Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Розробка інтерактивного лінгвістичного ресурсу за допомогою реактивного програмування з використанням фреймворку Spring Boot та мікросервісної архітектури

Виконав(ла): студент(ка) 6 курсу, групи СПм-61
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) (прізвище та ініціали)

Керівник (підпис) Дячук С.Ф. (прізвище та ініціали)
Нормоконтроль (підпис) Стоянов Ю.М. (прізвище та ініціали)
Завідувач кафедри (підпис) Петрик М.Р. (прізвище та ініціали)
Рецензент (підпис) Сверстюк А.С. (прізвище та ініціали)

Тернопіль
2024

АНОТАЦІЯ

Кваліфікаційна робота на здобуття освітнього ступеню «магістр» за спеціальністю 121 – Інженерія програмного забезпечення. Тернопільський національний технічний університет ім. Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СПм-61, 2024 рік. Атестаційна робота магістра містить: 100 с., 15 рис., 22 джер.

JAVA, SPRING BOOT, REACTIVE PROGRAMMING, SPRING CLOUD, MICROSERVICE ARCHITECTURE, GRPC, ANGULAR, POSTGRES, GRADLE, PROTOBUF, KUBERNETES, AWS.

Метою роботи є аналіз сучасних технологій для розробки високонавантажених систем на прикладі розробки лінгвістичного ресурсу.

Щоб справитись з колосальним навантаженням систему було спроектовано на основі найсучасніших архітектурних рішень з використанням передових технологій. До прикладу, на заміну всім відомого архітектурного партерну був використаний gRPC, який славиться своєю швидкість передачі та обробки даних. Він був використаний для комунікації між мікросервісами. Також, на відміну від стандартного блокуючого виконання програми, була використана парадигма реактивного програмування, що зменшує потреба на обчислювальні ресурси та пришвидшує обробку даних.

В результаті роботи розроблено лінгвістичний ресурс, що дозволяє гнучко та інтерактивно вивчати мови на основі найсучасніших технологій з використанням передових архітектурних рішень

ANNOTATION

Qualification work for obtaining a master's degree in specialty 121 – Software engineering. Ternopil National Technical University named after Ivan Pulyuya, faculty of computer and information systems and software engineering, department of software engineering, group CPm-61, 2024. The master's thesis contains: 100 pages, 15 figures, 22 figures.

JAVA, SPRING BOOT, REACTIVE PROGRAMMING, SPRING CLOUD, MICROSERVICE ARCHITECTURE, GRPC, ANGULAR, POSTGRES, GRADLE, PROTOBUF, KUBERNETES, AWS.

The purpose of the work is to analyze modern technologies for the development of high-load systems using the example of developing a linguistic resource.

To cope with the enormous load, the system was designed based on the most modern architectural solutions using advanced technologies. For example, gRPC, which is famous for its speed of data transmission and processing, was used to replace the well-known architectural partner. It was used for communication between microservices. Also, unlike the standard blocking program execution, the reactive programming paradigm was used, which reduces the need for computing resources and speeds up data processing.

As a result of the work, a linguistic resource was developed that allows for flexible and interactive learning of languages based on the most modern technologies using advanced architectural solutions.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

HTTP – HyperText Transfer Protocol

SQL – Structured Query Language

URL – Uniform Resource Locator

DB – Data Base

UI – User Interface

UX – User Experience

UML – Unified Modeling Language

JSON – JavaScript Object Notation

XML – Extensible Markup Language

SPA – Single Page Application

UML – Unified Modeling Language

CI – Continues Integration

CD – Continues Delivering

AOP – Aspect-Oriented Programming

MPA – Multiple pages application

REST – Representational State Transfer

ВСТУП

У все більш глобалізованому світі здатність спілкуватися кількома мовами стала важливою навичкою, що відкриває двері для культурного розуміння, міжнародної співпраці та професійних можливостей. Хоча традиційні методи вивчення мови залишаються ефективними, технологічний прогрес запровадив інноваційні підходи, які можуть покращити та персоналізувати процес вивчення мови. Цей проект має на меті задовольнити зростаючий попит на доступні, привабливі та настроювані онлайн-платформи для вивчення мов, який поєднує в собі найсучасніші технології

Основною метою цього проекту є створення зручної веб-програми з можливістю розширення, яка полегшить вивчення мови в інтерактивний та захоплюючий спосіб. Платформа буде призначена для людей, які прагнуть розширити свої лінгвістичні навички, незалежно від їхнього рівня володіння мовою, надаючи динамічні інструменти та ресурси. Користувачі матимуть доступ до таких функцій, як інтерактивні субтитри до мультимедійного вмісту, персоналізований словарь та інтеграція з цифровими словниками та курсами, створеними викладачами. Система буде розроблена таким чином, щоб заохочувати активну участь і безперервне навчання, роблячи процес приємним і ефективним.

Ключова відмінність цієї платформи полягає в її акценті на гнучкості та масштабованості. Архітектура системи буде розроблена з розширюваністю в її основі, що дозволить легко додавати нові мови, методики навчання та формати контенту. Цей модульний підхід гарантує, що платформа може розвиватися разом із досягненнями в технології та теорії освіти, залишаючись актуальною та ефективною для різноманітних потреб користувачів. Крім того, він використовуватиме сучасні парадигми розробки програмного забезпечення, такі як реактивне програмування, хмарна інфраструктура та мікросервіси, забезпечуючи надійну продуктивність і масштабованість для задоволення зростаючої бази користувачів.

Іншим фокусом цього проекту є досвід користувача (UX). Усвідомлюючи важливість інтуїтивно зрозумілої навігації та взаємодії, інтерфейс буде ретельно розроблений, щоб задовольнити потреби учнів будь-якого віку та рівня технічних навичок. Платформа включатиме принципи адаптивного дизайну, забезпечуючи доступність на різних пристроях, від настільних комп'ютерів до мобільних телефонів. Підхід, орієнтований на користувача, також поширюватиметься на налаштування, дозволяючи учням адаптувати свій досвід відповідно до особистих цілей і вподобань.

Щоб покращити навчальний процес, платформа буде інтегрувати такі функції, як мультимедійне навчання, що дозволить користувачам переглядати відео з динамічними субтитрами, які забезпечують відстеження слів і контекстно-залежний переклад. Крім того, включення інтерактивного конструктора словника дозволить користувачам зберігати нові слова та фрази в контексті, забезпечуючи ефективний перегляд і збереження. Для педагогів система надасть інструменти для розробки та проведення персоналізованих курсів, сприяючи співпраці та обміну знаннями в навчальному співтоваристві.

Завдяки поєднанню потужності передових технологій, таких як Spring Boot для бекенд-розробки, Angular для безперебійного використання інтерфейсу та AWS для масштабованого хмарного хостингу, платформа запропонує надійне та високопродуктивне середовище. Крім того, використання галузевих стандартних систем автентифікації, таких як Keycloak або Ory, забезпечить безпеку даних і конфіденційність користувачів.

РОЗДІЛ 1 ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

1.1 Аналіз функціональних вимог

Запропонована платформа вивчення мови охоплюватиме різні функціональні можливості, адаптовані до різних ролей користувачів: користувач, менеджер вмісту та адміністратор. Кожна роль матиме певні обов'язки та рівні доступу, забезпечуючи організовану та безпечну систему, одночасно сприяючи співпраці та взаємодії. У цьому розділі описано функціональні вимоги, пов'язані з цими суб'єктами, деталізовано важливі операції, такі як реєстрація та авторизація, керування вмістом та адміністративні завдання.

Для Користувача система забезпечить безперешкодний доступ до ресурсів для вивчення мови через зручний інтерфейс. Процес реєстрації призначений для забезпечення безпеки та дійсності, включаючи багатоетапну систему перевірки. Користувачі ініціюють реєстрацію, вказавши свою адресу електронної пошти, яка буде їхнім основним ідентифікатором. Для перевірки інформації система надішле одноразовий код електронною поштою. Користувачі введуть цей код, щоб підтвердити свою особу, перш ніж продовжити заповнення свого профілю, що включатиме встановлення пароля та надання додаткових відомостей, таких як бажана мова, цілі навчання. Цей перевірений процес реєстрації забезпечує автентичність облікових записів і закладає основу для персоналізованого навчання. Діаграму варіантів використання для даного актора можна побачити на рисунку 1.1.

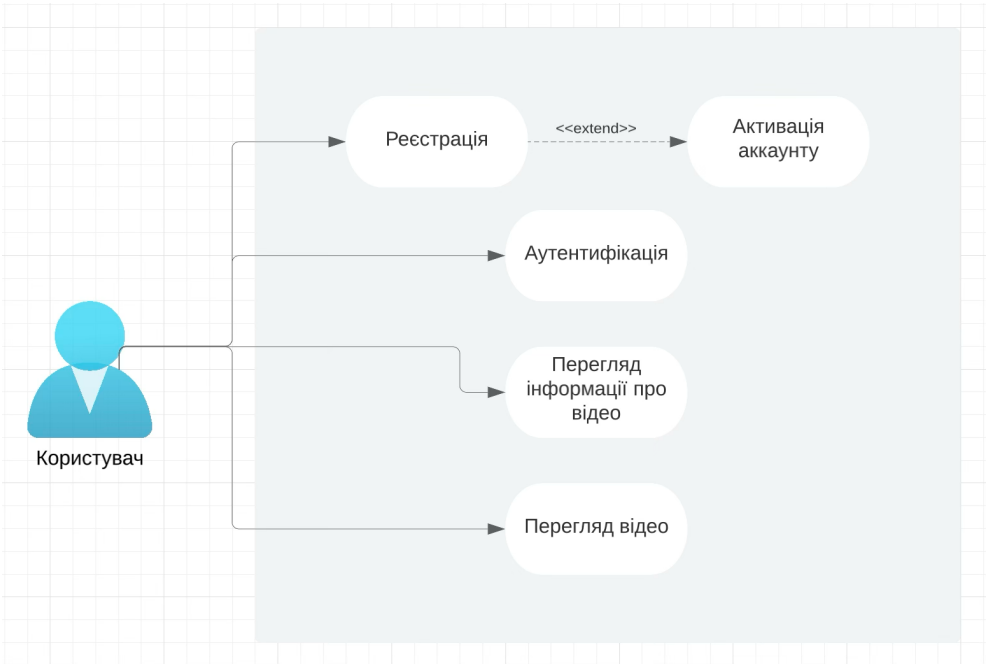

Рисунок 1.1 – Діаграма варіантів використання для Користувача

Під час процесу реєстрації система пропонує Користувачам можливість вибрати бажану мову, забезпечуючи персоналізований і доступний досвід із самого початку. За замовчуванням платформа намагатиметься визначити та попередньо вибрати найбільш підходящу мову на основі системних налаштувань клієнта, таких як мовні параметри браузера або локаль операційної системи. Ця функція забезпечує негайну локалізацію, роблячи інтерфейс реєстрації більш інтуїтивно зрозумілим і зменшуючи перешкоди для користувачів, які не знайомі з мовою сайту за замовчуванням.

Після того як користувач ініціює процес реєстрації, йому пропонується крок вибору мови. На цьому етапі інтерфейс відображає спадне меню або список доступних мов, кожна з яких супроводжується відповідною рідною назвою та піктограмою прапорця для легкої ідентифікації. Якщо система успішно визначила системну мову Користувача, ця мова вибирається попередньо, але Користувач зберігає можливість змінити її за бажанням. Ця гнучкість має вирішальне значення для сценаріїв, коли виявлена мова не відповідає перевагам користувача, наприклад, у багатомовних домогосподарствах або для осіб, які вивчають нову мову.

Вибрана мова не тільки визначає інтерфейс реєстрації, але й встановлює мову сайту за замовчуванням для користувача під час створення облікового запису. Якщо користувач пропускає чіткий вибір мови, система продовжить роботу з виявленою за замовчуванням. І навпаки, якщо користувач вибирає іншу мову, це налаштування негайно застосовується до процесу реєстрації, динамічно оновлюючи текст і мітки інтерфейсу без необхідності перезавантаження сторінки.

У серверній частині вибір мови записується як частина корисного навантаження реєстрації разом з іншими даними користувача та зберігається в базі даних. Потім ця інформація використовується для персоналізації подальших взаємодій, наприклад надання локалізованого вмісту, встановлення мови за замовчуванням для субтитрів і налаштування спілкування електронною поштою або сповіщень у додатку відповідно до вибраної Користувачем мови.

Процес огляду та перегляду відеоінформації на платформі створений для забезпечення захоплюючого, освітнього та зручного досвіду, покращеного наявністю живих перекладів для субтитрів. Ця функція є наріжним каменем інтерактивного підходу платформи до вивчення мови, що дозволяє користувачам отримувати лінгвістичне розуміння під час перегляду мультимедійного вмісту.

Перед переглядом відео користувачі можуть отримати доступ до детальної інформації про нього на спеціальній сторінці попереднього перегляду. На цій сторінці відображаються ключові метадані, як-от назва відео, опис, мова, рівень складності (наприклад, початковий, середній, просунутий) і теми, які розглядаються (наприклад, подорожі, бізнес, культура). Менеджери вмісту надають ці відомості під час процесу завантаження відео.

Крім того, сторінка може містити тривалість відео, мініатюру та короткий трейлер або зразок кліпу, щоб дати Користувачам уявлення про вміст. Якщо відео є частиною курсу, на сторінці також буде підкреслено його місце в навчальній програмі та надано посилання на інші відповідні матеріали. Список ключових словникових елементів або виразів, представлених у відео, може бути показано, щоб викликати інтерес Користувача.

Для відео з інтерактивними субтитрами сторінка попереднього перегляду пояснює цю функцію та заохочує користувачів до активної взаємодії, роблячи паузи, зберігаючи слова та вивчаючи живі переклади. Користувачі також можуть бачити створений користувачем рейтинг або розділ відгуків, щоб допомогти їм вирішити, чи відповідає вміст їхнім навчальним цілям.

Коли користувач починає переглядати відео, інтерфейс відтворення забезпечує чіткий інтерактивний дизайн, призначений для навчання. Відеопрогравач підтримує базові елементи керування, такі як відтворення, пауза, регулювання гучності та повноекранний режим, а також розширені функції, специфічні для вивчення мови.

Переклад субтитрів у реальному часі – це складна функція, призначена для підтримки користувачів різного рівня кваліфікації. Якщо ввімкнено, субтитри одночасно відображаються як мовою оригіналу, так і мовою перекладу, яку вибирає користувач. Цей дворівневий підхід гарантує, що користувачі можуть зрозуміти вміст, одночасно дотримуючись структури та словникового запасу цільової мови.

За лаштунками живий переклад забезпечується інтеграцією в реальному часі з API перекладу, таким як Google Translate або DeepL. Система динамічно обробляє текст субтитрів під час відтворення відео, забезпечуючи синхронізацію перекладу з оригінальним звуком. Щоб підвищити точність і відповідність контексту, менеджери вмісту мають можливість переглядати та вдосконалювати переклади для конкретних відео перед публікацією.

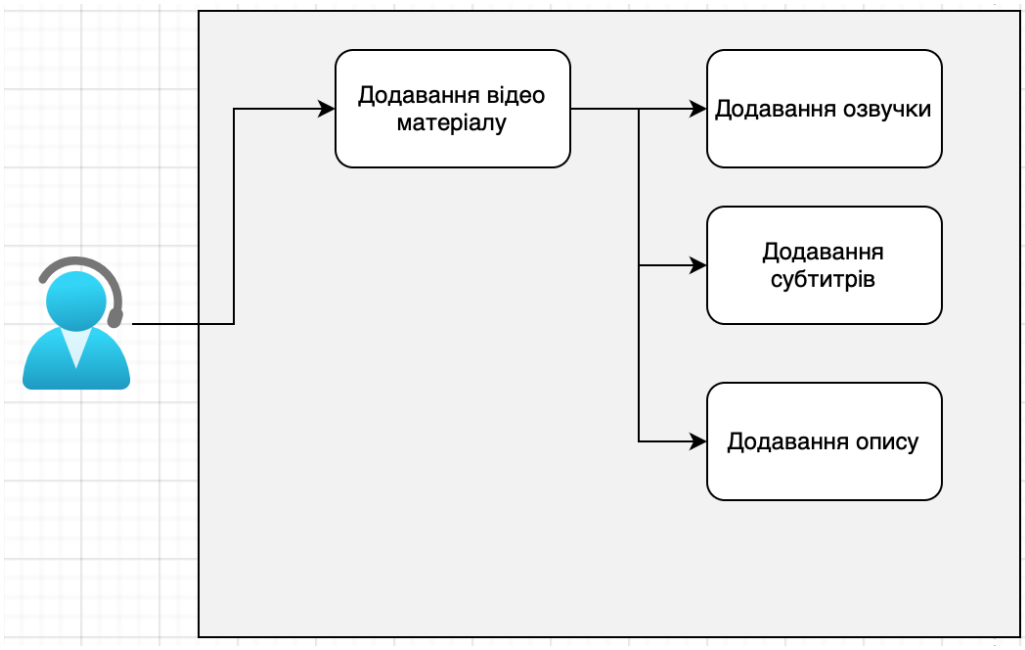

Рисунок 1.2 – Діаграма варіантів використання Контент Менеджера

Роль Контент Менеджер (діаграма варіантів використання якої зображено на рисунку 1.2) є невід'ємною частиною успіху платформи, відповідальною за кураторство та управління освітніми ресурсами, які формують основу досвіду вивчення мови. Контент-менеджери відіграють творчу та педагогічну роль, розробляючи та завантажуючи матеріали, які залучають користувачів і покращують засвоєння ними мови. Їхні внески гарантують, що платформа пропонує різноманітний, високоякісний контент, адаптований до потреб учнів на всіх рівнях кваліфікації.

Контент-менеджери починають свою роботу з підготовки відеоматеріалів, які є основним засобом вивчення мови. Цей процес включає пошук або створення відео, які відповідають освітнім цілям платформи, таким як розмовна практика, підручники з граматики, культурні ідеї або спеціалізовані теми, як-от бізнес чи подорожі. Коли відео готове, Content Manager завантажує його через спеціальний інтерфейс. Під час процесу завантаження система дотримується вказівок щодо формату файлу, роздільної здатності та розміру, щоб забезпечити сумісність і безперебійну роботу користувача.

Окрім відео, менеджери вмісту покращують вміст додатковими компонентами, які перетворюють пасивний перегляд на інтерактивне навчання.

Вони забезпечують детальні субтитри, які відповідають аудіо, забезпечуючи точність як часу, так і мовного вмісту. Ці субтитри є важливою особливістю платформи, що пропонує учням можливість стежити за текстом мовою оригіналу, а також мати доступ до перекладів, визначень і граматичних пояснень. Менеджери вмісту можуть використовувати інтегровані засоби редагування субтитрів або зовнішнє програмне забезпечення для забезпечення точності. Субтитри часто збагачені такими функціями, як слова, які можна натиснути, контекстні переклади фраз і спливаючі пояснення, які покращують розуміння.

Крім відео, менеджери вмісту завантажують допоміжні аудіоматеріали, які можуть включати посібники з вимови, вправи на аудіювання або окремі мовні вправи. Ці аудіоресурси доповнюють відео, дозволяючи учням зосередитися виключно на слухових навичках, сприяючи кращому розумінню на слух і вимові. Як і у випадку з відео, менеджери вмісту надають метадані та описи для аудіоматеріалів, допомагаючи користувачам орієнтуватися та розуміти призначення кожного ресурсу.

Для кожного вмісту менеджери вмісту створюють детальні описи, які скеровують користувачів у виборі матеріалів, які найкраще відповідають їхнім навчальним цілям. Ці описи підкреслюють рівень мови (наприклад, початковий, середній, просунутий), тематичний фокус (наприклад, бізнес, подорожі, культура) і конкретні навички, на які націлений вміст. Описи також можуть включати короткі виклади матеріалу, ключову лексику чи введені вирази та рекомендації щодо ефективного використання ресурсу. Ця інформація не тільки покращує видимість, але й допомагає користувачам визначити чіткі очікування перед тим, як працювати з вмістом.

Менеджери вмісту також відповідають за те, щоб матеріали, які вони завантажують, відповідали освітнім і технічним стандартам платформи. Вони переглядають і вдосконалюють субтитри, аудіо та описи, щоб підтримувати узгодженість, точність і відповідність потребам користувачів. Цей процес часто передбачає співпрацю з адміністраторами, які надають остаточне схвалення перед публікацією вмісту. Після опублікування матеріали стають частиною динамічної

бібліотеки, доступної для всіх користувачів, пропонуючи захоплюючий та інтерактивний спосіб вивчення нової мови.

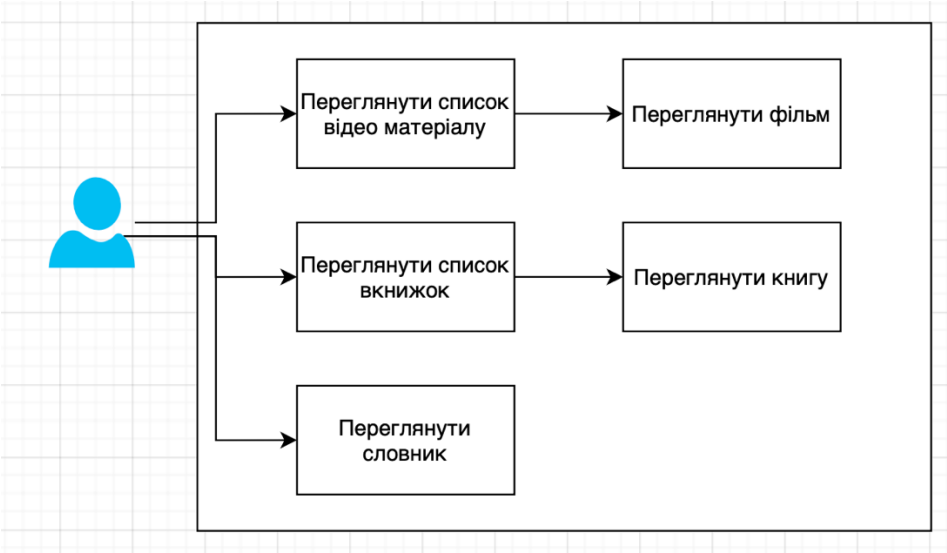

Рисунок 1.3 – Діаграма варіантів використання для клієнта

На діаграмі варіантів використання, що зображено на рисунку 1.3, описано можливі дії користувача. Користувачам надається каталог фільмів або відео з описами та метаданими, такими як рівень мови, субтитри та час виконання. Після вибору відео зі списку користувач переходить до інтерфейсу перегляду, де він може переглянути фільм. Під час відтворення інтерактивні субтитри покращують враження, дозволяючи користувачам наводити курсор на слова чи фрази, отримувати миттєві переклади та додавати їх до свого словника. Цей процес робить перегляд відео як освітнім, так і розважальним, сприяючи засвоєнню мови в контексті.

