Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

**МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА**

на тему:

# ЗАСОБИ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ В РОЗРОБЦІ СУЧАСНИХ ВЕБДОДАТКІВ

Виконав студента 2 курсу групи 1КІ-24м
спеціальності 123 — Комп'ютерна інженерія
_____ ГрішаД. Т.

Керівник: к.т.н., доц. кафедри ОТ
_____ Тарновський М. Г.
«12» 12 2025 р.

Опонент: к.т.н., доц. кафедри ПЗ
_____ Ткаченко О. М.
«12» 12 2025 р.

**Допущено до захисту**

Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
_____
«15» 12 2025 р

ВНТУ 2025

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

Рівень вищої освіти II—й (магістерський)

Галузь знань — 12 «Інформаційні технології»

Спеціальність — 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

д.т.н., професор Азаров О.Д.

"26" вересня 2025 року

**З А В Д А Н Н Я**

**НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ**

Гріші Данілу Тарасовичу

1 Тема роботи — Засоби мікросервісної архітектури в розробці сучасних вебдодатків, керівник роботи Тарновський Микола Генадійович, к.т.н., доцент кафедри ОТ, затверджені наказом ВНТУ від "24" вересня 2025 року № 313.

2 Строк подання студентом роботи — 4 грудня 2025 року.

3 Вихідні дані до роботи: Мікросервісний стиль архітектури (MSA), автоматизоване розгортання сервісів — Docker, обмін повідомленнями — RabbitMQ, реалізація single sign-on — Keycloak, управління модулями програми — Maven, реляційна база даних — PostgreSQL, реалізація кешування — Redis, реалізація специфікації OpenAPI — Swagger, середовище розробки — IntelliJ IDEA: Community Edition, мова розробки — Java, фреймворк Spring Boot 3.

4 Зміст текстової частини: вступ; аналіз методів та засобів створення сучасних вебдодатків; теоретичні аспекти проєктування та методи розробки мікросервісного вебдодатку; розробка програмних складових вебдодатку; аналіз результатів дослідження та тестування мікросервісного вебдодатку; економічна частина; висновки.

5 Перелік графічного матеріалу: структурна схема роботи програм
блок схема логіки авторизації, структурна схема бази даних.

6 Консультанти розділів роботи

Таблиця МКР.1 — Консультанти розділів

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | Завдання прийняв |
| 1— 4 | Тарновський М. Г., к.т.н., доц. | 29.02.25 | 12.12.25 |
| 5 | Ратушняк О.Г., к.т.н., доц. каф. | | |
| Нормоконтроль | асистент каф. ОТ Швець С.І. | | |

7 Дата видачі завдання — 29.09.2025 року

8 Календарний план

Таблиця МКР.2 — Календарний план

| № з/п | Назва етапів магістерської кваліфікаційної роботи | Строк виконання етапів роботи | Приміт |
|---|---|---|---|
| 1 | Аналіз методів та засобів створення сучасних вебдодатків | 08.09.25 — 12.09.25 | Вик. |
| 2 | Теоретичні аспекти проєктування та методи розробки мікросервісного вебдодатку | 15.09.25 — 19.09.25 | Вик. |
| 3 | Розробка програмних складових вебдодатку | 22.09.25 — 03.10.25 | Вик. |
| 4 | Аналіз результатів дослідження та тестування мікросервісного вебдодатку | 06.10.25 — 15.10.25 | Вик. |
| 5 | Економічна частина | 17.10.25 — 20.10.25 | Вик. |
| 6 | Оформлення пояснювальної записки, графічного матеріалу і презентації | 25.10.25 | Вик. |
| 7 | Підготовка супроводжуючих документів | 30.10.25 | Вик. |

Студент ( підпис ) Гріша Д.Т.
(прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи ( підпис ) Тарновський М.
(прізвище та ініціали)

# АНОТАЦІЯ

УДК 004.31

Гріша Д.Т. Засоби мікросервісної архітектури в розробці сучасних вебдодатків. Магістерська кваліфікаційна робота зі спеціальності 123 «Комп'ютерна інженерія». Вінниця: ВНТУ, 2025. 129 с.

На укр. мові. Бібліогр.: 37 назв; рис.: 55; табл.: 14.

У магістерській роботі досліджено сучасні методи та технології розробки вебдодатків із використанням мікросервісної архітектури. Розглянуто переваги, недоліки та особливості застосування мікросервісного підходу у порівнянні з монолітними та серверлес архітектурами.

У роботі розроблено демонстраційний мікросервісний вебдодаток, який складається з автономних сервісів, що реалізують бізнес-логіку системи, а також інфраструктурних компонентів для забезпечення маршрутизації, автентифікації, авторизації, кешування та взаємодії між сервісами. Реалізовано функціонал управління користувачами, обробки музичних композицій, імпорту даних у форматах XLSX, генерації PDF-файлів, а також асинхронного обміну повідомленнями.

Проведено тестування вебдодатку, яке підтвердило правильність роботи сервісів, стійкість до збоїв та можливість масштабування. Виконано оцінку економічної ефективності запропонованих рішень.

Ключові слова: Мікросервісна архітектура, сервіси, сервер, вебдодатки, масштабованість, хмарні технології, розгортання сервісів, безпека, моніторинг, метрика, мікромодульність, екосистема.

# ANNOTATION

UDC 004.31

Hrisha D.T. Microservice architecture tools in the development of modern web applications. Master’s qualification work in the specialty 123 "Computer Engineering". Vinnytsia: VNTU, 2025. 129 pages .

In Ukrainian. Bibliography: 37 titles; figures: 55; tables: 14.

The master's thesis explores modern methods and technologies for developing web applications using microservice architecture. The advantages, disadvantages, and features of using the microservice approach compared to monolithic and serverless architectures are considered.

The paper develops a demonstration microservice web application consisting of autonomous services that implement the business logic of the system, as well as infrastructure components to ensure routing, authentication, authorization, caching, and interaction between services. The functionality of user management, music composition processing, data import in XLSX formats, PDF file generation, and asynchronous messaging are implemented.

The web application was tested, which confirmed the correct operation of the services, fault tolerance, and scalability. The economic efficiency of the proposed solutions was assessed.

Keywords: Microservice architecture, services, server, web applications, scalability, cloud technologies, service deployment, security, monitoring, metrics, micromodularity, ecosystem.

# ЗМІСТ

# ВСТУП

Стрімкий розвиток інформаційних технологій створює умови для підвищення вимог до програмних продуктів. Відповідно розробники постають перед необхідністю пошуку та застосування рішень, які надійно працюватимуть в умовах високого навантаження, великих потоків даних, дозволяють ефективно здійснювати реалізацію нового функціоналу.

**Актуальність теми** обумовлена тим, що мікросервісна архітектура набуває усе більшої популярності серед розробників програмних продуктів через надання можливості створювати легко масштабовані, гнучкі та кросплатформні програмні застосунки, які можуть бути легко адаптовані під завдання різних користувачів. Головними її перевагами є незалежність створення та розгортання окремих елементів системи, а також високий рівень декомпозиції всього проекту [1].

Основна концепція архітектури полягає у розділенні складної програми на невеликі автономні модулі — мікросервіси, кожен з яких орієнтований на вирішення конкретного завдання і може бути змінений без внесення змін у зв’язані з ним компоненти. Це дозволяє легко оновлювати та змінювати функціонал системи, масштабувати її без ризику порушення цілісності та працездатності, що робить мікросервісну архітектуру оптимальним рішенням при створенні великих інтернет-сервісів, які обслуговують велику кількість користувачів.

Збільшення кількості сервісів тягне за собою зростання кількості баз даних та ускладнює тестування системи. Іншою проблемою є перенесення мікросервісів із середовища розробки на платформу, на якій вони будуть функціонувати, що ускладнює процес розгортання та масштабування мікросервісів.

**Метою роботи** є вдосконалення підходів до використання мікросервісної архітектури при розробці вебдодатків, що спрощує їх розгортання та організацію взаємодії між їхніми компонентами.

Для досягнення даної роботи необхідно виконати такі **задачі**:

— аналіз методів та засобів створення сучасних вебдодатків;

— огляд прикладів застосування мікросервісної архітектури;

— аналіз методів розробки мікросервісного вебдодатку;

— розробка програмних складових вебдодатку;

— тестування мікросервісного вебдодатку;

— оцінювання економічної ефективності запропонованих рішень.

**Об’єктом дослідження** є процеси проєктування та розробки сучасних вебдодатків.

**Предметом дослідження** є методи та засоби із застосування мікросервісної архітектури при розробці вебдодатків.

**Новизна роботи** полягає в тому, що набув подальшого розвитку метод інтеграції мікросервісів, при якому за рахунок застосування сучасної технології контейнеризації Docker усувається проблема несумісності програмних середовищ, що полегшує інтеграцію різних програмних компонентів у межах однієї системи.

Також вдосконалено методи роботи з базою даних, зокрема із системою управління базами даних PostgreSQL, завдяки впровадженню оптимізованих підходів до побудови запитів, індексації та кешування даних, що підвищує ефективність оброблення запитів і зменшує навантаження на сервер бази даних.

**Практичне значення** роботи полягає в тому, що запропоновані підходи надають можливість створювати гарно масштабовані та високо надійні вебдодатки різного призначення.

**Апробація** результатів роботи здійснена у доповіді на Міжнародній науково-практичній Інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи (МН 2026)».

Матеріали роботи доповідались та опубліковувались [2]:

Тарновський М.Г., Гріша Д.Т. ЗАСОБИ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ В РОЗРОБЦІ СУЧАСНИХ ВЕБДОДАТКІВ. Конференція

ВНТУ: Молодь в науці: дослідження, проблеми, перспективи (МН - 2026). [Електронний ресурс]. Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2026/paper/view/26248

# 1 АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ СТВОРЕННЯ СУЧАСНИХ ВЕБДОДАТКІВ

## 1.1 Аналіз сучасних підходів до розробки вебдодатків

За останній час вебдодатки зазнали суттєвих змін, які зумовлені як зростанням вимог користувачів, так і швидким розвитком технологій. Вебдодатоки стали більш розвиненими у своїй загальній функціональності відповідно до потреб та трендів. Тому аналіз сучасних підходів до розробки вебдодатків є важливим етапом даної роботи.

Однією з ключових характеристик сучасних вебдодатків є високий рівень інтерактивності. Вебпереглядач за певний час став більш використовуваним та набув багато можливостей. Тому завдяки широкому застосуванню JavaScript, WebSocket, WebRTC та інших технологій він став повноцінною платформою для виконання клієнтської логіки [3].

Користувачі очікують, що вебдодатки реагуватимуть на їхні дії так само швидко, як і настільні застосунки. Це зумовило появу таких концепцій, як Single Page Application (SPA), де взаємодія з сервером здійснюється переважно через API, а оновлення інтерфейсу відбувається без повного перезавантаження сторінки. Завдяки цьому покращується реактивність, сприйняття швидкості роботи та зменшується затримка між діями користувача та відповіддю системи [4].

Динамічність також виявляється в адаптації інтерфейсу до контексту користувача: мови, регіону, історії взаємодії, поточних запитів. Сучасні системи часто використовують персоналізацію на основі даних, що збільшує вимоги до швидкої доставки та обробки інформації.

Зі стрімким зростанням кількості користувачів та обсягів даних питання масштабованості стало одним із найважливіших у проєктуванні вебдодатків. Традиційна вертикальна масштабованість — збільшення потужності одного сервера — більше не задовольняє потреби сучасних розподілених систем.

Натомість поширилася горизонтальна масштабованість, яка передбачає збільшення кількості серверів або контейнерів у кластері.

Сучасні вебдодатки мають бути стійкими до навантажень, таких як різкий зріст популярності сервісу. Для цього застосовуються механізми балансування навантаження, кешування даних та відповідей API, реплікація сервісів, застосування хмарних платформ та автоматичного масштабування.

Стійкість до навантажень також охоплює здатність системи залишатися працездатною при часткових збоях окремих сервісів або інфраструктурних елементів. Концепції fault-tolerance, high availability та resilience стали стандартом для сучасних вебдодатків і багато в чому визначають архітектурний стиль мікросервісів [5].

Доступність вебдодатка — це один із ключових показників якості сервісу. Непередбачені збої, недоступність окремих частин системи або повне припинення роботи можуть призвести до негативних наслідків.

Доступність визначається не лише апаратною складовою, але й архітектурою застосунка.

Таким чином, забезпечення високої доступності стало фундаментальним завданням веброзробки, що вимагає правильного вибору архітектурних рішень, інфраструктурних інструментів і підходів до тестування.

Продуктивність вебдодатка визначає швидкість його роботи, здатність обробляти запити та масштабуватись із мінімальною затримкою. Розробники мають забезпечити, щоб система працювала швидко як на стороні сервера, так і на стороні клієнта.

Сучасні вебдодатки мають інтегрувати безпеку на всіх рівнях: від інфраструктури до бізнес-логіки, від контролю сесій до зберігання даних. До обов’язкових вимог входять шифрування даних, механізми авторизації, багатофакторна аутентифікація, контроль доступу. Також важливими вимогами є логування та захист від поширених атак таких як: XSS, CSRF, SQL injection [6]. Усі ці особливості є складовими майже кожного сучасного

вебдодатку, що орієнтований на масового користувача. Тому усі ці ознаки будуть прийняті до уваги в подальшій розробці.

## 1.2 Аналіз сучасних технологій розробки мікросервісних додатків

Оскільки подальша розробка вебдодатку використовуватиме мікросервісний архітектурний стиль, потрібно виділити основні технології, які необхідні для створення програми.

Для розробки мікросервісних додатків наявна велика кількість мов програмування. Мова Java є отимальним рішенням для такого роду програмного заезчпечення, оскільки має потужну екосистему фреймворків, велику кількість біліотек та інтеграцію з різними сервісами. Для мови програмування Java найбільш поширеним рішенням є використання Spring Boot у поєднанні з Spring Cloud. Даний фреймворк дозволяє багато можливостей для створення окремих сервісів, які легко інтегруються у складні системи. Також існують альтернативи Micronaut та Quarkus, які позиціонуються як сучасні та простої варіанти створення мікросервісів, орієнтовані на швидкий старт та інтеграцію з контейнеризованими середовищами. Micronaut демонструє високі показники продуктивності завдяки попередній компіляції залежностей, тоді як Quarkus орієнтований на запуск у Kubernetes [7].

JavaScript та TypeScript є також вдалим рішенням для створення мікросервісних додатків, оскільки вони мають велику екосистему пакетів (npm), активну підтриму спільноти та проста контейнеризація. Для цих мов програмування значного поширення набув Node.js. Найпростішим підходом вважається використання Express, що забезпечує базову інфраструктуру для побудови REST API. Для більш структурованого підходу часто застосовують NestJS, який підтримує принципи модульності та впроваджує патерни, характерні для корпоративних архітектур [8].

C# (C-Sharp) є дуже близьким родичем мови програмування Java. Тому вона має схожі переваги для розробки, але при цьому вирізняється легкою інтеграцією з хмарними сервісами, сильною типізацією та безпечністю. Для даної мови програмування використовується .NET Core. Тут мікросервіси реалізуються за допомогою ASP.NET Core, який забезпечує як побудову REST-сервісів, так і підтримку gRPC. Основна перевага даної технології полягає в інтеграції з Microsoft Azure, однак завдяки контейнеризації дане рішення активно застосовується також у поєднанні з Kubernetes [9].

Мова програмування Python вирізняється своєю загальною простотою та швидкістю. Також вона є має розвинуту екосистему біліотек що робить її вдалим рішенням для розробки мікросервісів. Загалом Python пропонує декілька рішень для створення мікросервісів. FastAPI вирізняється сучасним підходом до опису REST API, що дозволяє автоматично генерувати документації та підтримувати асинхронні обробки. Flask забезпечує простий підхід, який дає змогу швидко створювати невеликі сервіси, проте вимагає додаткових бібліотек для реалізації повноцінної інфраструктури. Django, у свою чергу, орієнтований на монолітні системи, однак у поєднанні з Django Rest Framework може застосовуватися для реалізації окремих мікросервісів у складі великої розподіленої системи [10].

Мова програмування GO (Golang) активно використовується у сфері мікросервісів завдяки своїй легкості та продуктивності. Також вона має вбудовану підтримку багатопоточності та вбудовані інструменти для реалізації різного роду функціоналу. GO має фреймворки Gin або Echo, дозволяють створювати сервіси з мінімальними витратами пам’яті та часу виконання [11].

Основним елементом будь-якого сучасного вебдодатку є оперування великою кількістю даних, яка повинна створюватися, змінюватися та видалятися. Саме тут використовується бази даних.

Найчастіше використовуються реляційні системи управління базами даних таких як PostgreSQL та MySQL. Перша має високий рівень відповідності

стандартам SQL та володіє розвинутими можливостями роботи з розподіленими даними. MySQL в своєю чергу застосовується завдяки простоті, швидкодії та широкій підтримці з боку хмарних платформ.

Інколи функціонал програми орієнтується на високу швидкість обробки даних у реальному часі та потребує масштабованості, тому доцільним стає використання нереляційних (або NoSQL) баз даних. Серед таких наявна MongoDB, яка належить до документно-орієнтованих баз даних і забезпечує зручність роботи з напівструктурованими даними [12].

Система управлінням баз даних Redis використовується як інструмент для кешування та тимчасового зберігання даних, що дає можливість значно зменшити затримки при обробці запитів.

Cassandra використовується у високонавантажених системах, де критично важливими є швидке горизонтальне масштабування та забезпечення відмовостійкості.

Існують також графові бази даних, серед яких найпопулярнішим інструментом є Neo4j. Вона застосовуються у тих випадках, коли дані мають складні взаємозв’язки. Завдяки оптимізованим алгоритмам пошуку по графам такі бази даних дозволяють суттєво знизити складність обробки запитів порівняно з реляційними підходами.

Мікросервісні додатки являють собою декілька окремих програм, що взаємодіють між собою та утворюють цілу й злагоджену екосистему. Тому для такого роду програмного забезпечення необхідна наявність точка єдиного входу. Також дуже важливими аспектами є безпека та перевірка даних користувача, оскільки без неї програмний продукт може стати доволі проблемним та небезпечним.

Найпоширенішим механізмом у сучасних мікросервісних вебдодатках є використання протоколів OAuth 2.0 та OpenID Connect. OAuth 2.0 дозволяє реалізувати авторизацію, коли користувач надає додатку обмежений доступ до своїх ресурсів без передачі пароля. OpenID Connect розширює цей підхід та

забезпечує автентифікацію на основі токенів. Завдяки використанню JSON Web Token (JWT) з’являється можливість передавати маркери доступу між сервісами у компактній та криптографічно захищеній формі.

Для організації єдиного входу (Single Sign-On) у мікросервісних програмах застосовуються спеціалізовані інструменти, серед яких Keycloak, Auth0 або Okta. Keycloak забезпечує керування користувачами та ролями до підтримки SAML та OpenID Connect, що дозволяє йому бути придатним для інтеграції у корпоративні середовища. Використання таких інструментів дозволяє централізувати політику безпеки, водночас не порушуючи автономності окремих мікросервісів [13 — 14].

Ще одним важливим елементом мікросервісної архітектури є використання API Gateway (Шлюз), який виконує роль єдиної точки входу для всіх запитів користувачів. Gateway не забезпечує маршрутизацію запитів до відповідних мікросервісів і реалізує взаємодії, пов’язані з безпекою, перевіркою токенів авторизації, шифрування даних. Тут відбувається інтеграція з сервісами ідентифікації, завдяки чому внутрішні мікросервіси позбавлені необхідності самостійно реалізовувати складні механізми безпеки [15].

Продовжуючи тему взаємодії та ідентифікації сервісів, далі буде розглянуто сервіси виявлення або Discovery services. Такі сервіси слугують для забезпечення динамічного виявлення інших сервісів, оскільки кількість екземплярів кожного з них може змінюватися в залежності від навантаження. Саме тому у мікросервісних середовищах застосовується концепція Discovery services, яка дозволяє автоматизувати процес пошуку та взаємодії між сервісами.

Архітектурно сервіс виявлення може бути реалізований за двома основними моделями — клієнтською та серверною. У клієнтській моделі кожен мікросервіс самостійно звертається до Discovery service для отримання актуальних даних про розташування інших сервісів. У серверній моделі запити

проходять через спеціальний маршрутизатор, який взаємодіє з сервісом виявлення і повертає клієнтам необхідні адреси.

Серед найпоширеніших інструментів для реалізації сервісів виявлення варто виділити Netflix Eureka, Consul та Apache Zookeeper. Eureka розвивався як частина екосистеми Netflix OSS і тісно інтегрується зі Spring Cloud. Його головною перевагою є простота у використанні та легкість інтеграції в Java-застосунки. Consul, розроблений компанією HashiCorp. Такий інструмент забезпечує зберігання конфігурацій та механізми перевірки стану сервісів. Apache Zookeeper використовується як універсальний координатор розподілених систем і забезпечує високу надійність завдяки механізмам узгодженості даних [16].

Як було описано вище, мікросервісний додаток це сукупність окремих програм які взаємодіючи між собою створюють цілу систему. Тому для взаємодії між ними треба розглянути механізм обміну повідомленнями.

Реалізація такого механізму полягає у використанні проміжного сервісу-брокера повідомлень, який приймає дані від одного сервісу і передає їх іншим за допомогою черг. Такий підхід дозволяє можливість обробки великої кількості подій та впровадження повторної доставки у випадку збоїв.

Одним із найпоширеніших інструментів для передачі повідомлень є RabbitMQ. Він базується на протоколі AMQP і надає гнучкі засоби маршрутизації повідомлень, дозволяючи налаштовувати як прості черги, так і складні топології взаємодії між сервісами. RabbitMQ широко використовується там де важливими є надійність доставки та можливість підтвердження отримання повідомлення.

Наступний інструмент Apache Kafka орієнтований на високошвидкісну обробку потоків даних у режимі реального часу. Kafka особливо ефективний в аналітичних системах, сервісах збору телеметрії та архітектурах. Здатність працювати з мільйонами подій на секунду робить Apache Kafka потужною технологією для створення мікросервісних додатків.

Apache Pulsar відзначається простою архітектурою та високою швидкодією, що робить його придатним для мікросервісів, де критично важливий час відгуку. Він поєднує можливості потокової обробки.

Розгортання мікросервісних вебдодатків є найважливішим етапом розробки та експлуатації подібного роду програмного забезпечення. Оскільки такі програми мають в собі інші сервіси та утворюють складну систему, необхідно застосувати інструмент, що дозволить запускати їх [17].

В цьому можуть допомогти технології по типу Docker, що дозволяють ізолювати середовище виконання кожного сервісу, зменшуючи залежність від конфігурації операційної системи. Кожен контейнер містить необхідні компоненти для забезпечення справної та ефективної роботи кожної складової мікросервісного додатку.

Коли потрібне управління великою кількістю контейнерів, застосовується оркестрація. Kubernetes надає інструменти для автоматичного масштабування, балансування навантаження, відновлення у випадку збоїв. Цей інструмент дозволяє зосередитися на розробці, покладаючи рутинні завдання управління інфраструктурою на платформу [18].

У хмарних середовищах часто використовуються керовані рішення для оркестрації контейнерів такі як Amazon Elastic Kubernetes Service (EKS), Google Kubernetes Engine (GKE) та Azure Kubernetes Service (AKS). Вони зменшують витрати на адміністрування та дозволяють інтегрувати розгортання мікросервісів із іншими сервісами відповідної платформи.

## 1.3 Огляд прикладів застосування мікросервісної архітектури

Мікросервісна архітектура є досить поширеною методикою для розробки сучасного програмного забезпечення. Багато відомих технологічних компаній використовують мікросервіси для різних цілей, включаючи спрощення архітектури, прискорення розробки програмного забезпечення, підвищення швидкості реакції та здатності систем до оновлення. Розвиток методів

автоматизації інфраструктури також сприяло поширенню цієї архітектури. Ось деякі лідери ринку, які використовують мікросервісну архітектуру у своїх системах.

Комерційний сайт компанії Amazon на момент свого створення був монолітом із заплутаними зв'язками між багаторівневими операціями. Це вимагало ретельної розробки програмного забезпечення при кожному оновленні або виконанні завдання масштабування, щоб не допустити збоїв. Така стратегія була звичайною на той час. Монолітна архітектура використовувалася розробки навіть масштабних технологічних ініціатив, здійснюваних великими корпораціями. На рисунку 1.1 зображено вигляд головної сторінки Amazon:

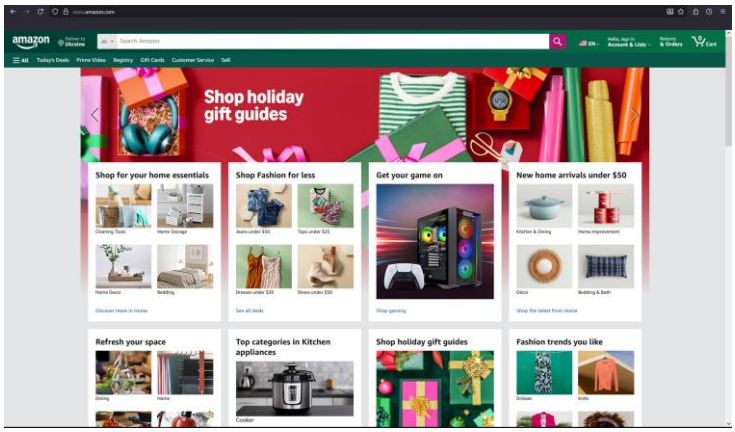


Рисунок 1.1 — Зображення головної сторінки Amazon

Але в міру зростання бази користувача Amazon наймала додаткових людей для роботи над нею, що призвело до збільшення кодової бази. В результаті архітектуру стало складніше змінювати, що збільшило витрати на обробку та подовжило життєвий цикл розробки.

Щоб вирішити ці проблеми, Amazon розбила свої великі монолітні системи на дрібніші автономні корпоративні програми. На перших етапах розробники вивчали вихідний код та виділяли ділянки коду, що виконують єдине завдання. Після цього ці блоки було укладено у шар вебсервісів. Наприклад, для різних кнопок та калькуляторів були створені різні модулі. В

даний час Amazon розробляє та розповсюджує такі продукти, як AWS та Apollo, що спрощує впровадження мікросервісів іншими підприємствами [19]. На рисунку 1.2 зображено структуру мікросервісів на базі AWS:

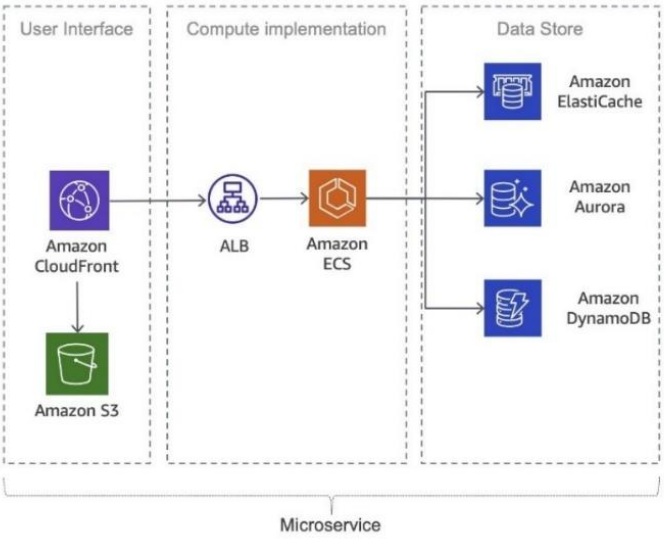


Рисунок 1.2 — Зображення структуру мікросервісів на базі AWS

Netflix, як і Amazon, є предтечею в галузі архітектури мікросервісів. Коли гігант потокового мовлення зіткнувся з кількома проблемами масштабованості та перебоями в обслуговуванні, у 2008 році розпочалося його переселення. На рисунку 1.3 зображено вигляд головної сторінки Netflix:

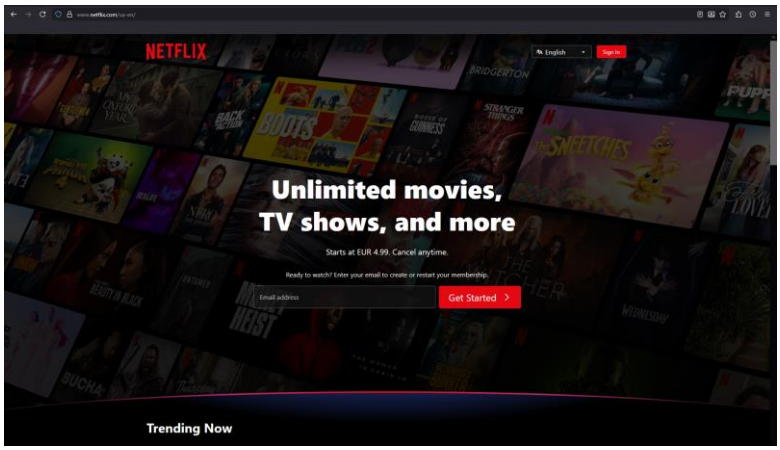


Рисунок 1.3 — Зображення головної сторінки Netflix

Коли система управління даними Netflix дала збій, заблокувавши доставку DVD-дисків передплатникам на три дні, компанія зрозуміла, що

настав час переходити на мікросервіси. Netflix вибрала Amazon Web Services (AWS) як постачальника хмарних сервісів для досягнення цілей міграції у хмару.

