**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Сумський державний університет**

Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
Завідувач кафедри
_____ Ігор ШЕЛЕХОВ
(підпис) (Ім'я та ПРІЗВИЩЕ)
____ _____ 20__ р.

# КВАЛІФІКАЦІЙНА РОБОТА

**на здобуття освітнього ступеня бакалавр**

зі спеціальності 122 Комп'ютерні науки,
(код та назва)
освітньо-професійної програми Інформатика_____
(освітньо-професійної / освітньо-наукової) (назва програми)

на тему: Інформаційна веб-орієнтована система керування навчальними курсами_____

Здобувача групи ІН-91__ ___Алексенка Артемія Сергійовича_____
(шифр групи) (прізвище, ім'я, по батькові)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Артемій АЛЕКСЕНКО
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник кандидат ф-м. наук, доцент Борис КУЗІКОВ _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)
Консультант[1)] _____ _____
(посада, науковий ступінь, вчене звання Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2023

**ЗАВДАННЯ**

**до кваліфікаційної роботи бакалавра**

Здобувача освіти 4-го курсу, групи ІН-91 спеціальності 122 - Комп’ютерні науки, денної форми навчання Алексенка А.С.

**Тема: Інформаційна веб-орієнтована система керування навчальними курсами**

Затверджена наказом по СумДУ

№_____ від _____2023 р.

**Зміст пояснювальної записки**: 1) літературний огляд за обраною тематикою роботи; 2) методика вирішення поставлених задач; 3) проектування та моделювання веб-сайту; 4) програмна реалізація поставленого завдання; 5) висновки

Дата видачі завдання “_____”_____2023 р.

Керівник випускної роботи _____ Кузіков Б.О.

Завдання прийняв до виконання _____ Алексенко А.С.

# АНОТАЦІЯ

Дані про обсяг кваліфікаційної роботи: 66 стр., 50 рис., 2 таблиці, 11 додатків, 20 літературних джерел.

Актуальність обраної теми, обґрунтована тим, що веб-системи з навчальними курсами допомагають оптимізувати навчальний процес, зробити навчання гнучким та доступним кожному, у кого є доступ до Інтернету. Завдяки навчальним-веб-системам можна легко оновити та додати корисний матеріал, відфільтрувати актуальні теми, адже сьогодні людина має обробляти велику кількість інформації, а інформаційні веб-системи керування навчальними курсами вирішують ці завдання.

Мета роботи — розробка додатку з використанням мов програмування Java і TypeScript, фреймворків Angular і Spring Boot, бази даних PostgreSQL, хмарного сховища AWS S3 для додавання, редагування і перегляду навчальних матеріалів на платформі онлайн курсів.

Результати — було проаналізовано принципи побудови освітніх IT сайтів на прикладі існуючих аналогів. На основі проведеного аналізу була поставлена задача, обрані мови програмування та фреймворки для програмної реалізації. На базі визначеного завдання, для кожного рівня веб-системи була спроектована та змодельована структура і основні бізнес-процеси. Використовуючи отримані результати, було програмно реалізовано усі необхідні компоненти веб-орієнтованої системи курування навчальними курсами.

ІНФОРМАЦІЙНА ВЕБ-ОРІЄНТОВАНА СИСТЕМА КЕРУВАННЯ НАВЧАЛЬНИМИ КУРСАМИ, JAVA, TYPESCRIPT, SPRING BOOT, ANGULAR, POSTGRESQL, AWS S3, МІКРОСЕРВІСНА АРХІТЕКТУРА

**ЗМІСТ**

# ВСТУП

Актуальність роботи. В епоху глобальної діджиталізації при збільшенні попиту на онлайн навчання вникає необхідність створення якісних освітніх сайтів, які допоможуть великій кількості людей набувати нових знань та вдосконалювати їх.

Важко не помітити як різко зросла популярність онлайн освіти в останні роки. Насамперед такі тенденції пов’язані з кризовими явищами в країні, необхідністю швидкої перекваліфікації робітників в бізнесі та бажанням людей покращити вже набуті навички з саморозвитку.

Освітні портали допомагають студентам та викладачам знайти один одного та ефективно організувати роботу, а іноді віддалені заняття є єдиною можливою формою навчання.

Дистанційне навчання є досить зручним та гнучким, адже студент самостійно обирає час занять та рухається в своєму ритмі. Онлайн платформи забезпечують більш ефективне засвоєння знань через застосування інтерактивних інструментів, таких як презентації, відео, супровідні аудіозаписи, спільні чати тощо.

Використання онлайн платформ руйнує бар’єри в навчанні для сором’язливих та закритих людей, адже уся комунікація може відбуватися в чаті, де викладач надасть фідбек. Особливу увагу звертаємо і на той факт, що подібне навчання можуть отримати люди з обмеженими фізичними можливостями.