Друга дія передбачає перегляд списку книг. Подібно до функції перегляду відео, користувачам надається бібліотека текстів, класифікованих за рівнем мови, жанром або темою. Вибравши книгу, вони потрапляють в інтерактивний інтерфейс читання, де вони можуть читати та взаємодіяти з вмістом. Досвід читання доповнюється інструментами для виділення слів або фраз, які забезпечують переклади та інформацію, залежну від контексту. Ці виділені терміни також можна зберегти безпосередньо в особистому словнику користувача, створюючи

інтегрований та ефективний спосіб розширити словниковий запас, насолоджуючись літературою.

Третя дія забезпечує доступ до особистого словника. Ця функція об'єднує всі слова і фрази, збережені користувачем під час перегляду відео або читання книг. Словник пропонує детальну інформацію для кожного запису, включаючи переклади, контекст (наприклад, речення чи абзац, де зустрічається термін) і будь-які додані користувачем примітки. Користувачі можуть упорядковувати, переглядати та практикувати свій збережений словниковий запас, зміцнюючи свій прогрес у вивченні мови за допомогою періодичних сеансів перегляду.

Потік підкреслює взаємопов’язаний характер цих функцій, демонструючи, як користувачі можуть переходити між вивченням нового вмісту та закріпленням того, що вони вивчили. Завдяки об’єднанню відеоматеріалів, книг та інтерактивного словника програма забезпечує цілісне та захоплююче середовище для вивчення мови, гарантуючи, що користувачі залишаються залученими та продуктивними протягом усього подорожі.

1.2 Аналіз нефункціональних вимог

Програма має відповідати надійному набору нефункціональних вимог, щоб гарантувати безперебійну, надійну та масштабовану роботу для своїх користувачів. Ці вимоги охоплюють кілька аспектів, включаючи продуктивність, безпеку, придатність до обслуговування, масштабованість, зручність використання та відповідність вимогам, кожна з яких є критично важливою для успіху платформи як глобального ресурсу для вивчення мов.

Продуктивність є наріжним каменем програми, оскільки користувачі очікують швидкого реагування та безперебійної роботи незалежно від свого місцезнаходження чи пристрою. Платформа має забезпечувати швидке завантаження відео та допоміжних матеріалів, навіть якщо доступ до них

здійснюється через повільніші мережеві з’єднання. Система повинна підтримувати низьку затримку в інтерактивних функціях, таких як динамічні субтитри, живі переклади та інструменти для збереження словникового запасу, щоб уникнути збоїв у процесі навчання. Ці можливості вимагають ефективної серверної архітектури, оптимізованих запитів до бази даних і ефективних стратегій кешування.

Масштабованість є ще одним важливим фактором, оскільки платформа повинна враховувати зростаючу базу користувачів і збільшення обсягу вмісту без шкоди для продуктивності. Це передбачає розробку масштабованої інфраструктури, здатної впоратися зі сплесками активності користувачів, наприклад під час рекламних заходів або запуску нових навчальних матеріалів. Архітектура повинна підтримувати плавне додавання функцій, мов і ресурсів з часом, забезпечуючи адаптацію системи до мінливих вимог.

Безпека має першочергове значення, враховуючи конфіденційність даних користувачів, зокрема облікових даних, платіжної інформації та особистих уподобань. Платформа повинна вживати суворих заходів для захисту від несанкціонованого доступу, витоку даних та інших кіберзагроз. Це включає використання надійних протоколів шифрування, регулярні перевірки безпеки та відповідність галузевим стандартам, таким як GDPR або CCPA. Крім того, необхідно запровадити контроль доступу на основі ролей, який гарантує, що користувачі, менеджери вмісту та адміністратори можуть виконувати лише дії, які відповідають їхнім ролям.

Підтримка має вирішальне значення для довговічності платформи, тому потрібен чистий модульний код, який полегшує оновлення та усунення несправностей. Команда розробників повинна дотримуватися найкращих практик у розробці програмного забезпечення, включаючи ретельну документацію, автоматизоване тестування та контроль версій. Ці заходи гарантують додавання нових функцій і виправлення помилок без регресії чи шкоди стабільності.

Надійність є ключовою нефункціональною вимогою, оскільки користувачі повинні довіряти платформі бути доступною щоразу, коли вони хочуть отримати до неї доступ. Система повинна мати високу доступність із задіяними механізмами

резервування та відновлення після збоїв, щоб мінімізувати час простою. Необхідно створити плани регулярного резервного копіювання та аварійного відновлення, щоб захистити від втрати даних і забезпечити швидке відновлення у разі збоїв апаратного забезпечення або інших непередбачених проблем.

Відповідність міжнародним стандартам і правилам є ще одним важливим фактором. Платформа повинна поважати закони про інтелектуальну власність під час обробки стороннього контенту, наприклад відео та субтитрів, а також гарантувати, що дані користувачів збираються та обробляються прозоро та законно. Дотримання цих стандартів створює довіру користувачів і дозволяє уникнути юридичних ускладнень.

Нарешті, додаток має виконувати свою освітню місію, забезпечуючи послідовний, привабливий досвід. Це включає забезпечення точності живих перекладів, підтримку високого стандарту якості вмісту та підтримку інноваційних функцій, які мотивують користувачів.

РОЗДІЛ 2 ВИКОРИСТАННІ ТЕХНОЛОГІЇ

2.1 Опис архітектури серверної частини

Проаналізувавши вимоги, можна дійти висновку, що мікросервісна архітектура буде застосована для розробки ресурсу. Це підхід до проектування програмного забезпечення, коли додаток розділено на невеликі незалежні служби, кожна з яких обробляє певну бізнес-функцію чи процес. Ці служби, відомі як мікросервіси, слабко пов’язані між собою та взаємодіють між собою через чітко визначені API, як правило, за допомогою легких протоколів, таких як HTTP або черги обміну повідомленнями. Кожен мікросервіс є автономним, тобто його можна розробляти, розгортати та підтримувати незалежно від інших, і зазвичай має власну базу даних і стратегію керування даними.

В архітектурі мікросервісів кожна служба має невеликий обсяг, зосереджена на одній частині функціональності, що полегшує розуміння, розробку та підтримку всієї системи. Наприклад, на платформі вивчення мови мікросервіси можна використовувати для керування обліковими записами користувачів, обробки платежів, обробки відеовмісту, надання субтитрів, керування тестами та навіть перекладу тексту в реальному часі. Ці служби можуть масштабуватися незалежно, при цьому кожна служба відповідає за певний аспект платформи [2].

Однією з ключових переваг впровадження мікросервісної архітектури є масштабованість. У міру того як кількість користувачів і трафіку зростає, конкретні мікросервіси, які мають більший попит, можна масштабувати незалежно. Наприклад, якщо потокове передавання відеоконтенту стане найбільш ресурсомісткою функцією, мікросервіс потокового відео можна розширити, не впливаючи на інші частини системи. Це забезпечує ефективне використання ресурсів і кращу продуктивність, особливо під час пікового використання.

Мікросервіси також пропонують гнучкість щодо вибору технологій. Оскільки кожен сервіс є незалежним, різні мікросервіси можна створити за допомогою найбільш підходящої технології або мови програмування для

конкретного завдання. Наприклад, послуга потокового відео може бути побудована з використанням технології, оптимізованої для обробки великих обсягів медіа, тоді як служба обробки платежів може використовувати структуру, що спеціалізується на безпечних транзакціях. Ця технологічна різноманітність дозволяє розробникам вибирати найкращий інструмент для кожного мікросервісу, що забезпечує кращу загальну продуктивність і надійність системи.

Крім того, мікросервіси забезпечують безперервне розгортання та швидку ітерацію. Оскільки кожен мікросервіс є окремою одиницею, нові функції, виправлення помилок або оновлення можна зробити для одного сервісу, не порушуючи роботу всієї системи. Таке відокремлення служб дозволяє командам випускати оновлення частіше, не ризикуючи простоєм усієї програми. Наприклад, якщо в мікросервісі перекладу виявлено помилку, її можна виправити та розгорнути незалежно від потокового відео або служб автентифікації користувача, зменшуючи час простою та покращуючи швидкість розробки.

Мікросервіси також покращують ремонтопридатність і усунення несправностей. Якщо мікросервіс дає збій або має проблеми, він ізольований від решти системи, запобігаючи повному виходу з ладу системи. Ця ізоляція дозволяє швидше ідентифікувати та вирішувати проблеми, а також спрощує моніторинг і налагодження. У випадку платформи вивчення мови, якщо служба генерації субтитрів має помилку, її можна вирішити, не впливаючи на потокове відео або служби вікторин. Це зменшує складність обробки великих проблем і забезпечує більш високу доступність для користувачів.

Крім того, мікросервіси підтримують гнучкі практики розробки та автономність команди. Оскільки кожна мікрослужба є невеликою незалежною одиницею, групи розробників можуть бути організовані навколо конкретних служб, що дозволяє спеціалізуватися та мати досвід. Ці команди можуть одночасно працювати над різними службами, не наступаючи одна одній навпроти, прискорюючи цикли розробки та підвищуючи продуктивність. Наприклад, одна команда може зосередитися на управлінні користувачами, інша — на відеоконтенті,

а трета — на перекладі в реальному часі. Такий розподіл праці дозволяє ефективніше розробляти та швидше виводити нові функції на ринок.

З точки зору бізнесу, мікросервіси також забезпечують гнучкість щодо розгортання та масштабування. За допомогою мікросервісів ви можете розгортати різні частини програми на окремих серверах, використовувати хмарні служби для масштабування та використовувати технології контейнеризації, такі як Docker і Kubernetes, для автоматизації процесу розгортання. Це гарантує, що програму можна легко поширювати та масштабувати в багатьох середовищах та інфраструктурах, таких як локальні сервери чи хмара. Він також надає можливість скористатися перевагами хмарних функцій, таких як автоматичне масштабування, балансування навантаження та оптимізація ресурсів.

2.2 Опис архітектури клієнтської частини

Рішення про використання архітектури Single Page Application (SPA) із зв’язком HTTP у розробці платформи вивчення мови обумовлено декількома стратегічними цілями, спрямованими на оптимізацію продуктивності, взаємодії з користувачем і масштабованості. У SPA клієнтська сторона програми побудована як єдина динамічна веб-сторінка, яка взаємодіє з компонентами на стороні сервера через запити HTTP. Цей підхід забезпечує більш сучасний і ефективний спосіб створення веб-додатків, оскільки дозволяє користувачам безперебійно взаємодіяти з платформою без необхідності перезавантажувати сторінки чи переривати робочий процес [3].

В основі цього дизайну лежить бажання покращити взаємодію з користувачем. У традиційних багатосторінкових програмах (MPA) кожного разу, коли користувач взаємодіє з інтерфейсом, наприклад натискає посилання або надсилає форму, вся сторінка перезавантажується, що призводить до затримок і призводить до неузгодженості роботи. З іншого боку, SPA заздалегідь завантажують

усі необхідні ресурси, включаючи HTML, JavaScript і CSS, і за потреби динамічно оновлюють вміст без перезавантаження всієї сторінки. Це дозволяє користувачам плавно переміщатися по платформі, миттєво переглядати новий вміст, взаємодіяти з відео, субтитрами, тестами чи будь-яким іншим навчальним матеріалом без затримок, які зазвичай пов’язані з перезавантаженням сторінок. Така поведінка особливо корисна на платформі вивчення мови, де користувачі очікують швидких відповідей і миттєвого зворотного зв’язку під час перегляду відео, читання динамічних субтитрів або тренування мовних навичок.

Використовуючи HTTP для зв’язку між SPA та сервером, ми можемо оптимізувати обмін даними між ними. HTTP — це легкий протокол, який дозволяє програмі на стороні клієнта робити асинхронні запити на сервер для даних, які потім використовуються для оновлення сторінки. Наприклад, коли користувач переглядає відео із субтитрами, SPA надсилає HTTP-запити до сервера, щоб отримати відповідний вміст субтитрів або відеодані, і лише цей конкретний вміст оновлюється на сторінці. Це означає, що всю сторінку не потрібно перезавантажувати щоразу, коли користувач виконує дію. Натомість сторінка динамічно налаштовується на основі відповіді сервера, що призводить до швидшого часу завантаження та кращої чутливості платформи.

Зв'язок HTTP також відіграє важливу роль у підвищенні ефективності сервера. У типовому SPA програма на стороні клієнта запитує лише певні дані, які їй потрібні в будь-який момент, на відміну від вилучення цілих HTML-сторінок або перезавантаження цілих розділів веб-сайту. Це мінімізує обсяг даних, що передаються між клієнтом і сервером, зменшуючи мережевий трафік і навантаження на ресурси сервера. Наприклад, якщо користувачі платформи беруть участь у сеансі вивчення мови, клієнт може запросити лише дані уроку, результати вікторин або оновлений вміст субтитрів із сервера замість того, щоб перезавантажувати цілу сторінку вмісту. У результаті загальна система стає більш ефективною, а продуктивність програми покращується.

Однією з значних переваг використання архітектури SPA з HTTP є те, що вона зменшує складність розробки та підтримки системи. Інтерфейсний і внутрішній

компоненти програми розділені, що означає, що інтерфейс користувача може розвиватися незалежно від логіки на стороні сервера. Це забезпечує швидшу розробку та більшу гнучкість, коли йдеться про оновлення або додавання нових функцій. Наприклад, до платформи можна додавати нові мови чи відеовміст, не вимагаючи жодних змін у базовій інфраструктурі, оскільки інтерфейс просто спілкується із сервером через стандартизовані HTTP API. Відокремлення компонентів на стороні клієнта та стороні сервера також гарантує, що система є більш модульною та зручною для обслуговування, оскільки оновлення однієї сторони програми не обов'язково впливають на іншу сторону.

Ще одна ключова перевага використання зв'язку HTTP полягає в тому, що він забезпечує кращу масштабованість. Внутрішній сервер, який зазвичай створюється з використанням RESTful API або мікросервісів, може бути розроблений для обробки кількох запитів одночасно, причому кожен мікросервіс виконує певну функцію, таку як потокове відео, керування користувачами або створення субтитрів у реальному часі. Це дозволяє системі легше масштабуватись у відповідь на підвищений попит. Наприклад, якщо спостерігається різке збільшення кількості користувачів, які отримують доступ до відеоконтенту, мікросервіс потокового відео можна масштабувати незалежно від інших сервісів, забезпечуючи оптимальну роботу програми навіть із зростанням трафіку. Подібним чином, якщо до платформи додаються нові функції або функції, сервер можна модифікувати або розширити, не впливаючи на клієнтську програму, доки протоколи зв'язку (HTTP API) залишаються послідовними.

Окрім масштабованості та гнучкості, використання SPA також підвищує загальну продуктивність програми. SPA, як правило, завантажуються швидше, ніж традиційні MPA, оскільки вони завантажують початкову сторінку лише один раз, а потім отримують лише необхідні дані. Це скорочує час очікування перезавантаження сторінок і дозволяє користувачам продовжити взаємодію з платформою майже негайно. Коли користувачі взаємодіють із відеовмістом, субтитрами, тестами чи іншими навчальними інструментами, вони очікують швидких відповідей, особливо якщо вони залучені в активні навчальні сесії.

Здатність SPA оновлювати вміст без оновлення всієї сторінки відповідає цим очікуванням і робить платформу більш привабливою.

Крім того, архітектура SPA також підтримує крос-платформну сумісність, що є важливою функцією для сучасних програм, які повинні працювати на кількох пристроях. Оскільки інтерфейс створено з використанням таких веб-технологій, як HTML, CSS і JavaScript, ту саму кодову базу можна використовувати на різних платформах, включаючи веб-браузери, смартфони та планшети. Використання зв'язку HTTP гарантує, що програма може взаємодіяти з внутрішнім сервером незалежно від пристрою, що робить платформу більш доступною для більш широкого кола користувачів. Це особливо цінно для платформи вивчення мови, оскільки користувачі можуть захотіти перемикатися між пристроями, наприклад, переходити з настільного комп'ютера на мобільний телефон або планшет, зберігаючи при цьому свій прогрес і вміст.

Нарешті, однією з причин вибору SPA з HTTP-зв'язком є простота та ефективність інтеграції сучасних інструментів розробки та фреймворків. Такі технології, як React, Angular або Vue.js, пропонують потужні можливості для створення динамічних, високопродуктивних інтерфейсів користувача, тоді як серверні служби можна створювати за допомогою сучасних серверних технологій, таких як Node.js, Spring Boot або Django. Завдяки використанню HTTP для зв'язку між клієнтом і сервером програма стає дуже гнучкою та легкою для інтеграції з різними зовнішніми службами чи інструментами, такими як платіжні шлюзи, служби обробки відео або API мовного перекладу. Ця гнучкість дає змогу розробникам швидко створювати й ітерувати функції, полегшуючи вдосконалення платформи новими можливостями в міру розвитку потреб користувачів.

Підсумовуючи, рішення запровадити платформу вивчення мови з використанням архітектури SPA із зв'язком HTTP було прийнято з метою створення швидкої, ефективної та масштабованої системи, яка забезпечує виняткову взаємодію з користувачем. Поєднання безперебійних динамічних оновлень SPA з простотою та гнучкістю зв'язку HTTP гарантує, що платформа може розвиватися з часом, задовольняючи вимоги користувачів, зберігаючи продуктивність і

надійність. Ця архітектура також підтримує зростання платформи, забезпечуючи легку інтеграцію з новими службами, технологіями та функціями, зберігаючи незалежність і масштабованість зовнішніх і внутрішніх компонентів. Як наслідок, користувачі отримають плавну та захоплюючу подорож до вивчення мови, а розробники отримають переваги від модульної системи, яку можна підтримувати.

2.3 Огляд використаних технологій при розробці клієнтської частини

Під час розробки програми було ретельно відібрано різноманітні клієнтські технології, щоб забезпечити сучасний, ефективний і зручний інтерфейс. Ці технології були вибрані на основі їхньої здатності підтримувати динамічну природу додатка, швидкість реагування та масштабованість, одночасно забезпечуючи бездоганний досвід для користувачів. Кожна технологія внесла унікальний внесок у процес розробки, вирішуючи конкретні проблеми та покращуючи загальну якість платформи.

Angular був основним фреймворком, який використовувався для розробки на стороні клієнта. Будучи надійною та широко поширеною структурою, яку підтримує Google, Angular забезпечує потужну структуру для створення динамічних односторінкових програм (SPA). Його компонентна архітектура дозволила розробникам створювати модульні багаторазові елементи, спрощуючи процес розробки та забезпечуючи узгодженість у програмі. Такі функції, як двостороннє зв’язування даних, ін’єкція залежностей і вбудована маршрутизація, відіграли важливу роль у створенні інтерактивного та чутливого користувача. Комплексна екосистема Angular також сприяла інтеграції бібліотек сторонніх розробників, покращуючи функціональність і скорочуючи час розробки.

HTML служив основою для структурування інтерфейсу користувача програми. Це дозволило розробникам чітко та семантично визначити макет і елементи платформи, забезпечивши доступність і сумісність на різних пристроях і

браузерах. Використання сучасних функцій HTML5, таких як власні аудіо- та відеоелементи, розширило мультимедійні можливості програми, зробивши її ідеальною для відображення відеовмісту з інтерактивними субтитрами. Це гарантувало, що користувачі могли безперешкодно працювати з навчальними матеріалами незалежно від свого пристрою чи браузера.

Для стилізації програми використовувався SCSS (Sassy CSS) як розширення CSS, що забезпечує такі потужні функції, як змінні, вкладення та міксини. Це дозволило розробникам писати чіткіші та зручніші таблиці стилів, зберігаючи при цьому високий ступінь налаштування. SCSS був особливо ефективним у забезпеченні узгодженого вигляду та відчуття на всій платформі, а також уможливлюючи швидке коригування дизайну, коли це необхідно. Його модульна природа доповнювала компонентний підхід Angular, дозволяючи легко визначати стилі для конкретних компонентів, не впливаючи на інші. Це сприяло створенню візуально привабливого та професійного інтерфейсу.

TypeScript відіграв вирішальну роль у покращенні процесу розробки, додавши до JavaScript статичний тип. Ця функція значно зменшила кількість помилок під час розробки, увімкнувши перевірку типів і забезпечивши кращу підтримку інструментів, наприклад автозавершення та інтелектуальну навігацію кодом. Об’єктно-орієнтовані можливості TypeScript добре узгоджуються з фреймворком Angular, що полегшує написання надійного та масштабованого коду. Використовуючи TypeScript, розробники можуть реалізувати складні функції, такі як динамічний рендеринг вмісту та взаємодія в реальному часі, з більшою впевненістю та ефективністю. Використання TypeScript також гарантувало, що кодова база була зручною для обслуговування та легшою для налагодження, що було особливо цінним у проекті з довгостроковим баченням.

Доповнюючи ці основні технології, JavaScript служив основою для динамічної поведінки та взаємодії на стороні клієнта. Незважаючи на те, що Angular і TypeScript абстрагували велику частину складності, універсальність JavaScript залишалася критичною для інтеграції додаткових функцій і забезпечення сумісності з існуючими бібліотеками. Розгалужена екосистема інструментів і

бібліотек дозволила розробникам розширити можливості програми за межі пропозицій за замовчуванням Angular, покращивши загальний досвід роботи з користувачем.

Сучасні веб-інтерфейси API також зіграли важливу роль у розробці клієнтської програми. Такі API, як Fetch API, використовувалися для ефективного зв'язку з серверною частиною, дозволяючи SPA динамічно оновлювати вміст без перезавантаження сторінки. Це мало вирішальне значення для забезпечення плавної взаємодії з користувачем, особливо коли користувачі взаємодіяли з відео, субтитрами чи тестами. Крім того, використання API для локального зберігання та керування сеансами забезпечило збереження налаштувань користувача та даних сеансу під час взаємодії, що ще більше підвищило зручність використання програми.

Нарешті, додаток розроблено таким чином, щоб бути оперативним і доступним, використовуючи фреймворки та методи для забезпечення сумісності з широким діапазоном пристроїв і розмірів екранів. Принципи адаптивного дизайну в поєднанні з інструментами Angular для обробки макета та маршрутизації дозволили додатку витончено адаптуватися до настільних ПК, планшетів і смартфонів. Це забезпечило послідовний і приємний досвід для всіх користувачів, незалежно від їхнього пристрою чи платформи.

Інтернаціоналізація (i18n) є ключовим аспектом сучасної розробки програмного забезпечення, особливо для програм, призначених для охоплення глобальної аудиторії. Це стосується процесу проектування та створення програмного забезпечення таким чином, щоб його було легко адаптувати до різних мов, культур і регіонів, не вимагаючи значних змін кодової бази. Для платформи вивчення мови i18n є не просто функціональною вимогою, а фундаментальною функцією, оскільки користувачі з різним лінгвістичним походженням покладаються на програму для ефективної та інтуїтивно зрозумілої доставки вмісту мовою, яку вони віддають перевагу.