У 2009 році Netflix почала перетворювати свою монолітну архітектуру по одній функції за раз на архітектуру мікросервісів. Вона розпочала перетворення своєї платформи для створення сценаріїв фільмів, не орієнтованої на користувача, для роботи в хмарі AWS з використанням одиночної архітектури мікросервісів. Незабаром після цього компанія розпочала переведення своїх споживчих систем на мікросервіси та завершила цей процес у 2012 році [20].

Через перешкоди для розширення компанія Uber також вирішила вийти з монолітної структури, подібно до Amazon і Netflix. Мережа спільних поїздок зіткнулася з труднощами, пов'язаними з об'єднанням міжнародних операцій, що швидко розширюються, а також з неефективністю при створенні та впровадженні нових послуг. Дійшло до того, що навіть базові оновлення та коригування системи вимагали залучення висококваліфікованих програмістів через складну структуру програми [21]. На рисунку 1.4 зображено вигляд головної сторінки Uber:

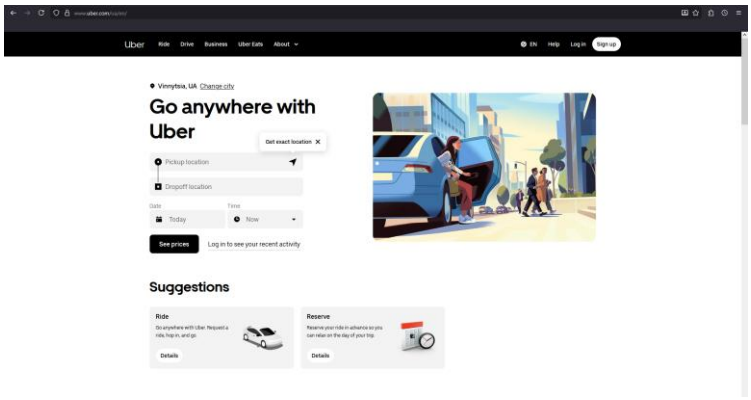


Рисунок 1.4 — Зображення головної сторінки Uber

Для вирішення проблем Uber розділив свій монолітний додаток на мікросервіси, що працюють на базі хмарної архітектури. Незабаром були

спеціальні мікросервіси для операцій компанії, таких як управління даними про поїздки і управління клієнтами.

### 1.4 Постановка задач дослідження

Провівши аналіз сучасного стану розвитку технологій побудови мікросервісних вебдодатків, було визначено основні завдання, які необхідно вирішити в процесі дослідження. У межах роботи передбачається створення демонстраційного вебдодатку, здатного показати основні принципи побудови мікросервісних програм із застосуванням сучасних технологій та методів, які були описані в минули пунктах даного розділу. В якості демонстрації будуть представлений функціонал програми який імітуватиме вебдодаток для збереження даних користувачів. В свою чергу користувачі можуть мати доступ до ресурсів та функціоналу, що надає розроблювана програма.

Для того щоб розробити подібний вебдодаток необхідно виділити основні задачі та вирішити їх.

Першою з них це буде визначення бізнес-логіки програми та її функціоналу.

Далі необхідно визначити загальну структуру програми та розподілити складові проєкту на окремі сервіси.

Як було зазначено в пункті 1.2, наступним етапом буде вибір стеку технологій для розробки вебдодатку.

В ході розробки можуть виникнути задачі: база даних та комунікація між сервісами, які необхідно вирішити аби розробити демонстраційний вебдодаток

Далі потрібно реалізувати кожен етап розробки, що описаний в попередньому пункті. Також важливо розробити документацію для легшого аналізу програми. На кінець необхідно здійснити тестування та огляд розробленого вебдодатку на предмет ефективності та принципу роботи та здійснити загальний аналіз результатів дослідження.

# 2. ТЕОРЕТИЧНІ АСПЕКТИ ПРОЄКТУВАННЯ ТА МЕТОДИ РОЗРОБКИ МІКРОСЕРВІСНОГО ВЕБДОДАТКУ

## 2.1 Аналіз архітектурних стилів для проєктування вебдодатків

Більшість вебдодатків мають доволі широкий функціонал та можливості, що забезпечують зручність та багатозадачність для найрізноманітніших користувачів. Такі ознаки потребують багато зусиль зі сторони розробників програмного забезпечення, що виливається в доволі масштабну кодову базу. Щоб забезпечити правильний менеджмент, розроблювальний проєкт потрібно правильно розробити структуру програми. Саме тут використовуються архітектурні стилі.

Побудова архітектури додатку — це комплекс методів, який спрямований на чітке визначення побудови система. Це визначає як різні компоненти програмного забезпечення взаємодіятимуть один з одним, і відіграє ключову роль у його продуктивності, масштабованості та зручності обслуговування.

Архітектурні стилі вебдодатків включають три основні типи верхньорівневої архітектури: моноліт, мікросервіси та серверлес, що відрізняються за способом структурування та розгортання, а також різні архітектурні патерни, такі як MVC, MVP та MVVM, які пропонують готові рішення для організації компонентів застосунку та бізнес-логік [22].

Монолітний стиль архітектури передбачає створення цілісного цілісного застосунку, в якому усі функціональні компоненти взаємопов’язані та працюють у межах одного проєкту. У такій архітектурі бізнес-логіка, користувацький інтерфейс, модулі роботи з базою даних утворюють цілу структуру, що забезпечує узгодженість процесів і спрощує розробку. Завдяки своїй цілісності монолітні додатки легше створювати на початкових фазах розробки, оскільки вони не потребують складних механізмів взаємодії між компонентами. Схему монолітної архітектури вебдодатку зображено на рисунку 2.1. Попри простоту застосування, подальший розвиток монолітної архітектури супроводжується низкою труднощів.

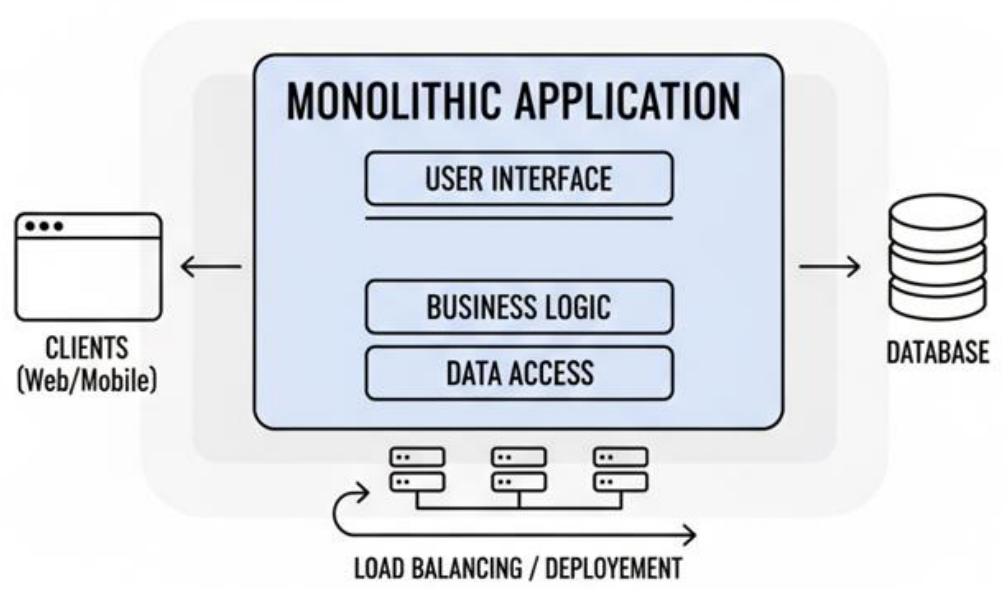


Рисунок 2.1 — Схема монолітного архітектурного стилю

Розширення функціональності такого застосунку призводить до поступового зростання його розмірів, що ускладнює процес підтримки, тестування та масштабування. Будь-які зміни потребують повторної компіляції та розгортання всієї системи. Зростання кількості користувачів і навантаження робить таку архітектуру неефективною [23].

Даний архітектурний стиль характеризується сукупною складністю системи, яка визначається сумою внутрішніх складностей окремих модулів та рівнем зв’язності між ними і може бути подана у вигляді формули (2.1):

$$C_m = \sum_{i=1}^{n} C(M_i) + \sum_{i \neq j} D(M_i, M_j), \tag{2.1}$$

де $C(M_i)$ — внутрішня складність модуля;

$D(M_i, M_j)$ — ступінь зв’язаності між модулями.

При зростанні кількості модулів складність зростає квадратично, що ускладнює масштабування.

Архітектура серверлес у сучасній сфері програмного забезпечення розглядається як підхід, що дозволяє зосередитися виключно на бізнес-логіці та функціональних можливостях системи. Основна ідея полягає в тому, що розробник створює програмні модулі у вигляді функцій, які виконуються у хмарному середовищі, а управління ресурсами, балансування навантаження, масштабування та забезпечення безперервної роботи здійснюється постачальником хмарних послуг. Такий підхід забезпечує високу гнучкість і дозволяє швидко адаптувати систему до змін у вимогах користувачів та умовах експлуатації. Схему серверлес архітектури вебдодатку зображено на рисунку 2.2. Однією з ключових характеристик серверлес є модель автоматичного масштабування, за якої кількість ресурсів динамічно змінюється залежно від фактичного навантаження на застосунок.

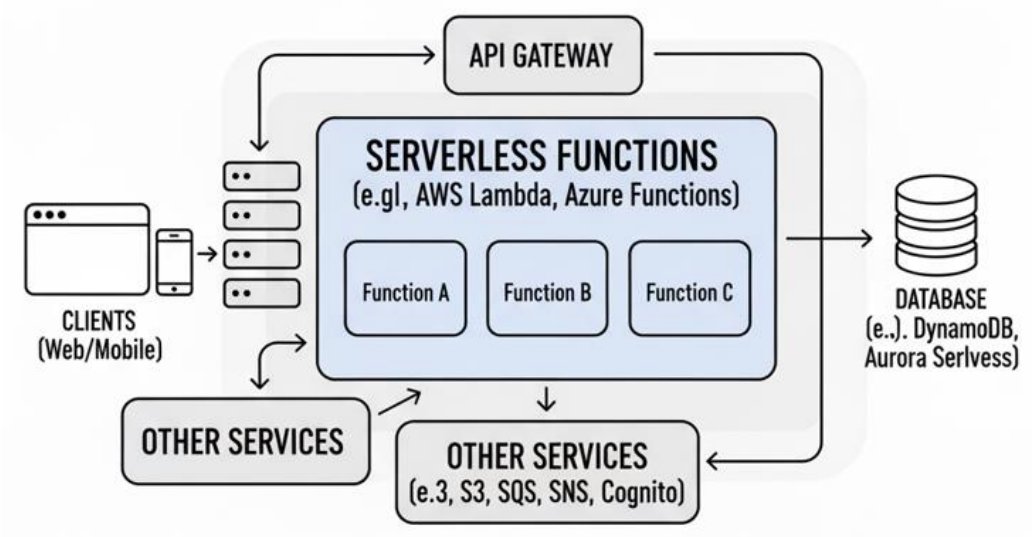


Рисунок 2.2 — Схема серверлес архітектури

Це означає, що система здатна ефективно обробляти як невелику кількість запитів, так і пікові навантаження без потреби у попередньому плануванні чи ручному розширенні інфраструктури. Такий підхід позитивно впливає на оптимізацію витрат, оскільки замовник сплачує лише за фактичне використання ресурсів, що робить архітектуру економічно привабливою у порівнянні з традиційними [24].

Водночас, серверлес архітектура має і певні обмеження. Виконання функцій у безсерверному середовищі залежить від постачальника хмарних сервісів, що створює ризик прив’язки до конкретної платформи та знижує рівень контролю над інфраструктурою.

Даний архітектурний стиль характеризується виконанням незалежних функцій, інтенсивність викликів яких визначає загальне навантаження на систему та може бути описана наступною формулою (2.2):

$$R = \sum_{i=1}^{n} \lambda_i, \tag{2.2}$$

де $\lambda_i$ — інтенсивність викликів функцій.

Мікросервісна архітектура є одним із найпоширеніших сучасних підходів до проєктування та розробки програмних систем, що ґрунтується на ідеї поділу застосунку на набір відносно невеликих, автономних сервісів, кожен з яких виконує чітко визначену бізнес-функцію. Кожен мікросервіс розробляється, тестується, розгортається та масштабується незалежно від інших, що забезпечує високу гнучкість системи та значно підвищує її стійкість до змін. Така архітектура дає можливість організувати процес розробки так, щоб окремі команди могли працювати паралельно над різними сервісами, використовуючи різні технології та підходи, що сприяє підвищенню ефективності й швидкості впровадження нових рішень. Схему серверлес архітектури вебдодатку зображено на рисунку 2.3.

Однією з головних переваг мікросервісної архітектури є її масштабованість. На відміну від монолітних систем, де збільшення ресурсів стосується всього застосунку, у мікросервісній архітектурі масштабування може здійснюватися вибірково для тих компонентів, які відчувають найбільше навантаження.

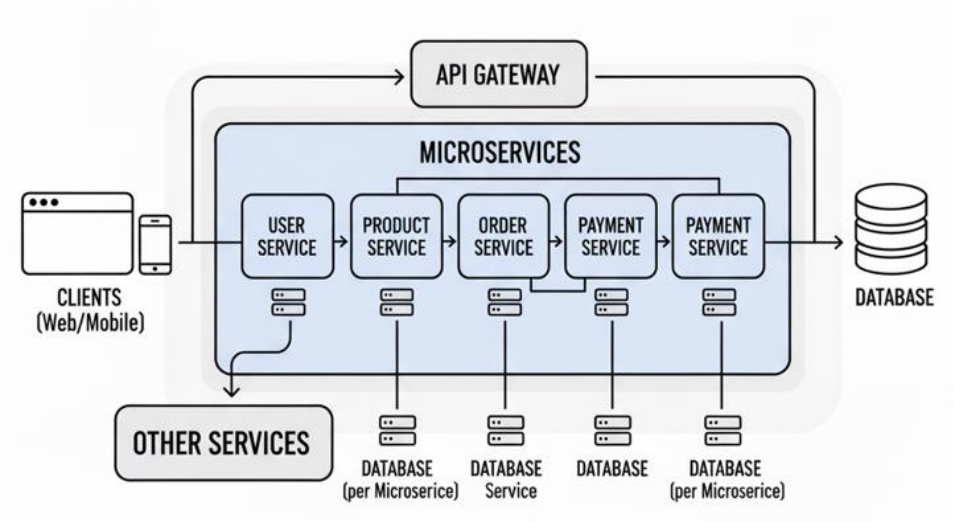


Рисунок 2.3 — Схема мікросервісної архітектури

Це дозволяє оптимізувати використання ресурсів та зменшити витрати на обслуговування інфраструктури. Можливість поступового оновлення або заміни окремих сервісів без зупинки всієї системи робить такий підхід надзвичайно зручним у процесі довгострокової експлуатації.

Завдяки своїй структурі мікросервісна архітектура значно підвищує надійність застосунку. Вихід з ладу одного сервісу, як правило, не призводить до зупинки всієї системи, оскільки інші її компоненти продовжують функціонувати. Це дозволяє створювати відмовостійкі рішення, що актуально для критично важливих сфер.

Разом з тим, впровадження мікросервісної архітектури супроводжується і певними недоліками. Розподілення системи призводить до зростання складності управління нею. Забезпечення надійності комунікацій, узгодженості даних та цілісності транзакцій потребує використання спеціалізованих технологій і ретельного планування. Ще одною проблемою є централізоване спостереження та моніторинг за сервісами, оскільки в такій архітектурі потрібно слідкувати за помилками та миттєво вирішувати їх. Важливою задачею є організація безпеки, адже збільшення кількості точок взаємодії підвищує ризик потенційних атак [25].

Даний архітектурний стиль характеризується внутрішньою складністю окремих сервісів та кількістю інтеграційних взаємодій між ними, що може бути подано у вигляді наступної формули (2.3):

$$C_{ms} = \sum_{i=1}^{n} C(S_i) + \sum_{i \neq j} I(S_i, S_j), \tag{2.3}$$

де $C(S_i)$ — показник внутрішньої складності $i$-го мікросервісу, що може визначатися кількістю реалізованих функцій;

$I(S_i, S_j)$ — кількість або інтенсивність інтеграційних взаємодій між $i$-м та $j$- м мікросервісами.

Мікросервісна архітектура сильно пов’язана з концепціями DevOps, вони забезпечують технічні засоби для швидкого розгортання і підтримки численних сервісів. Використання контейнеризації дозволяє автоматизувати процес управління інфраструктурою, забезпечити стабільність та масштабованість.

## 2.2 Етапи проєктування мікросервісного вебдодатку

Процес розробки мікросервісного вебдодатку має багато етапів, де кожен з них є логічним продовженням попереднього та формує фундамент для розробки наступних кроків. У сучасній сфері інформаційних технологій велике значення набуває моделювання архітектури, оскільки воно визначає структурну організацію коду, забезпечує масштабованість, стійкість до збоїв і можливість подальшого розвитку програми відповідно до нових вимог до розроблюваного проєкту.

Першим етапом в розробці є визначення цілей створюваного застосунку та формулювання його функціональних вимог. Цей крок пов’язаний з аналізом бізнес-логіки, однак у контексті мікросервісної архітектури він має специфічне

значення, оскільки потребує чіткого розмежування меж відповідальності окремих сервісів. Визначення вимог охоплює не лише перелік очікуваних функцій, а й характеристики надійності, доступності, рівня продуктивності, вимог до безпеки, здатності інтегруватися з іншими системами та підтримки гнучкого масштабування.

На наступному етапі опрацьовується логічна структура вебдодатку. Саме тут потрібно визначити поділ всього проєкту на окремі автономні сервіси, кожен з яких відповідає за певну бізнес-функцію або технічну складову. Важливою умовою є збереження зв'язку між сервісами та уникнення надмірної взаємозалежності, адже це визначає можливість незалежного розгортання, оновлення. У рамках даного процесу застосовується підхід доменно-орієнтованого проєктування, що дозволяє виділити чіткі межі відповідальності.

Після окреслення структури окремих сервісів, потрібно розробити архітектурну модель їхньої взаємодії. На цьому етапі потрібно налагодити маршрутизацію запитів та організацію єдиної точки доступу до системи. Тут допоможе технологія API Gateway, яка розглядалася в попередньому пункті цього розділу. Завдяки йому можна налагодити єдину точку входу завдяки, якій можна впровадити механізм авторизації та дозволити користувачеві з певними правами отримувати доступ до певних сервісів.

Важливим етапом розробки є управління даними. На відміну від монолітного архітектурного стилю, де база даних зазвичай є частиною самої програми, у мікросервісній архітектурі застосовується принцип незалежності збереження даних. Кожен сервіс має власне сховище, яке відповідає його функціональним завданням і може реалізовуватися за допомогою різних типів баз даних відповідно до характеру даних і вимог до швидкості обробки. Важливим аспектом є узгодження транзакцій та забезпечення цілісності інформації в умовах децентралізованої структури [26].

Далі формується технічна основа майбутнього середовища розгортання. На цьому етапі розглядаються процес контейнеризації та оркестрації, що є

невід’ємною частиною для мікросервісних систем. Використання контейнерів дозволяє забезпечити уніфіковане середовище виконання незалежно від інфраструктурних відмінностей. Тут також враховуються механізми спостереження, моніторингу та логування, які дозволяють оперативно реагувати на зміни стану системи та своєчасно усувати потенційні проблеми.

Наступним етапом проєктування мікросервісного додатку є створення механізму безпеки. Автентифікація, авторизація та захист каналів комунікації становлять ключові складові, які гарантують конфіденційність даних і запобігають несанкціонованому доступу. У мікросервісних додатках механізм безпеки реалізується  на рівні окремих сервісів або на рівні шлюзу [27].

Кінцевим етапом у проєктуванні є моделювання процесів розробки та підтримки вебдодатку. За допомогою концепції DevOps можна забезпечити інтеграцію процесів створення, тестування й експлуатації мікросервісних програм. Це вимагає безперервного вдосконалення та швидкого реагування на зміну умов. Автоматизація конвеєрів збірки, тестування та розгортання гарантує стабільність і скорочує час від моменту внесення змін до їхнього впровадження в реальне середовище.

## 2.3 Визначення загальної структури вебдодатку

Для якісного дослідження використання мікросервісної архітектури вебдодатків, є доцільним створення демонстраційного вебдодатку, що застосовує методи і технології що притаманні такій архітектурі.

Основними атрибутами для розроблюваного додатку є використання бази даних, робота з користувачами, Можливість автентифікуватися та авторизовуватися в систему, використовувати ресурси програми та можливість змінювати та створювати нові. Більшої різноманітності функціоналу може надати можливість роботи з файлами XSLX та PDF, що дозволить користувачеві зручніше працювати з даними.

В якості теми демонстраційної програми було прийнято рішення обрати музичний вебдодаток. Бізнес-логіка цього додатку передбачає наявність бази з музичними композиціями та повною інформацією про них. Користувач має змогу створити профіль та переглядати й додавати до власної бібліотеки композиції. Також він може переглядати інші профілі та дивитися усі композиції які вони додали в свою бібліотеку. Така тема може в повній мірі показати застосування мікросервісного архітектурного стилю.

Також у цього додатку будуть наявна роль адміністратора, який матиме змогу управляти користувачами та бібліотекою з музикою. Він також матиме доступ до управління користувачами за допомогою XSLX файлів, що зробить процес зручним та дозволить здійснювати звітність. Адміністратор матиме доступ до роботи з PDF файлами, що дозволить робити snapshot (зліпок) певного сегменту вебдодатку.

Кожен користувач матиме змогу реєструватися в системі та в подальшому матиме доступ до сторінок, відповідно зі своєю ролью, через механізм JWT. Програма формуватиме токен, завдяки якому можна буде переходити по різним ендпойнтам.

Оскільки вебдодаток буде розгортатися на сервері, то є можливість зробити його кросплатформенним, що дозволить користувачам інших операційних систем працювати з ним. Додаток передбачений для роботи в браузері, тому кросплатформенність буде впливати на збереження та завантаження файлів. В подальшому вебдодаток можна модифікувати в окреме програмне забезпечення, що зробить його незалежним від браузеру.

Визначення загальної структури та розподіл складових є важливим етапом в розробці демонстраційного вебдодатку. Не маючи чіткого алгоритму побудови архітектури, розробка подібного роду програмного забезпечення є досить проблематичною та має негативні наслідки. Тому проаналізувавши вихідні дані про тему проєкту та його функціонал, які були описані в попередньому пункті, можна виділити ключові складові вебдодатку.

Вебдодаток матиме три основні сутності, які зберігатимуть інформацію. Одна стосуватиметься користувачів і зберігатиме усі потрібні дані. В цю сутність також входитиме інформацію про ролі користувачів.

Далі буде сутність відносно музичних композицій. Тут буде поділ на самі композиції та їх лейбли (фірма звукозапису). Річ у тім, що один і той самий лейбл може бути причетним до різних композицій, тому доцільно буде їх розділити.

Далі потрібно зробити розподіл сервісів розроблюваного додатку. Для цього необхідно виділити основні групи компонентів додатку вебдодатку та здійснити розподіл.

Перша група сервісів буде стосуватися адміністрування та інфраструктури самого мікросервісного вебдодатку. В цю групу входитимуть: шлюз, сервіс виявлення, сервіс моніторингу. Шлюз (Gateway) є необхідним компонентом для створення точки входу. Окрім цього, через нього здійснюватиметься перевірка токенів та маршрутизація між сервісами проєкту. Далі буде розглянуто сервіс виявлення (discovery service), в якому буде визначення інших працюючих сервісів. Він дозволить відслідковувати які сервіси працюють та скільки екземплярів працює на даний момент. Даний модуль є необхідним елементом мікросервісної архітектури. Останнім компонентом з групи буде сервіс моніторингу. Його наявність ґрунтується на тому, що під час роботи мікросервісного додатку, можливо траплятимуться збої самих сервісів. Для полегшення вирішення проблем, застосовується сервіс моніторингу, що дозволить в режимі реального часу відслідковувати роботу, стан та продуктивність кожного сервісу додатку. Такий підхід дозволятиме швидко реагувати на несправності системи та вчасно усувати їх.

Наступна група компонентів являє собою сервіси що реалізують бізнес-логіку. Серед таких складових наявні сервіс користувача, сервіс адміністратора, сервіс музичних композицій та сервіс авторизації. Подібний розподіл опирається на особливості бізнес-логіки самого вебдодатку, яка була

розписана в минулому пункті. Тому відповідно до такого розподілу було створено саме чотири сервіси.

Останньою групою буде сукупність сторонніх програм, які допомагатимуть здійснювати фективну роботу вебдодатку. До цих сервісів можна віднести бази даних, брокери повідомлень та інше.

На рисунку 2.4 приведена приблизна схема сервісів вебдодатку та як вони повинні комунікувати. Точкою входу слугує шлюз (API-gateway) в якому також проходить валідація токену, для подальшого визначення доступу користувача. Після цього, відповідо до токену авторизації, користувач має або немає доступ до тих чи інших сервісів.

На рисунку 2.4 також показано яку саме бізнес-логіку мають сервіси. Зовнішніми сервісами позначені різні бази даних, брокери повідомлень та інші інструменти.

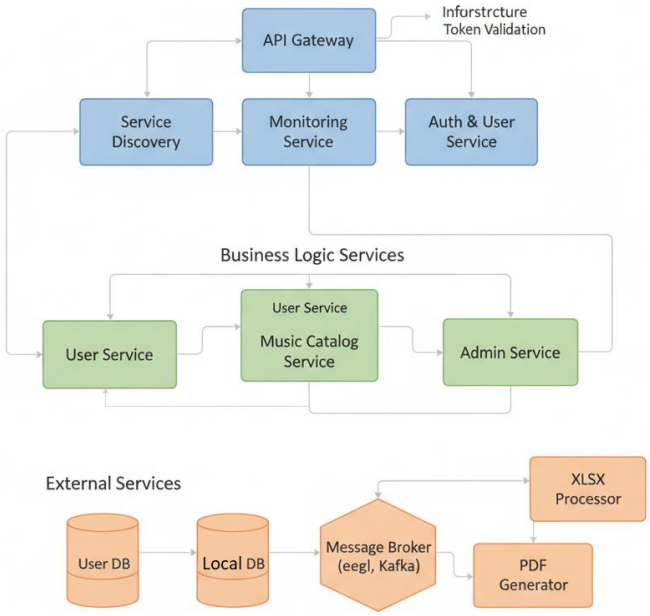


Рисунок 2.4 — Зображення приблизної схеми вебдодатку

Таким чином було визначено та розподілено основні сервіси розроблюваного вебдодатку. Виявлено три основних групи сервісів, що допоможуть структуровано розробити програму, та здійснювати підтримку роботи.

