файлу («Drag & Drop» або кнопка «Choose File»).
- Поля для введення метаданих: «Title» (Назва) та «Artist» (Виконавець).
- Кнопку «Upload Track».

Функціональність:

- Валідація файлу – перевірка формату (тільки .mp3/.wav) на стороні клієнта.
- Завантаження – відправка файлу на сервер для подальшої асинхронної обробки.

Форма №3: TracksPage (Список треків та плеєр)

Інтерфейс відображає:

- Список доступних композицій із зазначенням назви, виконавця та тривалості.
- Панель відтворення (Player Controls): кнопки Play/Pause, смуга прогресу, регулятор гучності.
- Статус треку (якщо він ще обробляється сервером).

Функціональність:

- Прослуховування – потокове відтворення аудіо через HTML5 Audio

API.

- Оновлення списку – відображення нових треків після завершення їх обробки бекендом.

Основна частина реалізованого коду

Нижче наведено лістинг коду, що реалізує механізм асинхронної обробки медіафайлів. Використано архітектурний патерн Chain of Responsibility (Ланцюжок обов'язків) для послідовного виконання етапів обробки (валідація, аналіз метаданих, збереження) у фоновому режимі.

Package: com.example.radio.service.processor

AudioProcessingListener.java Клас-слухач, який очікує повідомлення від RabbitMQ і запускає ланцюжок обробки.

```java
package com.example.radio.service.processor;

import com.example.radio.config.RabbitConfig;
import com.example.radio.model.TrackEntity;
import com.example.radio.service.TrackService;
import com.example.radio.service.processor.stage.ProcessingChain;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Service;

@Slf4j
@Service
@RequiredArgsConstructor
public class AudioProcessingListener {

    private final TrackService trackService;
    private final ProcessingChain chain;

    @RabbitListener(queues = RabbitConfig.TRACK_PROCESSING_QUEUE)
    public void receiveMessage(String trackId) {
        log.info("Received message from RabbitMQ for trackId: {}", trackId);

        TrackEntity track = trackService.findById(trackId);

        if (track != null && "PROCESSING".equals(track.getStatus())) {
            log.info("Starting processing chain for track: {}", track.getTitle());
            chain.startChain(track);
        } else {
            log.warn("Track not found or already processed: {}", trackId);
        }
    }
}
```

Package: com.example.radio.service.processor.stage

ProcessingChain.java Клас-оркестратор, який керує послідовністю виконання етапів

```java
import com.example.radio.model.TrackEntity;
import jakarta.annotation.PostConstruct;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Component;
import java.util.List;

@Slf4j
@Component
@RequiredArgsConstructor
public class ProcessingChain {

    private final FormatValidationStage formatStage;
    private final MetadataExtractionStage metadataStage;

    private List<StageExecutor> stages;
    private int currentStageIndex = 0;

    @PostConstruct
    private void initExecutorChain() {
        // Визначаємо порядок виконання етапів
        stages = List.of(
                formatStage,
                metadataStage
        );
    }

    public void startChain(TrackEntity track) {
        this.currentStageIndex = 0;
        this.doNext(track);
    }

    public void doNext(TrackEntity track) {
        if (currentStageIndex < stages.size()) {
            StageExecutor currentStage = stages.get(currentStageIndex++);
            currentStage.execute(track, this);
        } else {
            log.info("All processing stages completed successfully for track: {}",
            // Фіналізація: трек готовий до ефіру
            track.setStatus("READY");
        }
    }
}
```

StageExecutor.java (Interface)

```java
package com.example.radio.service.processor.stage;

import com.example.radio.model.TrackEntity;

public interface StageExecutor {
    void execute(TrackEntity track, ProcessingChain chain);
}
```

FormatValidationStage.java Перший етап: перевірка формату файлу та його цілісності.

```java
package com.example.radio.service.processor.stage;

import com.example.radio.model.TrackEntity;
import com.example.radio.service.TrackService;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Component;

@Slf4j
@Component
@RequiredArgsConstructor
public class FormatValidationStage implements StageExecutor {

    private final TrackService trackService;

    @Override
    public void execute(TrackEntity track, ProcessingChain chain) {
        log.info("Stage 1: Validating audio format for {}", track.getId());

        try {
            // Імітація перевірки заголовків файлу (Magic numbers)
            Thread.sleep(1000);

            // Тут була б реальна логіка перевірки бінарних даних
            boolean isValidMp3 = true;

            if (isValidMp3) {
                log.info("Format validation passed.");
                chain.doNext(track);
            } else {
                log.error("Invalid audio format.");
                track.setStatus("ERROR");
                trackService.save(track);
            }
        } catch (Exception e) {
            log.error("Error during validation stage", e);
            track.setStatus("ERROR");
            trackService.save(track);
        }
    }
}
```