У контексті цієї програми модуль i18n Angular використовувався для оптимізації реалізації інтернаціоналізації. Angular надає вбудовану підтримку для

перекладу текстів інтерфейсу користувача, дат, чисел та інших форматів, що залежать від локалі. Завдяки екстерналізації всього тексту, який відкриває користувач, у файли перекладу, розробники змогли відокремити логіку програми від її вмісту, що спростило додавання або оновлення мов. Ці файли перекладу, часто у таких форматах, як XLIFF або JSON, слугували сховищами для локалізованих рядків, дозволяючи програмі динамічно завантажувати відповідні переклади на основі вибраної користувачем мови або мови за замовчуванням.

Реалізація i18n на платформі вийшла за рамки простого перекладу тексту та включила підтримку форматування дат, часу, валют та іншої конфіденційної інформації щодо культурних особливостей. Наприклад, різні мови можуть використовувати різні формати для написання дат або відображення символів валют, і програма повинна була врахувати ці нюанси, щоб забезпечити природну та інтуїтивно зрозумілу роботу. Можливості Angular i18n включають вбудовані конвеєри та функції локалізації, які автоматично налаштовують ці формати відповідно до локалі користувача, зменшуючи потребу в настроюваній логіці та забезпечуючи точність.

Важливою частиною стратегії i18n було увімкнення визначення мови та динамічного перемикання. За замовчуванням програма може визначити мову системи або браузера користувача та автоматично завантажити відповідні файли локалізації. Це гарантувало, що нові користувачі користуються програмою улюбленою мовою без необхідності ручного вибору. Для користувачів, які хотіли змінити мову, була реалізована функція динамічного перемикання, яка дозволяла програмі перезавантажувати відповідні файли перекладу без оновлення сторінки. Це гарантувало, що перехід між мовами був плавним, зберігаючи контекст користувача та постійну взаємодію.

Інтернаціоналізація також поширилася на мультимедійний вміст, такий як субтитри та описи. Оскільки платформа забезпечує навчання на основі відео, субтитри повинні бути синхронізовані з вмістом і точно перекладені кількома мовами. Це вимагало інтеграції методів i18n у систему керування субтитрами, щоб користувачі могли перемикатися між мовами під час перегляду відео. Для курсів

або навчальних модулів метадані й описи також локалізовано, щоб підтримувати послідовність і доступність усього вмісту.

Нарешті, i18n розширив здатність платформи до глобального масштабування. Розробляючи систему з урахуванням інтернаціоналізації, можна додавати нові мови з мінімальними зусиллями, просто створюючи додаткові файли перекладу та оновлюючи конфігурацію. Цей модульний підхід гарантував, що програма може розширюватися на нові ринки, не вимагаючи значних переробок, позиціонуючи її як справді глобальне рішення для тих, хто вивчає мову.

Також було використано PrimeNG бібліотеку для Angular. Це комплексна та багатофункціональна бібліотека компонентів інтерфейсу користувача, розроблена спеціально для програм Angular. Він надає розробникам широкий спектр попередньо розроблених настроюваних компонентів, які оптимізовані для продуктивності та зручності використання. PrimeNG, розроблений компанією PrimeTek, широко поширений в екосистемі Angular і цінується за простоту використання, швидкість реагування та здатність значно пришвидшувати терміни розробки. Завдяки інтеграції PrimeNG розробники можуть створювати користувальницькі інтерфейси професійного рівня з мінімальними зусиллями, забезпечуючи узгодженість і масштабованість у програмі.

Однією з видатних особливостей PrimeNG є його велика колекція компонентів, яка охоплює практично всі аспекти розробки інтерфейсу користувача. До них належать основні елементи, такі як кнопки, спадні списки та поля введення, а також більш розширені компоненти, такі як таблиці даних, діаграми та засоби завантаження файлів. Кожен компонент розроблено з урахуванням як функціональності, так і естетики, пропонуючи сучасний і відшліфований вигляд із коробки. PrimeNG також підтримує реактивні форми Angular і форми, керовані шаблонами, що робить його легкою інтеграцією з перевіркою форм і механізмами зв’язування даних.

Компоненти PrimeNG легко налаштовуються, що дозволяє розробникам адаптувати їх зовнішній вигляд і поведінку відповідно до вимог конкретної програми. Бібліотека підтримує теми та налаштування на основі CSS, що дозволяє

розробникам створювати цілісну візуальну ідентичність для своїх проектів. Крім того, PrimeNG містить конструктор тем, спеціалізований інструмент, який спрощує процес налаштування тем. Ця гнучкість особливо цінна для створення додатків з унікальним брендом або стильовими рекомендаціями.

Ще однією ключовою перевагою PrimeNG є його зосередженість на доступності та оперативності. Усі компоненти розроблено таким чином, щоб повністю реагувати, забезпечуючи елегантну адаптацію інтерфейсу користувача до різних розмірів екрана та пристроїв. Це робить PrimeNG чудовим вибором для створення додатків, які вимагають узгодженого функціонування на настільних ПК, планшетах і мобільних пристроях. Крім того, бібліотека дотримується Рекомендацій щодо доступності веб-вмісту (WCAG), що дає змогу розробляти інклюзивні програми для людей з обмеженими можливостями.

2.4 Технології для розробки серверної частини

До вибору серверних технологій підходили з усією ретельністю, оскільки це безпосередньо впливало на продуктивність програми, її масштабованість, зручність обслуговування та загальну надійність. Технології на стороні сервера утворюють основу системи, обробляючи важливі операції, такі як обробка даних, маршрутизація запитів і виконання бізнес-логіки. Вибрані технології мали відповідати вимогам програми, забезпечуючи ефективне управління ресурсами, безперебійний зв’язок із стороною клієнта та адаптованість до майбутніх потреб.

2.4.1 Опис основної мови програмування та фреймворків

Java і Spring Boot були обрані як основні технології для розробки на стороні сервера через їх перевірену стабільність, масштабованість і розгалужену екосистему. Java, будучи однією з найпоширеніших мов програмування, забезпечила міцну та зрілу основу для створення програми. Його незалежність від платформи та широка підтримка бібліотеки зробили його ідеальним вибором для створення високопродуктивної серверної частини, здатної задовольнити вимоги програми.

Spring Boot було обрано через його здатність оптимізувати процес розробки та спростити конфігурацію складних програм на стороні сервера. Використовуючи Spring Boot, архітектуру програми було створено з упором на модульність, масштабованість і зручність обслуговування. Вбудовані сервери інфраструктури, можливості автоматичної конфігурації та попередньо створені залежності дозволили швидко розробляти та розгортати, зменшуючи накладні витрати та мінімізуючи шаблонний код. Це було особливо корисно при створенні мікросервісів для певних функцій, гарантуючи, що кожен сервіс можна розробляти, розгортати та масштабувати незалежно [4].

На рішення використовувати Spring Boot також вплинула його підтримка сучасних парадигм програмування, таких як реактивне програмування. Такі функції, як Spring WebFlux, дозволяли програмі ефективно обробляти неблокувальні асинхронні операції, що мало вирішальне значення для керування взаємодіями в реальному часі, такими як оновлення субтитрів і обробка даних у реальному часі. Крім того, інтеграція зі сторонніми службами та API була спрощена завдяки комплексній підтримці Spring Boot для служб RESTful і бездоганній інтеграції з такими інструментами, як OAuth для автентифікації.

Сильний набір тексту та ефективність виконання Java гарантували, що серверна частина може обробляти великі обсяги даних із надійністю та точністю. У поєднанні з вбудованими механізмами безпеки Spring Boot, такими як захист CSRF

і керування доступом на основі ролей, програма була оснащена необхідними засобами захисту для захисту даних користувачів і підтримки цілісності системи.

Таким чином, поєднання Java і Spring Boot забезпечило міцну основу для серверної архітектури. Їхній вибір дозволив створити продуктивну, масштабовану та безпечну серверну частину, яка відповідала б цілям програми, забезпечуючи довгострокову життєздатність і легкість вдосконалення відповідно до потреб користувачів.

Щоб увімкнути динамічне масштабування та безперебійне спілкування між службами, Spring Cloud було представлено як частину стратегії мікросервісу. Spring Cloud надав необхідні інструменти для керування розподіленими системами, включаючи виявлення служб, керування конфігурацією та розрив ланцюга. Використання Spring Cloud Config дозволило централізовано керувати властивостями конфігурації, забезпечуючи послідовність і легкість оновлень у мікросервісах. Такі функції, як Eureka для реєстру служб і стрічка для балансування навантаження на стороні клієнта, гарантували, що служби могли динамічно виявляти та спілкуватися одна з одною, зберігаючи стійкість і масштабованість.

Spring Kubernetes відіграв ключову роль у розгортанні та управлінні програмою в контейнерних середовищах. Завдяки використанню можливостей оркестровки Kubernetes мікросервіси програми було розгорнуто як контейнери, забезпечуючи переносимість і спрощене масштабування. Spring Kubernetes повністю інтегрував програми Spring із Kubernetes, увімкнувши такі функції, як ConfigMaps і Secrets для безпечної та динамічної конфігурації, а також горизонтальне автомасштабування модулів для ефективної обробки змінних навантажень. Ця інтеграція забезпечила горизонтальне масштабування мікросервісів для задоволення попиту, зберігаючи стабільну продуктивність.

Комбіноване використання Spring Boot, Spring Cloud і Spring Kubernetes створило потужну гнучку серверну інфраструктуру. Spring Boot забезпечив базову структуру, Spring Cloud покращив керування розподіленими службами, а Spring Kubernetes сприяв ефективному розгортанню та масштабуванню в хмарних середовищах. Разом ці технології дозволили розробити сучасну, стійку та

масштабовану програму, здатну розвиватися разом із вимогами користувачів і технологічним прогресом [5].

Реактивне програмування, зокрема Project Reactor, було прийнято для розробки додатка, щоб підвищити його чуйність і масштабованість, зокрема під час обробки асинхронних неблокуючих операцій. Реактивне програмування — це підхід, який дозволяє системам бути більш ефективними в обробці великих обсягів одночасних запитів, не блокуючи потоки для операцій вводу-виводу, що є критичним для програм із високою пропускною спроможністю та взаємодією даних у реальному часі. У цьому випадку Project Reactor, основоположна частина фреймворку Spring WebFlux, була обрана для створення реактивної магістралі програми.

Project Reactor забезпечує потужний і гнучкий спосіб обробки асинхронних потоків даних за допомогою реактивних потоків, які розроблені для підтримки неблокуючого зворотного тиску та керованих подіями архітектур. Це робить його дуже придатним для додатків, які потребують ефективної обробки даних, таких як переклад субтитрів у реальному часі, керування вмістом і взаємодія з користувачем. Бібліотека дозволяє розробникам працювати з типами Mono та Flux, які представляють одне та кілька асинхронних значень відповідно.

Завдяки застосуванню реактивного програмування з Project Reactor серверна частина програми змогла обробляти кілька запитів користувачів одночасно, не блокуючи потоки. Наприклад, коли користувач запитує відеовміст або оновлення субтитрів у реальному часі, система може обробити ці запити асинхронно. Під час очікування зовнішніх даних, таких як переклад субтитрів або завантаження вмісту із зовнішніх API, потоки програми залишаються вільними для обробки інших завдань, покращуючи загальну продуктивність і швидкість реагування. Це було особливо цінно для платформи вивчення мови, де користувачі взаємодіють із відео та живим вмістом, а затримки можуть негативно вплинути на взаємодію з користувачем [6].

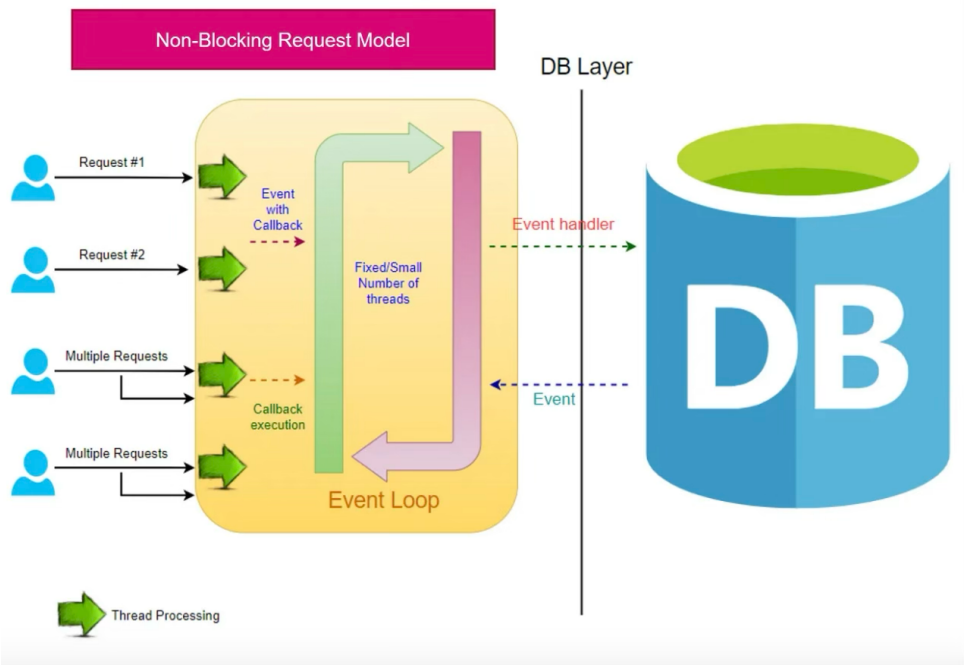

Рисунок 3.1 – Зображення неблокуючого підходу

Крім того, реактивне програмування за допомогою Project Reactor спрощує керування складними робочими процесами, які включають кілька асинхронних завдань (приклад зображено на рисунку 3.1). Наприклад, отримання субтитрів, їх переклад і синхронізація з відеовмістом оброблялися асинхронно за допомогою реактивних потоків. Використання зворотного тиску, функції Project Reactor, гарантувало, що система могла витончено керувати ситуаціями, коли дані створювалися швидше, ніж вони могли споживатися, таким чином запобігаючи перевантаженням і зберігаючи стабільність системи.

Інтеграція реактивного програмування забезпечила кращу масштабованість програми, оскільки вона могла обробляти все більшу кількість одночасних користувачів і запити даних із меншими ресурсами. Завдяки мінімізації кількості потоків, заблокованих операціями введення-виведення, було оптимізовано використання ресурсів сервера, забезпечуючи кращу продуктивність під час пікових навантажень. Крім того, неблокуючий характер реактивного програмування добре узгоджувався з архітектурою мікросервісів, дозволяючи службам самостійно масштабуватися та реагувати на дії користувачів у режимі реального часу.

Підсумовуючи, впровадження реактивного програмування з Project Reactor зіграло вирішальну роль у забезпеченні здатності програми ефективно обробляти великі обсяги одночасних запитів. Цей підхід не тільки покращив роботу користувача, забезпечивши швидшу та оперативнішу взаємодію, але й сприяв масштабованості програми та оптимізації ресурсів, завдяки чому вона краще відповідала вимогам глобальної платформи вивчення мови в реальному часі.

Проект використовує gRPC як основний протокол зв'язку між мікросервісами, що забезпечує ефективний, швидкий і безпечний обмін даними. GRPC було обрано через його здатність обробляти високопродуктивний зв'язок у розподілених системах, забезпечуючи безперебійну та надійну взаємодію різних мікросервісів у програмі. Це рішення було особливо важливим з огляду на мікросервісну архітектуру проекту, де такі служби, як Authority Service, Content Transferer Service і Routing Service, повинні часто обмінюватися інформацією, щоб підтримувати функціональність системи.

gRPC використовує буфери протоколів (Protobuf) як мову визначення інтерфейсу та формат серіалізації, що робить зв'язок легким та ефективним. Порівняно з традиційними API на основі REST, gRPC мінімізує накладні витрати, пов'язані з серіалізацією та десеріалізацією, що забезпечує швидшу передачу даних і меншу затримку. Це особливо корисно в сценаріях, коли сервіси обмінюються великими обсягами даних, наприклад потоковим вмістом, обробкою дозволів користувачів або обробкою взаємодії користувача з фільмами та книгами.

Двонаправлені потокові можливості gRPC також забезпечують значні переваги для проекту. Наприклад, взаємодія в реальному часі, така як призупинення фільмів і отримання перекладів для субтитрів або фраз за запитом, полегшується через потоки gRPC. Ці потоки дозволяють службам безперервно надсилати й отримувати дані, не чекаючи індивідуальних запитів і відповідей, створюючи плавну та чутливу роботу користувача [7].

Безпека є ще однією ключовою перевагою використання gRPC, оскільки він легко інтегрується з сучасними протоколами шифрування, такими як TLS (Transport Layer Security). Це гарантує, що весь зв'язок між мікросервісами зашифрований,

захищаючи конфіденційні дані користувача та підтримуючи відповідність стандартам безпеки. У цьому проекті такі служби, як Authentication Service і Authority Service, покладаються на захищений зв’язок gRPC для обміну обліковими даними користувачів і даними про дозволи, забезпечуючи цілісність даних і конфіденційність у всій системі.

Таким чином, використання gRPC для зв’язку між службами значно підвищило ефективність, продуктивність і безпеку програми. Його здатність підтримувати високошвидкісний обмін даними, двонаправлену потокову передачу та суворо типізовані контракти зробила його ідеальним вибором для архітектури мікросервісів проекту. Завдяки застосуванню gRPC проект створює надійну та масштабовану комунікаційну магістраль, що забезпечує безперебійну інтеграцію розподілених служб, забезпечуючи при цьому користувачам оперативну та надійну роботу [8].

2.4.2 Опис використаних баз даних

PostgreSQL було обрано як основну реляційну базу даних для тривалого зберігання, особливо для конфіденційних даних, таких як деталі облікового запису, керування повноваженнями та деталі медіаконтенту. Його надійність, розширені функції та продуктивність у обробці складних запитів і великих наборів даних зробили його ідеальним вибором для підтримки критичних потреб зберігання даних [9].

Для деталей облікового запису та керування повноваженнями PostgreSQL забезпечив надійне, безпечне й узгоджене середовище для зберігання даних, пов’язаних із користувачами. Підтримка базою даних транзакцій ACID (атомність, узгодженість, ізоляція, довговічність) забезпечила надійну обробку таких операцій, як реєстрація користувача, оновлення статусу облікового запису та керування ролями, без пошкодження даних [10]. Розробка схеми включала таблиці для

облікових записів користувачів, ролей, дозволів і терміну дії повноважень із зв'язками зовнішнього ключа, які гарантували цілісність даних. Використовуючи PostgreSQL, програма могла легко масштабуватися в міру додавання більшої кількості користувачів, а структура підтримувала гнучкі запити для ефективного отримання та зміни інформації облікового запису [11].

Розширені можливості PostgreSQL щодо індексування та запитів були використані для оптимізації керування повноваженнями користувачів. Оскільки системі потрібно було обробляти різні типи повноважень і дати закінчення терміну дії, складні запити можна було виконувати швидко, дозволяючи в режимі реального часу перевіряти активні повноваження для кожного користувача. Це гарантувало, що рівні доступу користувачів завжди були актуальними, а ролі та дозволи можна було динамічно змінювати відповідно до вимог програми. Використання полів JSONB у PostgreSQL додатково уможливило зберігання напівструктурованих даних, таких як користувацькі параметри користувача, уподобання або додаткові метадані, що зробило базу даних достатньо гнучкою, щоб виконувати нові вимоги без істотних змін у схемі.

Для медіаконтенту, включаючи відео, субтитри та пов'язані метадані, PostgreSQL використовувався для зберігання таких важливих деталей, як шляхи до файлів, формати, тривалість, параметри мови та описи. Ці записи були проіндексовані та запитувані, щоб забезпечити швидкий доступ до медіаконтенту під час взаємодії користувачів із програмою. Можливість зберігати двійкові дані, такі як посилання на мультимедійні файли або мініатюри зображень, у базі даних або як посилання на зовнішні системи зберігання файлів, була повністю використана для оптимізації керування вмістом. Підтримка PostgreSQL повнотекстового пошуку та індексування тексту також відіграла важливу роль у дозволі користувачам шукати описи вмісту та субтитри, що зробило пошук медіа-вмісту більш ефективним.

З точки зору масштабованості, PostgreSQL міг обробляти великі обсяги даних, що було вкрай важливо для зберігання інформації про медіаконтент, оскільки кількість відео, перекладів і пов'язаних даних з часом зростала. Крім того,

підтримка PostgreSQL для реплікації та високої доступності гарантувала, що база даних може масштабуватися горизонтально та залишатися стійкою у разі збоїв, гарантуючи, що користувачі завжди можуть отримати доступ до даних свого облікового запису та медіа-вмісту без перерв.

Завдяки використанню PostgreSQL програма отримала переваги від потужної системи керування реляційною базою даних, яка забезпечувала надійну узгодженість даних, гнучкий дизайн схеми та можливість масштабування із зростанням баз користувачів і медіа-вмісту. Це зробило його підходящим вибором для керування довгостроковим зберіганням даних, від облікових записів користувачів і повноважень до інформації, пов'язаної з медіа, забезпечуючи надійність, продуктивність і безпеку.

Elasticsearch було обрано як пошукову систему для програми для ефективної роботи з функціями пошуку вмісту, включаючи пошук медіа-контенту, субтитрів і пов'язаних метаданих. Його розподілений характер, можливості повнотекстового пошуку та індексування в реальному часі зробили його ідеальним вибором для швидкого отримання та фільтрації великих наборів даних, забезпечуючи швидкий і масштабований пошук для користувачів.

Потужні функції повнотекстового пошуку Elasticsearch були використані для індексування та пошуку описів вмісту, субтитрів, авторів і назв відео. Коли користувачі взаємодіяли з програмою, Elasticsearch дозволяв здійснювати швидкий, нечіткий і ранжований пошук, який міг знайти релевантний вміст, навіть якщо користувачі вводили ключові слова частково або неправильно. Здатність пошукової системи обробляти складні запити, включаючи фільтрацію за кількома параметрами, такими як мова, тип вмісту та теги, дозволила користувачам уточнювати пошук і легко знаходити саме те медіа, яке вони шукали [12].

Щоб забезпечити точність і релевантність результатів пошуку, аналізатори та токенізери Elasticsearch були використані для розбиття тексту на доступні для пошуку токени, дозволяючи розширений аналіз тексту, визначення основи та фільтрацію лексем. Ця функція гарантувала, що навіть якщо користувачі шукали синоніми або фрази в різних формах, результати пошуку були послідовними та

точними. Наприклад, пошук за запитом «вивчення мов» поверне вміст, пов’язаний як із «вивченням мови», так і з іншими відповідними термінами, покращуючи взаємодію з користувачем, надаючи ширші та всеохопніші результати.

Здатність Elasticsearch здійснювати швидке агрегування та сортування також була критично важливою для обробки складнішого пошуку вмісту, наприклад пошуку медіа за певними метаданими (наприклад, жанри, дати випуску, мови) і впорядкування результатів за релевантністю чи іншими факторами, як-от оцінки користувачів або кількість переглядів. Це гарантувало, що користувачі могли не лише знайти вміст, який вони шукали, але також могли швидко отримати доступ до найпопулярніших або найрелевантніших елементів відповідно до їхніх уподобань.