# 3 РОЗРОБКА ПРОГРАМНИХ СКЛАДОВИХ ВЕБДОДАТКУ

## 3.1 Вибір стеку технологій для розробки вебдодатку.

Починаючи з початку, необхідно визначитися з мовою програмування та її фреймворком. Серед різного роду аналогів було прийнято рішення використовувати мову Java 19 і фреймворк до неї Spring Boot 3.1.0. Такий вибір зумовлений зручною та сучасною реалізацією бізнес-логіки, підвищенням ефективності роботи з потоками, а також гарантуванням сумісністю з новітніми стандартами екосистеми Spring.

Мова Java є однією з найпоширеніших і найстабільніших платформ для корпоративної розробки. Її дев’ятнадцята версія пропонує суттєві покращення у сфері роботи з пам’яттю, оптимізації виконання коду та підтримки нових мовних конструкцій, таких як шаблонне зіставлення (pattern matching) і розширена система записів (records). Це дає змогу підвищити надійність коду та полегшити його підтримку, що особливо важливо для складних мікросервісних систем [28].

Фреймворк Spring Boot 3.1.0 є важливим компонентом Spring-екосистеми, адаптованим до сучасних вимог мікросервісної архітектури. Він забезпечує інтеграцію з такими технологіями, як Spring Cloud, Docker і Kubernetes, що сприяє побудові масштабованих і контейнеризованих рішень. Версія 3.1.0 базується на новій гілці Spring Framework 6 і підтримує виключно Jakarta EE 10, що відповідає сучасним стандартам розробки. Крім того, у ній реалізовано повноцінну підтримку GraalVM Native Image.

На рисунку 3.1 наведена схема побудови мікросервісних додатків на базі Spring Boot 3.1.0. Використання такого поєднання технологій має низку переваг. Серед них — висока стабільність платформи, зріла екосистема бібліотек, широкий спектр інструментів моніторингу та налагодження, а також активна підтримка спільноти. Це створює сприятливі умови для побудови гнучких, надійних і масштабованих систем.

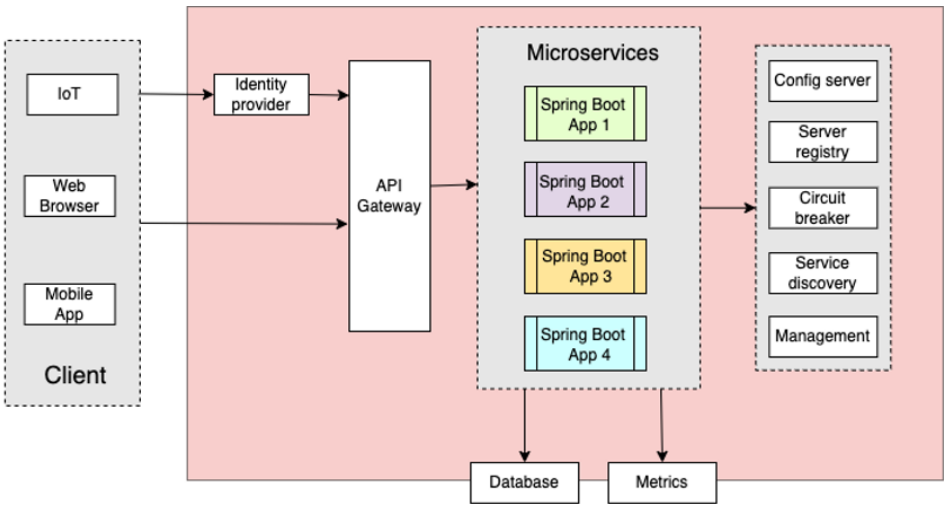


Рисунок 3.1 — Зображення схеми вебдодатку на базі Spring Boot 3.1.0

Водночас певні обмеження також мають бути враховані. Java як віртуальна машина потребує більшого обсягу ресурсів у порівнянні з мовами, орієнтованими на нативну компіляцію, а Spring Boot має відносно тривалий час запуску у традиційному режимі JVM.

У порівнянні з альтернативними технологіями, такими як Node.js чи Go, підхід на базі Java і Spring Boot вирізняється більшою формальністю, суворою типізацією та розвиненими засобами побудови корпоративної логіки [29].

Для роботи з даними потрібно застосувати систему управління базами даних. Серед усіх переглянутих технологій, було прийнято рішення використати PostgreSQL. Данна СУБД виступає основною системою керування реляційними базами даних, що забезпечує повну підтримку транзакційності, цілісності та консистентності даних. PostgreSQL відзначається високим рівнем гнучкості завдяки підтримці розширень, індексів різних типів, складних SQL-запитів, а також можливості роботи з напівструктурованими даними у форматах JSON і XML [30]. Додатково PostgreSQL забезпечує високу стабільність, має відкритий вихідний код і активно розвивається спільнотою, що робить її придатною для довготривалих корпоративних проєктів. Схема роботи PostgreSQL продемонстрована на рисунку 3.2. В якості заходів кешування та зниження навантаження на основну базу даних

використовуватиметься високопродуктивне сховище даних у пам’яті, оптимізоване для оброблення запитів із мінімальною затримкою — Redis. Дана технологія застосовується для кешування, що дозволяє суттєво скоротити час відповіді сервісів.

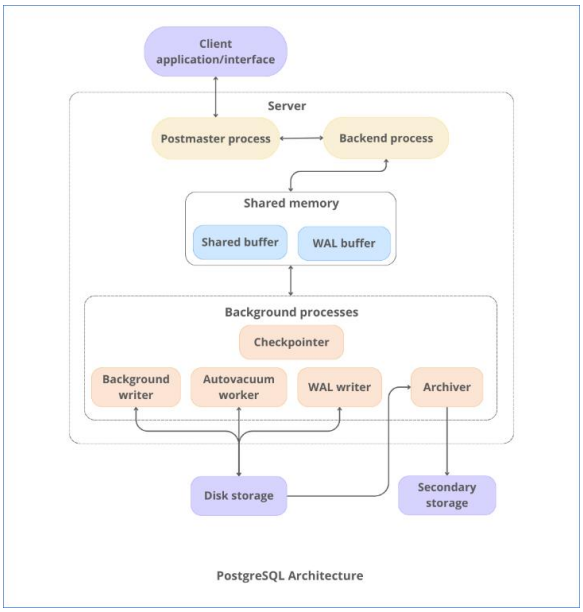


Рисунок 3.2 — Зображення інтерфейсу PostgreSQL

Redis підтримує структури даних, такі як множини, черги, хеші та списки, що розширює можливості його використання не лише як кешу, а й як інструменту для керування сесіями користувачів, черг повідомлень чи лічильників. Завдяки зберіганню даних у оперативній пам’яті Redis демонструє надзвичайно високу продуктивність [31]. Зображення роботи Redis показано на рисунку 3.3.

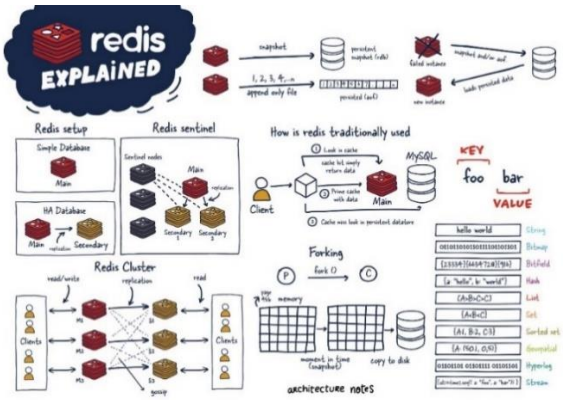


Рисунок 3.3 — Зображення роботи Redis

Комбінація PostgreSQL та Redis дозволяє досягти синергетичного ефекту: основні транзакційні дані зберігаються у стабільному реляційному середовищі, тоді як Redis забезпечує оперативний доступ до часто використовуваних об’єктів. Такий підхід відповідає сучасним архітектурним патернам, що передбачають розподіл обов’язків між різними сховищами залежно від характеру даних і вимог до швидкодії.

Разом з тим, використання цих технологій вимагає ретельного проєктування. PostgreSQL, попри свою гнучкість, може демонструвати нижчу продуктивність у сценаріях з надмірно високими навантаженнями на запис, порівняно з нативно масштабованими системами типу Cassandra чи MongoDB.

Для організації єдиного входу (Single Sign-On) у розроблюваному вебдодатку було прийнято рішення використати технологію Keycloak. Використання такого інструменту зумовлене необхідністю забезпечення єдиного, централізованого та безпечного механізму контролю доступу до ресурсів, що є фундаментальною вимогою для мікросервісних додатків. У межах мікросервісної архітектури, де кожен сервіс функціонує автономно та взаємодіє з іншими через мережеві інтерфейси, питання безпеки набуває особливого значення. Keycloak надає можливість централізовано реалізувати такі процеси, як реєстрація користувачів, автентифікація, авторизація, керування сесіями та інтеграція зі сторонніми постачальниками ідентифікації [32]. Схема застосування Keycloak у програмах, що використовують мікросервісну архітектуру зображено на рисунку 3.4. Завдяки підтримці відкритих стандартів, таких як OAuth 2.0, OpenID Connect та SAML 2.0, Keycloak забезпечує сумісність із широким спектром клієнтських додатків і сервісів. Це створює передумови для реалізації єдиної системи ідентифікації користувачів у межах усієї архітектури. Зокрема, використання токенів доступу дозволяє реалізувати механізм безпечного обміну даними між сервісами без необхідності постійного звернення до централізованого сховища

облікових записів. Такий підхід відповідає принципам zero trust security та сприяє підвищенню рівня захисту системи загалом.

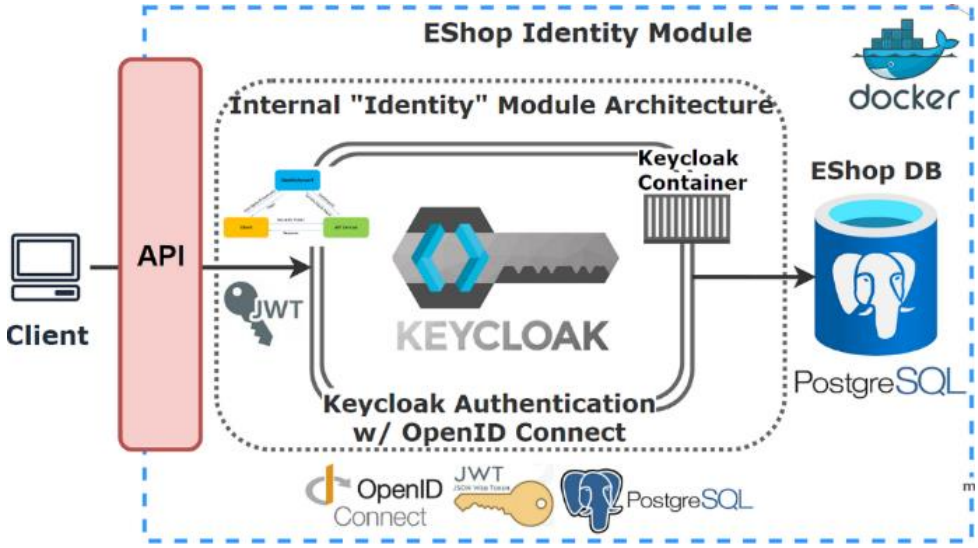


Рисунок 3.4 — Схема застосування Keycloak у програмах, що використовують мікросервісну архітектуру

Keycloak має розвинену систему керування ролями та політиками доступу, що дозволяє гнучко налаштовувати права користувачів, що дуже підходить для бізнес-логіки розроблюваного вебдодатку. Завдяки можливості інтеграції з LDAP та Active Directory він легко впроваджується в існуючу корпоративну інфраструктуру.

Як зазначалося вище, в якості фреймворка використовується Spring Boot 3.1.0. Сумісні компоненти (Spring Cloud) до нього мають інструменти для роботи зі шлюзом та виявленням сервісів.

Spring Cloud Gateway виступає сучасним рішенням для реалізації API Gateway, який виконує роль єдиної точки входу до всієї системи. Його застосування дозволяє централізовано обробляти вхідні запити, спрямовувати їх до відповідних сервісів залежно від маршруту, виконувати попередню фільтрацію, авторизацію, логування, а також трансформацію даних. Завдяки реактивній архітектурі, побудованій на базі Project Reactor, Gateway здатен ефективно обробляти велику кількість одночасних підключень, що особливо важливо для систем із високими вимогами до продуктивності та

масштабованості [33]. Схема роботи Spring Cloud Gateway показана на рисунку 3.5.

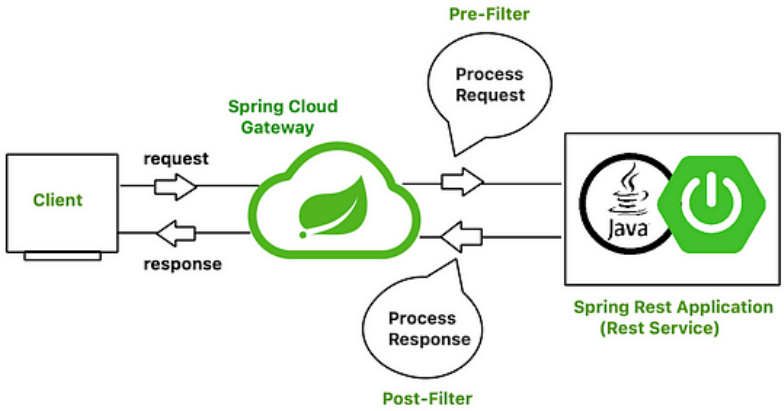


Рисунок 3.5 — Схема роботи Spring Cloud Gateway

Eureka Discovery виступає як система реєстрації та виявлення сервісів, яка автоматично відстежує активні екземпляри мікросервісів та надає інформацію про їх доступність. Це дозволяє уникнути статичної конфігурації адрес, що є неефективним у динамічному середовищі, де сервіси можуть запускатися, зупинятися або масштабуватися автоматично. Завдяки Eureka маршрутизація запитів відбувається на основі актуальної інформації про стан сервісів, що підвищує надійність системи та спрощує її супровід [34]. Схема роботи Spring Cloud Eureka Discovery показана на рисунку 3.6.

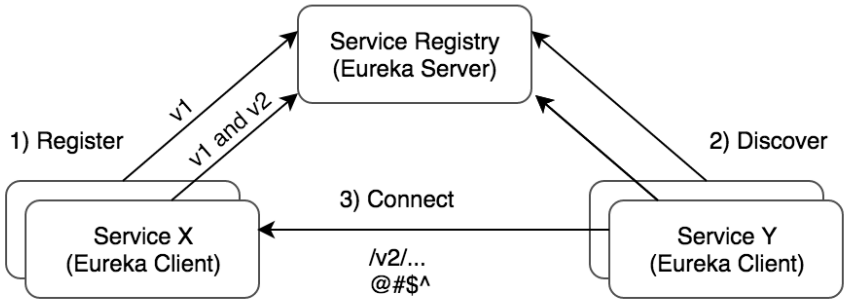


Рисунок 3.6 — .Схема роботи Spring Cloud Eureka Discovery

Оскільки розроблюваний вебдодаток має мікросервісну архітектуру, треба дати змогу комунікавти між сервісами самої програми. Це можна зробити

за допомогою брокерів повідомлень. В якості одного такого буде використаний RabbitMQ. Саме використання цього інструменту зумовлене потребою в ефективній, надійній та асинхронній взаємодії між компонентами системи. У межах мікросервісної архітектури, де кожен сервіс функціонує автономно та може мати власний цикл життя, важливо забезпечити обмін повідомленнями, який не залежить від безпосередньої доступності інших сервісів. RabbitMQ надає можливість організувати таку взаємодію через механізми черг, маршрутизації та підтвердження доставки. Схема роботи RabbitMQ показана на рисунку 3.7.

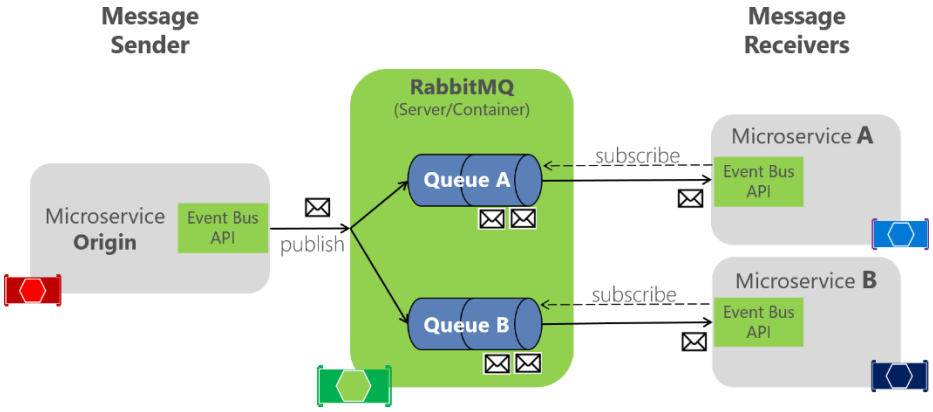


Рисунок 3.7 — Схема роботи RabbitMQ

RabbitMQ базується на протоколі AMQP (Advanced Message Queuing Protocol), який забезпечує чітку структуру обміну повідомленнями між відправниками (producers) та отримувачами (consumers) через проміжні елементи — черги та обмінники. Завдяки цьому сервіси можуть передавати повідомлення без необхідності прямої взаємодії, що сприяє зниженню зв’язності системи та підвищенню її гнучкості.

Однією з ключових переваг RabbitMQ є його здатність гарантувати доставку повідомлень завдяки механізмам підтвердження (acknowledgements), повторних спроб відправлення (retries) та збереження повідомлень на диск. Це дозволяє уникнути втрати даних навіть у випадку тимчасових збоїв сервісів-отримувачів. Крім того, RabbitMQ підтримує складні сценарії маршрутизації за допомогою різних типів обмінників, що дає змогу гнучко керувати потоками

повідомлень залежно від логіки бізнес-процесів. Завдяки цьому досягається масштабованість і стійкість системи, а навантаження може розподілятися між кількома споживачами [35].

У порівнянні з іншими технологіями, такими як Kafka, ActiveMQ чи Amazon SQS, RabbitMQ вирізняється простотою розгортання, широкою підтримкою клієнтських бібліотек і гнучкою конфігурацією. Kafka, хоч і демонструє кращу масштабованість у потокових сценаріях та аналітичних застосуваннях, має вищий поріг входу та менш гнучку модель маршрутизації. ActiveMQ є схожим за принципами, проте RabbitMQ має сучаснішу архітектуру, кращу підтримку кластеризації та вищу стабільність при роботі в хмарних середовищах.

Для розгортання вебдодатку буде застосовуватися Docker. Використання даного інструменту є обґрунтованим рішенням, оскільки ця технологія забезпечує ізольоване, відтворюване та масштабоване середовище для розгортання програмних компонентів. Docker надає можливість упаковувати кожен мікросервіс у контейнер, який містить усе необхідне для його функціонування, що усуває проблему несумісності середовищ і полегшує інтеграцію компонентів у межах однієї системи. Схема роботи Docker показана на рисунку 3.8.

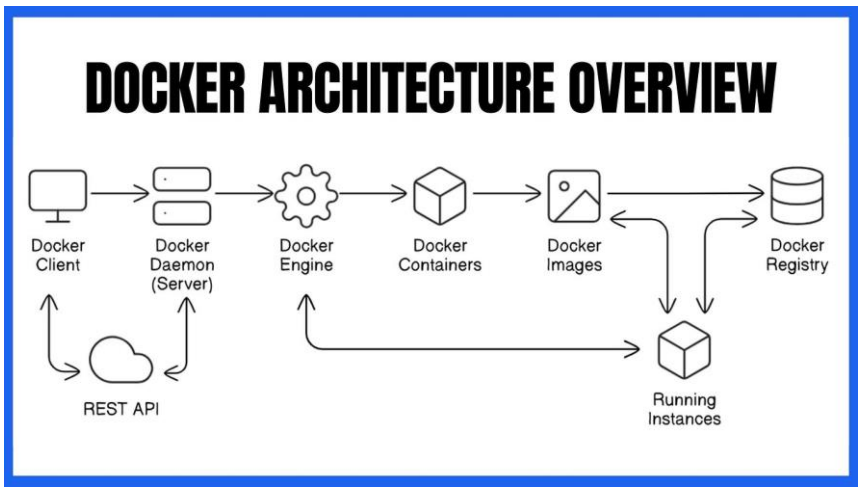


Рисунок 3.8 — Схема роботи Docker

Ключовою перевагою використання Docker є забезпечення ізоляції процесів, завдяки якій кожен контейнер працює незалежно від інших, використовуючи власні ресурси та залежності. Це дозволяє розгортати різні версії одного й того самого сервісу, експериментувати з новими технологічними стеком без ризику впливу на роботу інших компонентів і гарантує стабільність упродовж усього життєвого циклу системи. Завдяки контейнеризації досягається висока портативність: контейнери працюють однаково в середовищі розробника, на тестових стендах і у виробничому середовищі, що значно зменшує кількість помилок, пов’язаних із різницею конфігурацій.

Docker також сприяє автоматизації процесів розгортання та масштабування. Це особливо важливо для мікросервісних додатків, які мають забезпечувати безперебійну роботу в умовах змінного попиту та високих вимог до доступності. Висока швидкість розгортання контейнерів дозволяє реалізовувати підхід Continuous Integration/Continuous Deployment (CI/CD), скорочуючи час між розробкою та впровадженням нових версій сервісів [36].

До другорядних технологій які, також мають важливу роль в розробці мікросервісного вебдодатку можна віднести Swagger. е набір інструментів та стандарт для опису, документування, тестування та розробки RESTful API. Він дозволяє створити машиночитаний опис API (OpenAPI Specification), що полегшує взаємодію між службами та покращує розуміння їх функціоналу. Завдяки ньому можна автоматично зробити документацію до вебдодатку, та мати змогу без створення інтерфейсу користувача проводити тестування роботи додатку [37].

Наступними такими технологіями є Apache POI та wkhtmltopdf. Перший інструмент дозволяє працювати з документами Microsoft Office. Інший інструмент дозволить робити зліпок вебсторінки.

Останньою з технологій буде розглянуто Spring Boot Admin. Даний інструмент являє собою додаток для централізованого управління та

моніторингу Spring Boot проєктів, який надає зручний інтерфейс для доступу до функцій, що пропонуються компонентом Spring Boot Actuator. Він схожий по своєму функціоналу на Eureka Server, але має більше можливостей для моніторингу.

### 3.2 Вибір середовища розробки та налаштування компонентів

Наступним кроком буде безпосередньо розробка самого демонстраційного додатку. Для початку необхідно обрати середовище розробки. Було прийнято рішення використовувати IntelliJ IDEA: Community Edition. Інтерфейс даного середовища розробки зображено на рисунку 3.9.

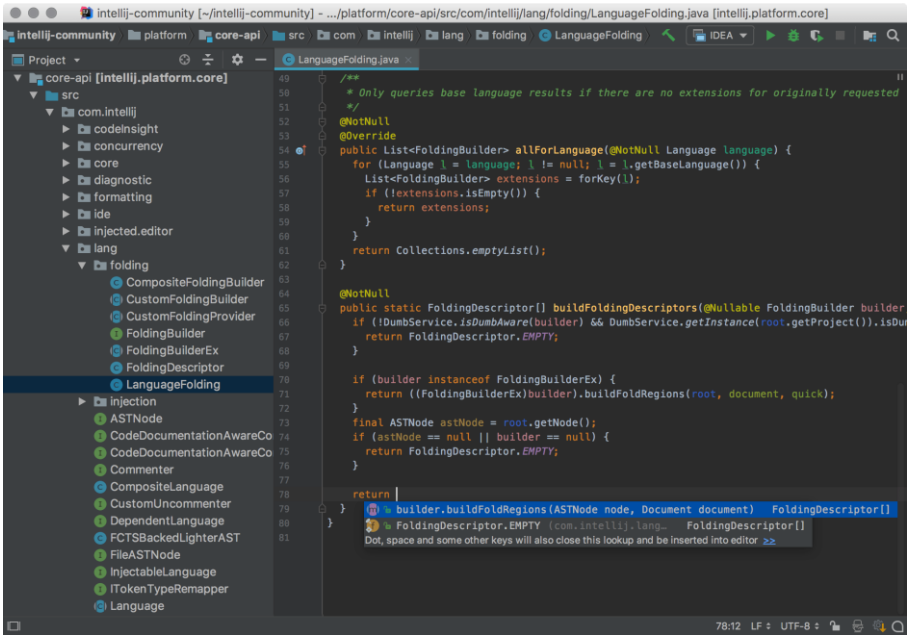


Рисунок 3.9 — Зображення інтерфейсу середовища розробки IntelliJ IDEA: Community Edition

Використання такої технології обґрунтоване стабільністю, продуктивністю та зручністю. Дане середовище розробки є оптимальною платформою для створення застосунків на базі мови Java. Позначення версії Community Edition безкоштовно надає всі необхідні інструменти для роботи з проєктами на Spring Boot, включно з підтримкою систем збирання Maven та Gradle, інтеграцією з системами контролю версій, а також можливістю

налагодження та тестування коду в реальному часі. Це забезпечує ефективне середовище для побудови сервісів, що взаємодіють між собою через REST API, черги повідомлень чи інші протоколи. Простота інтерфейсу, висока швидкодія та автоматичне доповнення коду сприяють скороченню часу розробки та підвищенню якості програмного забезпечення.

Серед головних переваг використання IntelliJ IDEA: Community Edition можна виокремити її відкритість і безкоштовність, що є особливо важливим у контексті академічних або демонстраційних проєктів. Вона підтримує роботу з багатьма популярними фреймворками, бібліотеками та мовами програмування, а також пропонує зручні інструменти для рефакторингу, аналізу коду та пошуку помилок. Крім того, інтеграція з тестовими фреймворками, такими як JUnit, дозволяє швидко перевіряти працездатність окремих сервісів у складі мікросервісного середовища. Проте, певним обмеженням Community Edition є відсутність вбудованої підтримки деяких корпоративних технологій, зокрема Spring Boot Actuator, Docker або баз даних через графічні інтерфейси, які доступні лише в Ultimate Edition. Це може ускладнити роботу з більш комплексними мікросервісними рішеннями, але в рамках розробки демонстраційного вебдодатку це не є суттєвою проблемою та не буде впливати на проєктування загалом.

У порівнянні з іншими середовищами розробки, такими як Eclipse чи Visual Studio Code, IntelliJ IDEA вирізняється більш глибокою інтеграцією з Java-екосистемою та розвиненими інструментами для роботи з Spring Boot, що робить її ефективним вибором для створення мікросервісів та додатків в цілому. Eclipse є потужним і розширюваним середовищем, однак часто поступається IntelliJ у швидкодії та інтуїтивності інтерфейсу. Visual Studio Code, у свою чергу, забезпечує більшу гнучкість і легкість, але потребує додаткової конфігурації через плагіни для досягнення еквівалентного функціоналу.