MetadataExtractionStage.java Другий етап: витягування тривалості, бітрейту та обкладинки.

```java
package com.example.radio.service.processor.stage;

import com.example.radio.model.TrackEntity;
import com.example.radio.service.TrackService;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Component;

@Slf4j
@Component
@RequiredArgsConstructor
public class MetadataExtractionStage implements StageExecutor {

    private final TrackService trackService;

    @Override
    public void execute(TrackEntity track, ProcessingChain chain) {
        log.info("Stage 2: Extracting metadata (Bitrate, Duration)");

        try {
            // Імітація роботи бібліотеки (наприклад, mp3agic або ffmpeg)
            Thread.sleep(1500);

            // Симуляція отриманих даних
            int calculatedDuration = 245; // 4 хв 05 сек
            String bitrate = "320kbps";

            track.setDuration(calculatedDuration);
            log.info("Metadata extracted: duration={}, bitrate={}", calculatedDur

            // Зберігаємо проміжні результати
            trackService.save(track);

            // Переходимо до наступного кроку (або завершуємо, якщо це останній)
            chain.doNext(track);

        } catch (Exception e) {
            log.error("Failed to extract metadata", e);
            track.setStatus("ERROR_METADATA");
            trackService.save(track);
        }
    }
}
```

Питання до лабораторної роботи

1. Що собою становить діаграма розгортання? Діаграма розгортання (Deployment Diagram) — це UML-діаграма, яка відображає фізичну архітектуру системи, тобто розподіл програмного забезпечення по апаратному обладнанню. Вона показує вузли (сервери, робочі станції), середовища виконання (Docker, браузер) та артефакти (файли програм), а також фізичні або мережеві зв'язки між ними (наприклад, HTTP-з'єднання між клієнтом та сервером).

2. Які бувають види вузлів на діаграмі розгортання? Вузли поділяються на

два типи:

- Пристрій (Device): фізичний ресурс із процесором та пам'яттю (наприклад, сервер, на якому хоститься Радіостанція, або комп'ютер користувача).
- Середовище виконання (Execution Environment): програмний вузол, який існує всередині пристрою і забезпечує запуск інших компонентів (наприклад, Docker Engine, що запускає контейнери, або Java Virtual Machine).

3. Які бувають зв'язки на діаграмі розгортання? Зв'язки зображуються лініями, що з'єднують вузли. Вони характеризують канал комунікації і часто містять стереотипи, що вказують протокол (наприклад, «HTTP», «TCP/IP», «AMQP»). Також можуть вказуватися атрибути множинності (наприклад, один сервер обслуговує багато клієнтів) .

4. Які елементи присутні на діаграмі компонентів? Основними елементами є:

- Компоненти: модульні частини системи (наприклад, React Web Client, Radio Core, Audio Processing Pipeline).
- Інтерфейси: точки взаємодії («lollipop» або гніздо), через які компоненти надають або отримують послуги.
- Залежності: пунктирні стрілки, що показують, як один компонент залежить від іншого.

5. Що становлять собою зв'язки на діаграмі компонентів? Це відносини залежності. Вони показують, що один компонент (наприклад, Radio Core) потребує функціоналу іншого компонента (наприклад, MongoDB або RabbitMQ) для своєї коректної роботи. Зміна в незалежному компоненті може вплинути на залежний.

6. Які бувають види діаграм взаємодії? В UML існує кілька видів діаграм взаємодії, основними з яких є:

- Діаграма послідовностей (Sequence Diagram): акцентує увагу на часовій послідовності повідомлень.
- Діаграма комунікації (Communication Diagram): акцентує увагу на структурних зв'язках між об'єктами, що обмінюються повідомленнями.

7. Для чого призначена діаграма послідовностей? Вона призначена для

моделювання динамічної поведінки системи в рамках одного сценарію використання. Вона візуалізує, у якому порядку об'єкти (Frontend, Backend, DB) обмінюються повідомленнями для досягнення конкретної мети (наприклад, завантаження треку).

8. Які ключові елементи можуть бути на діаграмі послідовностей?

- Актори: ініціатори дій (User).
- Лінії життя (Lifelines): вертикальні пунктири, що показують існування об'єкта в часі.
- Повідомлення: стрілки викликів методів або передачі даних.
- Фрагменти (Combined Fragments): блоки для логічних операцій, таких як alt (альтернатива/умова), loop (цикл), opt (необов'язкова дія).

9. Як діаграми послідовностей пов'язані з діаграмами варіантів використання? Діаграма послідовності є деталізацією конкретного варіанту використання (Use Case). Якщо Use Case каже «Що система робить» (наприклад, «Завантажити трек»), то діаграма послідовності показує «Як саме» це відбувається технічно, крок за кроком.

10. Як діаграми послідовностей пов'язані з діаграмами класів? Діаграми послідовностей оперують екземплярами класів (об'єктами), які були визначені на діаграмі класів. Повідомлення, що передаються між лініями життя, зазвичай відповідають методам, визначеним у класах (наприклад, метод save() у репозиторії або process() у сервісі).