Іншою значною перевагою використання Elasticsearch була його масштабованість і розподілена архітектура. Зі збільшенням кількості користувачів, відео та вмісту Elasticsearch зміг обробляти зростаючі обсяги даних, розподіляючи навантаження індексування та пошуку між кількома вузлами. Це забезпечило горизонтальне масштабування системи, забезпечуючи послідовний пошук, навіть якщо обсяг вмісту в базі даних збільшувався. Підтримка Elasticsearch індексування в реальному часі також гарантувала, що новий вміст, субтитри або метадані, додані в систему, можуть бути проіндексовані та негайно доступні для пошуку.

Крім того, інтеграція Elasticsearch із базою даних PostgreSQL забезпечила ефективну індексацію метаданих вмісту, що зберігаються в реляційній базі даних, і можливість пошуку через Elasticsearch без необхідності резервного зберігання даних. Синхронізація даних між PostgreSQL і Elasticsearch здійснювалася за допомогою спеціалізованих процесів індексування, які підтримували пошуковий індекс в актуальному стані з мінімальною затримкою, гарантуючи, що новий або оновлений вміст завжди включатиметься в результати пошуку.

Підводячи підсумок, Elasticsearch використовувався для забезпечення швидкого, масштабованого та високорелевантного пошуку для програми. Розширені можливості повнотекстового пошуку, індексування в реальному часі та розподілена архітектура забезпечили ефективне виявлення вмісту, дозволяючи користувачам швидко й точно знаходити медіа-вміст і субтитри. Використання

Elasticsearch гарантувало, що в міру зростання платформи пошук вмісту залишатиметься продуктивним і здатним обробляти велику кількість запитів, що значно покращить загальний досвід роботи з додатком.

Redis інтегровано в програму для керування кешуванням авторитетів та інших даних, до яких часто звертаються, покращуючи продуктивність системи та зменшуючи навантаження на основну базу даних. Як високопродуктивне сховище даних у пам’яті Redis було обрано завдяки його здатності швидко зберігати та отримувати дані, забезпечуючи майже миттєвий доступ до важливої інформації, як-от повноваження користувача, деталі сеансу та інші дані, які часто запитуються.

Для кешування повноважень Redis відіграє ключову роль у зберіганні ролей і дозволів користувача, дозволяючи програмі швидко отримувати деталі повноважень, не запитуючи базу даних за кожним запитом. Оскільки програма мала справу з динамічним керуванням повноваженнями, було важливо, щоб система могла швидко перевірити поточний рівень доступу користувача та забезпечити ефективне застосування політик контролю доступу. Кешування цих деталей у Redis дозволило значно зменшити кількість звернень до бази даних, що призвело до швидшого часу відповіді та покращення пропускної здатності системи. Це було особливо корисно для операцій великого обсягу, таких як ті, що включають автентифікацію та авторизацію користувачів, де швидкість і оперативність були важливими.

Кешовані дані повноважень у Redis підтримувалися в актуальному стані завдяки реалізації ефективного механізму анулювання або оновлення кешу щоразу, коли змінювалися ролі або дозволи користувача. Це гарантувало, що кеш завжди відображає останній стан повноважень користувача, зберігаючи високу продуктивність. Використовуючи функцію Redis TTL (Time to Live), прострочені дані можна було автоматично видаляти, запобігаючи використанню застарілої інформації та забезпечуючи ефективну роботу системи навіть із зростанням бази користувачів.

Окрім кешування повноважень, Redis також використовувався для інших цілей кешування у всій програмі. Наприклад, метадані медіаконтенту, до яких часто

звертаються, такі як деталі відео, описи та параметри мови, зберігалися в Redis, щоб пришвидшити час пошуку та зменшити навантаження на PostgreSQL. Кешуючи такі дані, програма змогла обробляти більшу кількість запитів користувачів, забезпечуючи стабільну продуктивність навіть за інтенсивного використання. Це було особливо важливо для мультимедійних функцій, таких як пошук і відтворення вмісту, де затримки в отриманні даних можуть погіршити роботу користувача.

Природа Redis у пам’яті також дозволила програмі ефективно керувати сеансами, зберігаючи деталі сеансу, такі як статус входу користувача, маркери та налаштування. Це зменшило необхідність частого запиту до бази даних щодо даних, пов’язаних із сеансом, ще більше підвищивши загальну продуктивність і досвід користувача. Швидкий шаблон доступу Redis до даних за ключем і значенням добре підходив для зберігання невеликих фрагментів даних, до яких часто зверталися, які потрібно було отримати швидко та з мінімальними витратами.

Підсумовуючи, Redis був ключовим компонентом стратегії кешування програми, допомагаючи оптимізувати продуктивність, зберігаючи дані, до яких часто звертаються, наприклад повноваження користувачів, медіа-метадані та деталі сеансу. Його швидкі можливості зберігання даних у пам’яті значно покращили час відгуку та зменшили навантаження на базову базу даних, зробивши програму більш ефективною та масштабованою. Завдяки інтеграції Redis в архітектуру програма змогла надати користувачам оперативну та безперебійну роботу, навіть коли попит на доступ до даних у реальному часі зростав.

2.4.3 Технології для розгортання застосунку

Docker і Kubernetes були невід’ємною частиною стратегії розгортання програми, створюючи гнучку, масштабовану та ефективну інфраструктуру, яка могла б відповідати вимогам сучасного хмарного середовища. Docker

використовувався для контейнеризації програми, тоді як Kubernetes надав можливості оркестровки для керування, масштабування та забезпечення високої доступності служб програми.

Docker було обрано через його здатність пакувати всю програму, включаючи всі її залежності, у легкі портативні контейнери. Це забезпечило послідовну роботу програми в різних середовищах, будь то на локальній машині розробки, проміжному сервері чи робочому середовищі. Використання контейнерів Docker дозволило краще ізолювати кожну службу, зробивши програму більш модульною та зменшивши потенційні конфлікти між залежностями. За допомогою Docker кожен мікросервіс у додатку, незалежно від того, відповідає він за автентифікацію користувачів, керування вмістом або відтворення відео, можна було незалежно упаковувати, тестувати та розгортати, зберігаючи узгодженість у різних середовищах.

Завдяки контейнеризації програми за допомогою Docker процес розгортання значно спростився. Розробникам більше не потрібно було турбуватися про відмінності між середовищами розробки, тестування та виробництва. Контейнерну програму можна легко тиражувати та запускати в будь-якому середовищі, яке підтримує Docker, забезпечуючи стабільну роботу програми на всіх етапах свого життєвого циклу. Крім того, легкий характер Docker дозволив пришвидшити час створення та швидші цикли ітерацій, що допомогло пришвидшити процеси розробки та розгортання.

Потім Kubernetes використовувався для оркестрування та керування масштабним розгортанням контейнерів Docker. Оскільки програма прийняла архітектуру мікросервісів, потреба в оркестрові, який міг би керувати життєвим циклом кількох контейнерів, стала важливою. Kubernetes надав інструменти, необхідні для керування розгортанням, масштабуванням і моніторингом цих контейнерів у розподіленому середовищі. За допомогою Kubernetes мікросервіси програми було розгорнуто таким чином, щоб забезпечити автоматичне масштабування, балансування навантаження та самовідновлення.

За допомогою Kubernetes служби можна було автоматично масштабувати відповідно до попиту, забезпечуючи ефективну роботу програми зі змінними навантаженнями. Kubernetes використовував горизонтальне автомасштабування модулів, щоб динамічно регулювати кількість запущених екземплярів кожного мікросервісу залежно від використання ресурсів або вхідного трафіку. Ця можливість була особливо корисною в ситуаціях, коли траплялися стрибки трафіку, гарантуючи, що програма залишалася чуйною навіть під великим навантаженням.

Kubernetes також забезпечив високу доступність, розподіливши контейнери між кількома вузлами в кластері. У разі збою або вичерпання ресурсів на одному вузлі Kubernetes автоматично перепланував уражені контейнери на інших доступних вузлах, гарантуючи, що програма залишатиметься онлайн без будь-яких простоїв. Ця відмовостійкість була життєво важливою для того, щоб служби програми, такі як відтворення відео в реальному часі або переклад субтитрів, були завжди доступні для користувачів.

Крім того, Kubernetes керував виявленням служб і балансуванням навантаження через Kubernetes Services, що забезпечувало безперебійний зв’язок між мікросервісами програми. Ця сервісно-орієнтована архітектура дозволяла іншим службам у системі отримувати доступ до кожного мікросервісу без необхідності знати його конкретне місцезнаходження чи IP-адресу, оскільки Kubernetes обробляв маршрутизацію автоматично. Це мало вирішальне значення для програми, де різні компоненти, як-от керування вмістом, обробка мультимедійних даних і автентифікація користувачів, були слабко пов’язані між собою, але повинні були безперебійно взаємодіяти [13].

Завдяки Kubernetes моніторинг і ведення журналів також були централізовані, що полегшило відстеження справності окремих служб і системи в цілому. Використання Kubernetes Dashboard і інструментів сторонніх розробників, таких як Prometheus і Grafana, дозволило команді спостерігати за системними показниками в режимі реального часу та приймати обґрунтовані рішення щодо масштабування або оптимізації використання ресурсів.

Підсумовуючи, Docker і Kubernetes були необхідними для розгортання та керування архітектурою мікросервісів програми. Docker забезпечив послідовний, портативний та ефективний спосіб контейнеризації програми, а Kubernetes забезпечив ефективне управління та масштабування контейнерів. Разом ці технології дозволили розгорнути програму в хмарному середовищі, забезпечуючи масштабованість, надійність і високу доступність, одночасно дозволяючи спрощувати процеси розробки та розгортання. Такий підхід гарантував, що додаток зможе задовольнити зростаючі вимоги користувачів, залишаючись при цьому гнучким і стійким до динамічного середовища.

Terraform використовувався як частина стратегії управління інфраструктурою для автоматизації розгортання та налаштування хмарних ресурсів програми. Будучи потужним інструментом «Інфраструктура як код» (IaC), Terraform дозволив визначити інфраструктуру в коді, дозволивши керувати нею, версії та відтворювати її узгоджено та передбачувано. Використовуючи Terraform, всю хмарну інфраструктуру, включаючи обчислювальні ресурси, мережу, сховище та керовані служби, можна надавати, оновлювати та підтримувати за допомогою автоматизованих сценаріїв, зменшуючи складність і потенційні людські помилки в ручних конфігураціях.

Використання Terraform дозволило створювати шаблони інфраструктури, які можна повторно використовувати в різних середовищах, будь то для розробки, постановки чи виробництва. Ці шаблони забезпечували узгодженість основної інфраструктури в кожному середовищі, усуваючи розбіжності та гарантуючи, що мікросервіси програми працюватимуть на інфраструктурі того самого типу, незалежно від того, де вони були розгорнуті. Ця послідовність була вирішальною для підтримки безперебійного робочого процесу від розробки до виробництва.

Декларативний підхід Terraform до управління інфраструктурою означав, що бажаний стан інфраструктури програми можна було описати у файлах конфігурації. Ці файли зберігалися в системі керування версіями, що дозволяло команді відстежувати зміни з часом, повертатися до попередніх версій, якщо потрібно, і співпрацювати над модифікаціями інфраструктури, як і з кодом програми. Коли

потрібні були зміни в інфраструктурі, як-от масштабування кількості вузлів у кластері Kubernetes або додавання нових ресурсів, як-от баз даних чи сегментів зберігання, Terraform автоматично обчислював необхідні зміни та застосовував їх, забезпечуючи, щоб інфраструктура завжди була синхронізована з указаною конфігурація.

У контексті програми Terraform був особливо корисним для надання та керування хмарними ресурсами, необхідними для архітектури мікросервісів. Наприклад, кластер Kubernetes, який працює в хмарному провайдері, можна визначити та керувати ним за допомогою сценаріїв Terraform разом з іншими важливими ресурсами. Розглядаючи інфраструктуру як код, процес розгортання став більш автоматизованим, а інфраструктура надавалась і налаштовувалася повторюваним способом у різних середовищах.

Terraform також використовувався для керування зовнішніми службами, такими як хмарні бази даних, сховище об’єктів і служби моніторингу. Наприклад, бази даних PostgreSQL і кластери Elasticsearch були підготовлені за допомогою Terraform, а ресурси Kubernetes були налаштовані для безперебійного обміну даними з цими службами. Це гарантувало, що інфраструктура була не тільки автоматизованою, але й надійною, дозволяючи швидко масштабувати та керувати критично важливими ресурсами в міру зростання програми.

Крім того, Terraform сприяла впровадженню найкращих практик у сфері управління інфраструктурою, як-от управління версіями, документація та безпека. Зберігаючи файли конфігурації інфраструктури в репозиторії з контролем версій, команда могла відстежувати та переглядати зміни з часом, забезпечуючи належне документування оновлень інфраструктури та можливість перевірки. Крім того, інтеграція Terraform із системами автентифікації та авторизації хмарних провайдерів забезпечила безпечне надання конфіденційних ресурсів із належним контролем доступу.

Таким чином, Terraform відіграв важливу роль в управлінні та автоматизації розгортання хмарних ресурсів для програми. Використовуючи Terraform, інфраструктура може бути визначена як код, що забезпечує послідовне,

повторюване та масштабоване розгортання. Можливість надавати та керувати всією інфраструктурою автоматично та з контролем версій значно оптимізувала процес розгортання, зменшила ручне втручання та гарантувала, що інфраструктура завжди буде відповідати бажаному стану. Це не тільки спростило керування хмарними ресурсами, але й сприяло загальній масштабованості, безпеці та ефективності програми.

AWS було обрано хмарним провайдером для розміщення додатка завдяки широкому набору послуг, масштабованості, надійності та гнучкості. Кілька служб AWS було використано для забезпечення безперебійної роботи додатка, високої доступності та безперебійного масштабування, оскільки попит користувачів зростав. Ключові служби AWS, такі як Elastic Kubernetes Service (EKS), RDS, API Gateway і S3, були інтегровані в архітектуру для забезпечення надійного керування інфраструктурою, зберігання даних і безперебійного зв’язку між різними компонентами.

Служба Elastic Kubernetes (EKS) була використана для керування та масштабування кластерів Kubernetes, які запускали мікросервіси програми. EKS надала повністю керовану службу Kubernetes, яка дозволяла автоматизоване розгортання, масштабування та керування контейнерними програмами. Переклавши керування площиною керування Kubernetes на AWS, команда змогла зосередитися на розробці та розгортанні програми, не турбуючись про складнощі підтримки інфраструктури Kubernetes. EKS гарантує, що мікросервіси можуть автоматично масштабуватися у відповідь на зміни трафіку, забезпечуючи стійке та високодоступне середовище для ефективної роботи програми.

Для тривалого зберігання та керування даними Amazon RDS (Relational Database Service) використовувався для обробки бази даних PostgreSQL, яка зберігала важливі дані, такі як деталі облікового запису користувача, інформація про повноваження та метадані медіаконтенту. RDS запропонував автоматичне резервне копіювання, програмне забезпечення та можливість легко масштабувати екземпляр бази даних для задоволення зростаючих потреб у даних. Завдяки використанню RDS програма отримала переваги високодоступної та керованої

служби реляційної бази даних без накладних витрат на підтримку інфраструктури та операційних завдань, пов’язаних із традиційним керуванням базами даних. Повна інтеграція RDS з іншими службами AWS, такими як EC2 і S3, також спростила створення загальної архітектури програми та керування нею [14].

Amazon S3 (Simple Storage Service) використовувався для зберігання великих медіафайлів, зокрема відеовмісту, субтитрів та інших ресурсів. S3 надав масштабоване, довговічне та економічно ефективне рішення для зберігання статичного вмісту програми. За допомогою S3 відео можна безпечно зберігати та обслуговувати з низькою затримкою, забезпечуючи плавний і безперервний перегляд для користувачів. Сервіс також інтегрований із системою керування вмістом програми, дозволяючи менеджерам вмісту легко завантажувати та керувати медіа-активами. Крім того, S3 пропонує вбудоване керування версіями та довговічність даних, забезпечуючи безпечне зберігання вмісту та доступність у разі потреби.

Щоб полегшити безперебійний зв’язок між клієнтською програмою та серверними мікросервісами, Amazon API Gateway використовувався для створення та керування RESTful API. Шлюз API діяв як центральна точка входу для всіх запитів клієнтів, надаючи такі функції, як керування трафіком, авторизація та контроль доступу. Ця служба дозволила створювати безпечні та масштабовані API, забезпечуючи ефективну взаємодію клієнтських програм із серверними службами. Завдяки здатності API Gateway до автоматичного масштабування програма змогла обробляти різні рівні трафіку та забезпечувати адаптивний рівень API для підтримки інтерфейсу.

Крім того, AWS забезпечила критично важливу інфраструктуру для керування безпекою та контролем доступу. Такі служби, як AWS IAM (Identity and Access Management), використовувалися для керування дозволами користувачів, забезпечуючи доступ до конфіденційних ресурсів лише авторизованим користувачам і службам.

Для моніторингу та оперативного керування набір інструментів AWS, як-от CloudWatch і X-Ray, був інтегрований для моніторингу справності та

продуктивності програми. CloudWatch дозволив команді відстежувати показники, журналювати дані та налаштовувати сигнали тривоги для використання ресурсів і продуктивності додатків. Це допомогло завчасно виявити потенційні проблеми та забезпечити ефективну роботу системи.

Підсумовуючи, AWS був невід'ємною частиною інфраструктури програми, надаючи надійну та масштабовану хмарну платформу, яка підтримувала розгортання, керування та масштабування мікросервісів. Завдяки таким службам, як Elastic Kubernetes Service, RDS, S3, API Gateway, а також різноманітним функціям моніторингу та безпеки, AWS дозволила команді розробників створити надійну, безпечну та високодоступну архітектуру додатків. Бездоганна інтеграція між цими службами гарантувала, що програма може обробляти великі обсяги даних, забезпечувати користувальницький досвід високої якості та ефективно масштабуватися відповідно до зростання попиту, зменшуючи операційну складність керування інфраструктурою.

РОЗДІЛ 3 РОЗРОБКА ЗАСТОСУНКУ

3.1 Аналіз завдань для розробки проєкту

Підхід, заснований на Scrum, був прийнятий для керівництва процесом збору та уточнення вимог під час розробки програми. Ця гнучка методологія наголошує на повторюваних циклах зворотного зв'язку, постійному залученні зацікавлених сторін і швидкій адаптації до мінливих потреб. Вимоги були зібрані, проаналізовані та визначені пріоритети під час регулярних заходів, забезпечуючи чітку зосередженість команди розробників на поступовому забезпеченні цінності та першочерговому задоволенні найнагальніших потреб користувачів.

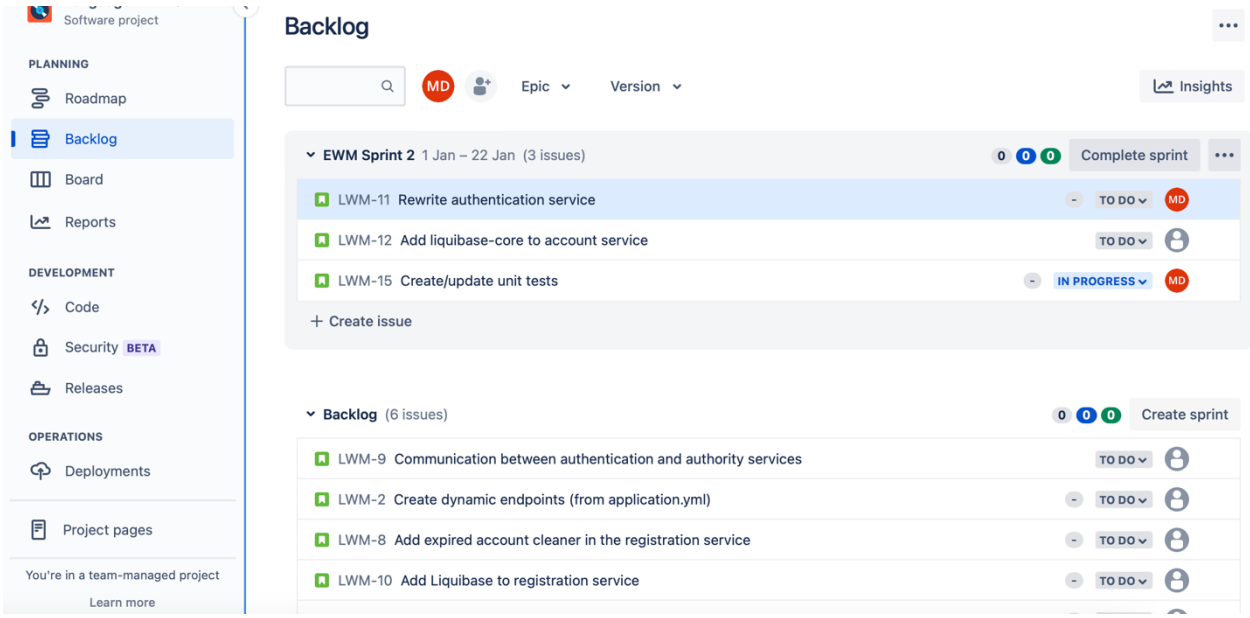

Рисунок 3.1 – Набір визначених завдань

На початку проекту було створено початковий план продукту шляхом збору вимог – зображений на рисунку 3.1. У міру розробки вимоги уточнювалися під час кожного сеансу планування спринту. Власник продукту відіграв важливу роль у забезпеченні ясності, визначенні пріоритетів і забезпеченні відповідності резерву продукту загальному баченню. Команда розробників тісно співпрацювала з власником продукту, щоб розбити великі або складні вимоги на менші, більш керовані історії користувачів. Під час цього процесу було встановлено критерії

прийнятності, щоб гарантувати, що кожна вимога добре зрозуміла, перевірена та відповідає визначенню «Виконано». Поступово уточнюючи відставання, команда переконалася, що лише найцінніші, здійсненні та добре зрозумілі вимоги були обрані для впровадження в майбутніх спринтах.

Регулярні цикли зворотного зв'язку були критично важливими для підтримки оперативного підходу до збору вимог. Наприкінці кожного спринту проводився сеанс перегляду, щоб продемонструвати зацікавленим сторонам робочі прирости продукту. Ці збільшення дозволили зацікавленим сторонам надати прямий відгук, який потім використовувався для коригування пріоритетів, уточнення вимог або додавання нових історій користувачів до резерву. Цей безперервний цикл гарантував, що продукт відображає реальні потреби та очікування користувачів, а не керується статичними специфікаціями.

Щоденні робочі зустрічі ще більше підтримали постійне уточнення вимог. Хоча ці зустрічі були зосереджені на найближчому прогресі команди, блокувальниках і запланованих діях, вони забезпечили команду неформальною обстановкою, щоб висвітлити будь-які запитання чи невизначеності щодо вимог. Якщо команда визначила неоднозначні або суперечливі потреби, ці проблеми можна було негайно вирішити під час подальших розмов або передати власнику продукту для вирішення.