Тому, IntelliJ IDEA: Community Edition поєднує в собі оптимальний баланс між функціональністю, стабільністю та зручністю, що робить її ідеальним середовищем розробки для демонстраційного мікросервісного вебдодатку на базі Java Spring Boot.

Як зазначалося вище, вебдодаток розробляється за допомогою мови програмування Java та його фреймворку Spring Boot 3.1.0. Тому для застосування усіх потрібних інструментів, які забезпечать ефективну та стабільну роботу вебдодатку, потрібно використати систему, яка допоможе правильно їх інтегрувати безпосередньо в проєкт.

Для такої задачі було прийнято рішення використати Apache Maven. Дана технологія є засобом автоматизації роботи з програмними проєктами, який спочатку використовувався для Java проєктів. Використовується для керування та складання програм. Структура роботи Apache Maven зображена на рисунку 3.10.

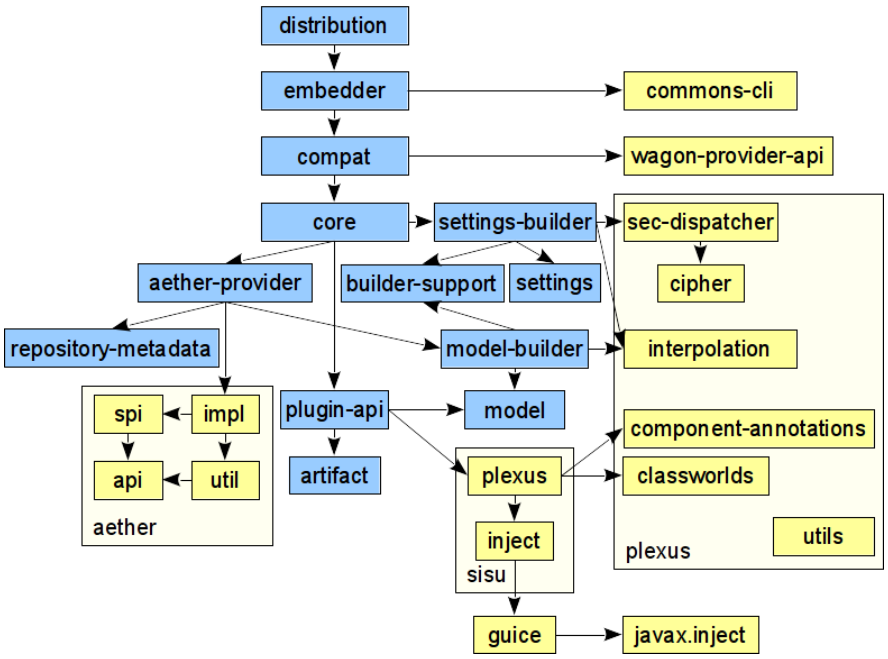


Рисунок 3.10 — Структура роботи Apache Maven

Використання Maven є обґрунтованим рішенням, оскільки цей інструмент забезпечує ефективне управління залежностями, автоматизацію процесу збирання проєкту та уніфіковану структуру, що особливо важливо для

систем із багатьма незалежними сервісами. У межах мікросервісної архітектури кожен сервіс зазвичай є окремим проєктом зі своїм життєвим циклом, набором бібліотек і конфігурацій. Maven дозволяє централізовано контролювати версії залежностей, спрощує оновлення компонентів і зменшує ризики конфліктів між ними. Крім того, використання стандартної структури каталогів і сценаріїв збирання забезпечує передбачуваність і сумісність між різними сервісами.

Завдяки декларативному підходу до конфігурації, реалізованому у файлі pom.xml, Maven забезпечує прозорість процесу збирання та дозволяє легко інтегрувати інструменти тестування, перевірки якості коду та розгортання. Це особливо актуально для мікросервісів. Крім того, наявність розгалуженої екосистеми плагінів і підтримка великої кількості середовищ розробки роблять Maven універсальним рішенням для різних сценаріїв використання. Проте основним недоліком може бути громіздкість конфігураційних файлів та менша гнучкість у порівнянні з деякими сучасними інструментами. У великих проєктах структура залежностей може стати складною, що потребує ретельного управління версіями та додаткових налаштувань для запобігання конфліктам.

Порівнюючи Maven з альтернативами, такими як Gradle чи Ant, можна відзначити, що Gradle забезпечує більшу швидкість збирання та гнучкість завдяки використанню DSL на базі Groovy або Kotlin, що дозволяє реалізовувати складні сценарії конфігурацій. Однак Maven вирізняється стабільністю, передбачуваністю та широкою підтримкою у спільноті, що робить його кращим вибором для проєктів, де важлива стандартизація та мінімізація людських помилок.

Також потрібно зазначити про наявність такого вебсервісу як Maven Repository, який містить базу всіх доступних бібліотек для підключення. Обравши потрібний інструмент можна переглянути версії його бібліотек та які версії вважаються популярними та найбільш використовуваними у спільноті.

Підключення компонентів через Maven відбувається шляхом копіювання з вебсервісу запису. Далі це вставляється в pom.xml, після чого іде автоматичне оновлення. На цьому етапі розробник уже має доступ до ресурсів встановленої бібліотеки. Інтерфейсу Maven Repository зображено на рисунку 3.11. Стосовно розроблюваного вебдодатку Apache Maven матиме ключову роль. Оскільки застосунок є мікросервісним, для більшої зручності структура проєкту в IntelliJ IDEA матиме мікромодульний вигляд, тобто всі сервіси та компоненти будуть знаходитися в одному місці.


Рисунок 3.11 — Зображення інтерфейсу Maven Repository

У разі потреби ці самі сервіси можна окремо переносити. Зображення структури вебдодатку в середовищі розробки IntelliJ IDEA продемонстровано на рисунку 3.12.

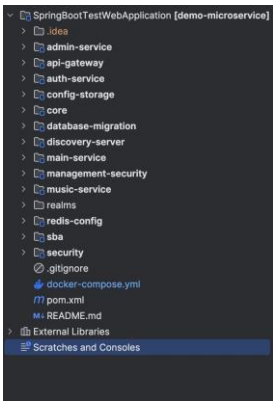


Рисунок 3.12 — Зображення структури вебдодатку в середовищі розробки IntelliJ IDEA

Тут окрім спроєктованих у минулому розділі сервісів також наявні загальні модулі з налаштуваннями. Деякі сервіси використовують однакові ресурси, тому задля простоти та ефективності, наявні модулі з цими ресурсами. В разі переносу сервісу модуль з налаштуваннями потрібно переносити також.

До спільних модулів належать модулі core, config-storage, database-migration, management-security, realms, redis-config, security. Потрібно зазначити, що при переносі конкретного сервісу не потрібно переносити усі ці модулі, лише тільки ті що пов’язані з даним сервісом. Детальніше про ці модулі та сервіси буде описано в наступному пункті даного розділу.

В якості базового налаштування компонентів для всіх проєктів, використовуватиметься загальний pom.xml, який зображений на рисунку 3.13.

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.1.0</version>
    <relativePath/>
</parent>


<groupId>com.demo-microservice</groupId>
<artifactId>demo-microservice</artifactId>
<packaging>pom</packaging>
<version>1.0-SNAPSHOT</version>
<modules>
    <module>main-service</module>
    <module>discovery-server</module>
    <module>api-gateway</module>
    <module>admin-service</module>
    <module>management-security</module>
    <module>core</module>
    <module>config-storage</module>
    <module>security</module>
    <module>auth-service</module>
    <module>database-migration</module>
    <module>sba</module>
    <module>redis-config</module>
    <module>music-service</module>
</modules>

<name>demo-microservice</name>
<url>http://www.example.com</url>
```

Рисунок 3.13 — Зображення фрагменту загального pom.xml

У цьому фрагменті, показано які модулі використовуються в додатку. Вище знаходиться spring-boot-starter-parent який надає набір конфігурацій за замовчуванням для проектів Spring Boot, керуючи версіями залежностей, налаштовує плагіни Maven та встановлює стандартні параметри для збірки проекту.

До інших компонентів можна віднести spring-cloud-dependencies — керує версіями різних компонентів Spring Cloud, spring-boot-starter-actuator — додає функціональність для моніторингу та управління додатком у виробничому середовищі та spring-boot-maven-plugin — є основним компонентом для застосунків Spring Boot, побудованих за допомогою Apache Maven. Підключення цих компонентів зображено на рисунку 3.14.


Рисунок 3.14 — Зображення фрагменту підключення компонентів в загальний pom.xml

Далі відносно функціоналу сервісу, будуть підключатися потрібні бібліотеки. Про які буде описано в наступному пункті.

До додаткових компонентів, що не увійшли у цей pom.xml, але які глобально використовуватимуться в розробці можна віднести Lombok, ModelMapper, Springdoc, spring-boot-starter-web.

Lombok — бібліотека для програмування на Java, який автоматизує написання шаблонного коду, такого як геттери, сеттери та конструктори, використовуючи анотації, що суттєво спрощує розробку.

ModelMapper — бібліотека для відображення (мапінгу) об'єктів, яка дозволяє легко та динамічно перетворювати один об'єкт іншого типу на об'єкт іншого типу в робочому середовищі.

Springdoc — бібліотека для проектів Spring Boot, яка автоматично генерує документацію REST API на основі вихідного коду та спеціальних анотацій, забезпечуючи її актуальність.

spring-boot-starter-web — стартер (starter) для Spring Boot, який містить набір стандартних залежностей для створення веб-додатків і REST сервісів.

## 3.3 Проєктування структури баз даних

Визначившись з основними технологіями, потрібно вирішити задачі, які передують розробці вебдодатку. Було визначено що структура бази даних міститиме три основні сутності: користувачі, композиції та альбоми.

Саме навколо них буде розв'язуватимуться задачі пов'язані з проєктуванням бази даних. Проєктування баз даних здійснюватиметься за допомогою СУБД PostgreSQL.

Перша сутність, що стосується користувачів, матиме три таблички, які будуть взаємопов'язані між собою. Перша з них це власне самі користувачі (user_entity). Дану табличку можна побачити на рисунку 3.15.

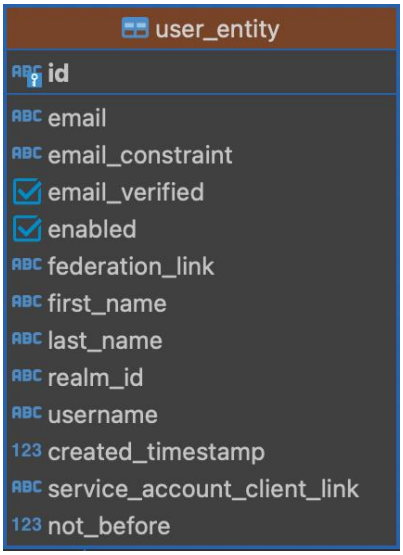


Рисунок 3.15 — Табличка user_entity

Серед різних полів, можна виділити ідентифікатор, електронну пошту, ім’я, прізвище, ім’я користувача та інше. Такий вигляд таблиці користувачів зумовлений тим, що цю базу даних в подальшому використовуватиме Keycloak. Тому поля таблиці повинні відповідати формату та вигляду бази, з якою працює Keycloak. Подібний підхід стосуватиметься інших таблиць, що пов’язані з користувачами.

Наступною буде розглянуто таблицю паролів (credential). Окрім, первинного ключа (ідентифікатор) тут наявний зовнішній ключ, що пов’язаний з ідентифікатором user_entity. Таким чином формується зв’язок між паролем та користувачем, який його використовує. Таким чином формується зв’язок між таблицями один до одного. Таблиця credential зображена на рисунку 3.16.

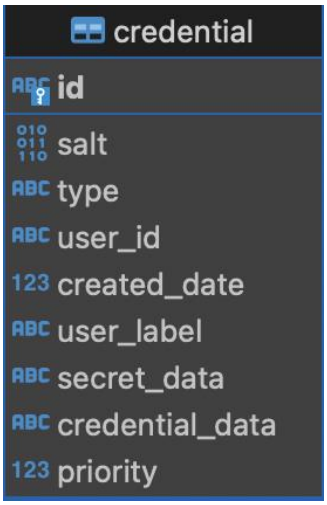


Рисунок 3.16 — Таблиця credential

Серед цих полів потрібно виділити secret_data, який містить пароль, але в цілях захисту даних користувача, пароль зберігається в хешованому вигляді, аби сторонні особи не могли мати прямий доступ до нього.

Наступна таблиця містить інформацію про ролі користувача (keycloak_role). Така таблиця вже містить дані, оскільки вони статичні, і не повинні змінюватися. Таблиця keycloak_role зображена на рисунку 3.17.

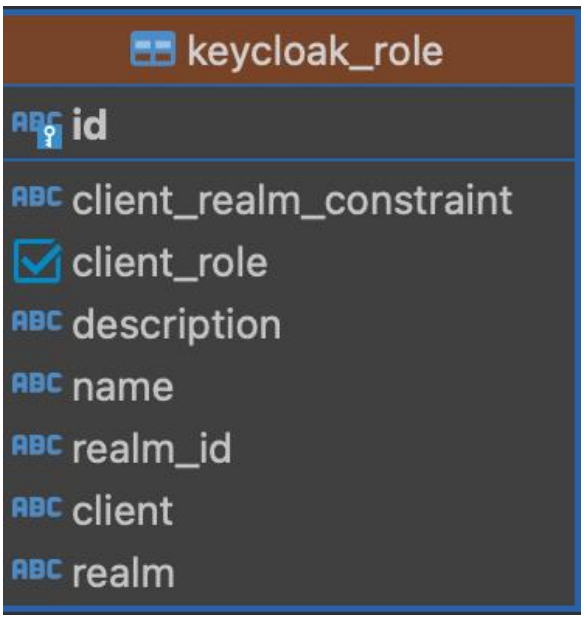


Рисунок 3.17 — Таблиця keycloak_role

Найважливіше поле даної таблиці це власне назва ролі. Дане поле матиме значення USER та ADMIN, для доступу користувачеві та адміністратору відповідно. Зв’язок між user_entity та keycloak_role здійснюється через проміжну таблицю user_role_mapping, що зображено на рисунку 3.18.

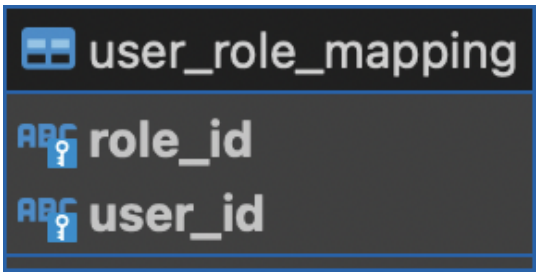


Рисунок 3.18 — Таблиця user_role_mapping

Наступна сутність стосуватиметься музичних композиції. Вона також реалізована таблицею через СУБД PostgreSQL. На цей раз вона не стосуватиметься Keycloak, тому її можна зробити в тому вигляді, який буде найбільше зручним. Тут є поля з інформацією про назву, виконавця рік, альбом, тощо. Дана таблиця називається songs, та показана на рисунку 3.19.

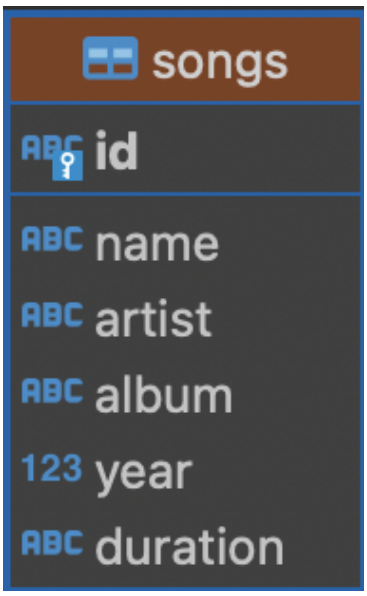


Рисунок 3.19 — Таблиця songs

Наступна сутність являє собою звукозаписну компанію (label), яка представлена одноіменною таблицею. В ній зберігається назва та ідентифікатори пісень, які відносяться до конкретного лейблу. Таблиця label зображена на рисунку 3.20. Таким чином була сформована уся структура бази даних яка представлена п’ятьма таблицями, так чи інакше пов’язані між собою.


Рисунок 3.20 — Таблиця label

В подальшому для перенесення занесених даних, без їх втрати застосовуватиметься програмний компонент FlyWay, який допоможе автоматизовано здійснювати міграцію бази даних. Загальний вигляд сутності користувача бази зображено на рисунку 3.21.

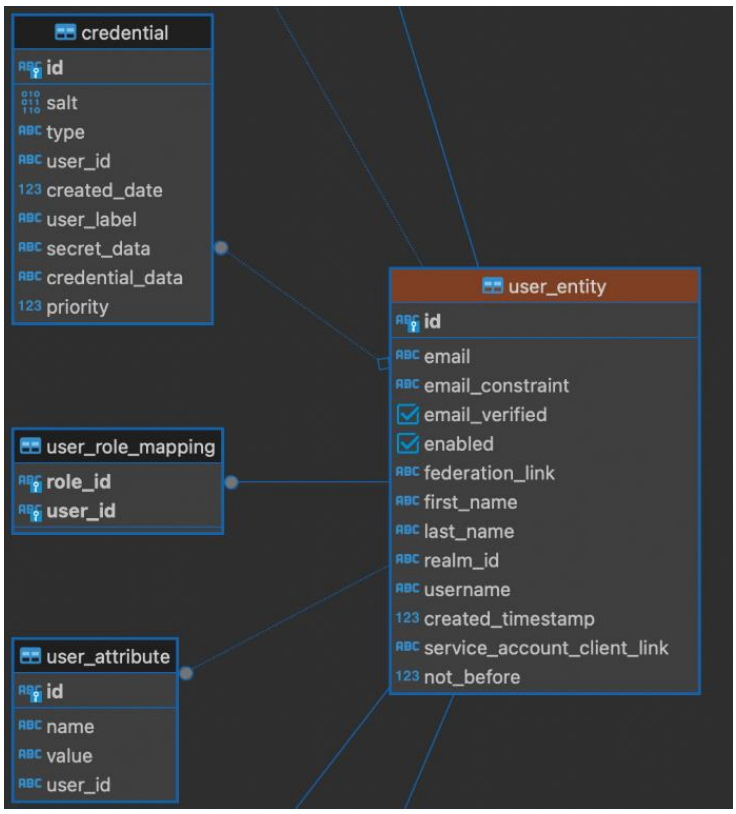


Рисунок 3.21 — Загальний вигляд бази даних сутності користувача

Потрібно зазначити, що загальна база даних пов’язаної з користувачем, містить велику кількість інших таблиць. Це пов’язано з інтеграцією Keycloak, який в свою чергу дозволяє розподіляти велику кількість інформації. В рамках розробки демонстраційного вебдодатку, огляд інших таблиць баз даних проводитися не буде.

## 3.4 Проєктування комунікації між сервісами

Наступною задачею, яку необхідно вирішити аби розробити демонстраційний вебдодаток, є реалізація обміну повідомленнями між сервісами. Подібний механізм є необхідним, оскільки це дозволить максимально ізолювати компоненти програми один від одного.

Зв’язок буде між сервісом адміністратора та сервісом авторизації. Задумка такого підходу полягає в тому що користувач, який буде реєструватися або просто входити в систему, повинен пройти відповідну процедуру, після чого сервіс авторизації передасть повідомлення сервісу

адміністратора відповідне повідомлення. Передачу такого повідомлення можливо здійснити за допомогою RabbitMQ.

Процес передачі повідомлення між сервісом авторизації та сервісом адміністратора реалізується через механізм асинхронного обміну повідомленнями, що дозволяє забезпечити слабке зв’язування між компонентами системи та підвищити її масштабованість. У межах такого підходу сервіс авторизації після успішного виконання операції реєстрації або входу користувача формує спеціальне повідомлення, яке містить необхідні дані, наприклад ім’я користувача (username), тип операції та часову мітку. Це повідомлення передається до черги повідомлень, створеної в RabbitMQ, через обмінник (exchange), який маршрутизує повідомлення згідно з визначеними правилами.

Сервіс адміністратора, у свою чергу, виступає у ролі підписника (subscriber) на відповідну чергу. Як тільки в черзі з’являється нове повідомлення, RabbitMQ передає його цьому сервісу, який обробляє отримані дані. Важливо, що передача повідомлень здійснюється незалежно від стану отримувача: навіть якщо сервіс адміністратора тимчасово недоступний, повідомлення залишатиметься в черзі до моменту його відновлення, що гарантує надійність доставки.

Таким чином між двома сервісами буде обмінник з двома чергами authQueue та registrationQueue. Через перший обмінники будуть передавитися дані стосовно авторизації користувача, а через другий — реєстрації. Схему передачі даних зображено на рисунку 3.22. Отже, механізм брокера повідомлення дозволяє уникнути прямої залежності між сервісами, оскільки сервіс авторизації не потребує знати про внутрішню логіку або стан сервісу адміністратора.

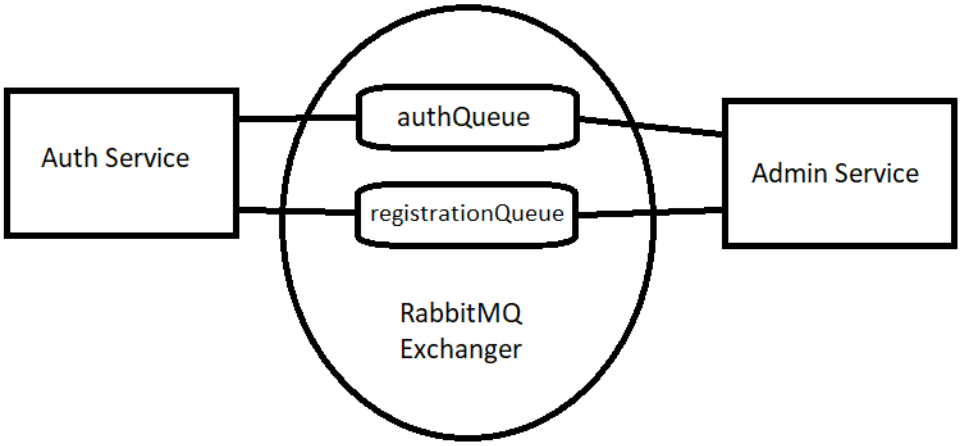


Рисунок 3.22 — Схема передачі даних

Завдяки цьому забезпечується гнучкість архітектури, можливість масштабування окремих компонентів і зниження ризику збоїв у роботі системи через відмову одного з її елементів. RabbitMQ, як брокер повідомлень, виступає центральним посередником, який керує чергами, гарантує доставку та підтримує різні шаблони маршрутизації, що дозволяє ефективно реалізувати асинхронну взаємодію між мікросервісами.

В подальшому цей брокер повідомлень можна використовувати для інших цілей, тільки для цього потрібно створити нову чергу.

## 3.5 Розробка спільних модулів вебдодатку

Як зазначалося вище, структура розроблюваної програми в середовищі розробки є мікромодульною, тобто це мікросервіс, сервіси якого на даному етапі знаходяться в одному місці. Певні сервіси використовують однакові ресурси, тому ці ресурси виокремлені в окремі модулі задля спрощення розробки, та зменшення кодової бази. При перенесені окремих модулів в інше місце, спільний модуль дублюється та переноситься разом з ним. Тому наступним кроком буде розробка та огляд спільних модулів.

Модуль core, є одним з найголовніших модулів, оскільки, він має ресурси, що використовують майже всі сервіси. Його функціонал полягає в підключенні

та роботі з базою даних користувачів. Для того щоб працювати з ресурсами бази даних необхідно підключити необхідні бібліотеки, а саме JPA (Java Persistence API) та драйвер для роботи з СУБД PostgreSQL.

Для роботи з базою даних необхідно визначити певні параметри, що зображені на рисунку 3.23.

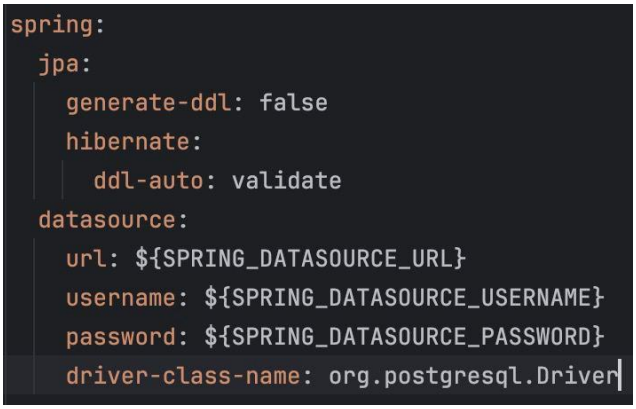


Рисунок 3.23 — Зображення параметрів для роботи з базою даних

Далі необхідно представити таблиці бази у вигляді класів Java. Для цього необхідно створити клас з відповідними полями та позначити його анотаціями @Entity та @Table. Таким чином можна зв’язати таблицю і даний клас, для того щоб оперувати інформацією можна було програмно. Подібний підхід застосовується і для інших таблиць таких як credentials keycloak_role. Представлення сутності user_entity зображено в лістингу 3.1. Далі необхідно ввести механізм транзакцій. Це можна зробити за допомогою створення інтерфейсів Java та анотацій @Repository. Також цей інтерфейс необхідно наслідувати від CrudRepository та задати потрібні методи для транзакцій.

Лістинг 3.1 — Сутність user_entity в Java

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "user_entity")
public class KeycloakEntity {
    @Id
    private String id;
```

```
    private String username;
    private String email;
    private String emailConstraint;
    private String lastName;
    private String firstName;
    private String realmId;
    private String serviceAccountClientLink;
    private String federationLink;
    private boolean emailVerified;
    private long createdTimestamp;
    private boolean enabled;
    private int notBefore;
    @ManyToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    @JoinTable(
        name = "user_role_mapping",
        joinColumns = @JoinColumn(name = "user_id"),
        inverseJoinColumns = @JoinColumn(name = "role_id")
    )
    private Set<KeycloakRole> roles = new HashSet<>();
      @OneToMany(mappedBy = "user", cascade = CascadeType.ALL,
orphanRemoval = true)
    private List<KeycloakCredential> credentials;
    public void removeCredential(KeycloakCredential credential){
      this.credentials.remove(credential);
    }
    public void removeRole(KeycloakRole role) {
      this.roles.remove(role);
    }
  }
```

Таким чином можна вносити, видаляти та оновлювати інформацію в базі даних через код Java. Аналогічно створюються такі репозиторії для інших таблиць. Механізм транзакції для таблиці user_entity зображено в лістингу 3.2.

Лістинг 3.2 —Механізм транзакції для таблиці user_entity

```
@Repository
public interface KeycloakEntityRepository extends
CrudRepository<KeycloakEntity, String> {
    Page<KeycloakEntity> findAll(Pageable pageable);
    @NonNull List<KeycloakEntity> findAll();
    @Query(
        value = "select * from user_entity where username = :username",
```

```
        nativeQuery = true
    )
    Optional<KeycloakEntity> findByUsername(String username);
    @Query(
        value = "select * from user_entity order by id asc",
        nativeQuery = true
    )
    Page<KeycloakEntity> ascSorted(Pageable pageable);

    @Query(
        value = "select * from user_entity order by id desc",
        nativeQuery = true
    )
    Page<KeycloakEntity> descSorted(Pageable pageable);
}
```

Робота з інформацією бази даних вимагає уважності та обережності, оскільки є ризик випадкового видалення чи зміни потрібних даних у ході роботи з класами, що репрезентують таблицю. Тому за для безпеки такого процесу використовується DTO (Data Transfer Object) — простий об'єкт-контейнер без бізнес-логіки, який використовується для передачі даних між різними підсистемами, рівнями або компонентами програмної системи. Такий підхід дозволяє не тільки ізолювати основний клас, а й зменшує кількість звернення до бази даних, що в свою чергу підвищує продуктивність роботи вебдодатку. DTO для user_entity зображено в лістингу 3.3.