Процес збору вимог залишався динамічним, гнучким і орієнтованим на користувача завдяки використанню підходу на основі Scrum. Замість того, щоб покладатися на вичерпні попередні специфікації, команда прийняла ітераційне навчання та постійну співпрацю із зацікавленими сторонами. Це дозволило функціям програми розвиватися органічно, забезпечуючи, щоб кінцевий продукт відповідав потребам користувачів у вивченні мови та забезпечував постійну цінність протягом життєвого циклу розробки.

3.2 Огляд мікросервісної архітектури

Оскільки для розробки проєкту використовується мікросервісна архітектура то, відповідно, наша програма складається із набору незалежних сервісів, основні з яких зображено на рисунку 3.2.

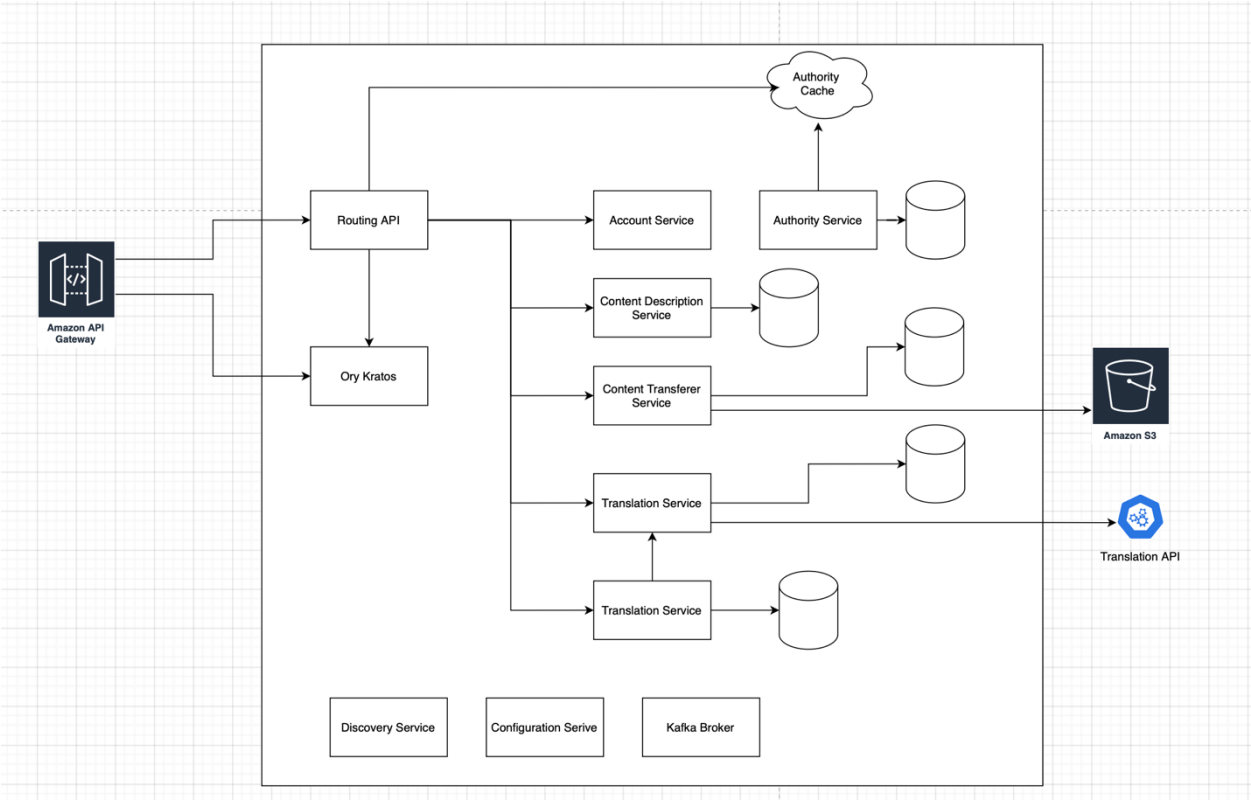

Рисунок 3.2 – Зображення кластера

На рисунку 3.2 можна ознайомитись з архітектурою мікросервісів та зв’язків між ними. Система розроблена для роботи з високими навантаженнями, підтримки безперебійної взаємодії з користувачем і динамічної адаптації до змін попиту. Кожна мікрослужба контейнеризована та організована Kubernetes, одночасно використовуючи служби AWS для додаткових функцій та оптимізації.

Точкою входу програми керує Amazon API Gateway, який безпечно обробляє клієнтські запити та направляє їх до відповідних внутрішніх служб. Шлюз забезпечує безпечний доступ, надаючи такі функції, як регулювання, обмеження

швидкості та перевірка запитів. Крім того, було представлено Amazon Simple Email Service (SES) для обробки сповіщень електронною поштою, таких як підтвердження реєстрації користувачів, скидання пароля та оновлення транзакцій, що ще більше покращує роботу користувача.

Запити від шлюзу API спрямовуються до API маршрутизації, який виконує роль центрального оркестратора системи, делегуючи завдання відповідним серверним службам. Компонент Ory Kratos керує робочими процесами автентифікації та ідентифікації користувачів, забезпечуючи безпечний контроль доступу та надійне керування сеансами. Він тісно інтегрований з API маршрутизації, що дозволяє ефективно перевіряти користувачів і керувати ідентифікацією, зберігаючи відповідність протоколам безпеки.

Життєвий цикл облікового запису користувача обробляється службою облікових записів, яка керує профілями користувачів, параметрами та деталями реєстрації. Управління ролями та дозволами передається на службу Authority Service, яка інтегрується з високошвидкісним кеш-пам’яттю Redis для зберігання та перевірки дозволів користувачів. Такий підхід забезпечує низьку затримку під час перевірок авторизації, зменшуючи навантаження на базову базу даних і підтримуючи високопродуктивний контроль доступу.

Функціональність керування контентом розподілена між спеціалізованими мікросервісами. Служба опису вмісту зберігає та керує метаданими для медіа, включаючи заголовки, описи та пов’язані теги, у спеціальній базі даних. Ці метадані необхідні для організації, категоризації та ефективного отримання медіафайлів. Служба Content Transferer полегшує зберігання та доставку великих медіафайлів, використовуючи Amazon S3 для масштабованого, економічного та безпечного зберігання. Це гарантує, що файли відео та субтитрів надійно зберігаються та доступні з мінімальною затримкою, підтримуючи плавну та чутливу роботу користувача.

Служба перекладу займається обробкою та перекладом субтитрів та іншого текстового вмісту. Завдяки інтеграції із зовнішнім API перекладу він забезпечує точні переклади з урахуванням контексту для доставки багатомовного вмісту.

Кілька екземплярів Translation Service працюють паралельно, кожен із яких підтримується спеціальною системою зберігання для тимчасової обробки даних. Ця розподілена архітектура забезпечує ефективне виконання робочих навантажень перекладу навіть під час пікового використання.

Основні компоненти інфраструктури, зокрема служба виявлення, служба конфігурації та брокер Kafka, ще більше підвищують надійність архітектури. Служба Discovery полегшує динамічну реєстрацію служби та зв'язок, забезпечуючи безперебійну взаємодію між мікросервісами. Служба конфігурації централізує керування налаштуваннями додатків, забезпечуючи узгодженість серед середовищ. Kafka використовується для зв'язку, керованого подіями, уможливлюючи асинхронну обробку таких завдань, як транскодування медіафайлів, сповіщення та фонові робочі процеси.

У цю оновлену архітектуру Amazon SES інтегровано як важливе вдосконалення для спілкування користувачів на основі електронної пошти, що дозволяє програмі надсилати безпечні та надійні електронні листи для перевірки облікового запису та оновлень. Крім того, Kubernetes керує загальною оркестровкою та масштабованістю, гарантуючи, що кожен мікросервіс може динамічно масштабуватися відповідно до потреб користувачів. Поєднання таких служб Kubernetes і AWS, як API Gateway, SES, S3 і Redis, забезпечує надійну хмарну платформу, здатну задовольняти постійні потреби користувачів, зберігаючи високу доступність, продуктивність і безпеку.

3.1.1 Детальний огляд компоненту Ory Kratos

Ory Kratos є основною службою керування ідентифікацією та автентифікацією в додатку, забезпечуючи безпечну та ефективну обробку реєстрації користувачів, автентифікації та робочих процесів відновлення

облікового запису. З точки зору програми, Ory Kratos глибоко інтегровано з іншими службами, щоб підтримувати безперебійну взаємодію з користувачем, а також сприяє модульності та безпеці загальної системи. Він діє як централізована служба для керування обліковими даними користувачів, перевірки особи та організації багатоетапних процесів автентифікації.

Під час процесу реєстрації користувача Ory Kratos не лише збирає та перевіряє інформацію про користувача, але й запускає важливі подальші події, щоб забезпечити плавну інтеграцію даних користувача в екосистему. Після успішної реєстрації користувача Орі Кратос генерує подію та публікує її в Kafka, повідомляючи службу ролей для початку створення базових ролей користувача. Цей керований подіями підхід гарантує, що кожному новому користувачу належним чином налаштовано ролі та дозволи за замовчуванням у системі. Завдяки відокремленню процесу ініціалізації авторизації від логіки реєстрації система залишається модульною та масштабованою, що дозволяє службі авторизації обробляти ці події незалежно та асинхронно. Приклад етапів реєстрації користувача зображено на рисунку 3.3.

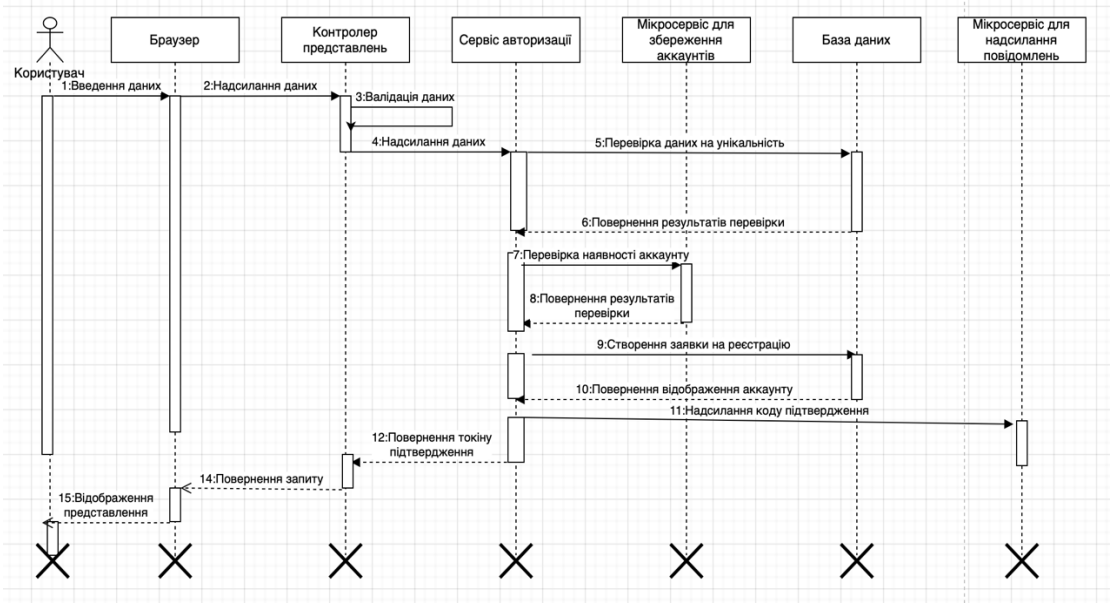

Рисунок 3.3 – Діаграма послідовності для реєстрації

Діаграма послідовності – яка зображена на рисунку 3.3 – описує потік реєстрації користувача в програмі, демонструючи взаємодію між браузером, контролером презентацій, службою автентифікації, мікрослужбою керування обліковим записом, базою даних і мікрослужбою сповіщень.

Процес починається з того, що користувач вводить свої реєстраційні дані в браузері. Ці дані, які включають такі обов'язкові поля, як ім'я користувача, адреса електронної пошти чи пароль, надсилаються до контролера презентації. Контролер перевіряє отримані дані, гарантуючи, що вони відповідають заздалегідь визначеним критеріям, таким як правильне форматування та повнота. Після перевірки контролер пересилає дані до служби автентифікації для подальшої обробки.

Служба автентифікації ініціює перевірку унікальності, надсилаючи запит до мікросервісу керування обліковим записом. Цей крок гарантує, що надані реєстраційні дані, такі як електронна адреса чи ім'я користувача, ще не пов'язані з існуючим обліковим записом. Мікросервіс запитує базу даних, щоб перевірити унікальність даних, і повертає результати до служби автентифікації.

Якщо дані проходять перевірку унікальності, служба автентифікації перевіряє, чи обліковий запис із подібними атрибутами очікує на розгляд або активний. Це гарантує уникнення повторних або конфліктних реєстрацій. Після підтвердження служба автентифікації переходить до створення нового запиту на реєстрацію шляхом взаємодії з мікросервісом керування обліковим записом. Мікросервіс оновлює базу даних деталями щойно зареєстрованого облікового запису, позначаючи його як очікуваний до завершення підтвердження.

Потім служба автентифікації повертає маркер підтвердження реєстрації та інформацію про обліковий запис до контролера презентації. У цей момент контролер презентації надсилає дані до мікросервісу сповіщень, який створює та надсилає повідомлення підтвердження, наприклад електронний лист або SMS, що містить код підтвердження або посилання. Цей крок гарантує, що користувачеві буде запропоновано підтвердити право власності на обліковий запис перед завершенням процесу реєстрації.

Після отримання підтвердження від мікросервісу сповіщень інтерфейс користувача оновлюється в браузері, щоб відобразити статус створення облікового запису. Маркер підтвердження повертається в браузер, безпечно зберігається та пов’язується з щойно зареєстрованим сеансом. Нарешті, браузер оновлює інтерфейс, вказуючи на успішне надсилання та очікуючи підтвердження користувача за допомогою коду підтвердження.

Цей потік гарантує, що всі аспекти процесу реєстрації, включаючи перевірку даних, перевірку унікальності, створення облікового запису та сповіщення про підтвердження, обробляються без проблем. Інтеграція кількох служб забезпечує безпечний, ефективний і зручний процес реєстрації, мінімізуючи помилки та гарантуючи, що лише перевірені користувачі отримають доступ до системи.

Орі Кратос також відповідає за виконання критично важливого завдання щодо надсилання електронних листів із підтвердженням реєстрації та кодів підтвердження номера телефону. Використовуючи Amazon Simple Email Service (SES), Ory Kratos гарантує надійність і масштабованість електронного зв’язку. Цей механізм не лише захищає процес створення облікового запису, але й забезпечує відповідність найкращим практикам перевірки користувачів, захищаючи платформу від підроблених або неавторизованих облікових записів.

Після автентифікації користувача Ory Kratos відіграє ще одну ключову роль у забезпеченні актуальності дозволів і ролей користувача в програмі. Після успішного входу Ory Kratos надсилає другу подію до Kafka, повідомляючи службу Authority Service про завантаження повноважень автентифікованого користувача в кеш Redis. Цей крок гарантує, що найновіші дозволи для користувача є легкодоступними, мінімізуючи запити до бази даних і зменшуючи затримку під час перевірок контролю доступу. Зберігаючи дані авторизації у високошвидкісному кеші Redis, програма досягає балансу між безпекою та продуктивністю, що дозволяє швидко приймати рішення щодо авторизації без шкоди для цілісності базових даних.

Інтеграція Ory Kratos із Kafka також забезпечує надійну та керовану подіями архітектуру, яка може ефективно обробляти складні робочі процеси. Наприклад,

коли користувач оновлює свій профіль або скидає пароль, Ory Kratos може публікувати відповідні події в Kafka, дозволяючи іншим службам реагувати відповідним чином. Цей дизайн гарантує, що всі зміни в облікових записах користувачів послідовно поширюються в системі. На рисунку 3.4 зображено діаграму послідовності для авторизації.

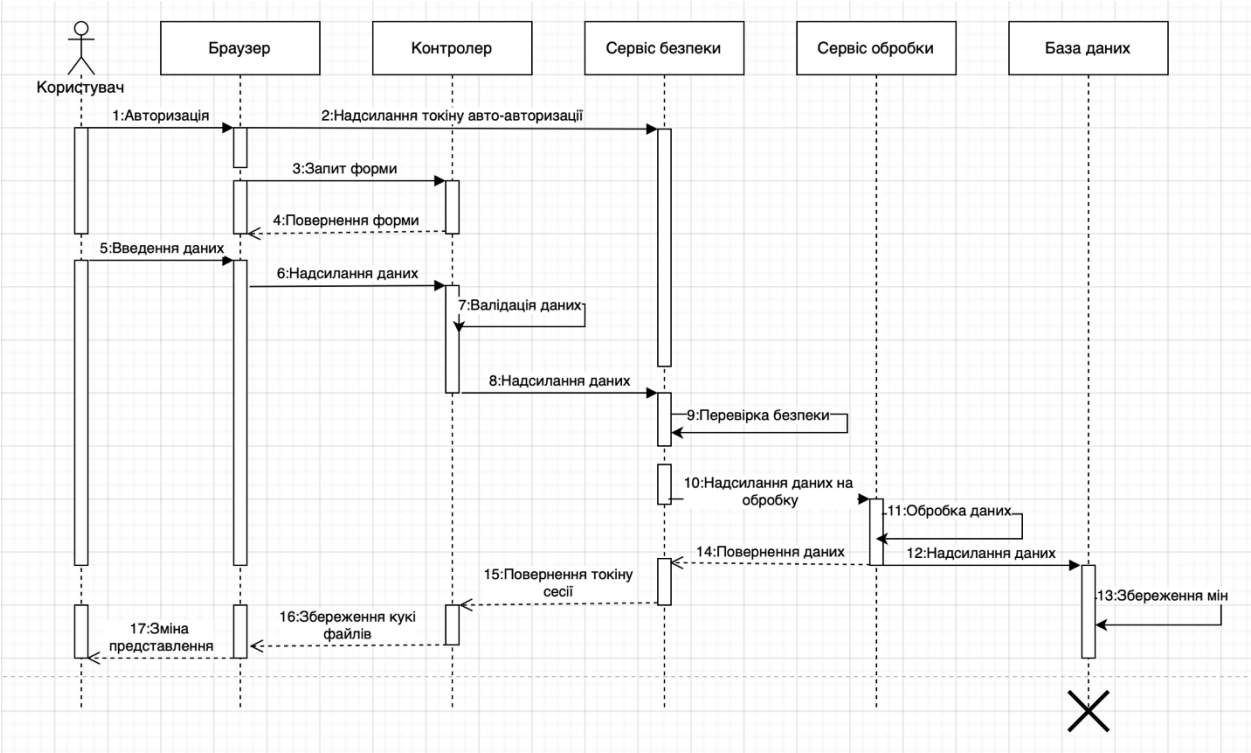

Рисунок 3.4 – Діаграма послідовності для авторизації

Діаграма – зобрадена на рисунку 3.4 – ілюструє потік реєстрації користувача, докладно описуючи, як дані переміщуються через різні компоненти програми, включаючи контролер презентацій, службу підтвердження, службу безпеки та базу даних. Процес починається з того, що користувач ініціює запит на реєстрацію через свій браузер. Браузер спілкується з контролером презентацій, який запитує реєстраційну форму на сервері. Форма повертається до браузера, що дозволяє користувачеві ввести необхідні реєстраційні дані, такі як електронна адреса, номер телефону чи інші облікові дані.

Коли користувач надсилає реєстраційні дані, вони надсилаються до контролера презентацій для перевірки. Контролер гарантує, що дані відповідають

заздалегідь визначеним критеріям, таким як формат і повнота, перш ніж передавати їх до служби підтвердження. Служба підтвердження відповідає за створення та зберігання коду підтвердження, який надсилається користувачеві електронною поштою або SMS для підтвердження. Потім користувач повинен ввести код підтвердження, який повертається до служби підтвердження для перевірки. Сервіс перевіряє код на збережене значення, щоб забезпечити точність і автентичність.

Після успішної перевірки коду підтвердження дані передаються на обробку в службу безпеки. Цей крок може передбачати шифрування конфіденційної інформації, наприклад паролів, для забезпечення безпечного зберігання. Потім оброблені дані надсилаються в базу даних для збереження, забезпечуючи надійне зберігання інформації про обліковий запис користувача та належний зв’язок із пов’язаними деталями перевірки.

Після збереження даних служба безпеки генерує маркер сесії для щойно зареєстрованого користувача. Цей маркер повертається до контролера презентації, який потім зв’язується з браузером, щоб зберегти маркер сеансу як файл cookie або в локальному сховищі для майбутньої автентифікації. Нарешті, браузер оновлює інтерфейс користувача, щоб відобразити успішну реєстрацію, дозволяючи користувачеві продовжити роботу з програмою.

Цей потік підкреслює інтеграцію між контролером презентації, службою підтвердження та службою безпеки, забезпечуючи безпечну, ефективну та зручну реєстрацію. Використання кодів підтвердження додає додатковий рівень безпеки, а етапи обробки та шифрування захищають конфіденційні дані користувача. Останні кроки гарантують, що користувач має автентифікований сеанс, що забезпечує безперебійний доступ до програми.

Крім того, Ory Kratos підтримує розширені функції автентифікації, такі як багатофакторна автентифікація (MFA) і керування сеансами, що ще більше підвищує безпеку платформи. З точки зору додатків, його розширюваність і гнучкість дозволяють легко додавати нові механізми автентифікації або інтегрувати із зовнішніми постачальниками ідентифікаційної інформації, забезпечуючи адаптацію системи до мінливих вимог безпеки.

Підсумовуючи, Ory Kratos є наріжним каменем системи керування ідентифікацією додатка, що забезпечує безпечні, масштабовані та розширювані рішення для реєстрації користувачів, автентифікації та керування обліковими записами. Завдяки інтеграції з Kafka та такими службами, як Authority Service і Amazon SES, він забезпечує безперебійну координацію всієї платформи, забезпечуючи ефективне підключення користувачів, ініціалізацію повноважень і контроль доступу на основі ролей. Його здатність керувати робочими процесами автентифікації та поширювати критичні події по всій системі підкреслює його важливість у підтримці цілісності та швидкості реагування програми.

3.1.2 Огляд компоненту Routing API

Служба маршрутизації виконує роль центрального координатора в архітектурі мікросервісів програми, керуючи потоком запитів між клієнтами та внутрішніми службами. Він відіграє ключову роль у забезпеченні безперебійного зв’язку між різними компонентами, реалізуючи логіку для ефективного делегування завдань. Діючи як рівень проміжного програмного забезпечення, Routing Service відокремлює клієнтські запити від серверних мікросервісів, забезпечуючи модульність і масштабованість. Основну частину конфігурації Routing API можна знайти у лістингу 3.1 додатку А.