Лістинг 3.3 — DTO для user_entity

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class KeycloakEntityDto implements Serializable {
    @Serial
    private static final long serialVersionUID = 123L;
    private String id;
    private String username;
    private String email;
    private String emailConstraint;
    private String lastName;
```

```
    private String firstName;
    private String realmId;
    private String serviceAccountClientLink;
    private String federationLink;
    private boolean emailVerified;
    private long createdTimestamp;
    private boolean enabled;
    private int notBefore;
    private Set<KeycloakRoleDto> roles = new HashSet<>();
    private List<KeycloakCredentialDto> credentials;
    public void removeCredential(KeycloakCredentialDto credential){
       this.credentials.remove(credential);
    }
    public void removeRole(KeycloakRoleDto role) {
       this.roles.remove(role);
    }
}
```

Модуль core, має допоміжні механізми роботи, серед яких є обробка виключень. При певних обставинах можуть відбуватися помилки пов’язані з відкриттям сторінок. Аби ці помилки не з’являлися необхідно розробити власний обробник виключень. Завдяки ним образу можна отримати конкретний код помилки та її опис, що суттєво спрощує вирішення проблем під час розробки розроблюваного вебдодатку. Коди помилок, які використовує обробник виключень показано в лістингу 3.4.

Лістинг 3.4 — Коди помилок, які використовує обробник виключень

```
public enum ErrorCode {
    DEMO_BAD_REQUEST("BAD_REQUEST", "Bad request. Incorrect method or params!", HttpStatus.BAD_REQUEST),
    DEMO_NOT_FOUND("NOT_FOUND", "Not found. Incorrect endpoint description!", HttpStatus.NOT_FOUND),
    DEMO_USER_NOT_FOUND("USER_NOT_FOUND", "User not found. Incorrect id or name!", HttpStatus.NOT_FOUND),
    DEMO_ROLE_NOT_FOUND("ROLE_NOT_FOUND", "Role not found. Incorrect id or name! ", HttpStatus.NOT_FOUND),
    DEMO_CREDENTIAL_NOT_FOUND("CREDENTIAL_NOT_FOUND", "Credential not found. Incorrect id or name! ", HttpStatus.NOT_FOUND);
    private final String code;
    private final String description;
```

```
    private final HttpStatus status;
    ErrorCode(String code, String description, HttpStatus status) {
        this.description = description;
        this.code = code;
        this.status = status;
    }
    public String getCode() {
        return code;
    }
    public String getDescription() {
        return description;
    }
    public HttpStatus getStatus() {
        return status;
    }
}
```

Наступний буде розглянуто модуль security. Його компоненти відповідають за кодування паролів та налаштування певних класів програм. Також тут наявні допоміжні класи, що обслуговують базу даних, тобто через них можна здійснювати маніпуляцію з даними. Сенс полягає в тому щоб під час роботи не працювати на пряму з репозиторіями, а працювати з стороннім екземпляром класу, що в свою чергу має потрібні методи даного репозиторію. Таким чином при подальшій розробці йде ізоляція між прямою роботою з базою даних та бізнес-логікою. Такого роду класи є для інших таблиць бази даних. Приклад методу допоміжного класу показано в лістингу 3.5.

Лістинг 3.4 — Коди помилок, які використовує обробник виключень

```
@Cacheable(cacheNames = "user")
  public boolean saveUser(BaseUserDto baseUserDto) {
                                Optional<KeycloakEntity>  userFromDB  =
keycloakEntityRepository.findByUsername(baseUserDto.getUsername());
        if (userFromDB.isPresent()) {
            return false;
        }

                                        KeycloakEntity  currentUser  =
modelMapper.map(buildUser(baseUserDto), KeycloakEntity.class);
        keycloakEntityRepository.save(currentUser);
```

```
                KeycloakCredential userCredential =
modelMapper.map(currentUser.getCredentials().iterator().next(),
KeycloakCredential.class);
                userCredential.setUser(currentUser);
                keycloakCredentialRepository.save(userCredential);
                return true;
            }
```

Наступним буде розглянуто модуль config-storage. Його особливість полягає в тому, що він складається з YAML файлів, в яких знаходяться налаштування до сервісів. Коли сервер запускається, від підтягує потрібні дані з цього модуля. Структура config-storage зображена на рисунку 3.24.

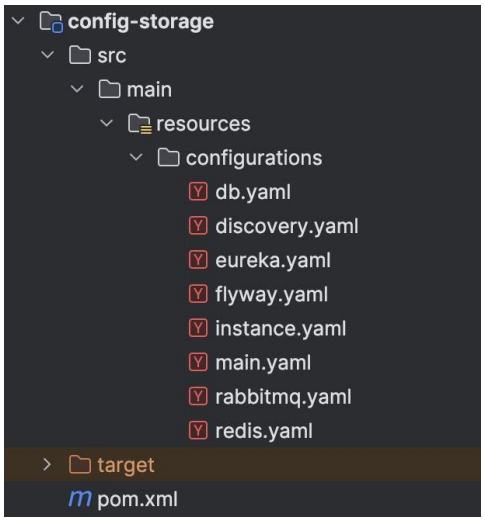


Рисунок 3.24 — Зображення структура config-storage

Модуль redis-config слугує для налаштування noSQL технології під назвою Redis. В роботі демонстраційного вебдодатку цей інструмент здійснюватиме кешування та буде додатковою базою даних для композиції. Продовжуючи тему СУБД, ще одним модулем який потрібно зазначити є database-migration. Цей модуль складається з скриптів, які створюють потрібні бази даних та їх таблиці. За допомогою технології FlyWay здійснюється міграція даних, завдяки якій при неполадках та видаленні даних, можна було відновити стан самої бази.

Останній спільний модуль який наявний в проєкті є realms. Це скоріше навіть не модуль на JSON файл який містить в собі налаштування для образу Keycloak. Цей файл не відноситься ні до одного з сервісів вебдодатку.

## 3.6 Розробка сервісів вебдодатку

Розібравши будову спільних модулів можна перейти до розробки власне самих сервісів демонстраційного вебдодатку. Першим з них буде розглянуто discovery-server. Він слугує для виявлення та реєстрації сервісів на базі Netflix Eureka Discovery. Це окремий сервіс, який захищений логіном та паролем. Маючи до нього доступ можна побачити сервіси що вже працюють та по скільки екземплярів вони мають. Його Java клас з main методом зображено в лістингу 3.6.

Лістинг 3.6 — Java клас з main методом discovery-server

```
@SpringBootApplication
@EnableEurekaServer
@ComponentScan({"com.management.security"})
public class DiscoveryServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(DiscoveryServerApplication.class, args);
    }
}
```

Наступний сервіс є шлюзом та має назву api-gateway. Як зазначалося в минулих розділах, його він здійснює механізм єдиної точки входу для користувача. Таким чином через нього можна отримати доступ до інших сервісів, за рахунок маршрутизації на самому шлюзі. Також api-gateway має механізм токенів, тому авторизувавшись, користувач на основі своєї ролі, матиме або не матиме доступ до тих чи інших сервісів. Налаштування доступу через ролі зображено в лістингу 3.7. Таким чином кожен з ендпойнтів буде захищений відповідно до своєї ролі.

Лістинг 3.7 — налаштування доступу через ролі в api—gateway

```
@Bean
```

```
                                    public SecurityWebFilterChain
springSecurityFilterChain(ServerHttpSecurity serverHttpSecurity) {
        serverHttpSecurity
                .cors(cors -> cors.configurationSource(request -> new
CorsConfiguration().applyPermitDefaultValues()))
            .csrf(ServerHttpSecurity.CsrfSpec::disable)
            .authorizeExchange(exchange -> exchange
                .pathMatchers("/eureka/**").permitAll() // Eureka Discovery
Access
                .pathMatchers("/actuator/info").permitAll()// Management
descriptions Access
                .pathMatchers("/actuator/health/**").permitAll()
                .pathMatchers("/sba/**").permitAll() // Spring Boot Admin
Access
                .pathMatchers(SWAGGER_WHITELIST).permitAll()
                .pathMatchers("/auth/keycloak/**").permitAll()
                .pathMatchers("/admin/keycloak/**").hasAnyRole("ADMIN")
                .pathMatchers("/admin/profile/**").hasAnyRole("ADMIN")
                .pathMatchers("/admin/pdf/**").hasAnyRole("ADMIN")
                .pathMatchers("/music/songs/**").hasAnyRole("ADMIN")
                .pathMatchers("/music/labels/**").hasAnyRole("ADMIN")
                .pathMatchers("/main/keycloak/**").hasAnyRole("USER")
                .anyExchange()
                .authenticated()
            )
            .oauth2ResourceServer(oauth2 -> oauth2
                .jwt(jwt -> jwt.jwtAuthenticationConverter(jwtAuthConverter)));
        return serverHttpSecurity.build();
    }
```

Також саме через шлюз відбувається доступ до документації OpenAPI. Для цього використовується конфігураційний файл application.yaml, що зображено на рисунку 3.25. За допомогою такої конфігурації, адміністратору через шлюз будуть доступні інструменти документації. Тут відбувається підключення функцій OpenAPI та режиму інтерфейсу, що дозволить зручніше запускати потрібні сервіси та перевіряти їх на працездатність. Тому у кожного сервісу буде своя вкладка, по якій можна перейти та перевіряти як саме він працює. Найрізноманітнішим сервісом по функціоналу є сервіс адміністратора, або admin-service. Відповідно до минулих розділів, він здійснює управління над користувачами, профілями та бібліотекою композицій. Його робота

базується на архітектурному шаблоні Model-View-Controller (MVC), що розділяє програму на три окремі компоненти: модель (дані та бізнес-логіка), вигляд (інтерфейс користувача) та контролер (обробник вхідних запитів).

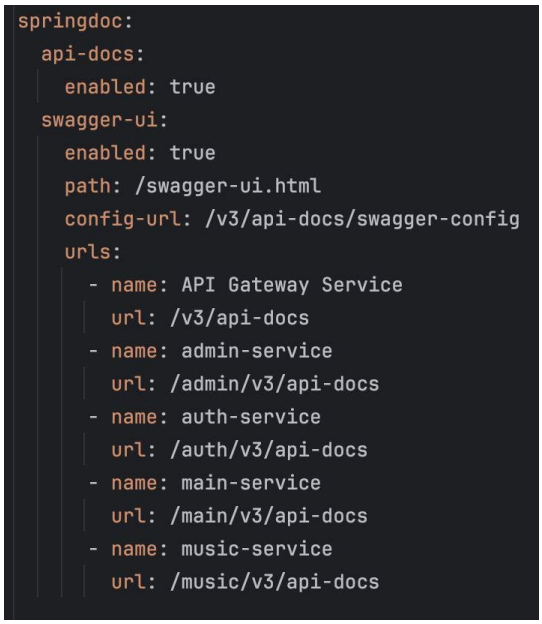


Рисунок 3.25 — Зображення фрагменту конфігураційного файлу api-gateway для здійснення автоматичної документації

В якості моделі використовується DTO сутностей, які обробляються через допоміжні класи. Вигляд представлений документацією Swagger, що допоможе переглядати та бачити функціонал в дії. Контролер в свою чергу представлений окремим класом, методи якого репрезентують HTTP запити. Для цього використовуються анотації @GetMapping, @PostMapping, @PutMapping, @DeleteMappuing. Наприклад користувач, який має доступ до даного сервісу хоче переглянути усіх користувачів, що є в системі. Для цього він вводить в браузері домен, та такий ендпойнт “…keycloak/all”, запит оброблюється і виводить потрібну інформацію. Зображення методу що виводить усіх користувачів зображено в лістингу 3.8.

Лістинг 3.8 — Метод що виводить усіх користувачів

```
@ApiResponses({
        @ApiResponse(responseCode="200", description ="Success", content = {@Content(mediaType = "application/json")}),
```

```
            @ApiResponse(responseCode = "500", description = "Server Error")
        })
        @GetMapping("/keycloak/all")
        public Page<KeycloakEntityDto> getAllUsers(@RequestParam(required =
false) int page,
                                                   @RequestParam(required = false) int size) {
          return keycloakUserService.findAllUsers(page, size);
        }
```

Схожим чином контролер може відповідно до запитів маніпулювати інформацією стосовно користувачів, профілів та користувачів.

Admin-service має можливість працювати з файлами формату XLSX та PDF. Для цього існує два окремих контролера. В першому випадку наявні два методи для читання та запису файлів, що зображені в лістингу 3.9.

Лістинг 3.9 — Клас для роботи з файлами формату XLSX

```
@RestController
@SecurityRequirements({
      @SecurityRequirement(name = "Direct Access Grants"),
      @SecurityRequirement(name = "Bearer Authentication")
})
public class ExcelController {
   @Autowired
   private KeycloakUserService keycloakUserService;
   @ApiResponses({
         @ApiResponse(responseCode="200", description ="Success", content
= {@Content(mediaType = "application/json")}),
         @ApiResponse(responseCode = "500", description = "Server Error")
   })
   @GetMapping("/keycloak/excel/download")
   public ResponseEntity<Resource> downloadFile() {
                                  InputStreamResource file = new
InputStreamResource(ExcelWriter.usersToExcel(keycloakUserService.getUserList(
)));
      String filename = "users.xlsx";
      return ResponseEntity.ok()
                .header(HttpHeaders.CONTENT_DISPOSITION, "attachment;
filename=" + filename)
               .contentType(MediaType.parseMediaType("application/vnd.ms-
excel"))
            .body(file);
```

```
        }
        @ApiResponses({
            @ApiResponse(responseCode="200", description ="Success", content
= {@Content(mediaType = "application/json")}),
            @ApiResponse(responseCode = "500", description = "Server Error")
        })
        @PostMapping("/keycloak/excel/upload")
        public List<KeycloakEntityDto> uploadFile(@RequestParam MultipartFile
file) throws Exception {
          return ExcelReader.excelToObject(file);
        }
    }
```

Здійснюється це все за допомогою бібліотеки Apache POI, яка при налаштуванні дає змогу працювати не тільки з файлами Excel, а й іншими файлами що використовуються в Microsoft Office.

Другий контролер дозволяє робити знімок вебсторінки та зберегти його. Контролер для роботи з PDF файлами зображено в лістингу 3.10.

Лістинг 3.10 — Контролер для роботи з PDF файлами

```
@RestController
@SecurityRequirements({
    @SecurityRequirement(name = "Direct Access Grants"),
    @SecurityRequirement(name = "Bearer Authentication")
})

public class PdfController {
    @ApiResponses({
        @ApiResponse(responseCode="200", description ="Success", content
= {@Content(mediaType = "application/json")}),
        @ApiResponse(responseCode = "500", description = "Server Error")
    })
    @GetMapping("/pdf/download")
     public ResponseEntity<Resource> downloadFile(@RequestParam String
fileName) {

                                ByteArrayResource resource = new
ByteArrayResource(PdfGenerator.generatePdfFromHtml(fileName));
        return ResponseEntity.ok()
                .header(HttpHeaders.CONTENT_DISPOSITION, "attachment;
filename=\"download.pdf\"")
```

```
                .contentType(MediaType.parseMediaType("application/pdf"))
                .body(resource);}}
```

Як зазначалося між сервісом авторизації та сервісом адміністратор є зв’язок через брокер повідомлень RabbitMQ. Сервіс авторизації фіксує реєстрацію або вхід користувача та передає ці дані сервісу адміністрування. Зображення класів які формують та передають повідомлення зображено в лістингах 3.11 та 3.12.

Лістинг 3.11 — Класів які отримує повідомлення при вході в систему

```
@Component
@Slf4j
public class AuthLog {
   @Bean
   public Consumer<KeycloakEntityDto> authReceiver() {
      return message -> log.info(message.getUsername() + " is logged in!");
   }
}
```

Лістинг 3.12 — Класів які отримує повідомлення при реєстрації в систему

```
@Component
@Slf4j
public class RegistrationLog {
   @Bean
   public Consumer<KeycloakEntityDto> registrationReceiver() {
      return message -> log.info(message.getUsername() + " has registered to the system!");
   }
}
```

Налаштування черг RabbitMQ при яких admin-service отримує повідомлення знаходиться в конфігураційному файлі application.yaml. Фрагмент налаштування черг зображено на рисунку 3.26. В ньому налаштовуються відправники, отримувачі та черги через які мають проходити аовідомлення. Також тут відбувається налаштування самих черг, наприклад, тривалість збергіання повідомлення. Таким чином можна задати потрібні параметри для передачі повідомлень через черги.


Рисунок 3.26 — Зображення фрагменту налаштування черг RabbitMQ для admin-service

Наступним буде розглянуто сервіс авторизації під назвою auth-service. Як було зазначено в попередніх розділ, його роль полягає в механізмі реєстрації нових користувачів та авторизації в системі. Авторизація відбувається за допомогою JWT токенів та валідується за допомогою сервісу Keycloak. Він також має свій контролер, через які він може здійснювати необхідні операції. Метод, який дозволяє отримати токен зображено в лістингу 3.13.

Лістинг 3.13 — Метод, який дозволяє отримати токен

```
@ApiResponses({
        @ApiResponse(responseCode="200", description ="Success", content = {@Content(mediaType = "application/json")}),
        @ApiResponse(responseCode = "500", description = "Server Error")
    })
    @PostMapping("/keycloak/token")
    public UserToken auth(@RequestBody UserAccess userAccess) {
        //Header
        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
        //Body
        MultiValueMap<String, String> body = userAccess.toMultiValueMap();
                                KeycloakEntityDto currentUser = keycloakUserService.findUserByUsername(userAccess.getUsername());
```

```
        authUserSender.sendAuthUser(currentUser);
          return restTemplate.postForEntity(keycloakUserService.getTokenUrl(),
new HttpEntity<>(body, headers), UserToken.class).getBody();
    }
```

Оскільки, даний сервіс працює з RabbitMQ та отримує повідомлення від admin-service, то він має аналогічні класи та методи тільки на цей раз для формування та передачі повідомлень, що зображено в лістингах 3.14 та 3.15.

Лістинг 3.14 — Клас який формує та передає повідомлення при вході в систему

```
@Service
@Slf4j
public class AuthUserSender {
   @Autowired
   private StreamBridge streamBridge;
   @Value("${rabbitmq.destination.auth}")
   private String destination;
   public void sendAuthUser(KeycloakEntityDto keycloakEntityDto) {
      streamBridge.send(destination, keycloakEntityDto);
      log.info(keycloakEntityDto.getUsername() + " is logged in!");
   }
}
```

Лістинг 3.15 — Клас який формує та передає повідомлення при реєстрації в систему

```
@Service
@Slf4j
public class RegisteredUserSender {
   @Autowired
   private StreamBridge streamBridge;
   @Value("${rabbitmq.destination.registration}")
   private String destination;
   public void sendRegisteredUser(KeycloakEntityDto keycloakEntityDto) {
      streamBridge.send(destination, keycloakEntityDto);
        log.info(keycloakEntityDto.getUsername() + " has registered to the
system!");}}
```

Даний сервіс передає повідомлення через черги до admin-service. Тому тут також потрібно прописати певні налаштування для роботи з RabbitMQ, але

в даному випадку, тут всього лише потрібно вказати назви черг на які потрібно передавати те чи інше повідомлення. Фрагмент налаштування черг зображено на рисунку 3.27.

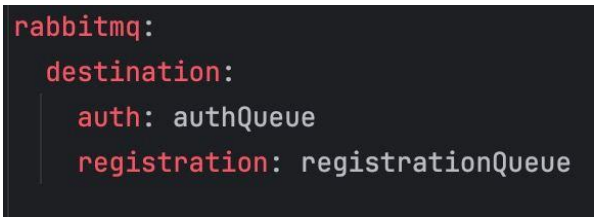


Рисунок 3.27 — Зображення фрагменту налаштування черг RabbitMQ для auth-service

Наступний на черзі сервіс має назву main-service. Даний сервіс призначений для звичайних користувачів. На даний момент через його контролер можна авторизуватися, та переглянути профіль, в подальшому його можна модифікувати та додати функціонал. Авторизація тут доступна завдяки auth-service та технології OpenFeign, декларативного HTTP-клієнта для Java, який спрощує створення інтеграцій з REST API в Spring-додатках. Таким чином сервіс користувача має доступ до інтерфейсу auth-service без прямої взаємодії.

Останній сервіс призначений для моніторингу за іншими сервісами на базі Spring Boot Admin, який має назву sba. Аналогічно до discovery-server він має захист у вигляді логіна та паролю. Завдяки цьому сервісу можна переглядати стан, статус та метрику сервісу що зареєстрований через сервіс виявлення.

Таким чином було розроблено усі потрібні сервіси для функціонування мікросервісного вебдодатку.

## 3.7 Розгортання вебдодатку

Виконавши усі необхідні кроки для розробки мікросервісного додатку, можна перейти до наступного етапу — розгортання вебдодатку. Цей процес полягає в опублікуванні програмного забезпечення в мережі, щоб користувачі

могли отримати до нього доступ через браузер. Процес включає розміщення коду додатку на вебсервері, що дозволяє йому працювати та бути доступним усім користувачам в мережі Інтернеті.

Відповідно до минулого розділу, було визначено, що процес розгортання проводитиметься за допомогою інструменту для управління ізольованими Linux-контейнерами Docker. Це дозволить запустити вебдодаток на будь-якому сервері без установки додаткового програмного забезпечення. Все що потрібно мати на сервері це встановлену програму Docker.

Для того щоб реалізувати розгортання демонстраційного вебдодатку, необхідно зробити певні кроки. Першим з цих кроків є створення файлу під назвою Dockerfile (без розширення). В директорії кожного сервісу. Даний файл — це текстовий документ, що містить інструкції, які Docker використовує для автоматичного створення образу Docker. Він діє як план для створення контейнерного середовища програми, визначаючи все: від базової операційної системи до залежностей програми, конфігурації та команд запуску. Наприклад, для admin-service буде актуальний такий Dockerfile, зображений на рисунку 3.28.

```
FROM ubuntu

RUN apt-get update && apt-get install -y openjdk-17-jre-headless wkhtmltopdf xvfb libfontconfig