Служба маршрутизації зв’язується безпосередньо з Орі Кратосом для обробки автентифікації користувачів і керування сеансами. Коли користувач робить запит, служба маршрутизації перенаправляє виклики, пов’язані з автентифікацією, Орі Кратосу для перевірки облікових даних користувача та підтвердження сеансу. Після успішної автентифікації Ory Kratos надсилає подію Kafka, повідомляючи службу Authority Service про завантаження даних авторизації автентифікованого користувача в Authority Cache. Потім служба маршрутизації отримує інформацію про повноваження з кешу, щоб забезпечити контроль доступу на основі ролей,

гарантуючи, що користувач має необхідні дозволи для доступу до певних ресурсів або виконання певних дій. Ця інтеграція забезпечує швидку та ефективну перевірку авторизації, мінімізуючи затримку, уникаючи частих запитів до бази даних.

Вона також організовує взаємодію з іншими ключовими компонентами системи. Наприклад, він пересилає запити до служби облікових записів для оновлення профілю користувача або отримання інформації, пов'язаної з обліковим записом. Під час обробки запитів, пов'язаних із вмістом, служба маршрутизації взаємодіє зі службою опису вмісту, щоб отримати метадані, і службою передачі вмісту, щоб керувати зберіганням і отриманням медіафайлів, що зберігаються в Amazon S3. Цією взаємодією керують чітко визначені API, що забезпечує чіткий розподіл обов'язків між службами.

Для операцій, пов'язаних із перекладом, служба маршрутизації делегує запити службі перекладу, яка взаємодіє із зовнішніми API перекладу для обробки субтитрів або іншого текстового вмісту. Служба маршрутизації гарантує, що відповіді від служби перекладу безперебійно направляються назад до клієнта, зберігаючи загальний потік взаємодії користувачів.

Крім того, служба маршрутизації використовує службу виявлення для динамічного визначення місцезнаходження та зв'язку з іншими мікросервісами в кластері Kubernetes. Ця функція гарантує, що система залишається стійкою та адаптованою до змін, таких як масштабування або перерозподіл окремих служб. Параметри конфігурації для всіх взаємодій централізовано керуються через службу конфігурації, що дозволяє службі маршрутизації працювати узгоджено в різних середовищах.

Підсумовуючи, служба маршрутизації виступає основою комунікаційного рівня програми, забезпечуючи ефективну обробку запитів, безпечну автентифікацію та бездоганну інтеграцію між мікросервісами. Завдяки координації з Ory Kratos, Authority Cache та іншими серверними службами він забезпечує надійний і масштабований механізм для керування взаємодією користувачів і підтримки цілісності системи. Ця конструкція не тільки оптимізує продуктивність, але й підтримує модульний і керований подіями характер загальної архітектури.

3.1.3 Огляд компоненту Authority Service

Служба авторизації є критично важливим компонентом у програмі, відповідальним за керування та застосування прав користувачів, ролей і дозволів. Його основна роль полягає в підтримці централізованого сховища всіх повноважень облікових записів, які безпечно зберігаються в базі даних PostgreSQL, одночасно забезпечуючи ефективний і динамічний доступ до цих повноважень для активних користувачів через механізми кешування. Цей дворівневий підхід врівноважує потребу в надійному сховищі даних із високопродуктивним доступом. На рисунку 3.5 зображено діаграму класів для Authority Service.

Рисунок 3.5 – Діаграма класів для Authority Service

Ця діаграма класів, що зображено на рисунку 3.5, описує структуру та взаємодію компонентів у службі авторизації, демонструючи її модульну та багаторівневу структуру для керування ролями користувачів, дозволами та колекціями повноважень. Архітектура базується на різних службах і сховищах, кожен з яких має чіткий розподіл завдань, що забезпечує масштабованість і

зручність обслуговування. SQL запит, що створює відповідні таблиці в базі даних можна зустріти у лістингу 3.2 додатку А.

В основі лежить AuthorityService, який служить основною точкою входу для керування окремими повноваженнями. Він покладається на AuthorityRepository – який зображений в лістингу 3.3 додатку А – для збереження даних і взаємодіє з AuthorityMapper для обробки перетворень між об’єктами даних і DTO. Ця служба відповідає за отримання, додавання або зміну окремих повноважень у системі.

Доповнює AuthorityService BlockerService, який зосереджується на управлінні заблокованими повноваженнями. Ця служба взаємодіє з BlockerRepository для збереження та використовує BlockerMapper для операцій зіставлення. Це гарантує, що певні повноваження або колекції можуть бути тимчасово або постійно обмежені для користувачів або облікових записів на основі конкретних бізнес-правил.

AccountAuthorityRelationService керує зв’язками між обліковими записами користувачів і призначеними їм повноваженнями або колекціями. Ця служба використовує AccountAuthorityRepository для збереження цих зв’язків і надає методи призначення або скасування повноважень для певних користувачів. Він відіграє вирішальну роль у зв’язуванні облікових записів із детальними або згрупованими дозволами, визначеними в інших місцях системи.

Для керування колекціями повноважень використовується AuthorityCollectionService. Ця служба працює з AuthorityCollectionRepository для обробки операцій, пов’язаних із колекціями повноважень. Він дає змогу отримувати колекції повноважень за їхніми ідентифікаторами чи іншими атрибутами та взаємодіє з AuthorityCollectionAuthorityRepository для керування зв’язками між колекціями та окремими повноваженнями.

AuthorityManagerService служить абстракцією вищого рівня, яка поєднує функції інших служб. Він інтегрує AccountAuthorityRelationService, AuthorityService та AuthorityCollectionService для забезпечення узгодженого керування повноваженнями та колекціями для облікових записів користувачів. Ця

служба надає методи призначення цілих колекцій або окремих повноважень обліковим записам, обробки всіх базових зв’язків і збереження.

На схемі також виділено репозиторії (AuthorityRepository, BlockerRepository, AccountAuthorityRepository, AuthorityCollectionRepository та AuthorityCollectionAuthorityRepository) як інтерфейси, відповідальні за абстрагування операцій бази даних. Ці репозиторії забезпечують чітке відокремлення між бізнес-логікою та рівнем збереження, що спрощує тестування та обслуговування.

Конструкція гарантує, що кожен аспект керування повноваженнями та дозволами, від окремих повноважень до згрупованих колекцій і заблокованих дозволів, модульовано в окремі служби. Кожна служба взаємодіє з відповідними репозиторіями та картографами, щоб керувати збереженням і перетвореннями даних, забезпечуючи ефективне та безпечне керування ролями користувачів і дозволами в програмі. Ця структура підтримує масштабованість і гнучкість, дозволяючи Authority Service адаптуватися до нових вимог бізнесу.

По суті, служба авторизації забезпечує створення, керування та застосування повноважень користувачів. Повноваження зберігаються в реляційній структурі в PostgreSQL, що дозволяє асоціювати конкретні ролі та дозволи з окремими обліковими записами користувачів або групами користувачів. Така конструкція забезпечує узгодженість даних і підтримує складні запити, наприклад отримання всіх повноважень, призначених користувачеві, або фільтрацію користувачів на основі певних ролей. Відповідність PostgreSQL ACID гарантує надійність і точність цих даних навіть під час одночасних оновлень або сценаріїв із високим трафіком. Саму діаграму бази даних можна побачити на рисунку 3.5.

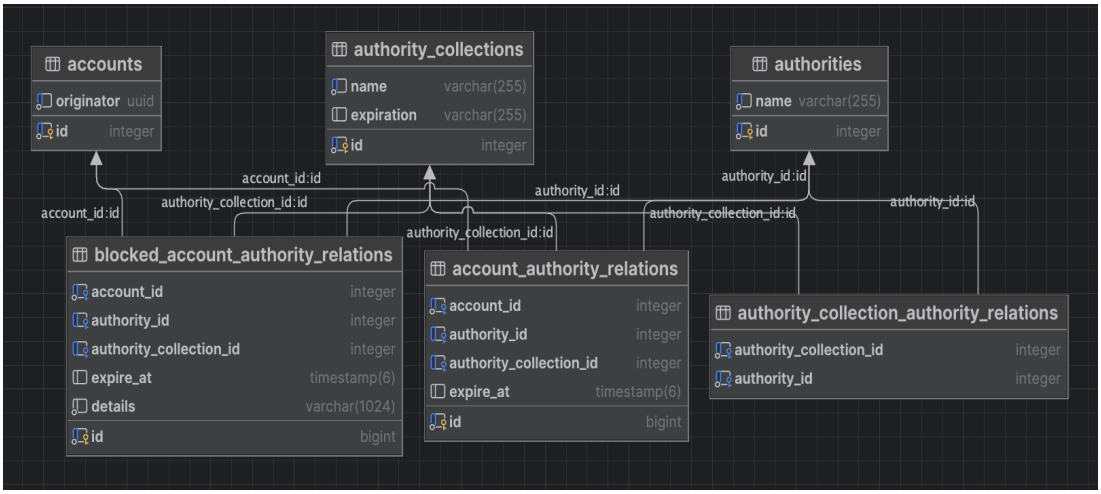

Рисунок 3.6 – Діаграма бази даних

Схема бази даних (яку зображено на рисунку 3.6) для служби авторизації призначена для керування повноваженнями користувачів, наборами дозволів і їхніми зв’язками з обліковими записами. У своїй основі схема підтримує детальний контроль над окремими повноваженнями, згрупованими колекціями та їх призначенням обліковим записам користувачів, забезпечуючи гнучкість і масштабованість системи авторизації.

Таблиця «Облікові записи» містить основну інформацію про всіх користувачів у системі. Кожен обліковий запис має унікальний ідентифікатор («ідентифікатор») і необов’язкове поле «оригінатор», яке може посилатися на зовнішню систему або забезпечувати відстеження для створення облікового запису. Ця таблиця є основою для всіх зв’язків, пов’язаних із дозволами користувача.

Самі повноваження визначаються в таблиці «повноваження». Кожне повноваження представляє окремий дозвіл, визначений унікальним «id» і пов’язаний з описовим «ім’ям». Ці повноваження можна призначити безпосередньо користувачам або згрупувати в колекції для полегшення керування. Колекціями керуються в таблиці "authority_collections", яка групує пов’язані повноваження в одній сутності. Кожна колекція має «ім’я» та необов’язкове поле «термін дії», що дозволяє автоматично анулювати дозволи після певного періоду.

Зв’язки між авторитетами та колекціями встановлюються в таблиці «authority_collection_authority_relations». Це відображення забезпечує зв’язки

«один-до-багатьох» і «багато-до-багатьох», дозволяючи повноваженням належати до кількох колекцій, а колекціям інкапсулювати кілька повноважень. Ця конструкція забезпечує модульне та багаторазове керування дозволами.

Щоб призначити повноваження або колекції обліковим записам користувачів, використовується таблиця "account_authority_relations". Ця таблиця відстежує призначення певних повноважень або зборів обліковим записам із необов’язковим полем «expire_at», щоб визначити період дії призначення. Ця структура дозволяє надавати як постійні, так і тимчасові ролі або дозволи відповідно до різноманітних бізнес-вимог.

Таблиця "blocked_account_authority_relations" доповнює структуру призначення, керуючи заблокованими дозволами. Ця таблиця відстежує повноваження або колекції, які були явно обмежені для облікового запису, гарантуючи, що певні дозволи можна відкликати або перевизначати, незважаючи на те, що вони призначені іншим чином. Це особливо корисно для впровадження винятків або динамічного керування проблемами безпеки. Кожен блок містить поле "expire_at" і додаткові "подробиці", щоб задокументувати обґрунтування або контекст для обмеження.

Загалом, схема розроблена таким чином, щоб збалансувати гнучкість і ефективність, використовуючи реляційні можливості PostgreSQL для підтримки цілісності даних і підтримки складних запитів. Розділивши прямі призначення, колекції та заблоковані дозволи на окремі, але взаємопов’язані таблиці, база даних забезпечує надійне та масштабоване керування ролями та повноваженнями користувачів, задовольняючи потреби динамічних та ієрархічних систем дозволів.

Служба повноважень підтримує як детальне, так і згруповане керування повноваженнями. Це означає, що адміністратори можуть надавати або блокувати окремі повноваження для певного користувача, наприклад доступ до певної функції, або керувати повноваженнями групою, застосовуючи зміни до групи користувачів або ролей. Наприклад, адміністратор може призначити всім користувачам у певному відділі набір дозволів або тимчасово скасувати доступ для групи у відповідь на проблеми безпеки чи організаційні зміни. Ці операції

ефективно обробляються службою Authority Service, використовуючи реляційні можливості PostgreSQL для оновлення окремих записів або виконання пакетних оновлень за потреби.

Щоб оптимізувати продуктивність і зменшити затримку під час частих перевірок доступу, Authority Service інтегрується з високошвидкісним кеш-пам’яттю Redis. Дані про повноваження активних користувачів завантажуються в кеш, що дозволяє системі виконувати швидку перевірку ролей і дозволів без безпосереднього запиту до бази даних PostgreSQL. Це особливо корисно для сценаріїв із високим трафіком, наприклад, коли багато користувачів одночасно взаємодіють із програмою, оскільки це значно зменшує навантаження на базу даних, зберігаючи зручність взаємодії з користувачем.

Служба Authority також тісно інтегрована з шиною подій Kafka, щоб забезпечити динамічні та керовані подіями робочі процеси. Наприклад, коли користувач успішно проходить автентифікацію за допомогою Ory Kratos, генерується подія Kafka, щоб сповістити службу Authority Service про завантаження повноважень користувача в кеш Redis. Це гарантує, що найновіші повноважні дані будуть доступні під час сеансу користувача. Подібним чином, коли адміністратори змінюють повноваження — надають або відкликають дозволи — події Kafka поширюють ці зміни, запускаючи оновлення як бази даних, так і кешу.

Сервіс також включає засоби захисту для забезпечення узгодженості між кешованими даними та офіційними записами в PostgreSQL. Періодичні процеси синхронізації або оновлення в реальному часі, викликані подіями, гарантують, що кешовані повноваження залишаються узгодженими з базою даних. Крім того, служба використовує функцію Redis TTL (Time to Live) для керування життєвим циклом кешованих даних, видалення прострочених записів і зменшення витрат пам’яті.

Абстрагуючи складність повноважень керування, служба Authority Service надає простий API для інших компонентів програми для перевірки, зміни або запиту дозволів користувача. Це забезпечує повну інтеграцію з такими службами,

як Routing Service, яка покладається на Authority Service для забезпечення рольового контролю доступу під час обробки запитів.

Таким чином, Authority Service є основою системи авторизації програми, яка поєднує надійність і структуру PostgreSQL для тривалого зберігання зі швидкістю та гнучкістю Redis для продуктивності в реальному часі. Його здатність керувати як індивідуальними, так і груповими повноваженнями в поєднанні з його інтеграцією в керовану подіями архітектуру програми робить його масштабованим і ефективним рішенням для підтримки надійного контролю доступу. Ця конструкція гарантує, що користувачам надаються відповідні дозволи динамічно та безпечно, а адміністратори зберігають точний контроль над керуванням повноваженнями.

3.1.4 Огляд компоненту Content Transferer Service

Content Transferer Service є ключовим компонентом програми, який відповідає за керування та полегшення передачі мультимедійного вмісту та пов’язаних даних користувачам. Ця послуга доповнює розрив між рівнем зберігання, де фізично зберігаються медіа та книги, і клієнтськими програмами, які споживають цей вміст. Це забезпечує ефективний пошук, безперебійну потокову передачу та безпечний доступ до фільмів, субтитрів, звуків і книг.

В основі Content Transferer Service лежить його база даних, яка зберігає детальні метадані про весь керований вміст. Ці метадані містять важливу інформацію, таку як шлях до файлів в Amazon S3, тривалість фільмів, інформацію про субтитри та звукові доріжки для багатомовної доступності. Крім того, база даних також містить тексти книг, що гарантує, що користувачі можуть взаємодіяти та отримувати як відео, так і текстові навчальні матеріали. Це централізоване зберігання метаданих дозволяє службі ефективно запитувати та обслуговувати

вміст користувачам, забезпечуючи такі функції, як динамічне завантаження субтитрів і адаптивна доставка вмісту.

Сервіс інтегрується з Amazon S3, який служить основним рішенням для зберігання всіх великих медіафайлів, включаючи фільми, файли субтитрів, аудіодоріжки та книги. Фільми, які зберігаються в S3, попередньо обробляються та розбиваються на менші сегменти, що дозволяє службі Content Transferer передавати їх користувачам поступово. Ця сегментація забезпечує плавне відтворення та зменшує споживання пропускної здатності, особливо для користувачів із обмеженим доступом або в мобільних мережах. Субтитри та пов’язані звукові доріжки також витягуються з S3 і синхронізуються з сегментами відео, забезпечуючи цілісне враження від перегляду для кінцевого користувача.

Для книг повний текст зберігається як у базі даних (для метаданих і пошуку вмісту), так і в S3 (для великомасштабного зберігання та розповсюдження). Сервіс гарантує, що книги можна завантажувати або представляти сегментами, подібно до фільмів, оптимізуючи процес передачі для користувачів, які можуть забажати отримати доступ до певних розділів або частин вмісту.

Служба передачі вмісту також забезпечує безпеку вмісту та контроль доступу шляхом інтеграції зі службою авторизації. Перед наданням будь-якого вмісту служба перевіряє, чи має користувач, який запитує, відповідні дозволи для доступу до запитуваного ресурсу. Це забезпечує дотримання ліцензійних угод і захист конфіденційної інтелектуальної власності. Наприклад, якщо користувач запитує фільм або книгу, служба консультується зі службою авторизації, щоб перевірити ролі та дозволи користувача перед тим, як почати передачу.

Крім того, служба оптимізує доставку контенту, підтримуючи механізми кешування та інтеграцію мережі доставки контенту (CDN). Ця архітектура забезпечує масштабованість, дозволяючи службі обробляти великі обсяги трафіку без шкоди для швидкості доставки чи взаємодії з користувачем.

Таким чином, Content Transferer Service – код якого наведено в лістингу 3.4 додатку А – розроблено для керування метаданими та ефективної доставки мультимедійного та текстового вмісту, що зберігається в Amazon S3.

Використовуючи надійну базу даних для метаданих, розділяючи великі мультимедійні файли для поступової передачі та забезпечуючи безпечний доступ через Authority Service, він забезпечує безперебійне та оптимізоване використання вмісту для користувачів. Його здатність працювати з фільмами, субтитрами, аудіодоріжками та книгами в єдиній структурі робить його наріжним каменем системи доставки вмісту програми.

3.2 Тестування під час розробки

Процес тестування для цього проекту був розроблений для забезпечення надійності, функціональності та продуктивності всіх компонентів програми. Було застосовано комплексну стратегію тестування, що включає модульне тестування, інтеграційне тестування та smoke тестування для перевірки системи на різних рівнях, від окремих модулів до програми в цілому. Платформа тестування використовувала такі галузеві стандартні інструменти, як Spring Mockito для модульних тестів і Testcontainers для інтеграційних тестів, забезпечуючи послідовність і точність протягом життєвого циклу розробки.

Модульне тестування послужило основою піраміди тестування, зосереджуючись на перевірці правильності окремих компонентів окремо. Використовуючи Spring Mockito, кожна служба та репозиторій були протестовані незалежно, висміюючи їхні залежності. Цей підхід дозволив розробникам перевірити поведінку бізнес-логіки та забезпечити коректність операцій, не залежачи від зовнішніх систем, таких як бази даних або API. Наприклад, службу Authority Service було ретельно перевірено, щоб переконатися, що її методи призначення, скасування та блокування дозволів ведуть себе належним чином за різних сценаріїв. Подібним чином, методи репозиторію були протестовані з використанням імітованих даних, щоб перевірити, чи запити були правильно структуровані та дали очікувані результати. Завдяки ізоляції окремих

функціональних одиниць помилки можна було виявити та усунути на ранніх стадіях процесу розробки, зменшивши ризик комплексних помилок.

Інтеграційне тестування було ще одним критичним рівнем стратегії тестування, перевіряючи взаємодію між кількома компонентами в межах програми. Testcontainers використовувався для створення легких, ефемерних екземплярів зовнішніх залежностей, таких як PostgreSQL для бази даних, Redis для кешування та Kafka для посередництва повідомлень. Такий підхід дозволив тестам імітувати реальне середовище, не потребуючи спеціальної інфраструктури тестування. Наприклад, потік подій від Ory Kratos через Kafka до Authority Service було ретельно перевірено, щоб переконатися, що дозволи користувача правильно створено, завантажено в кеш і синхронізовано з базою даних. Тести інтеграції також підтвердили бездоганну взаємодію служб із зовнішніми API, такими як Translation API або AWS S3, забезпечуючи обробку та отримання даних належним чином. Випробовуючи ці взаємодії в середовищі, яке дуже нагадувало робоче, інтеграційні тести забезпечили високий ступінь впевненості в наскрізній функціональності програми.

Smoke тестування було використано як останній рівень перевірки, особливо корисний під час розгортання та після значних оновлень програми. Димові тести були зосереджені на перевірці найважливіших функціональних можливостей, щоб переконатися, що система працює стабільно та готова до подальшого тестування або використання у виробництві. Ці тести охоплювали основні робочі процеси, такі як реєстрація користувачів, автентифікація, пошук вмісту та керування повноваженнями. Наприклад, після розгортання нової версії програми димові тести перевірятимуть, чи можуть користувачі входити в систему, отримувати доступ до своїх дозволів, переглядати вміст і взаємодіяти з основними функціями програми без серйозних проблем. Димове тестування діяло як швидка контрольна точка для виявлення будь-яких серйозних регресій або помилок конфігурації перед більш глибоким тестуванням або випуском.

Завдяки поєднанню цих методологій тестування проект досяг надійного процесу перевірки, який мінімізував ризики та забезпечив високу якість

результатів. Модульне тестування за допомогою Spring Mockito забезпечило впевненість в окремих компонентах, інтеграційне тестування за допомогою Testcontainers підтвердило складні взаємодії, а димове тестування забезпечило стабільність ключових робочих процесів. Цей багаторівневий підхід дозволив групі розробників завчасно виявляти проблеми, всебічно перевіряти функціональність і підтримувати високий рівень надійності протягом життєвого циклу програми.

3.3 Розгортання ресурсу

Процес розгортання цього проекту використовує AWS як постачальника хмарної інфраструктури, забезпечуючи масштабованість, надійність і повну інтеграцію з керованими службами. Стратегія розгортання використовує кластери Kubernetes для оркестровки контейнерів, а також Terraform для інфраструктури як коду (IaC) для автоматизації та стандартизації надання та налаштування ресурсів. Цей підхід гарантує узгодженість у різних середовищах, від розробки до виробництва, одночасно забезпечуючи швидку ітерацію та масштабування.