LABEL maintance = "admin-service"
ADD target/admin-service-1.0-SNAPSHOT.jar admin-service.jar
ENTRYPOINT ["java", "-jar", "admin-service.jar"]
```

Рисунок 3.28 — Зображення Dockerfile для admin-service

Перший рядок означає, що базовим образом для створення контейнера буде операційна система Ubuntu. Усі наступні команди виконуватимуться на основі цього образу.

Другий рядок додатково інсталює утиліту та всі потрібні компоненти для роботи з PDF файлами та, оскільки, цього потребує бізнес-логіка admin-service. Даний підхід дозволяє запускати утиліти разом цим сервісом, що не потребує окремого контейнеру в Docker.

Третій рядок, що починається з LABEL, додає метадані до образу. У даному випадку ключ maintance має значення admin-service Це лише описова інформація, яка не впливає на роботу контейнера.

Наступний рядок виконує копіювання файлу компіляції з локальної машини (target/admin-service-1.0-SNAPSHOT.jar) у контейнер з новою назвою admin-service.jar. Щоб отримати скомпільований файл сервісу необхідно виконати команду mvn install, в директорії самого сервісу.

Останній рядок в даному файлі, починається з ENTRYPOINT. Він вказує команду, яка буде виконана після запуску контейнера. Це означає, що після запуску контейнеру виконається команда java -jar admin-service.jar, де admin-service.jar це скомпільований файл, який дозволить запустити даний сервіс. Таким чином admin-service готовий жо розгортання. Аналогічним чином Dockerfile виглядає в інших сервісах. Виключення становить перший рядок де замість Ubuntu потрібно вказати openjdk:19, оскільки, сторонні програми які потребують спеціалізовану ОС не потрібні. Рядок RUN видаляється. Стосовно іншого все лишається без змін, за винятком назв та метаданих контейнера.

Далі для спрощеного запуску всіх сервісів та компонентів, необхідно створити файл docker-compose.yml. Даний файл дозволяє визначити та запустити багатоконтейнерні Docker-застосунки. Він дає змогу керувати кількома контейнерами як єдиним цілим, забезпечуючи їх безперебійну роботу разом.

У випадку з демонстраційним вебдодатком, це є важливим етапом розгортання, оскільки, тут наявна велика кількість сервісів, у тому числі ті що надають додатковий функціонал (RabbitMQ, Keycloak, Redis, PostgreSQL). В docker-compose.yml необхідно внести усі потрібні налаштування для запуску цех сервісів. В якості прикладу буде admin-service, налаштування до якого зображено на рисунку 3.29.

В секції image показано, що даний образ матиме ім’я admin-service:latest. Нижче в секції build буде збиратися потрібний образ, якщо він ще не зібраний.

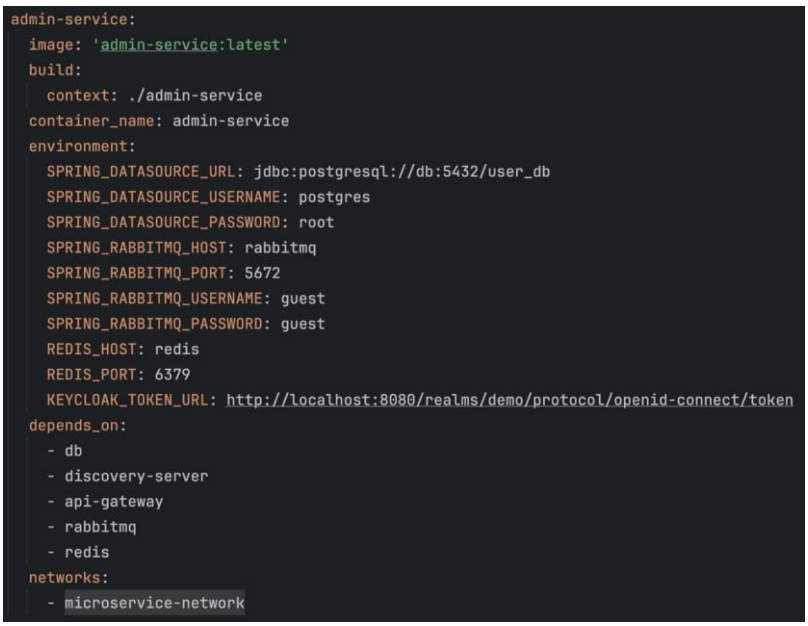


Рисунок 3.29 — Зображення фрагменту налаштування admin-service в docker-compose.yml

Секція environment дозволяє визначити певні параметри для налаштування конкретних складових сервісу. В даному випадку тут налаштовується зв’язок з базою даних, налаштування RabbitMQ, підключення до Redis, та підключення до сервісу Keycloak.

Наступна секція має назву depends_on. Вона позначає що даний сервіс залежний від тих, що внесені в цю секцію. Тобто в даному випадку admin-service запуститься після того як будуть запущені база даних, dsicovery-server, api-gateway, брокер повідомлень та Redis.

Остання секція network підключає до загальної мережі де запущені через Docker сервіси можуть комуніквати між собою за іменами.

Таким чином налаштовуються інші сервіси, відповідно до пторібних параметрів.

Після всіх потрібних налаштувань, весь вебдодаток можна запустити за допомогою команди docker-compose up. Після цього програма запускає усі потрібні сервіси, і далі вона готова до експлуатації.

# 4 АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ ТА ТЕСТУВАННЯ МІКРОСЕРВІСНОГО ВЕБДОДАТКУ

## 4.1 Тестування вебдодатку

Наступний етап даної магістерської роботи полягає в тестуванні розробленого вебдодатку. Тестування мікросервісного вебдодатку є критичною фазою життєвого циклу розробки будь-якого програмного забезпечення.

У контексті мікросервісної архітектури тестування виконує не лише функцію виявлення дефектів, а й слугує механізмом підтвердження коректності інтеграцій між сервісами та стабільності системи під навантаженням. Кожен мікросервіс тестується окремо як самостійний доменний компонент, однак його поведінка набуває повної значущості лише в рамках інтеграції із суміжними сервісами та зовнішніми інфраструктурними елементами, такими як брокер повідомлень, база даних або служба автентифікації.

Особливістю тестування мікросервісного програмного забезпечення є потреба в забезпеченні стабільності комунікаційного компоненту, що базується на протоколах REST, gRPC чи повідомленнях у чергах RabbitMQ. Кожен сервіс може бути розроблений різними командами із використанням відмінних технологій, підходів до оброблення даних та форматів відповідей.

Не менш значущим аспектом є тестування безпеки, яке має враховувати специфіку розподіленої архітектури. Оскільки кожен мікросервіс може мати власний канал доступу, окремі точки автентифікації та різні політики авторизації, необхідно гарантувати цілісність та конфіденційність даних на кожному етапі їх передавання.

Оскільки розроблена демонстраційна програма не є масивною відносно повноцінних додатків, було прийнято рішення провести ручне тестування. Такий підхід зумовлений кращим спостереженням за функціоналом ПЗ та

дозволить зібрати більшу кількість даних, щоб проаналізувати результат застосування компонентів мікросервісної архітектури.

Відправною точкою в обраному методі тестуванні розробленого вебдодатку є запуск самої програми за допомогою інструментарію розгортання. Для того щоб запустити це все, на персональний комп'ютер, на якому вебдодаток повинен запуститися, необхідно установити клієнт Docker. Дана програма доступна для операційних систем Linux, Windows, MacOS. Далі за допомогою файлу docker-compose.yaml можна запустити весь проєкт. Це може зайняти певний час, оскільки окрім розроблених сервісів повинні запуститися допоміжні. Стан працюючих сервісів можна відстежувати через клієнт Docker, що показано на рисунку 4.1.

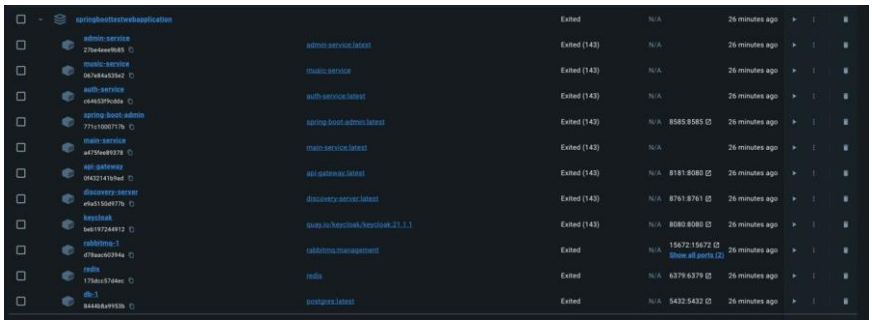

Рисунок 4.1 — Зображення сервісів вебдодатку в клієнті Docker

Коли усі сервіси запустяться, першочерговою задачею для перевірки працездатності це огляд Discovery Server. Доступ до нього здійснюється за такою адресою «http://loclahost:8761», яку можна ввести в браузері. Зображення роботи серверу виявлення показано на рисунку 4.2.

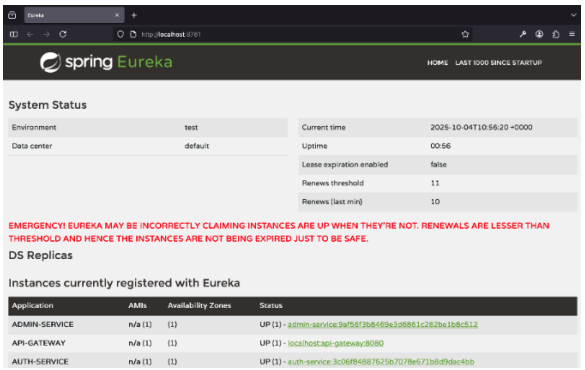


Рисунок 4.2 — Зображення роботи серверу виявлення

Тут можна переглянути які сервіси наразі працюють та скільки їх екземплярів вони мають. Завдяки цьому сервісу можна відстежувати працездатність основних сервісів вебдодатку. На рисунку 4.3 зображено сервіси які наразі працездатні.

Instances currently registered with Eureka

| Application | AMIs | Availability Zones | Status |
|---|---|---|---|
| ADMIN-SERVICE | n/a (1) | (1) | UP (1) - admin-service:9af56f3b8469e3d6861c282be1b8c512 |
| API-GATEWAY | n/a (1) | (1) | UP (1) - localhost:api-gateway:8080 |
| AUTH-SERVICE | n/a (1) | (1) | UP (1) - auth-service:3c06f84887625b7078e671b8d9dac4bb |
| MAIN-SERVICE | n/a (1) | (1) | UP (1) - main-service:bcfada1e81f1812b8bf203a0daedd20a |
| MUSIC-SERVICE | n/a (1) | (1) | UP (1) - music-service:f520c7301129ca750eb421429751744f |
| SPRING-BOOT-ADMIN | n/a (1) | (1) | UP (1) - spring-boot-admin:78435b907373d4da93063c9652839290 |

Рисунок 4.3 — Зображення працездатних сервісів

До сервісів що відстежують роботу всього додатку належить Spring Boot Admin. Він потрібне для того щоб побачити в якому стані той чи інший сервіс, скільки він вже працює та іншу потрібну інформацію. На рисунку 4.4 зображено зовнішній вигляд сервісу, на якому позначено інші робочі сервіси. Потрібно зазначити, що плитки, які позначають сервіси мають зелений колір. Це показує що сервіс в порядку та працює, в іншому випадку плитка стає червоного кольору.

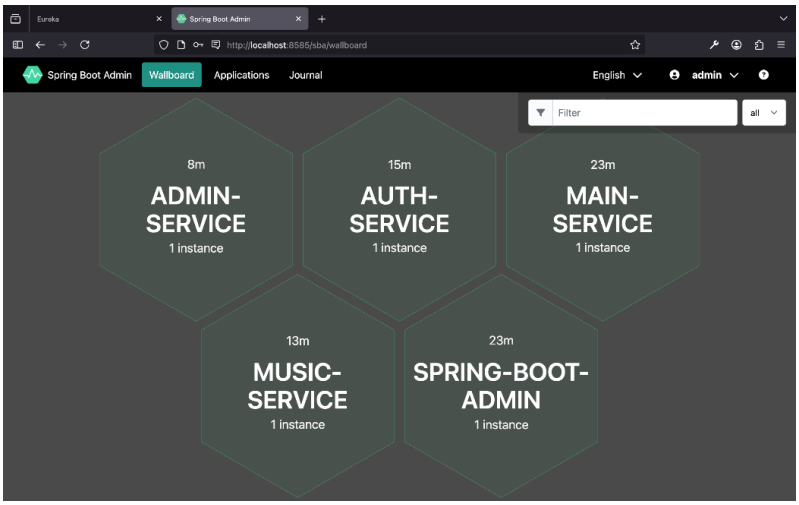


Рисунок 4.4 — Зображення сервісу Spring Boot Admin

Щоб подивитися детальну інформацію про сервіс, необхідно натиснути

на плитку, після чого відкриється нова сторінка. Наприклад можна обрати admin-service, та оглянути які дані він про себе містить. На рисунку 4.5 зображено інформацію про admin-service через Spring Boot Admin.

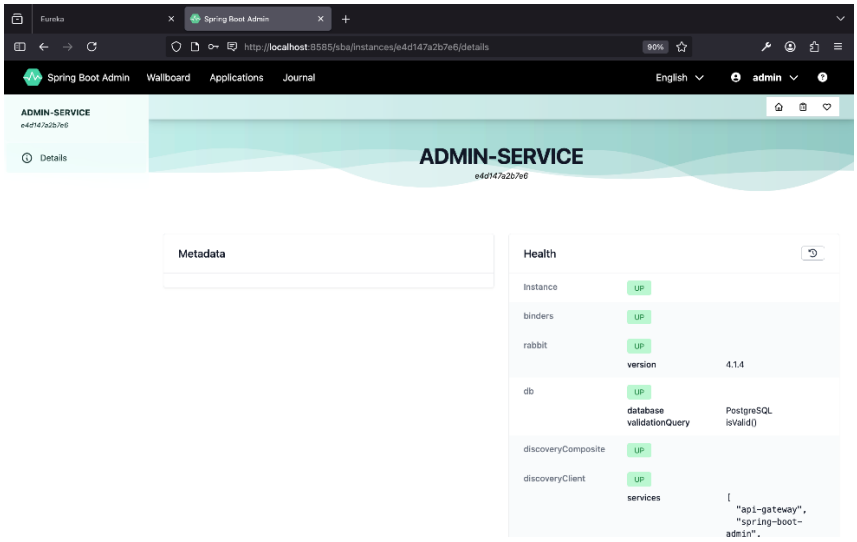


Рисунок 4.5 — Зображення сервісу Spring Boot Admin

Далі буде оглянуто шлюз вебдодатку. Оскільки він йде окремим сервісом, то до нього можна також звернутися за допомогою Url. Якщо не вказати ендпойнт, то сторінка видасть форму авторизації в Keycloak, що зображено на рисунку 4.6.

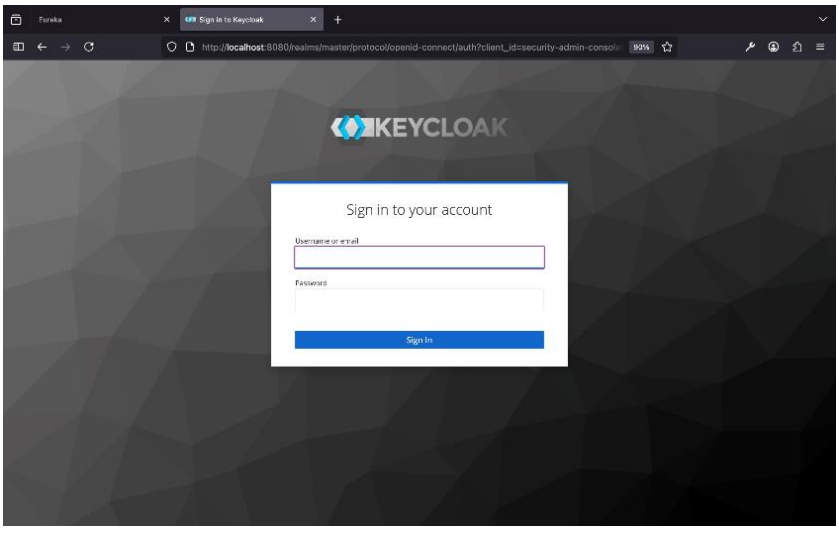


Рисунок 4.6 — Зображення форми авторизації в Keycloak

Вхід до даного сервісу доступний за певним логіном та паролем. В цілях спрощення тестування, такий обліковий запис вже створений в базі даних. Якщо внести необхідні дані можна отримати доступ до Keycloak, в якому можна налаштувати безпеку вебдодатку, та додати нових користувачів у систему. Вікно налаштувань Keycloak зображено на рисунок 4.7.

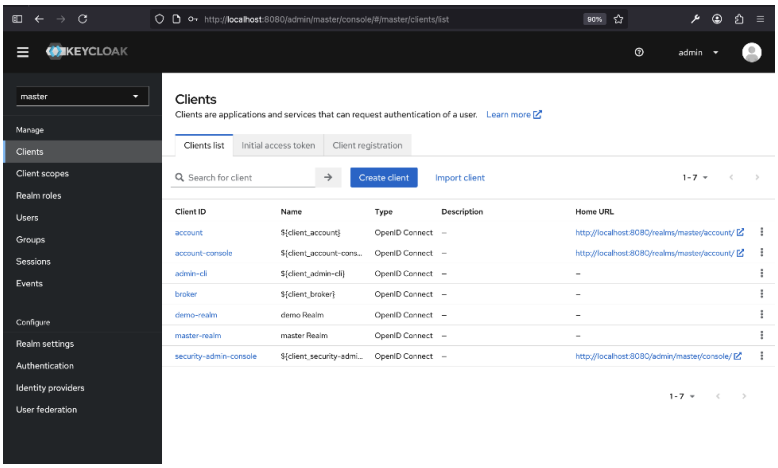


Рисунок 4.7 — Зображення вікна налаштувань Keycloak

Завдяки шлюзу можна отримати доступ і до інших сервісів. Для простішого тестування можна використати Swagger, який згенерує інтрефейс для кожного сервісу. Такий підхід дозволить простіше здійснювати перевірку працездатності при відсутності інтерфейсу користувача. На рисунку 4.8 зображено вікно Swagger.

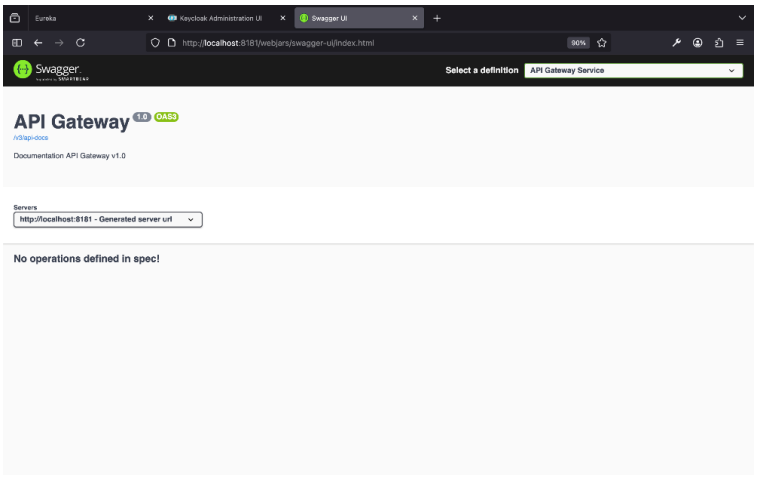


Рисунок 4.8 — Зображення вікна Swagger

У верхньому правому куті, можна обрати сервіс, що підтримує дану документацію. Наприклад можна оглянути admin-service, в якому буде доступний функціонал, який можна перевірити. На рисунку 4.9 зображено сервіс admin-service через Swagger.

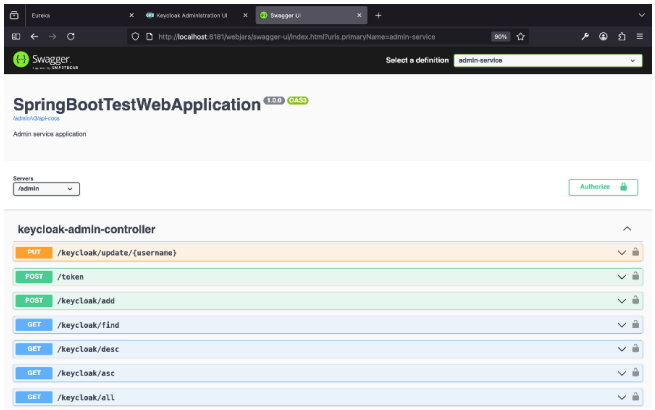


Рисунок 4.9 — Зображення сервісу admin-service через Swagger

Таким чином можна перевірити необхідний функціонал сервісів. Наприклад тут можна переглянути список усіх користувачів. Для цього потрібно вибрати «keycloak/all» та внести необхідні дані. Потрібно зазначити що це сервіс адміністратора, тому доступ до функціоналу можна отримати лише через авторизацію, про яку детальніше буде описано далі в цьому пункті. На рисунку 4.10 зображено результат роботи отримання даних про користувачів.

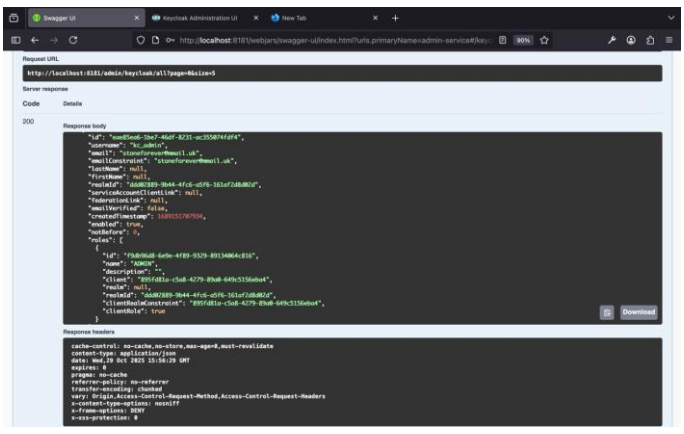


Рисунок 4.10 — Зображення результату роботи отримання даних про користувачів

Аналогічним чином можна отримати доступ до сервіс авторизації, де через потрібний метод можна отримати токен користувача. Вибравши потрібний метод та ввівши логін і пароль, можна отримати токен. На рисунку 4.11 зображено отримання токену через auth-service.

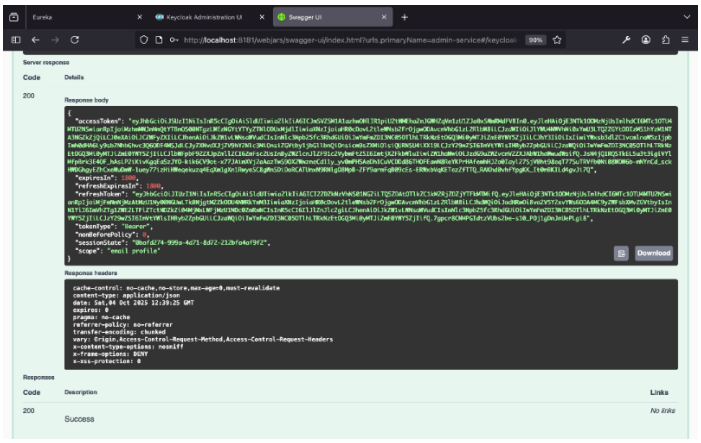

Рисунок 4.11 — Зображення отримання токену через auth-service

Маючи токен, можна перевірити його через відповідний метод. Таким працює авторизація в даному вебдодатку. Auth-service детально демонструє як працює цей процес.

Music-service працює аналогічно до admin-service, різниця полягає у даних з якими він працює. Так через відповідний метод можна отримати список всіх музикальних композицій, що наявні в бібліотеці самого додатку. На рисунку 4.12 зображено отримання списку музикальних композицій.

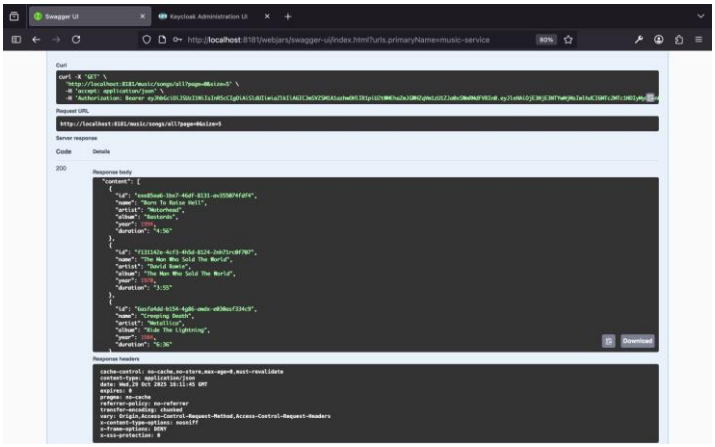

Рисунок 4.12 — Зображення отримання списку музикальних композицій

Таким чином працюють сервіси в розробленому демонстраційному вебдодатку. Переглянувши та перевіривши їх всіх, можна стверджувати що розроблений вебдодаток працює коректно та виконує задуманий функціонал.

## 4.2 Аналіз отриманих результатів дослідження

Розробивши та перевіривши вебдодаток, можна визначити основні положення щодо отриманих результатів.

В ході тестування було показано як вебдодаток за допомогою спеціалізованого файлу запускає усі свої складові. На практиці все це запускалося з одного локального персонального комп’ютера. В теорії даний вебдодаток може працювати за допомогою хмарних технологій, або в умовах роздільності сервісів. Тобто одна група сервісів може знаходитися на одній машині, а інша на другій. Подібний підхід не зможе завадити роботі всього вебдодатку, потрібно зазначити що для такого процесу необхідно здійснити певні налаштування аби система була працездатна.

Маючи спеціалізований доступ, відповідальна особа має змогу моніторити роботу вебдодатку в реальному часі, та в разі чого усувати неполадки. Також мікросервісний підхід до розробки такої програми дозволяє при оновленні або редагуванні компонентів, компілювати не весь проєкт, а лише тільки той в якому відбуваються зміни, що забезпечує більш гнучкий процес розробки.

Оскільки вебдодаток є мікросервісним, то певний функціонал може виконувати вже готовий сервіс, який всього лише потрібно підключити та налаштувати. Так, наприклад, в якості механізму безпеки використовується Keycloak, завдяки якому вебдодаток має механізми авторизації, без створення власних реалізацій.

Серед недоліків до вебдодатку можна віднести його масивність та складність при невеликій кількості функціоналу. Для простіших додатків можна використовувати монолітну архітектуру, але якщо проєкт вимагає

комплексність, то мікросервісний архітектурний стиль є більш раціональним рішенням. В подальшій перспективі розроблений вебдодаток можна модифікувати та покращити, що зробить його більш універсальним, а наявність саме такої архітектури, дозволить легше інтегрувати нові рішення та функціонал.

## 4.3 Розробка інструкції запуску вебдодатку

Для роботи з таким вебдодатком потрібно мати певні інструменти та знання. Спочатку потрібно перенести код програми на персональний комп'ютер, який виконуватиме роль сервера для даного застосунку. Це можна зробити двома способами: за допомогою Git або просто перенести архівом код. Потрібно зазначити, що код програми знаходиться на GitHub репозиторії.

Коли код вебдодатку знаходиться на ПК. Необхідно через редактор коду скомпілювати усі необхідні файли сервісів за допомогою команди clean install.

Як зазначалося в минулому пункті, для запуску даної програми потрібно використати клієнт Docker. Тому після його установки через docker-compose.yaml запускаються усі потрібні сервіси автоматично.

Коли усі сервіси запустилися, можна почати з ними працювати. Щоб перевірити чи певний сервіс працездатний, можна звернутися до Discovery Server. До нього можна потрапити за адресою «http://loclahost:8761». Доступ до нього захищений логіном та паролем, які прописані в самому коді сервіса. У спеціалізоване спливаюче вікно потрібно внести логін «eureka» та пароль «password», після чого можна отримати доступ до сервіса. Аналогічним чином можна отримати доступ до Spring Boot Admin, адреса до якого «http://loclahost:8585/sba», логін та пароль «admin» та «password» відповідно.

Щоб отримати доступ до Keycloak, необхідно перейти на адресу «http://loclahost:8080». Коли з'явиться вікно, необхідно ввести логін «kc_admin» та пароль «admin». Таким чином з'явиться можливість

налаштувати компоненти Keycloak, а також можна переглянути усіх наявних користувачів у системі.

Для перевірки сервісів на працездатність за допомогою Swagger, потрібно перейти за такою адресою «http://loclahost:8181/swagger-ui.html», а далі обрати потрібний сервіс.

Даний вебдодаток може працювати на операційних системах Windows, Linux, MacOS що робить його універсальним.

# 5 ЕКОНОМІЧНА ЧАСТИНА

## 5.1 Оцінювання комерційного потенціалу розробки

Метою проведення комерційного та технологічного аудиту є дослідження та аналіз компонентів мікросервісної архітектури в розробці сучасних вебдодатків на прикладі розроблюваного за таким принципом програми, що слугує демонстрацією ефективності даного метода розробки.

Для проведення технологічного аудиту було залучено 3-х незалежних експертів:

— Експерт 1: Тарновський М.Г., к.т.н, доцент кафедри обчислювальної техніки Вінницького національного технічного університету.

— Експерт 2: Курбатов О.А., інженер-програміст, КНП «ВМКЛ №1».

— Експерт 3: Причепа А.О., інженер-програміст, КНП «ВМКЛ №1».

Для проведення технологічного аудиту було використано таблицю 5.1 [38] в якій за п’ятибальною шкалою використовуючи 12 критеріїв здійснено оцінку комерційного потенціалу.

Таблиця 5.1 — Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

| Критерії оцінювання та бали (за 5-ти бальною шкалою) | | | | | |
|---|---|---|---|---|---|
| критерій | 0 | 1 | 2 | 3 | 4 |
| Технічна здійсненність концепції: | | | | | |
| 1 | Достовірність концепції не підтверджена | Концепція підтверджена експертними висновками | Концепція підтверджена розрахунками | Концепція перевірена на практиці | Перевірено роботоздатність продукту в реальних умовах |
| 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 великому ринку |
| 3 | Ціна продукту значно вища за ціни аналогів | Ціна продукту дещо вища за ціни аналогів | Ціна продукту приблизно дорівнює цінам аналогів | Ціна продукту дещо нижче за ціни аналогів | Ціна продукту значно нижче за ціни аналогів |
| 4 | Технічні та споживчі властивості продукту значно гірші, ніж в аналогів | Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів | Технічні та споживчі властивості продукту на рівні аналогів | Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів | Технічні та споживчі властивості продукту значно кращі, ніж в аналогів |

Продовження табл. 5.1

| | | | | | |
|---|---|---|---|---|---|
| 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 компаній на ринку | Активна конкуренція | Помірна конкуренція | Незначна конкуренція | Конкурентів немає |
| 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 наявних фахівців | 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 років | 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 продукту | 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 |

Таблиця 5.2 — Рівні комерційного потенціалу розробки

| Середньоарифметична сума балів СБ, розрахована на основі висновків експертів | Рівень комерційного потенціалу розробки |
|---|---|
| 0-10 | Низький |
| 11-20 | Нижче середнього |
| 21-30 | Середній |
| 31-40 | Вище середнього |
| 41-48 | Високий |

В таблиці 5.3 наведено результати оцінювання експертами комерційного потенціалу розробки.

Таблиця 5.3 — Результати оцінювання комерційного потенціалу розробки

| Критерії | Прізвище, ініціали, посада експерта | | |
|---|---|---|---|
| | Тарновський М.Г. | Курбатов О.А. | Причепа А.О. |
| | Бали, виставлені експертами: | | |
| 1 | 4 | 4 | 4 |
| 2 | 4 | 4 | 4 |
| 3 | 4 | 4 | 4 |
| 4 | 4 | 4 | 3 |
| 5 | 4 | 4 | 3 |
| 6 | 4 | 4 | 3 |
| 7 | 2 | 4 | 3 |
| 8 | 1 | 3 | 3 |
| 9 | 1 | 3 | 3 |
| 10 | 1 | 1 | 2 |
| 11 | 1 | 1 | 1 |
| 12 | 1 | 1 | 1 |
| Сума балів | $СБ_1$=31 | $СБ_2$=34 | $СБ_3$=34 |
| Середньоарифметична сума балів $\overline{СБ}$ | $\overline{СБ} = \frac{\sum_{1}^{3} СБ_i}{3} = \frac{31+34+34}{3} = 33$ | | |

Середньоарифметична оцінка, отримана на основі експертних висновків, тановить 33 бали, і згідно з таблицею 5.2, це вказує на рівень вище сиреднього комерційного потенціалу результатів проведених досліджень.

Розроблений продукт буде реалізований за допомогою середовища IntelliJ IDEA: Community Edition, мовою програмування Java, на базі фреймворку Spring Boot 3. Даний вебдодаток є кросплатформенним та призначений для розгортання його на сервері за допомогою програмного інструменту Docker. Після розгортання та налаштування програма може бути доступна в мережі інтернет, де користувачі можуть використовувати її для своїх цілей.

Результатами роботи можуть користуватися розробники програмного забезпечення, особливо розробники вебдодатків.

Проведемо оцінку якості і конкурентоспроможності нової розробки порівняно з аналогом.

В якості аналога для розробки було обрано Amazon(AWS). Основними недоліками аналога є обмеження контролю, ризики конфіденційності та загальна складність. Також до недоліків можна віднести низьку продуктивність в деяких випадках, регіональні обмеження ресурсів.

У розробці дана проблема вирішується спрощенням інтерфейсу користувача, автоматизація інструментів для моніторингу, надання клієнтам більшого контролю над конфігурацією безпеки, пріоритетність на відкриті стандарти, забезпечення рівномірної доступності ключових ресурсів у всіх регіонах.

Дані проблеми вирішуються за допомогою використання сучасних інструментів контейнеризації та оркестрації, управління через FinOps інструменти, використання Flat Rate або All-in-One пакети, використання Multi-Cloud/Hybrid Cloud стратегії та використання Open Source рішень.

Також система випереджає аналог за такими параметрами як Простота використання інтерфейсу користувача, використання екосистеми Open Source, більша продуктивність та оптимізація програми за рахунок своєї вузької специфікації, швидша технічна підтримка в разі неполадок.

В таблиці 5.4 наведені основні техніко-економічні показники аналога і нової розробки.

Проведемо оцінку якості продукції, яка є найефективнішим засобом забезпечення вимог споживачів та порівняємо її з аналогом.

Таблиця 5.4 — Основні параметри нової розробки та товару-конкурента

| Показник | Варіанти | | Відносний показник якості | Коефіцієнт вагомості параметра |
|---|---|---|---|---|
| | Базовий(товар конкурент) | Новий(інноваційне рішення) | | |
| 1 | 2 | 3 | 4 | 5 |
| Швидкість завантаження сторінки, сек. | 2,5 | 1,5 | 1,67 | 25% |
| Функціонал (Кількість інтерактивних функцій), бал | 40 | 65 | 1,63 | 35% |
| Безвідмовність, год. | 4000 | 7000 | 1,75 | 25% |
| Розмір застосунку, МБ | 450 | 350 | 1,29 | 15% |

Визначимо відносні одиничні показники якості по кожному параметру за формулами (5.1) та (5.2) і занесемо їх у відповідну колонку таблиці 5.5.

$$q_i = \frac{P_{Hi}}{P_{Бi}}, \tag{5.1}$$

або

$$q_i = \frac{P_{Бi}}{P_{Hi}}, \tag{5.2}$$

де $P_{Hi}$, $P_{Бi}$ — числові значення і-го параметру відповідно нового і базового виробів.

$$q_1 = \frac{2,5}{1,5} = 1,67,$$

$$q_2 = \frac{65}{40} = 1{,}63,$$

$$q_3 = \frac{7000}{4000} = 1{,}75,$$

$$q_4 = \frac{450}{350} = 1{,}29.$$

Відносний рівень якості нової розробки визначаємо за формулою:

$$К_{я.в.} = \sum_{i=1}^{n} q_i \cdot \alpha_i, \tag{5.3}$$

$$К_{я.в.} = 1{,}67 \cdot 0{,}25 + 1{,}3 \cdot 0{,}35 + 1{,}75 \cdot 0{,}25 + 1{,}29 \cdot 0{,}15 = 1{,}5.$$

Відносний коефіцієнт показника якості нової розробки більший одиниці, отже нова розробка якісніший базового товару-конкурента.

Наступним кроком є визначення конкурентоспроможності товару. Конкурентоспроможність товару є головною умовою конкурентоспроможності підприємства на ринку і важливою основою прибутковості його діяльності.

Однією із умов вибору товару споживачем є збіг основних ринкових характеристик виробу з умовними характеристиками конкретної потреби покупця. Такими характеристиками найчастіше вважають нормативні та технічні параметри, а також ціну придбання та вартість споживання товару.

В таблиці 5.5 наведено технічні та економічні показники для розрахунку конкурентоспроможності нової розробки відносно товару-аналога, технічні дані взяті з попередніх розрахунків.

Таблиця 5.5 — Нормативні, технічні та економічні параметри нової розробки і товару-виробника

| Показники | Варіанти | |
|---|---|---|
| | Базовий (товар-конкурент) | Новий (інноваційне рішення) |
| 1 | 2 | 3 |
| *1. Нормативно-технічні показники* | | |
| Швидкість завантаження сторінки, сек. | 2,5 | 1,5 |
| Функціонал (Кількість інтерактивних функцій), бал | 40 | 65 |
| Безвідмовність, год. | 4000 | 7000 |
| Розмір застосунку, МБ | 450 | 350 |
| *2. Економічні показники* | | |
| Ціна придбання, грн | 2000 | 1500 |

Загальний показник конкурентоспроможності інноваційного рішення (К) з урахуванням вищезазначених груп показників можна визначити за формулою:

$$K = \frac{I_{m.n.}}{I_{e.n.}}, \tag{5.4}$$

де $I_{m.n.}$ — індекс технічних параметрів;

$I_{e.n.}$ — індекс економічних параметрів.

Індекс технічних параметрів є відносним рівнем якості інноваційного рішення. Індекс економічних параметрів визначається за формулою (5.5)

$$I_{e.n.} = \frac{\sum_{i=1}^{n} P_{Hei}}{\sum_{i=1}^{n} P_{Бei}}, \tag{5.5}$$

де $P_{Нei}$, $P_{Бei}$ — економічні параметри (ціна придбання та споживання товару) відповідно нового та базового товарів.

$$I_{е.п.} = \frac{1500}{2000} = 0{,}75,$$

$$К = \frac{1{,}5}{0{,}75} = 2.$$

Зважаючи на розрахунки, можна зробити висновок, що нова розробка буде конкурентоспроможніше, ніж конкурентний товар.

## 5.2 Прогнозування витрат на виконання науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи групуються за такими статтями: витрати на оплату праці, витрати на соціальні заходи, матеріали, паливо та енергія для науково-виробничих цілей, витрати на службові відрядження, програмне забезпечення для наукових робіт, інші витрати, накладні витрати.

Основна заробітна плата кожного із дослідників $З_О$, якщо вони працюють в наукових установах бюджетної сфери визначається за формулою:

$$З_О = \frac{М}{Т_р} * t \text{ (грн)}, \quad (5.6)$$

де $M$ — місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.;

$T_P$ — число робочих днів в місяці; приблизно $T_P \approx 21...23$ дні;

$t$ — число робочих днів роботи дослідника.

Зведемо сумарні розрахунки до таблиці 5.6.

Таблиця 5.6 — Заробітна плата дослідника в науковій установі бюджетної сфери

| Найменування посади | Місячний посадовий оклад, грн. | Оплата за робочий день, грн. | Число днів роботи | Витрати на заробітну плату грн. |
|---|---|---|---|---|
| 1. Керівник проекту | 17000 | 809,5 | 5 | 4048 |
| 2. Інженер-програміст | 25000 | 1190,5 | 21 | 25000 |
| 3. Фахівець з тестування | 24000 | 1142,9 | 21 | 24000 |
| 4. Дизайнер інтерфейсу користувача | 2300 | 109,5 | 21 | 2300 |
| Всього | | | | 31348 |

Розрахунок додаткової заробітної плати робітників

Додаткова заробітна плата $З_д$ всіх розробників та робітників, які приймали устають в розробці нового технічного рішення розраховується як 10 - 12 % від основної заробітної плати робітників.

На даному підприємстві додаткова заробітна плата начисляється в розмірі 11% від основної заробітної плати.

$$З_д = (З_о + З_р) * \frac{Н_{доп}}{100\%}, \quad (5.7)$$

$$З_д = 0{,}11 * (31348) = 3448{,}24 \text{ (грн)}.$$

Нарахування на заробітну плату $Н_{ЗП}$ дослідників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою (5.8):

$$Н_{ЗП} = (З_о + З_р + З_д) * \frac{\beta}{100} \text{ (грн)}. \quad (5.8)$$

де $З_о$ — основна заробітна плата розробників, грн.;

$З_д$ — додаткова заробітна плата всіх розробників та робітників, грн.;

$З_р$ — основну заробітну плату робітників, грн.;

$\beta$ — ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, % .

Дана діяльність відноситься до бюджетної сфери, тому ставка єдиного внеску на загальнообов'язкове державне соціальне страхування буде складати 22%, тоді:

$$Н_{ЗП} = (31348 + 3448{,}24\ ) * \frac{22}{100} = 7655{,}09\ (грн).$$

Сировина та матеріали.

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби й предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за прямим призначенням згідно з нормами їх витрачання, а також витрачені придбані напівфабрикати, що підлягають монтажу або виготовленню й додатковій обробці в цій організації, чи дослідні зразки, що виготовляються виробниками за документацією наукової організації.

Витрати на матеріали (М) у вартісному вираженні розраховуються окремо для кожного виду матеріалів за формулою:

$$М = \sum_{i=1}^{n} H_j \cdot Ц_j \cdot К_j - \sum_{i=1}^{n} В_j \cdot Ц_{вj},\ (5.9)$$

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

n — кількість видів матеріалів;

Цj — вартість матеріалу j-го найменування, грн/кг;

Кj — коефіцієнт транспортних витрат, (Кj = 1,1 … 1,15);

Вj — маса відходів j-го найменування, кг;

Цвj — вартість відходів j-го найменування, грн/кг.

Проведені розрахунки зведені в таблицю 5.7.

Таблиця 5.7 — Витрати на матеріали

| Найменування матеріалу, марка, тип, сорт | Ціна за 1 кг, грн | Норма витрат, шт | Вартість витраченого матеріалу, грн |
|---|---|---|---|
| Папір офісний А4 | 180 | 0,25 | 45 |
| Картридж для принтера | 800 | 0,5 | 400 |
| Елементи живлення | 80 | 3 | 240 |
| Флеш-накопичувач | 250 | 2 | 500 |
| З врахуванням коефіцієнта транспортування | | | 1303,5 |

Розрахунок витрат на комплектуючі

Витрати на комплектуючі вироби (Кв), які використовують при дослідженні нового технічного рішення, розраховуються, згідно з їхньою номенклатурою, за формулою:

$$К_в = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j \,, \qquad (5.10)$$

де Нj — кількість комплектуючих j-го виду, шт.;

Цj — покупна ціна комплектуючих j-го виду, грн;

Кj — коефіцієнт транспортних витрат, (Кj = 1,1 … 1,15).

Проведені розрахунки бажано звести до таблиці 5.8.

Таблиця 5.8 — Витрати на комплектуючі

| Найменування комплектуючих | Кількість, шт. | Ціна за штуку, грн | Сума, грн |
|---|---|---|---|
| Хостинг бази даних | 1 | 5000 | 5000 |
| Доменне ім’я | 1 | 150 | 150 |

Продовження табл. 5.8

| | | | |
|---|---|---|---|
| CDN трафік | 1 | 2500 | 2500 |
| Тестові облікові записи | 3 | 250 | 750 |
| Ліцензія на сторонню бібліотеку | 1 | 1000 | 1000 |
| Послуги моніторингу та логування | 1 | 800 | 800 |
| Всього з врахуванням транспортних витрат | | | 11220,00 |

Програмне забезпечення для наукових (експериментальних) робіт

Балансову вартість програмного забезпечення розраховують за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i , \qquad (5.11)$$

де Ціпрг — ціна придбання одиниці програмного засобу даного виду, грн;

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

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

k — кількість найменувань програмних засобів.

Отримані результати необхідно звести до таблиці:

Таблиця 5.9 — Витрати на придбання програмних засобів по кожному виду

| Найменування програмного засобу | Кількість, шт | Ціна за одиницю, грн | Вартість, грн |
|---|---|---|---|
| Ліцензія IDE | 4 | 7500 | 30000 |
| Операційна система для сервера | 1 | 12000 | 12000 |
| Система управління версіями | 1 | 5000 | 5000 |
| Всього з врахуванням налагодження | | | 51700 |

Витрати на «Спецустаткування для наукових (експериментальних) робіт»

Балансову вартість спецустаткування розраховують за формулою:

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

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

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

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

k — кількість найменувань устаткування.

Отримані результати необхідно звести до таблиці 5.1:

Таблиця 5.10 — Витрати на придбання спецустаткування по кожному виду

| Найменування устаткування | Кількість, шт | Ціна за одиницю, грн | Вартість, грн |
|---|---|---|---|
| Серверне обладнання | 1 | 25000 | 25000 |
| Професійні монітори | 2 | 8500 | 17000 |
| Резервний носі інформації | 1 | 2500 | 42000 |
| Всього з врахуванням транспортних засобів | | | 46200 |

Амортизація обладнання, програмних засобів та приміщень

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

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (5.13)$$

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

tвик — термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

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

Проведені розрахунки необхідно звести до таблиці 5.11.

Таблиця 5.11 — Амортизаційні відрахування по кожному виду обладнання

| Найменування обладнання | Балансова вартість, грн | Строк корисного використання, років | Термін використання обладнання, місяців | Амортизаційні відрахування, грн |
|---|---|---|---|---|
| Комп'ютер | 25000 | 2 | 3 | 1041,67 |
| Всього | | | | 1041,67 |

До статті «Паливо та енергія для науково-виробничих цілей» відносяться витрати на всі види палива й енергії, що безпосередньо використовуються з технологічною метою на проведення досліджень.

$$В_е = \sum_{i=1}^{n} \frac{W_{yt} \cdot t_i \cdot Ц_е \cdot К_{впі}}{\eta_і}, \quad (5.14)$$

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

$t_i$ — тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ — вартість 1 кВт-години електроенергії, грн;

$К_{впі}$ — коефіцієнт, що враховує використання потужності, $К_{впі} < 1$;

$\eta_і$ — коефіцієнт корисної дії обладнання, $\eta_і < 1$.

Для написання магістерської роботи використовується персональний комп'ютер для якого розрахуємо витрати на електроенергію.

$$В_е = \frac{0{,}5 \cdot 195 \cdot 12{,}69 \cdot 0{,}5}{0{,}8} = 773{,}30.$$

Службові відрядження.

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

$$В_{св} = (З_о + З_р) * \frac{Н_{св}}{100\%}, \quad (5.15)$$

де Нсв — норма нарахування за статтею «Службові відрядження».

$$В_{св} = 0{,}2 * (31348) = 6269{,}52.$$

Накладні (загальновиробничі) витрати Внзв охоплюють: витрати на управління організацією, оплата службових відряджень, витрати на утримання, ремонт та експлуатацію основних засобів, витрати на опалення, освітлення, водопостачання, охорону праці тощо. Накладні (загальновиробничі) витрати Внзв можна прийняти як (100…150)% від суми основної заробітної плати розробників та робітників, які виконували дану МКНР, тобто:

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

де $Н_{нзв}$ — норма нарахування за статтею «Інші витрати».

$$В_{нзв} = (31348) \cdot \frac{100}{100\%} = 31347{,}62 \text{ грн.}$$

Сума всіх попередніх статей витрат дає витрати, які безпосередньо стосуються даного розділу МКНР

В=31348+3448,24+7655,09+1303,5+11220+51799+46200+1041,67+773,30 +6269,52 +31347,62= 140606,55грн.

Прогнозування загальних втрат ЗВ на виконання та впровадження результатів виконаної МКНР здійснюється за формулою:

$$ЗВ = \frac{В}{\eta}, \quad (5.17)$$

де η — коефіцієнт, який характеризує стадію виконання даної НДР.

Оскільки, робота знаходиться на стадії науково-дослідних робіт, то коефіцієнт $\beta$ = 0,7.

Звідси:

$$ЗВ = \frac{140606,55}{0,7} = 200866,5 \text{ грн.}$$

5.3 Розрахунок економічної ефективності науково-технічної розробки

У даному підрозділі кількісно спрогнозуємо, яку вигоду, зиск можна отримати у майбутньому від впровадження результатів виконаної наукової роботи. Розрахуємо збільшення чистого прибутку підприємства ΔПі, для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, за формулою

$$\Delta П_i = \sum_1^n \left(\Delta Ц_o \cdot N + Ц_o \cdot \Delta N\right)_i \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\upsilon}{100}\right), \quad (5.18)$$

де ΔЦ0 — покращення основного оціночного показника від впровадження результатів розробки у даному році.

N — основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN — покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки:

Цо — основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n — кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки:

л — коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт л = 0,8333.

р — коефіцієнт, який враховує рентабельність продукту. р = 0,25;

х — ставка податку на прибуток. У 2025 році — 18%.

Припустимо, що ціна зросте на 500 грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року на 3000 шт., протягом другого року — на 3200 шт., протягом третього року на 3300 шт. Реалізація продукції до впровадження розробки складала 1 шт., а її ціна до 1500 грн. Розрахуємо прибуток, яке отримає підприємство протягом трьох років.

$$\Delta П_1 = [500 \cdot 1 + (1500 + 500) \cdot 3000] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) =$$

1025044,4 грн.,

$$\Delta П_2 = [500 \cdot 1 + (1500 + 500) \cdot (3000 + 3200)] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right)$$

$$= 2118748{,}6 \text{ грн.},$$

$$\Delta П_3 = [500 \cdot 1 + (1500 + 500) \cdot (3000 + 3200 + 3300)] \cdot 0{,}833 \cdot 0{,}25$$

$$\cdot \left(1 + \frac{18}{100}\right) = 3246203{,}5 \text{ грн.}$$

## 5.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Розрахуємо основні показники, які визначають доцільність фінансування наукової розробки певним інвестором, є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності.

Розрахуємо величину початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки.

$$PV = k_{\text{інв}} \cdot \text{ЗВ}, \tag{5.19}$$

$k_{\text{інв}}$ — коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо ($k_{\text{інв}}$ = 2…5).

$$PV = 2 \cdot 200866{,}5 \ = 401733{,}01.$$

Розрахуємо абсолютну ефективність вкладених інвестицій Еабс згідно наступної формули:

$$E_{абс} = (ПП - PV), \tag{5.20}$$

де ПП — приведена вартість всіх чистих прибутків, що їх отримає підприємство від реалізації результатів наукової розробки, грн.

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \tag{5.21}$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДЦКР, грн.;

Т – період часу, протягом якою виявляються результати впровадженої НДДКР, роки;

$\tau$ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,2;

t — період часу (в роках).

$$ПП = \frac{1025044,4}{(1+0,2)^1} + \frac{2118748,6}{(1+0,2)^2} + \frac{3246203,5}{(1+0,2)^3} = 4212884,49 \text{ грн.},$$

$$Е_{абс} = (4212884,49 - 401733,01) = 3811151,48 \text{ грн.}$$

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів НДДКР може бути доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього користуються формулою:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{5.22}$$

де $T_ж$ – життєвий цикл наукової розробки, роки.

$$Е_в = \sqrt[3]{1 + \frac{3811151,48}{401733,01}} - 1 = 1,71 = 171\%.$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f,$$

(5.23)

де d — середньозважена ставка за депозитними операціями в комерційних банках; в 2025 році в Україні d= (0,14…0,2);

f — показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,1).

$$\tau_{\min} = 0,18 + 0,05 = 0,23.$$

Так як Ев > $\tau_{\min}$ то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_в},$$

$$\text{Т}_{\text{ок}} = \frac{1}{1,71} = 0,6 \text{ роки.}$$

(5.24)

Так як $T_{ок} \leq$ 3...5-ти років, то фінансування даної наукової розробки в принципі є доцільним.

# ВИСНОВКИ

Аналіз сучасних технологій створення вебдодатків показав, що сьогодні найбільш широкого застосування знаходять такі, які надають можливість досягнути високої продуктивності, масштабованості, безпеки та стійкості до збоїв кінцевого програмного продукту.

Аналіз архітектурних підходів, що використовуються для побудови сучасних вебдодатків показав, що мікросервісний підхід забезпечує найвищий рівень гнучкості, незалежності розвитку компонентів, можливість горизонтального масштабування та високу стійкість до відмов складних і високонавантажених систем.

Визначено загальну структуру демонстраційного вебдодатку, розроблено архітектурну модель системи, яка складається з окремих сервісів користувачів, авторизації, адміністрування, музичних композицій, а також інфраструктурних компонентів — шлюзу, сервісу виявлення, системи моніторингу та зовнішніх інструментів підтримки.

Проведено обґрунтований вибір технологічного стеку для створення розподіленого вебдодатку, що дозволило створити систему з високим рівнем інтеграції, незалежності компонентів та стабільності роботи.

Розроблено програмні компоненти демонстраційного мікросервісного вебдодатку, який реалізує функціонал роботи з користувачами, бібліотекою музичних композицій, ролевою моделлю доступу, імпортом даних у форматах XLSX, формуванням PDF-файлів, а також підтримкою взаємодії між сервісами через черги повідомлень.

Виконано тестування роботи вебдодатку, у ході якого підтверджено коректність функціоналу, стійкість до збоїв окремих сервісів, можливість динамічного масштабування та стабільність роботи при взаємодії між контейнеризованими компонентами системи.

Результати здійсненого технологічного аудиту вказують на високий рівень комерційного потенціалу. У порівнянні з аналогічним виробом

виявлено, що нова розробка вищої якості і більш конкурентоспроможна, як з технічних, так і економічних позначень.

Вкладені інвестиції в даний проект окупляться через 0,6 роки. Загальні витрати складають 200866,5 грн. Прогнозований прибуток за три роки 4212884,49 тис. грн.

Результати роботи можуть бути використані для проєктування і розробки корпоративних вебдодатків, впровадження розподілених обчислювальних систем та оптимізації інфраструктури програмних комплексів.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Улічев О.С. Інноваційні рішення та переваги мікросервісної архітектури програмних продуктів / Улічев О.С., О.П. Доренський, В.П. Кулагін // Центральноукраїнський науковий вісник. Технічні науки., 2024., Вип. 10 (41), ч.I, С. 16-29.

2. Тарновський М.Г., Гріша Д.Т. ЗАСОБИ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ В РОЗРОБЦІ СУЧАСНИХ ВЕБДОДАТКІВ. Конференція ВНТУ: Молодь в науці: дослідження, проблеми, перспективи (МН - 2026). [Електронний ресурс]. Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2026/paper/view/26248

3. Інтерактивний веб-дизайн. [Електронний ресурс]. Режим доступу: https://coi.ua/blog/DesignCo/interactive-web-design-entertainment-and-user-engagement/

4. Що таке Single Page Application та як працює SPA сайт. [Електронний ресурс]. Режим доступу: https://wezom.com.ua/ua/blog/chto-takoe-spa-prilozheniya

5. Unpacking System Resilience: High Availability, Fault Tolerance, and Disaster Recovery. [Електронний ресурс]. Режим доступу: https://medium.com/@helmi.confo/unpacking-system-resilience-high-availability-fault-tolerance-and-disaster-recovery-b522e5d80903

6. Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), SQL Injection, HTML Injection, Etc. [Електронний ресурс]. Режим доступу: https://www.zscaler.com/blogs/security-research/cross-site-scripting-xss-cross-site-request-forgery-csrf-sql-injection-html-injection-etc

7. Java Documentation. [Електронний ресурс]. Режим доступу: https://docs.oracle.com/en/java/

8. TypeScript is JavaScript with syntax for types. [Електронний ресурс]. Режим доступу: https://www.typescriptlang.org/

9. C# — a modern, open— source programming language. [Електронний ресурс]. Режим доступу: https://dotnet.microsoft.com/en- us/languages/csharp

10. Python Documentation. [Електронний ресурс]. Режим доступу: https://docs.python.org/3/

11. GO Documentation. [Електронний ресурс]. Режим доступу: https://go.dev/doc/

12. Introduction of DBMS (Database Management System). [Електронний ресурс]. Режим доступу: https://www.geeksforgeeks.org/dbms/introduction-of-dbms- database- management- system- set- 1/

13. OAuth 2.0. [Електронний ресурс]. Режим доступу: https://oauth.net/2/

14. What is OpenID Connect. [Електронний ресурс]. Режим доступу: https://openid.net/developers/how- connect- works/

15. Gateway API. [Електронний ресурс]. Режим доступу: https://kubernetes.io/docs/concepts/services- networking/gateway/

16. Service Discovery Explained. [Електронний ресурс]. Режим доступу: https://developer.hashicorp.com/consul/docs/use- case/service- discovery

17. AMQP 0- 9- 1 Model Explained. [Електронний ресурс]. Режим доступу: https://www.rabbitmq.com/tutorials/amqp- concepts

18. 5 Options for Deploying Microservices. [Електронний ресурс]. Режим доступу: https://semaphore.io/blog/deploy- microservices

19. Amazon (Company). [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/Amazon_(company)

20. Netflix. [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/Netflix

21. Uber. [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/Uber

22. Architecture Patterns for Beginners: MVC, MVP and MVVM. [Електронний ресурс]. Режим доступу: https://dev.to/chiragagg5k/architecture-patterns- for- beginners- mvc- mvp- and- mvvm- 2pe7

23. Monolithic Architecture — System Design. [Електронний ресурс]. Режим доступу: https://www.geeksforgeeks.org/system- design/monolithic-architecture- system— design/

24. Serverless Architecture: Pros, Cons and Use Cases. [Електронний ресурс]. Режим доступу: https://medium.com/@dave- patten/serverless-architecture- pros- cons- and- use- cases- 4a0769744ca2

25. Microservices architecture style. [Електронний ресурс]. Режим доступу: https://learn.microsoft.com/en- us/azure/architecture/guide/architecture-styles/microservices

26. Cervantes H., Kazman R. Designing Software Architectures: A Practical Approach. Boston: Addison-Wesley, 2016. 368 p.

27. Kruchten P. Architectural Blueprints: The 4 + 1 View Model of Software Architecture. IEEE Software, Vol. 12, No. 6, 1995. pp. 42—50. [Електронний ресурс]. Режим доступу: https://arxiv.org/abs/2006.04975

28. Java Microservices. [Електронний ресурс]. Режим доступу: https://hazelcast.com/foundations/software- architecture/java- microservices/

29. Spring Boot. Documentation overview. [Електронний ресурс]. Режим доступу: https://docs.spring.io/spring-boot/documentation.html

30. PostgreSQL. Documentation. [Електронний ресурс]. Режим доступу: https://www.postgresql.org/docs/

31. Redis. Docs. [Електронний ресурс]. Режим доступу: https://redis.io/docs/latest/

32. Keycloak. Documentation. [Електронний ресурс]. Режим доступу: https://www.keycloak.org/documentation

33. Spring cloud. Documentation. [Електронний ресурс]. Режим доступу: https://spring.io/projects/spring-cloud

34. Service Registration and Discovery. [Електронний ресурс]. Режим доступу: https://spring.io/guides/gs/service-registration-and-discovery

35. RabbitMQ Documentation. [Електронний ресурс]. Режим доступу: https://www.rabbitmq.com/docs

36. Docker docs. [Електронний ресурс]. Режим доступу: https://docs.docker.com/

37. Swagger Documentation. [Електронний ресурс]. Режим доступу: https://swagger.io/docs/

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

# ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

ЗАТВЕРДЖУЮ

Завідувач кафедри

ОТ д.т.н., професор

Азаров О.Д.

_____

“3” жовтня 2025 року

## ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

«Засоби мікросервісної архітектури в розробці сучасних вебдодатків»

123 — Комп’ютерна інженерія

Науковий керівник: доцент к.т.н.

_____Тарновський М.Г.

Студент групи 1КІ—24м

_____Гріша Д.Т.

ВНТУ 2025

1 Підстава для виконання магістерської кваліфікаційної роботи (МКР)

1.1 Актуальність роботи обумовлена тим, що мікросервісна архітектура набуває усе більшої популярності серед розробників програмних продуктів через надання можливості створювати легко масштабовані, гнучкі та кросплатформні програмні застосунки, які можуть бути легко адаптовані під завдання різних користувачів. Поряд із цим потребують вирішення проблеми перенесення мікросервісів із середовища розробки на платформу, на якій вони будуть функціонувати, що ускладнює процес розгортання та масштабування.

1.2 Наказ про затвердження теми МКР.

2 Мета МКР і призначення розробки

2.1 Мета роботи — вдосконалення підходів до використання мікросервісної архітектури при розробці вебдодатків, що спрощує їх розгортання та організацію взаємодії між їхніми компонентами.

2.2 Призначення розробки — застосування засобів мікросервісної архітектури для розробки вебдодатку, що є відкритою бібліотекою музикальних композицій, де користувач може створити обліковий запис та самостійно додавати до нього музичні композиції..

3 Вихідні дані для виконання МКР

3.1 Мікросервісний стиль архітектури (MSA);

3.2 Автоматизоване розгортання сервісів — Docker;

3.3 Обмін повідомленнями — RabbitMQ;

3.4 Реалізація single sign-on — Keycloak;

3.5 Управління модулями програми — Maven;

3.6 Реляційна база даних — PostgreSQ;

3.7 Реалізація кешування — Redis;

3.8 Реалізація специфікації OpenAPI — Swagger;

3.9 Середовище розробки — IntelliJ IDEA: Community Edition;

3.10 Засоби розробки — мова Java, фреймворк Spring Boot 3.

4 Вимоги до виконання МКР

4.1 Провести аналіз методів та засобів створення сучасних вебдодатків, прикладів застосування мікросервісної архітектури.

4.2 Розглянути етапи проєктування мікросервісного вебдодатку.

4.3 Визначити загальну структуру вебдодатку.

4.4 Вибрати стек технологій для розробки демонстраційного вебдодатку.

4.5 Здійснити проектування структури баз даних, сервісів вебдодатку та комунікації між ними, основного функціоналу демонстраційної програми.

4.6 Провести тестування розробленого додатку та проаналізувати отримані результати, розробити інструкції запуску програми.

4.7 Провести оцінку економічної ефективності запропонованих рішень.

5 Етапи МКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

6 Матеріали, що подаються до захисту МКР

До захисту подаються: пояснювальна записка МКР, графічні і ілюстративні матеріали, протокол попереднього захисту МКР на кафедрі, відгук наукового керівника, відгук опонента, анотації до МКР українською та іноземною мовами.

7 Порядок контролю виконання та захисту МКР

Виконання етапів графічної та розрахункової документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

Таблиця А.1 — Етапи МКР

| № етапу | Назва етапу | Термін виконання | | Очікувані результати |
|---|---|---|---|---|
| | | Початок | Кінець | |
| 1 | Аналіз методів та засобів створення сучасних вебдодатків | 08.09.25 | 12.09.25 | Розділ 1 |
| 2 | Теоретичні аспекти проєктування та методи розробки мікросервісного вебдодатку | 15.09.25 | 19.09.25 | Розділ 2 |
| 3 | Розробка програмних складових вебдодатку | 22.09.25 | 03.10.25 | Розділ 3 |
| 4 | Аналіз результатів дослідження та тестування мікросервісного вебдодатку | 06.10.25 | 15.10.25 | Розділ 4 |
| 5 | Економічна частина | 17.10.25 | 20.10.25 | Розділ 5 |
| 6 | Оформлення пояснювальної записки, графічного матеріалу і презентації | 25.10.25 | 25.10.25 | ПЗ, графічний матеріал і презентація |
| 7 | Підготовка супроводжуючих документів, їх підписування, проходження нормоконтролю та тесту на плагіат | 30.10.25 | 30.10.25 | Оформлені документи |

8 Вимоги до оформлювання та порядок виконання МКР

8.1 При оформлювання МКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— міждержавний ГОСТ 2.104—2006 «Єдина система конструкторської документації. Основні написи»;

— методичні вказівки до виконання магістерських кваліфікаційних робіт зі спеціальності 123 — «Комп’ютерна інженерія»;

— документами на які посилаються у вище вказаних.

8.2 Порядок виконання МКР викладено в «Положення про кваліфікаційні роботи на другому (магістерському) рівні вищої освіти СУЯ ВНТУ—03.02.02—П.001.01:21».

# ДОДАТОК Б

Протокол перевірки кваліфікаційної роботи

Назва роботи: Засоби мікросервісної архітектури в розробці сучасних вебдодатків

Тип роботи: магістерська кваліфікаційна робота
(бакалаврська кваліфікаційна робота/магістерська кваліфікаційна робота)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism (КП1) 1 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак академічного плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту.
- ☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- ☐ У роботі виявлено ознаки академічного плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень. Робота до захисту не приймається.

Експертна комісія:

Азаров О. Д., д.т.н., зав. каф. ОТ _____
(прізвище, ініціали, посада) (підпис)

Мартинюк Т. Б., д.т.н., проф. каф. ОТ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Захарченко С. М.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Тарновський М. Г., доц. каф. ОТ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Гріша Д. Т.
(підпис) (прізвище, ініціали)

# ДОДАТОК В

Структурна схема роботи програми

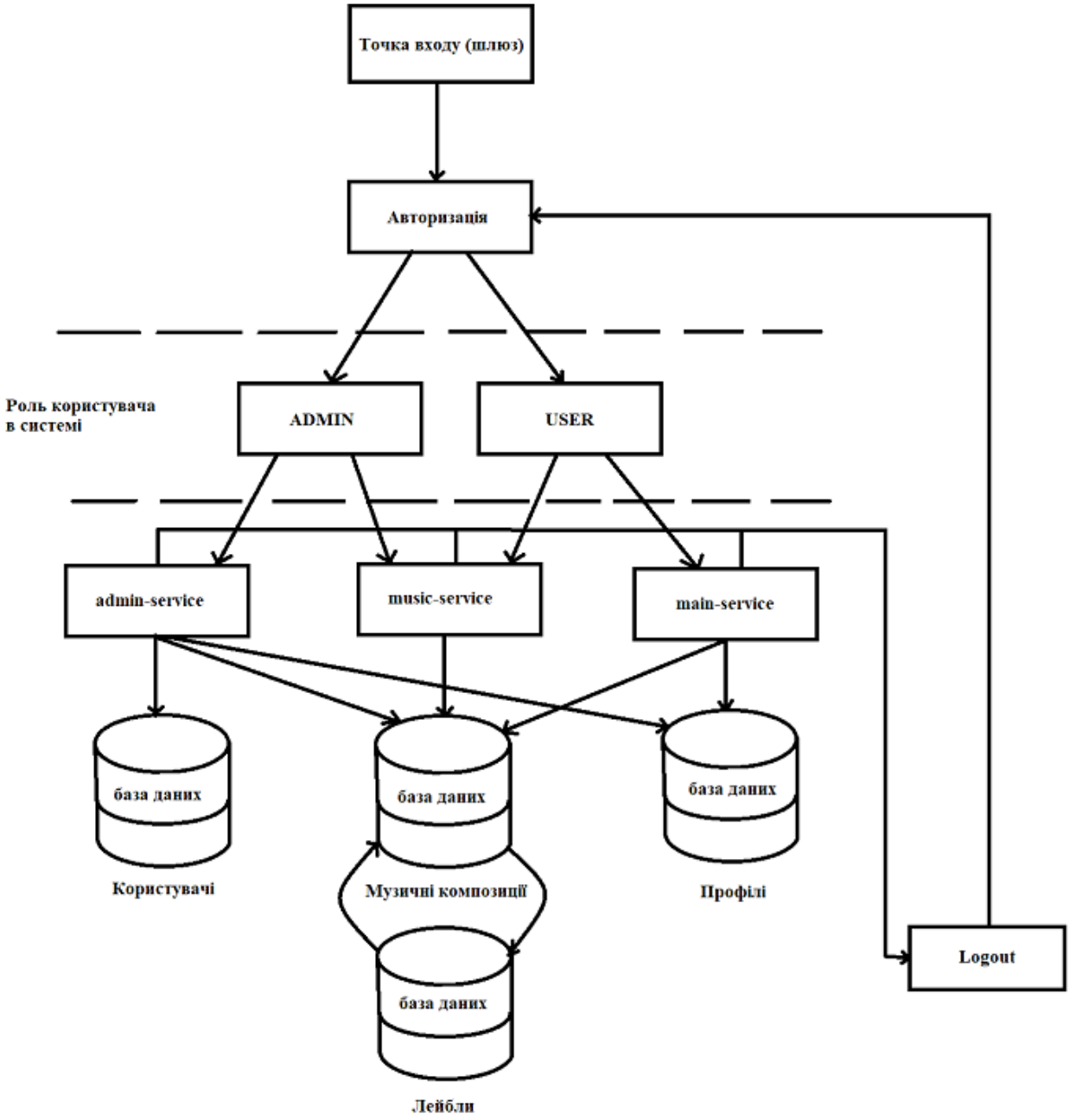


Рисунок Б.1 — Структура роботи програми

# ДОДАТОК Г

Блок схема логіки авторизації

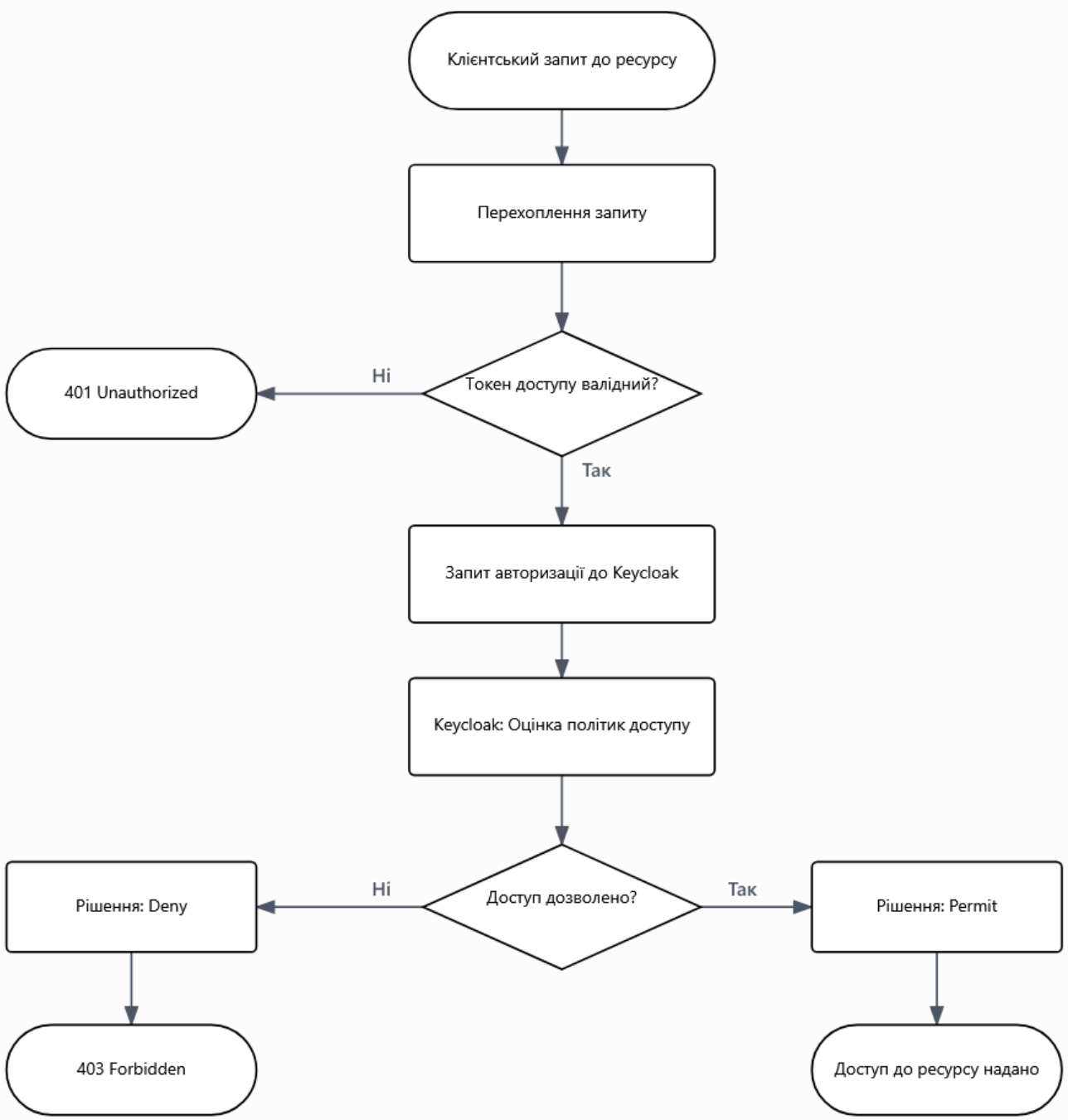


Рисунок В.1 — Блок схема логіки авторизації

# ДОДАТОК Д

Структурна схема бази даних

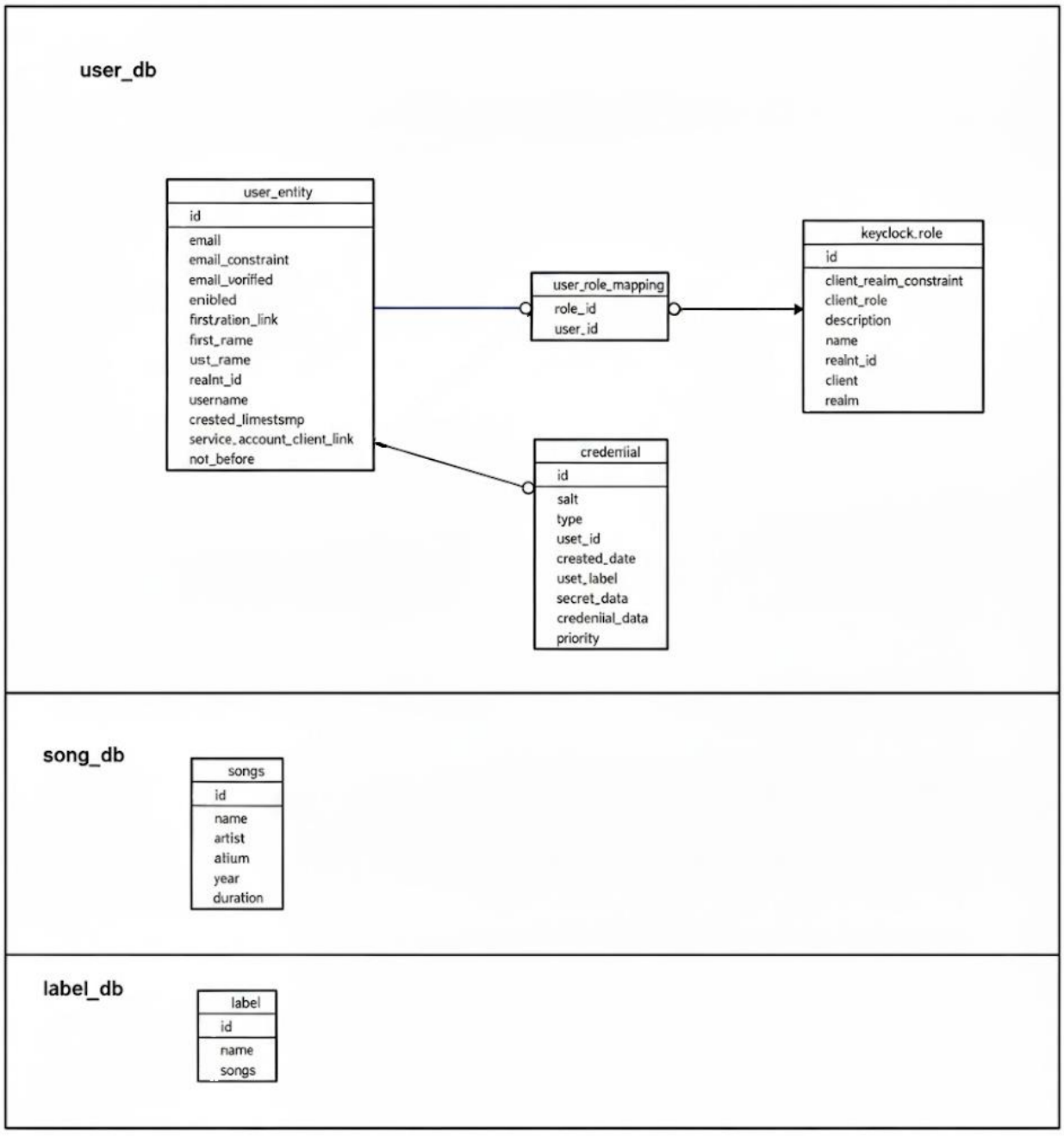


Рисунок Г.1 — Структурна схема бази даних

# ДОДАТОК Е

Лістинг програми

Код головного pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.1.0</version>
    <relativePath/>
  </parent>

  <groupId>com.demo-microservice</groupId>
  <artifactId>demo-microservice</artifactId>
  <packaging>pom</packaging>
  <version>1.0-SNAPSHOT</version>
  <modules>
    <module>main-service</module>
    <module>discovery-server</module>
    <module>api-gateway</module>
    <module>admin-service</module>
    <module>management-security</module>
    <module>core</module>
    <module>config-storage</module>
    <module>security</module>
    <module>auth-service</module>
    <module>database-migration</module>
    <module>sba</module>
    <module>redis-config</module>
    <module>music-service</module>
  </modules>

  <name>demo-microservice</name>
  <url>http://www.example.com</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
```

```
        <maven.compiler.source>19</maven.compiler.source>
        <maven.compiler.target>19</maven.compiler.target>
        <spring.boot.maven.plugin.version>3.1.0</spring.boot.maven.plugin.version>
        <spring.boot.dependencies.version>3.1.0</spring.boot.dependencies.version>
        <spring-cloud.version>2022.0.2</spring-cloud.version>
        <snakeyaml.version>2.2</snakeyaml.version>
    </properties>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>${spring-cloud.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-actuator</artifactId>
            <optional>true</optional>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <version>${spring.boot.maven.plugin.version}</version>
            </plugin>
        </plugins>
    </build>
</project>
```

Код файлу розгортування docker-compose.yaml

```
version: '3.7'
services:

  db:
```

```
    image: 'postgres:latest'
    ports:
      - "5432:5432"
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: root
    volumes:
      - ./database-migration/src/main/resources/db-scripts:/docker-entrypoint-initdb.d
    networks:
      - microservice-network

  discovery-server:
    image: 'discovery-server:latest'
    build:
      context: ./discovery-server
    container_name: discovery-server
    ports:
      - "8761:8761"
    expose:
      - "8761"
    networks:
      - microservice-network

  api-gateway:
    image: 'api-gateway:latest'
    build:
      context: ./api-gateway
    container_name: api-gateway
    hostname: localhost
    environment:
        SPRING_SECURITY_OAUTH2_RESOURCESERVER_JWT_ISSUER_URI:
http://keycloak:8080/realms/demo
      SPRING_SECURITY_OAUTH2_RESOURCESERVER_JWT_JWK_SET_URI:
http://keycloak:8080/realms/demo/protocol/openid-connect/certs
    ports:
      - "8181:8080"
    expose:
      - "8181"
    depends_on:
      - discovery-server
    networks:
      - microservice-network

  keycloak:
```

```
    image: quay.io/keycloak/keycloak:21.1.1
    container_name: keycloak
    command: [ "start-dev", "--import-realm"]
    environment:
      KC_DB_URL: jdbc:postgresql://db:5432/user_db
      KC_DB_USERNAME: postgres
      KC_DB_PASSWORD: root
      KC_DB: postgres
      KEYCLOAK_ADMIN: admin
      KEYCLOAK_ADMIN_PASSWORD: admin
    ports:
      - "8080:8080"
    volumes:
      - ./realms/:/opt/keycloak/data/import/
    depends_on:
      - db
    networks:
      - microservice-network

  admin-service:
    image: 'admin-service:latest'
    build:
      context: ./admin-service
    container_name: admin-service
    environment:
      SPRING_DATASOURCE_URL: jdbc:postgresql://db:5432/user_db
      SPRING_DATASOURCE_USERNAME: postgres
      SPRING_DATASOURCE_PASSWORD: root
      SPRING_RABBITMQ_HOST: rabbitmq
      SPRING_RABBITMQ_PORT: 5672
      SPRING_RABBITMQ_USERNAME: guest
      SPRING_RABBITMQ_PASSWORD: guest
      REDIS_HOST: redis
      REDIS_PORT: 6379
      KEYCLOAK_TOKEN_URL: http://localhost:8080/realms/demo/protocol/openid-
connect/token
    depends_on:
      - db
      - discovery-server
      - api-gateway
      - rabbitmq
      - redis
    networks:
      - microservice-network
```

```
  auth-service:
    container_name: auth-service
    build:
      context: ./auth-service
    image: 'auth-service:latest'
    environment:
      SPRING_DATASOURCE_URL: jdbc:postgresql://db:5432/user_db
      SPRING_DATASOURCE_USERNAME: postgres
      SPRING_DATASOURCE_PASSWORD: root
      SPRING_RABBITMQ_HOST: rabbitmq
      SPRING_RABBITMQ_PORT: 5672
      SPRING_RABBITMQ_USERNAME: guest
      SPRING_RABBITMQ_PASSWORD: guest
      REDIS_HOST: redis
      REDIS_PORT: 6379
      KEYCLOAK_TOKEN_URL: http://keycloak:8080/realms/demo/protocol/openid-
connect/token
    depends_on:
      - db
      - discovery-server
      - api-gateway
      - rabbitmq
      - redis
    networks:
      - microservice-network

  main-service:
    container_name: main-service
    build:
      context: ./main-service
    image: 'main-service:latest'
    environment:
      SPRING_DATASOURCE_URL: jdbc:postgresql://db:5432/user_db
      SPRING_DATASOURCE_USERNAME: postgres
      SPRING_DATASOURCE_PASSWORD: root
      REDIS_HOST: redis
      REDIS_PORT: 6379
      KEYCLOAK_TOKEN_URL: http://localhost:8080/realms/demo/protocol/openid-
connect/token
    depends_on:
      - db
      - discovery-server
      - api-gateway
```

```
      - redis
    networks:
      - microservice-network

  music-service:
    image: 'music-service'
    build:
      context: ./music-service
    container_name: music-service
    environment:
      SPRING_DATASOURCE_URL_SONG: jdbc:postgresql://db:5432/song_db
      SPRING_DATASOURCE_URL_LABEL: jdbc:postgresql://db:5432/label_db
      SPRING_DATASOURCE_USERNAME: postgres
      SPRING_DATASOURCE_PASSWORD: root
      REDIS_HOST: redis
      REDIS_PORT: 6379
      KEYCLOAK_TOKEN_URL: http://localhost:8080/realms/demo/protocol/openid-
connect/token
    depends_on:
      - db
      - discovery-server
      - api-gateway
      - redis
    networks:
      - microservice-network

  spring-boot-admin:
    image: 'spring-boot-admin:latest'
    build:
      context: ./sba
    container_name: spring-boot-admin
    ports:
      - "8585:8585"
    expose:
      - "8585"
    depends_on:
      - discovery-server
      - api-gateway
    networks:
      - microservice-network

  rabbitmq:
    image: rabbitmq:management
    ports:
```

```
      - "5672:5672"
      - "15672:15672"
    networks:
      - microservice-network

  redis:
    container_name: redis
    image: 'redis'
    ports:
      - "6379:6379"
    networks:
      - microservice-network

networks:
  microservice-network:
    driver: bridge
```