В основі розгортання лежить кластер Amazon Elastic Kubernetes Service (EKS), який керує мікросервісами програми. Кластер призначений для динамічного масштабування, самовідновлення та високої доступності, гарантуючи, що програма залишається продуктивною за змінних навантажень. Kubernetes керує контейнерними службами, розгортаючи їх в ізольованих пакетах і розумно розподіляючи трафік між вузлами. Служба маршрутизації в кластері налаштована за допомогою вхідного контролера, що забезпечує точну маршрутизацію зовнішніх запитів до відповідних внутрішніх мікросервісів. Наприклад, запити API для автентифікації користувача направляються до Ory Kratos, тоді як запити для отримання вмісту спрямовуються до Content Transferer Service. Ця маршрутизація забезпечує ефективну та безпечну обробку вхідного трафіку, використовуючи мережеві можливості AWS для оптимізації продуктивності.

Інфраструктура повністю налаштована та керована за допомогою Terraform, що дозволяє визначити весь процес розгортання як код. Сценарії Terraform описують бажаний стан ресурсів, включаючи кластер EKS, пов’язані віртуальні приватні хмари (VPC), підмережі, групи безпеки та ролі IAM. Цей декларативний підхід гарантує, що зміни інфраструктури контролюються версіями, перевіряються та відтворюються в різних середовищах. Наприклад, сценарії Terraform створюють необхідні ролі IAM, щоб дозволити вузлам Kubernetes безпечно взаємодіяти зі службами AWS, такими як S3 для зберігання медіафайлів або RDS для керування базами даних.

Terraform також надає додаткові ресурси, необхідні для розгортання. Це включає в себе налаштування Amazon Route 53 для керування DNS, дозволяючи додатку направляти трафік через спеціальні доменні імена. Балансувальники навантаження налаштовані для розподілу трафіку в кластері Kubernetes, забезпечуючи високу доступність і відмовостійкість. Terraform забезпечує створення моніторингу Amazon CloudWatch для інфраструктури, що дозволяє в режимі реального часу аналізувати продуктивність системи та використання ресурсів.

Конвеєр розгортання інтегрується з процесом CI/CD, автоматизуючи етапи створення, тестування та розгортання програми. Після контейнеризації програми за допомогою Docker отримані зображення надсилаються до Amazon Elastic Container Registry (ECR). Потім Terraform інтегрує ці зображення в маніфести Kubernetes, оновлюючи кластер новими версіями програми. Розгортання керуються за допомогою стратегій розгортання Kubernetes, таких як поточні оновлення, щоб забезпечити нульовий час простою під час оновлень. Це забезпечує плавний перехід між версіями додатків, мінімізуючи збої в роботі користувачів [15].

Маршрутизація в інфраструктурі призначена для оптимізації потоку трафіку та підвищення безпеки. Зовнішній трафік спочатку проходить через Amazon API Gateway, який діє як основна точка входу та надає такі функції, як перевірка запитів, обмеження швидкості та примусова автентифікація. Шлюз направляє запити до вхідного контролера Kubernetes, який далі направляє їх до відповідних

мікросервісів на основі попередньо визначених правил. Ця багаторівнева маршрутизація забезпечує ефективний розподіл трафіку, зберігаючи суворий контроль доступу.

Для підвищення масштабованості та стійкості до кластера EKS застосовуються політики автоматичного масштабування. Вузли Kubernetes динамічно масштабуються залежно від потреб ресурсів, гарантуючи, що інфраструктура може впоратися зі сплесками трафіку без ручного втручання. Сценарії Terraform визначають ці політики разом із наданням груп автоматичного масштабування Amazon для базових екземплярів EC2.

Таким чином, процес розгортання поєднує в собі гнучкість Kubernetes, масштабованість AWS і можливості автоматизації Terraform. Завдяки визначенню всієї інфраструктури як коду розгортання є повторюваним, надійним і простим у управлінні. Використання EKS і вдосконалених механізмів маршрутизації гарантує, що програма залишається високодоступною та продуктивною, тоді як інтеграція конвеєрів CI/CD забезпечує швидку ітерацію та безперебійне оновлення. Ця надійна стратегія розгортання гарантує, що програма може легко масштабуватися відповідно до вимог користувачів, зберігаючи бездоганну роботу.

3.4 Огляд готового проєкту

З точки зору клієнта, додаток створено для забезпечення безперебійного та інтуїтивно зрозумілого процесу реєстрації, автентифікації та підтвердження. Користувач взаємодіє з цими компонентами через адаптивний, зручний інтерфейс, який забезпечує безпеку та зручність під час керування робочими процесами, пов’язаними з обліковим записом. Ці функції підкріплені надійними серверними

службами та продумано розробленим зовнішнім інтерфейсом, які надають перевагу взаємодії з користувачем та цілісності системи.

Процес реєстрації — це перша взаємодія нового клієнта з системою. Інтерфейс представляє зрозумілу та зрозумілу форму, де користувачі можуть вводити необхідні дані, такі як ім’я, адреса електронної пошти, пароль і, за бажанням, бажана мова. Система динамічно адаптує мову реєстраційної форми на основі налаштувань браузера або пристрою клієнта, забезпечуючи доступність для користувачів у різних регіонах. Після надсилання деталей програма перевіряє введені дані в режимі реального часу, забезпечуючи миттєвий зворотний зв’язок для виправлень, таких як ненадійні паролі чи неправильно відформатовані адреси електронної пошти. Форму для проходження реєстрації можна побачити на рисунку 3.7.

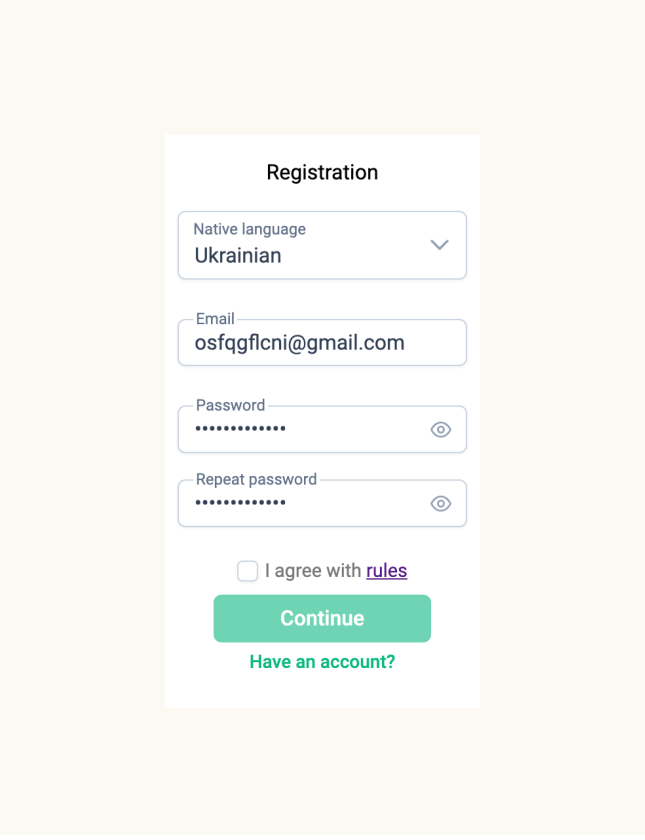

Рисунок 3.7 – Форма реєстрації

Після успішного надсилання бекенд створює попередній запис користувача та запускає процес підтвердження, що забезпечує автентичність наданих

контактних даних. Клієнт отримує електронний лист із підтвердженням або SMS із одноразовим кодом підтвердження або безпечним посиланням. Потім користувач переходить на сторінку підтвердження, де він може ввести код або натиснути посилання, щоб підтвердити свій обліковий запис. Приклад листа наведено на рисунку 3.8.

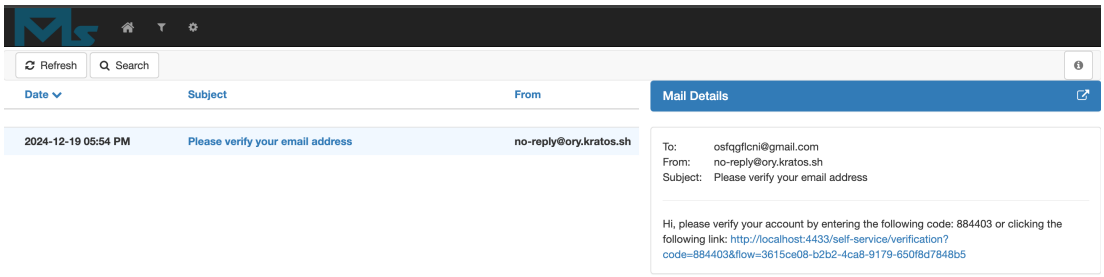

Рисунок 3.8 – Приклад листа для верифікації електронної адреси

Цей процес додає критичний рівень безпеки, запобігаючи несанкціонованому доступу та гарантуючи, що лише законні користувачі можуть завершити реєстрацію. Якщо користувач стикається з будь-якими проблемами під час підтвердження, як-от прострочений або недійсний код, програма надає можливість повторно надіслати повідомлення підтвердження, зберігаючи безперебійну роботу користувача. На рисунку 3.9 зображено форму, що дозволяє підтвердити електронну адресу.

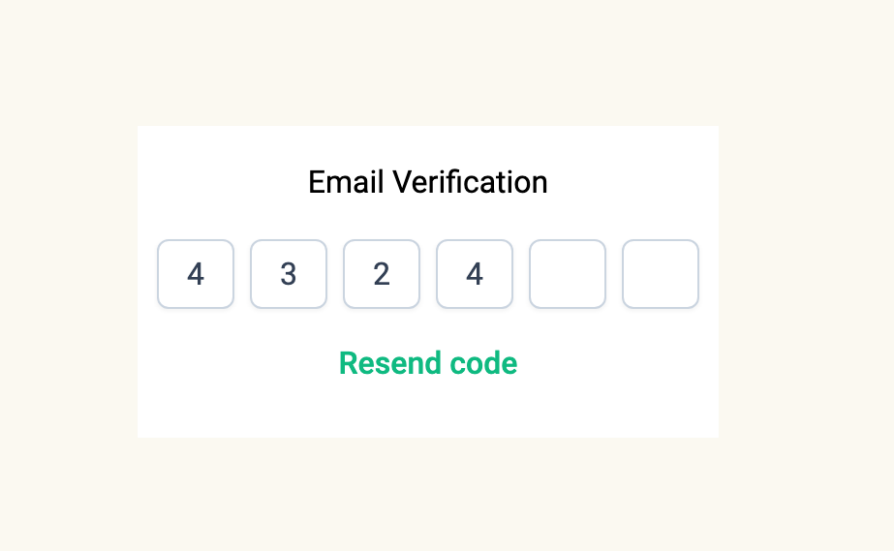

Рисунок 3.9 – Форма підтвердження електронної адреси

Після підтвердження облікового запису компонент автентифікації стає центральним у взаємодії користувача з програмою. Користувачі входять, ввівши адресу електронної пошти та пароль на адаптивній сторінці входу. Для підвищення безпеки програма також може пропонувати багатофакторну автентифікацію (MFA), вимагаючи від користувачів введення коду підтвердження, надісланого на їх зареєстровану електронну адресу або телефон. Система перевіряє облікові дані за допомогою безпечних серверних служб і надає негайний зворотний зв’язок, якщо спроба входу виявляється невдалою. Успішна автентифікація надає користувачеві доступ до основної програми, а система безпечно керує їхнім сеансом за допомогою токенів, що зберігаються в браузері.

Протягом усього шляху клієнта програма забезпечує послідовність і надійність обробки даних користувача. Усі запити захищені за допомогою HTTPS, а конфіденційні дані, наприклад паролі, ніколи не зберігаються у вигляді звичайного тексту. Натомість паролі безпечно хешуються, а для керування сеансами користувачів використовується автентифікація на основі маркерів. Крім того, інтерфейсні компоненти динамічно підлаштовуються під потреби користувача через локалізацію мови, функції доступності або адаптивний дизайн для різних пристроїв.

Таким чином, програма з точки зору клієнта ретельно розроблена, щоб запропонувати ефективну, безпечну та зручну роботу. Його процес реєстрації простий і адаптований, механізм підтвердження забезпечує безпеку та автентичність, а система автентифікації забезпечує надійний контроль доступу, зберігаючи простоту використання. Разом ці компоненти формують основу безперебійної взаємодії з клієнтом, що дозволяє користувачам швидко працювати з повним набором функцій програми, відчуваючи впевненість у безпеці та надійності платформи.

Після успішної автентифікації користувачі отримують доступ до багатого та динамічного середовища, де вони можуть досліджувати та взаємодіяти з великою колекцією фільмів і книг, спеціально підібраних для покращення їхнього досвіду вивчення мови. Дизайн надає пріоритет інтерактивності та персоналізації, що дозволяє користувачам плавно переходити між розвагами та навчанням. До прикладу, рисунок 3.10 відображає сторінку з фільмами та рівне мови, необхідним для перегляду.

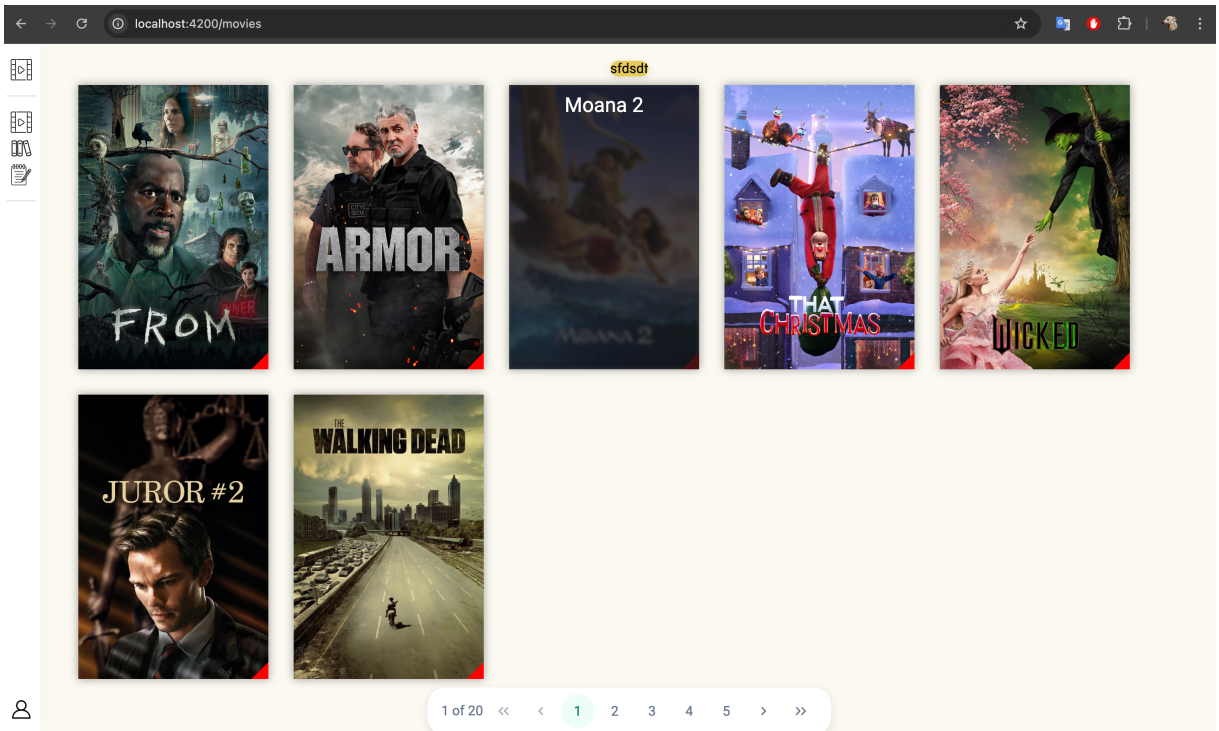

Рисунок 3.11 – Приклад сторінки з фільмами

Також, на рисунку 3.11 можна помітити, що при наведені на фільм користувачеві відображається короткий опис контенту.

Після доступу до розділу фільмів користувачам пропонується візуально організований каталог із афішами фільмів, назвами та короткими описами. Кожний фільм позначається такими метаданими, як час виконання, жанр, рівень володіння мовою, а також наявність субтитрів і звукових доріжок різними мовами. Користувачі можуть легко фільтрувати або здійснювати пошук у каталозі за жанром, рівнем мови чи ключовими словами, що дозволяє їм швидко знаходити вміст, який відповідає їхнім навчальним цілям чи особистим інтересам.

Коли користувач вибирає фільм, він переходить до захоплюючого інтерфейсу відеоплеєра, оснащеного інструментами для вивчення мови. Під час відтворення субтитри відображаються вибраною мовою, допомагаючи в реальному часі зрозуміти діалог. Однак субтитри не просто статичні; вони дуже інтерактивні та створені для заохочення активної взаємодії. Користувачі можуть навести курсор на будь-яке слово чи фразу в субтитрах, що автоматично призупинить відео саме в цей момент. Ця функція гарантує, що користувач має достатньо часу, щоб зосередитися на слові чи фразі, не пропускаючи жодної частини фільму.

Коли користувач наводить курсор на слово чи фразу, з’являється спливаюче вікно з його перекладом, посібником з вимови та будь-якою додатковою контекстною інформацією, такою як граматичне використання чи синоніми. Якщо користувач вважає слово або фразу цінними для навчання, він може клацнути на ньому, щоб зберегти його в особистому словнику. Система інтелектуально фіксує не лише слово, але й речення чи фразу, де воно було використано, зберігаючи контекст для кращого запам’ятовування під час майбутніх переглядів. Наприклад, якщо користувач наводить курсор на фразу «розбити лід», система збереже її як ідіоматичний вираз із контекстом використання, що полегшить розуміння та запам’ятовування.

Окрім збереження слів, користувачі можуть додавати власні примітки до кожного збереженого запису, наприклад, власний переклад або мнемонічні пристрої, ще більше персоналізуючи свій досвід навчання. Можливість взаємодії із

субтитрами таким чином перетворює пасивний перегляд на активну навчальну діяльність, роблячи засвоєння мови привабливим і ефективним.

Розділ книг надає подібний інтерактивний досвід. Користувачів вітає інтерфейс бібліотеки, який дозволяє їм переглядати книги за категоріями, такими як рівень складності, жанр чи автор. Кожна книга містить короткий опис, інформацію про рівень мови та приблизний час, необхідний для її прочитання. Коли користувач вибирає книгу, він переходить до інтерактивного інтерфейсу для читання.

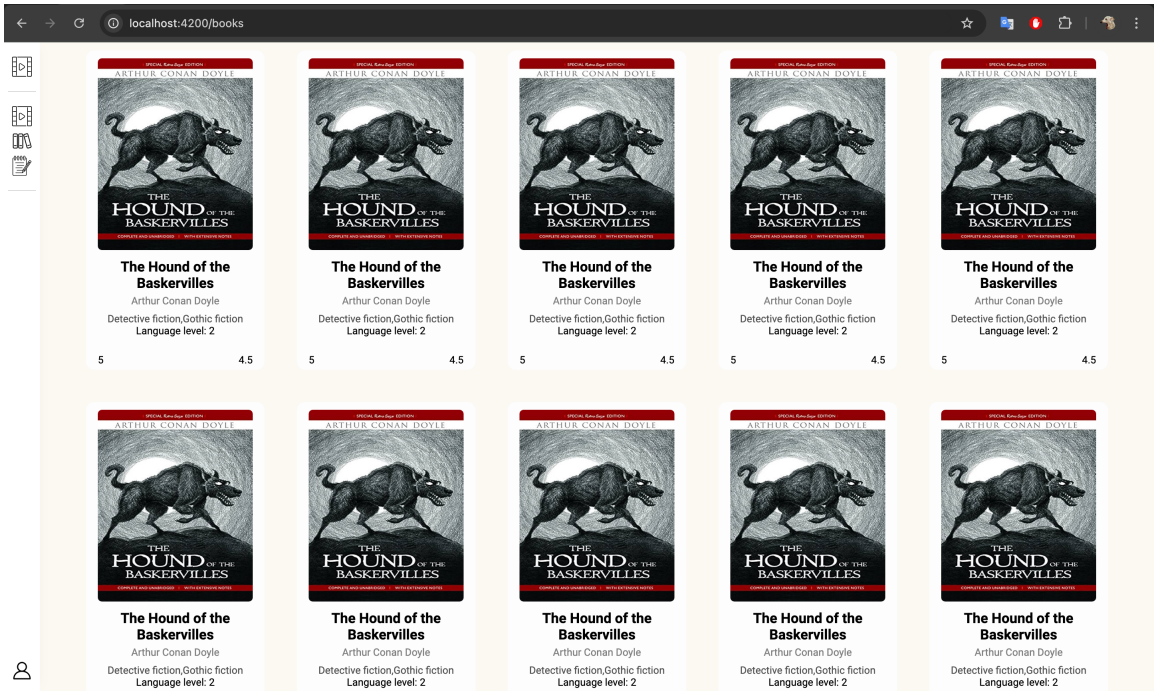

Рисунок 3.12 – Зображення сторінки з книгами.

Інтерфейс для читання створено для емуляції тієї ж інтерактивності, що й субтитри до фільму. Користувачі можуть виділити будь-яке слово чи фразу в книзі, викликавши спливаюче вікно з його перекладом, посібником з вимови та контекстним використанням. Подібно до субтитрів до фільму, текст книги автоматично призупиняє прокручування, коли слово виділяється, що гарантує, що користувачі мають достатньо часу, щоб зосередитися на слові чи фразі, не втрачаючи місця в тексті. Якщо користувач вирішує зберегти слово, система

зберігає його в його особистому словнику разом із реченням або абзацом, у якому воно з’явилося, надаючи широкий контекст для подальшого використання.

Для фраз або ідіоматичних виразів система розумно визначає їхнє значення в контексті, гарантуючи, що користувачі навчаться правильному використанню, а не просто дослівному перекладу. Наприклад, виділення слова «хіт книжок» збереже його як ідіоматичний вираз, що означає «вивчати», а не тлумачить його буквально.

Збережені слова та фрази з фільмів і книг зведені в особистий словник користувача, доступний у будь-який час. Словник надає повне уявлення про словниковий запас користувача разом із перекладами, посібниками з вимови та оригінальним контекстом. Користувачі можуть упорядковувати свої збережені записи за спеціальними категоріями, наприклад «загальні фрази» або «технічні терміни», і періодично їх переглядати. У словнику також є практичні інструменти, такі як картки або тести, що дозволяє користувачам з часом закріплювати знання.

Разом ці функції створюють інтерактивне та персоналізоване навчальне середовище. Дозволяючи користувачам глибоко взаємодіяти з контентом, будь то перегляд фільмів чи читання книг, програма перетворює традиційне вивчення мови на захоплюючий і приємний досвід. Можливість робити паузи, перекладати та зберігати слова чи фрази в контексті долає розрив між розвагами та навчанням, сприяючи ефективному засвоєнню мови природним і привабливим способом. Ця інтеграція інтерактивності, контекстного навчання та персоналізованих інструментів робить програму потужним ресурсом для тих, хто вивчає мову на всіх рівнях.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці

Запропоноване у цій роботі рішення, що передбачає розробку лінгвістичного ресурсу на основі мікросервісної архітектури та реактивного програмування, відповідає вимогам чинного законодавства України у сфері охорони праці, техніки безпеки та протипожежної безпеки. Під час організації робочого процесу були враховані нормативні акти, зокрема Закон України «Про охорону праці», Кодекс законів про працю України, Державні санітарні норми і правила роботи з комп'ютерною технікою (ДСНС 3.3.2-007-98) та інші актуальні вимоги, які діють у 2024 році [16].