Код модулю admin-service (KeycloakADminController.java)

```
package com.admin.controller;

import com.core.model.BaseUserDto;
import com.core.model.KeycloakEntityDto;
import com.core.model.template.UserAccess;
import com.core.model.template.UserToken;
import com.security.service.KeycloakUserService;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.responses.ApiResponses;
import io.swagger.v3.oas.annotations.security.SecurityRequirement;
import io.swagger.v3.oas.annotations.security.SecurityRequirements;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.web.bind.annotation.*;

@RestController
@SecurityRequirements({
        @SecurityRequirement(name = "Direct Access Grants"),
        @SecurityRequirement(name = "Bearer Authentication")
})
public class KeycloakAdminController {

    @Autowired
    private KeycloakUserService keycloakUserService;
```

```
    @ApiResponses({
            @ApiResponse(responseCode="200", description ="Success", content =
{@Content(mediaType = "application/json")}),
        @ApiResponse(responseCode = "500", description = "Server Error")
    })
    @PostMapping("/keycloak/add")
      public KeycloakEntityDto createKeycloakUser(@RequestBody BaseUserDto
baseUserDto) {
      if (!keycloakUserService.saveUser(baseUserDto)) {
         System.out.println("ERROR");
         return null;
      }
      return keycloakUserService.findUserByUsername(baseUserDto.getUsername());
    }

    @ApiResponses({
            @ApiResponse(responseCode="200", description ="Success", content =
{@Content(mediaType = "application/json")}),
        @ApiResponse(responseCode = "500", description = "Server Error")
    })
    @GetMapping("/keycloak/all")
     public Page<KeycloakEntityDto> getAllUsers(@RequestParam(required = false)
int page,
                                    @RequestParam(required = false) int size) {
      return keycloakUserService.findAllUsers(page, size);
    }

    // Sort users by ascending id
    @ApiResponses({
            @ApiResponse(responseCode="200", description ="Success", content =
{@Content(mediaType = "application/json")}),
        @ApiResponse(responseCode = "500", description = "Server Error")
    })
    @GetMapping("/keycloak/asc")
     public Page<KeycloakEntityDto> getAscUsers(@RequestParam(required = false)
int page,
                                    @RequestParam(required = false) int size) {
      return keycloakUserService.sortAllUsersByAsc(page, size);
    }

    // Sort users by descending id
    @ApiResponses({
            @ApiResponse(responseCode="200", description ="Success", content =
{@Content(mediaType = "application/json")}),
```

```
        @ApiResponse(responseCode = "500", description = "Server Error")
    })
    @GetMapping("/keycloak/desc")
    public Page<KeycloakEntityDto> getDescUsers(@RequestParam(required = false)
int page,
                                                @RequestParam(required = false) int size) {
        return keycloakUserService.sortAllUsersByDesc(page, size);
    }

    //Find user by login
    @ApiResponses({
            @ApiResponse(responseCode="200", description ="Success", content =
{@Content(mediaType = "application/json")}),
        @ApiResponse(responseCode = "500", description = "Server Error")
    })
    @GetMapping("/keycloak/find")
    public KeycloakEntityDto findUserByString(@RequestParam String value) {
        return keycloakUserService.findUserByUsername(value);
    }

    @ApiResponses({
            @ApiResponse(responseCode="200", description ="Success", content =
{@Content(mediaType = "application/json")}),
        @ApiResponse(responseCode = "500", description = "Server Error")
    })
    @PutMapping("/keycloak/update/{username}")
        public KeycloakEntityDto updateUser(@PathVariable("username") String
username, @RequestBody BaseUserDto baseUserDto) {
                                    if (!keycloakUserService.updateUser(baseUserDto,
keycloakUserService.findUserByUsername(username).getId())) {
            System.out.println("ERROR");
            return null;
        }
        return keycloakUserService.findUserByUsername(baseUserDto.getUsername());
    }

    @ApiResponses({
            @ApiResponse(responseCode="200", description ="Success", content =
{@Content(mediaType = "application/json")}),
        @ApiResponse(responseCode = "500", description = "Server Error")
    })
    @DeleteMapping("/keycloak/delete/{username}")
    public String deleteUser(@PathVariable("username") String username) {
```

```
        keycloakUserService.deleteUser(keycloakUserService.findUserByUsername(us
ername).getId());
        return username + " has been deleted";
    }


    @ApiResponses({
            @ApiResponse(responseCode="200", description ="Success", content =
{@Content(mediaType = "application/json")}),
        @ApiResponse(responseCode = "500", description = "Server Error")
    })
    @PostMapping("/token")
    public UserToken auth(@RequestBody UserAccess userAccess) {
        return keycloakUserService.authUser(userAccess);
    }
}
```