Процес створення лінгвістичного ресурсу вимагає тривалої роботи фахівців у комп’ютерних середовищах, що створює потенційні ризики для здоров'я працівників. Для їх мінімізації робочі місця розробників були організовані відповідно до Державних санітарних норм і правил роботи з персональними електронно-обчислювальними машинами (ДСНС 3.3.2-007-98) та вимог ДСТУ 2.09.04-87 «Адміністративні та побутові будівлі» [17].

Робочі місця обладнані ергономічними кріслами з регульованою спинкою та підтримкою поперекового відділу, а також регульованими столами для зручного розташування обладнання. Монітори розташовані на рівні очей з мінімальною відстанню у 50–70 см від обличчя розробника, що відповідає вимогам щодо профілактики комп'ютерного зорового синдрому. Програмне середовище, включаючи темні режими інтерфейсу у середовищах розробки (IDE) та адаптивні налаштування яскравості, сприяє зниженню навантаження на зір під час роботи у вечірній час.

Для профілактики втоми розробників впроваджено систему регулярних перерв, під час яких рекомендується виконувати фізичні вправи та вправи для очей. Це відповідає нормам пункту 8.4 ДСН 3.3.2-007-98, які регламентують перерви для відпочинку та розминки. Крім того, вказані норми встановлюють, що тривалість

роботи з комп'ютером повинна чергуватися з іншими видами діяльності для запобігання фізичному і психічному перевантаженню.

Розробка програмного забезпечення відбувається з дотриманням вимог техніки безпеки, що включають як роботу з комп'ютерним обладнанням, так і заходи для мінімізації електричних ризиків. Відповідно до вимог Правил безпечної експлуатації електроустановок споживачів (НПАОП 40.1-1.21-98), усі робочі місця розробників обладнані справною електричною інфраструктурою, а комп'ютерна техніка регулярно перевіряється на наявність ушкоджень та збоїв [18].

Електричні розетки та проводка відповідають нормам ДСТУ EN 50110-1:2020, а персонал проінструктований щодо правил безпечної експлуатації електрообладнання. У випадку технічного збою передбачено можливість автоматичного аварійного вимкнення живлення для уникнення короткого замикання чи перевантаження системи. Кожен працівник був проінструктований щодо порядку дій у разі виявлення несправності технічного обладнання або аварійних ситуацій.

Для безпечної роботи в офісі забезпечено заземлення комп'ютерного обладнання, а також стабільне електропостачання з використанням блоків безперебійного живлення (UPS). Додатково, середовище розробки підтримується у серверних приміщеннях, де встановлено системи моніторингу температури, вентиляції та пожежної сигналізації, що відповідає нормам ДБН В.2.5-23-2020 «Проектування електрообладнання» [19].

Особлива увага приділена протипожежній безпеці, оскільки робота з комп'ютерною технікою та електронними мережами підвищує ризик виникнення пожеж. Приміщення, у яких проводиться розробка програмного забезпечення, відповідають вимогам Правил пожежної безпеки в Україні (НАПБ А.01.001-2021). Офісні приміщення оснащені сучасними системами пожежної сигналізації, автоматичними установками пожежогасіння та первинними засобами пожежогасіння, зокрема вогнегасниками в кількості, що відповідає встановленим нормам [20].

Усі працівники пройшли інструктаж з пожежної безпеки відповідно до вимог Наказу МВС №1417 від 30.12.2014 року, включно з відпрацюванням евакуації на випадок пожежі. Шляхи евакуації позначені світловими індикаторами та вказівниками, а у кожному приміщенні розміщено плани евакуації. Комп'ютерне обладнання забезпечено системами теплового моніторингу, що автоматично сигналізують про перегрівання чи інші ризики виникнення пожежонебезпечних ситуацій.

Окрім фізичних ризиків, важливим аспектом є психологічне навантаження на розробників, що виникає через інтенсивну роботу з мікросервісами, тестування та усунення помилок. У процесі роботи активно використовується Scrum-методологія, яка дозволяє рівномірно розподіляти завдання між членами команди, уникати перевантаження та дотримуватися встановлених термінів без зайвого стресу.

Запроваджено практику гнучкого робочого графіка, що дозволяє розробникам планувати свій час з урахуванням індивідуальної продуктивності. Крім того, регулярні зустрічі та ретроспективи допомагають виявляти проблеми на ранніх етапах і знаходити ефективні рішення, що позитивно впливає на загальний психологічний стан команди.

Таким чином, під час розробки лінгвістичного ресурсу було враховано всі ключові аспекти охорони праці, техніки безпеки та протипожежної безпеки відповідно до чинного законодавства України станом на 2024 рік. Організація ергономічних робочих місць, дотримання санітарно-гігієнічних вимог, впровадження систем моніторингу та протипожежної безпеки дозволяють мінімізувати ризики для здоров'я працівників і забезпечити комфортне та безпечне робоче середовище. Усі рішення, запропоновані в цій роботі, відповідають нормативним актам України, а також сприяють ефективній та безпечній роботі команди розробників.

4.2 Безпека в надзвичайних ситуаціях

Тема кваліфікаційної роботи освітнього рівня «Магістр» присвячена розробці інтерактивного лінгвістичного ресурсу за парадигмою реактивного програмування з використанням фреймворку Spring Boot та мікросервісної архітектури. Тому, доцільно розглянути питання факторів ризику і можливі порушення здоров’я користувачів комп’ютерної мережі.

У сучасному цифровому середовищі, де комп'ютерні мережі та програмні ресурси відіграють ключову роль у повсякденній діяльності, особливу увагу необхідно приділяти факторам ризику, що можуть вплинути на здоров’я користувачів. Розроблений лінгвістичний ресурс, побудований на парадигмі реактивного програмування та мікросервісній архітектурі, орієнтований на тривалу та інтенсивну взаємодію користувачів із цифровим контентом. Це створює потенційні загрози для здоров'я, які повинні бути вчасно ідентифіковані та мінімізовані шляхом впровадження відповідних заходів безпеки [21].

Тривале перебування користувача перед екраном, пов’язане з переглядом фільмів, читанням книг та активною взаємодією з системою, може призвести до комп’ютерного зорового синдрому. Симптоми включають втому очей, головний біль, нечіткість зору та сухість очей. Використання інтерактивних функцій, як-от зупинка фільму для перекладу слів чи активне підсвічування тексту в книгах, може вимагати зосередженого погляду на екран, що додатково підвищує навантаження на зір.

Інший фізіологічний ризик стосується порушення постави та опорно-рухового апарату через тривале сидіння в статичній позі. Користувачі, які активно працюють з лінгвістичним ресурсом, часто перебувають у нерухомому стані, що призводить до болю в спині, шиї та кистях рук. Такі симптоми посилюються через відсутність регулярних перерв і невідповідну ергономіку робочого місця.

Надмірна взаємодія з комп'ютерною мережею може призвести до психоемоційного виснаження. Тривале користування ресурсом, особливо під час

інтенсивного навчання іноземної мови, може викликати когнітивну перевтому. Потреба у зосередженні на перекладі слів, перегляді фільмів чи читанні книг у режимі реального часу на додачу до інтерактивних функцій, як-от збереження слів у словнику, створює значне ментальне навантаження.

Крім того, в умовах ізоляції або обмеженої соціальної активності зростає ризик цифрової залежності. Користувачі, захоплюючись використанням функцій ресурсу, таких як перегляд медіаконтенту або глибокий аналіз текстів, можуть втратити баланс між цифровим і реальним світом. Це негативно впливає на психічне здоров'я та емоційний стан.

Важливим фактором ризику є стрес, спричинений технічними збоями у роботі мережі або програмного забезпечення. Оскільки лінгвістичний ресурс базується на мікросервісній архітектурі, в якій кожна служба відповідає за окремий функціонал, потенційні перебої або затримки у взаємодії між сервісами можуть створювати фрустрацію користувачів. Наприклад, затримка в завантаженні фільму чи синхронізації інтерактивних субтитрів може викликати роздратування, знижуючи ефективність навчального процесу [22].

Також стрес-фактори пов’язані із безпекою даних користувачів. Незважаючи на використання сучасних технологій, як-от gRPC для комунікації між мікросервісами та безпечне зберігання даних у AWS S3, користувачі можуть відчувати занепокоєння щодо витоків інформації, особливо під час реєстрації та авторизації.

Для зниження ризиків, пов’язаних із використанням лінгвістичного ресурсу, було передбачено впровадження регулярних нагадувань про перерви, які сприяють зменшенню навантаження на очі та опорно-руховий апарат користувачів. Це дозволяє підтримувати здоровий ритм роботи та уникати перенапруги під час тривалого навчання. Адаптивна яскравість інтерфейсу та підтримка темних режимів відображення допомагають знизити напругу на зір, особливо при тривалій взаємодії з екраном у різних умовах освітлення. Додатково було реалізовано оптимізацію продуктивності системи для мінімізації затримок і технічних збоїв, що забезпечується завдяки використанню переваг мікросервісної архітектури та

реактивного програмування. Це дозволяє системі швидко обробляти дані та забезпечувати безперебійне функціонування під час високих навантажень. Важливим аспектом також є захист даних користувачів та прозора політика конфіденційності, що досягається завдяки використанню безпечних методів комунікації на основі gRPC. Усі ці заходи спрямовані на створення безпечного, надійного та комфортного середовища для користувачів, яке відповідає сучасним вимогам продуктивності та охорони здоров'я.

Загалом, проект лінгвістичного ресурсу, розроблений із використанням сучасних технологій, таких як мікросервісна архітектура, реактивне програмування та SPA, враховує високі вимоги до продуктивності та надійності. Проте усвідомлення можливих ризиків для здоров'я користувачів дозволяє впровадити превентивні заходи, які сприяють безпечному та комфортному користуванню системою в довгостроковій перспективі.

ВИСНОВКИ

Підсумовуючи, розробка цього ресурсу привела до створення надзвичайно складної але зручної для користувача платформи, яка ефективно поєднує розваги та освіту, дозволяючи користувачам вдосконалювати свої мовні навички за допомогою ефектного інтерактивного досвіду. Впровадження сучасних технологій сприяло досягненню масштабованості, швидкості та безперебійної функціональності ресурсу.

Вибір архітектури мікросервісу дозволив програмі бути модульною, підтримуваною та масштабованою, причому кожна служба несе певну відповідальність. Ця архітектура гарантує, що система може розвиватися з часом, адаптуючись до нових вимог або масштабуючись, щоб задовольнити зростаючий попит користувачів, не впливаючи на загальну функціональність.

Щоб реалізувати ефективний зв’язок між службами, було використано gRPC, що забезпечує швидкий, безпечний і надійний обмін даними між мікросервісами. Цей вибір підвищив загальну продуктивність і забезпечив бездоганну інтеграцію таких компонентів, зберігаючи сувору безпеку та узгодженість даних. У інтерфейсі було реалізовано архітектуру SPA, що забезпечує користувачам гладкий динамічний інтерфейс.

Загалом ресурс представляє сучасний та інноваційний підхід до вивчення мови, підкріплений передовими технологіями. Поєднання мікросервісів, парадигм реактивного програмування, ефективного міжсервісного зв’язку з gRPC і високоінтерактивного інтерфейсу SPA створило надійну та привабливу платформу. Цей ресурс не тільки відповідає сучасним технологічним стандартам, але також має хороші можливості для майбутнього зростання та адаптації, що робить його цінним інструментом для тих, хто вивчає мови.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. М.Р. Петрик, Д.М. Михалик, О.Ю. Петрик, Г.Б. Цуприк. Методичні вказівки до виконання атестаційної роботи магістра за спеціальністю 121 – “Інженерія програмного забезпечення” для усіх форм навчання [Текст] – Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя – 2020 – 27 с.

2. Harris C. Microservices architecture [Електронний ресурс] / C. Harris. – 2024. – Режим доступу до ресурсу: https://www.atlassian.com/microservices/microservices-architecture.

3. Ramotion. Understanding The Single-Page Application Architecture [Електронний ресурс] / Ramotion. – 2024. – Режим доступу до ресурсу: https://www.ramotion.com/blog/single-page-application-architecture/.

4. Weder S. Spring Boot: Everything you need to know, and what nobody told you [Електронний ресурс] / Sousa Weder. – 2023. – Режим доступу до ресурсу: https://dev.to/weder96/spring-boot-everything-you-need-to-know-and-what-nobody-told-you-o4j.

5. Mauricio S. Developing and deploying Spring Boot microservices on Kubernetes [Електронний ресурс] / Salatino Mauricio. – 2022. – Режим доступу до ресурсу: https://learnk8s.io/spring-boot-kubernetes-guide.

6. Martin E. What Is Reactive Programming? [Електронний ресурс] / Eric Martin. – 2024. – Режим доступу до ресурсу: https://www.baeldung.com/cs/reactive-programming.

7. Introduction to gRPC [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://grpc.io/docs/what-is-grpc/introduction/.

8. What’s the Difference Between gRPC and REST? [Електронний ресурс]. – 2014. – Режим доступу до ресурсу: https://aws.amazon.com/compare/the-difference-between-grpc-and-rest/.

9. What is PostgreSQL? Everything You Need to Know [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://www.percona.com/blog/what-is-postgresql-used-for/.

10. Stefanyshyn, I., Pastukh, O., Stefanyshyn, V., Baran, I., Boyko, I. Robustness of AI algorithms for neurocomputer interfaces based on software and hardware technologies [Текст] / I. Stefanyshyn, O. Pastukh, V. Stefanyshyn, I. Baran, I. Boyko // CEUR Workshop Proceedings. – 2024. – №3742. – С. 137–149.

11. ACID Transactions [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://www.databricks.com/glossary/acid-transactions.

12. What is Elasticsearch? [Електронний ресурс] // AWS. – 2024. – Режим доступу до ресурсу: https://aws.amazon.com/what-is/elasticsearch/.

13. Що таке Kubernetes? [Електронний ресурс] // Kubernetes. – 2024. – Режим доступу до ресурсу: https://kubernetes.io/uk/docs/concepts/overview/what-is-kubernetes/.

14. Why is AWS Complicated? Simplifying AWS for Users [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://www.digitalocean.com/resources/articles/aws-complicated.

15. Tyson M. Understanding Terraform: From Basics to Execution [Електронний ресурс] / Tyson. – 2022. – Режим доступу до ресурсу: https://blog.brainboard.co/terraform-explained-in-15-mins-50411dd3a29e?gi=0db68b38321c.

16. Закон України «Про охорону праці» [Текст] – 14.10.1992 № 2694-XII. – Режим доступу: https://zakon.rada.gov.ua/laws/show/2694-12.

17. Державні санітарні норми і правила роботи з персональними електронно-обчислювальними машинами (ДСН 3.3.2-007-98) [Текст]. – 1998. – Режим доступу: https://zakon.rada.gov.ua/laws/show/z0203-98.

18. Правила пожежної безпеки в Україні (НАПБ А.01.001-2021) [Електронний ресурс]. – 2021. – Режим доступу: https://zakon.rada.gov.ua/laws/show/z1241-21.

19. Правила безпечної експлуатації електроустановок споживачів (НПАОП 40.1-1.21-98) [Електронний ресурс]. – 1998. – Режим доступу: https://zakon.rada.gov.ua/laws/show/z0093-98.

20. ДБН В.2.5-23-2020 «Проектування електрообладнання» [Електронний ресурс]. – 2020. – Режим доступу: https://dbn.co.ua.

21. Методичний посібник для здобувачів освітнього ступеня «магістр» всіх спеціальностей денної та заочної (дистанційної) форм навчання «Безпека в надзвичайних ситуаціях» [Текст] / В.С. Стручок. – Тернопіль: ФОП Паляниця В. А., 2024. – 156 с. – Режим доступу: https://elartu.tntu.edu.ua/handle/lib/39196.

22. Навчальний посібник «Техноекологія та цивільна безпека. Частина «Цивільна безпека»» [Текст] / автор-укладач В.С. Стручок. – Тернопіль: ФОП Паляниця В. А., 2024. – 156 с. – Режим доступу: http://elartu.tntu.edu.ua/handle/lib/39424.

ДОДАТКИ

ДОДАТОК А ЛІСТИНГИ НАЙВАЖЛИВІШИХ МОДУЛІВ

Лістинг 3.1:

```
@Configuration
@RequiredArgsConstructor
public class RoutingConfiguration {
    @Bean
    public RouteLocator routeLocator(RouteLocatorBuilder
routeLocatorBuilder) {
        return routeLocatorBuilder.routes()
                .route(route -> route
                        .path("/registrations", "/registrations/*",
"/registrations/*/confirm", "/registrations/*/resend-code",
"/accounts/authenticate")
                        .filters(gatewayFilterSpec ->
gatewayFilterSpec.rewritePath("/(?<remaining>.*)", "/account-
service/${remaining}"))
                        .uri("lb://ACCOUNT-SERVICE"))
                .build();
    }
}
```

Лістинг 3.2:

```
CREATE TABLE accounts
(
    id         SERIAL PRIMARY KEY,
    originator UUID NOT NULL UNIQUE
);

REVOKE UPDATE (originator) ON accounts FROM PUBLIC;

CREATE UNIQUE INDEX account_uuid ON accounts (originator);

-----------------------------------------------------------------------
--------------

CREATE TABLE authorities
(
    id   SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL UNIQUE
);

CREATE UNIQUE INDEX authority_name ON authorities (name);
```

```
-------------------------------------------------------------------------
--------------

CREATE TABLE authority_collections
(
    id         SERIAL PRIMARY KEY,
    name       VARCHAR(255) NOT NULL UNIQUE,
    expiration VARCHAR(255)
);

CREATE UNIQUE INDEX authority_collection_name ON
authority_collections (name);

-------------------------------------------------------------------------
--------------

CREATE TABLE authority_collection_authority_relations
(
    authority_collection_id INT NOT NULL,
    authority_id            INT NOT NULL,

    FOREIGN KEY (authority_collection_id) REFERENCES
authority_collections (id) ON DELETE CASCADE,
    FOREIGN KEY (authority_id) REFERENCES authorities (id) ON DELETE
CASCADE,

    CONSTRAINT
unique_combination_of_authority_collection_id_and_authority_id UNIQUE
(authority_collection_id, authority_id)
);

-------------------------------------------------------------------------
--------------

CREATE TABLE account_authority_relations
(
    id                      BIGSERIAL PRIMARY KEY,
    account_id              INT NOT NULL,
    authority_id            INT,
    authority_collection_id INT,
    expire_at               TIMESTAMP(6),

    FOREIGN KEY (account_id) REFERENCES accounts (id) ON DELETE
CASCADE,
    FOREIGN KEY (authority_id) REFERENCES authorities (id) ON DELETE
CASCADE,
    FOREIGN KEY (authority_collection_id) REFERENCES
authority_collections (id) ON DELETE CASCADE,

    CONSTRAINT
account_authority_relations_unique_combination_of_fields UNIQUE
(account_id, authority_id, authority_collection_id),
    CONSTRAINT account_authority_relations_check_one_not_null CHECK
((authority_id IS NOT NULL AND authority_collection_id IS NULL) OR

(authority_id IS NULL AND authority_collection_id IS NOT NULL))
);
```

```
CREATE INDEX account_authority_relations_account_id ON
account_authority_relations (account_id);
CREATE INDEX account_authority_relations_authority_id ON
account_authority_relations (authority_id);
CREATE INDEX account_authority_relations_authority_collection_id ON
account_authority_relations (authority_collection_id);

-------------------------------------------------------------------
--------------

CREATE TABLE blocked_account_authority_relations
(
    id                      BIGSERIAL PRIMARY KEY,
    account_id              INT           NOT NULL,
    authority_id            INT,
    authority_collection_id INT,
    expire_at               TIMESTAMP(6),
    details                 VARCHAR(1024) NOT NULL,

    FOREIGN KEY (account_id) REFERENCES accounts (id) ON DELETE
CASCADE,
    FOREIGN KEY (authority_id) REFERENCES authorities (id) ON DELETE
CASCADE,
    FOREIGN KEY (authority_collection_id) REFERENCES
authority_collections (id) ON DELETE CASCADE,

    CONSTRAINT blockers_unique_combination_of_fields UNIQUE
(account_id, authority_id, authority_collection_id),
    CONSTRAINT blockers_check_one_not_null CHECK ((authority_id IS
NOT NULL AND authority_collection_id IS NULL) OR
                                                  (authority_id IS
NULL AND authority_collection_id IS NOT NULL))
);

CREATE UNIQUE INDEX blockers_account_id ON
blocked_account_authority_relations (account_id);
CREATE UNIQUE INDEX blockers_authority_id ON
blocked_account_authority_relations (authority_id);
CREATE UNIQUE INDEX blockers_authority_collection_id ON
blocked_account_authority_relations (authority_collection_id);
```

Лістинг 3.3:

```
public interface AuthorityRepository extends
ReactiveCrudRepository<Authority, Long> {

    @Query("""
            INSERT INTO authorities (name) SELECT temp.name FROM
unnest(:authorities) as temp(name)
            WHERE NOT EXISTS(SELECT 1 FROM authorities sa WHERE
sa.name = temp.name)
            """)
```

```
    Mono<Void> insertAll(String[] authorities);

    @Query("INSERT INTO authorities (name) SELECT :authorityName AS
an WHERE NOT EXISTS ( SELECT 1 FROM authorities AS a WHERE a.name =
:authorityName )")
    Mono<Void> insertIfNotExists(@NonNull String authorityName);
}
```

Лістинг 3.3:

```
@GrpcService
@RequiredArgsConstructor
public class AuthorityGrpcService extends
ReactorAuthorityServiceGrpc.AuthorityServiceImplBase {

    private final AccountService accountService;
    private final AuthorityCacheService authorityCacheService;

    @PostConstruct
    public void init() {

initAuthority(AccountGrpc.newBuilder().setOriginator(UUID.randomUUID(
).toString()).build()).subscribe();
    }

    @Override
    public Mono<Empty> initAuthority(AccountGrpc request) {
        return
accountService.create(UUID.fromString(request.getOriginator()))
                .then(Mono.just(Empty.newBuilder().build()));
    }

    @Override
    public Mono<Empty> loadToCache(AccountGrpc request) {
        return
authorityCacheService.loadAuthorityForAccountIfAbsent(UUID.fromString
(request.getOriginator()))
                .then(Mono.just(Empty.newBuilder().build()));
    }
}
```

ДОДАТОК Б Тези конференції

ДОДАТОК В Диск із кваліфікаційною роботою