Хоча використання онлайн порталів має безліч переваг, однак є й недоліки. Часто перешкодою у використанні платформи виявляється незрозумілість процесу створення курсу, розміщення основних матеріалів, їх структуризація тощо.

Об’єкт дослідження – LMS (онлайн системи керування навчанням).

Предмет дослідження – веб-орієнтована система навчальних курсів.

Метою роботи є створення веб-орієнтованої системи керування онлайн курсами.

Завдання роботи:

- здійснити порівняльний аналіз сайтів з навчальними курсами;
- обрати фреймворки та мови програмування для створення навчальної веб-системи;
- дослідити особливості взаємодії компонентів веб-орієнтованої системи;
- провести моделювання та проектування веб-системи;
- розробити веб-орієнтовану систему керування навчальними курсами.

Структура роботи. У першому розділі розглянуто вже існуючі веб-системи, орієнтовані на навчання, та виділено основні переваги та недоліки кожної з них.

У другому розділі проаналізовано сучасні підходи побудови веб-орієнтованих систем, розглянута клієнтська та серверна частина додатку, способи та інструменти їх побудови.

У третьому розділі спроектовано та змодельовано функціонал та основні модулі додатку, розглянуті взаємодії між компонентами системи, їх розгортання та роль акторів при використанні додатку.

У четвертому розділі реалізовано увесь зазначений функціонал системи, виконані всі необхідні налаштування для безперебійного функціонування системи.

У п’ятому розділі виконано огляд усіх реалізованих функцій веб-орієнтованої системи керування навчальними курсами.

# 1 ЛІТЕРАТУРНИЙ ОГЛЯД ЗА ОБРАНОЮ ТЕМАТИКОЮ РОБОТИ

## 1.1 Аналіз принципів побудови освітніх IT сайтів на прикладі існуючих аналогів

Незважаючи на пандемію, економічний спад та війну в Україні, IT сектор продовжує стрімко розвиватись. IT-експорт за 5 років зріс на 5,1 млрд дол. США і зростає далі попри війну.

За прогнозом, у 2022 році він становитиме 7,6 млрд дол. США, що є втричі більшим, ніж показник 2017 року. Частка експорту IT-послуг зростає щорічно, і у 2022 на тлі війни становила 12,9% [1].

У 2022 році у галузі працюють 331 тисяча фахівців. За 5 років IT-індустрія створила більше 180 тисяч робочих місць. Для порівняння, це дорівнює кількості робочих місць у 7 групах компаній МХП, або 9 меткомбінатах АрселорМіттал, або 3 мережах магазинів АТБ. [1].

Враховуючи зазначені тенденції створення LMS (learning management system) в сфері IT набуває значної актуальності.

Тож визначмо сутність та основні функції майбутнього навчального сайту для фахівців у сфері інформаційних технологій.

Сайт з навчальними курсами – це веб платформа, що поєднує у собі набір навчальних матеріалів, спрямованих на формування знань, вмінь та навичок в обраній сфері. Основними форматами відображення інформації на сайті виступають відео, аудіо та текст.

Зазвичай освітній сайт надає такі базові функцій:

- завантаження власного курсу або курсів зі сторони адміністрації;
- перегляд навчальних матеріалів студентами;
- проходження тестових завдань, перегляд відео уроків та отримання супровідного текстового матеріалу;
- формування рейтингу курсів;

- список додаткової літератури за темами;
- комунікація з автором курсу, тощо.

Для більш детального ознайомлення з функціоналом та структурою сайтів пропонуємо розглянути декілька аналогів, що дозволить виділити основні принципи побудові остатнього сайту для ІТ-фахівців.

Для порівняння ми виділили три найбільш популярні навчальні ресурси в сфері ІТ серед студентів, зокрема Coursehunter, Amigoscode та Mate Academy.

Дані проведеного аналізу представлено в таблиці 1.1

Таблиця 1.1 – Порівняльний аналіз сайтів з ІТ курсами за базовими критеріями якісного сайту

| **Критерій** | **Coursehunter** | **Amigoscode** | **Mate Academy** |
|---|---|---|---|
| Зміст та наповнення сайту корисною інформацією | + | + | + |
| Вдала структура | - | + | + |
| Належне оформлення та дизайн | + | + | + |
| Оновлення контенту | + | + | + |
| Адреса домену або розміщення в Інтернеті | + | + | + |
| Швидкість завантаження сайту | + | - | - |
| Продумана система зв’язку з користувачем | + | + | + |
| Адаптивність | + | - | + |
| Інтеграція з соціальними мережами | + | + | + |
| Захист від неавторизованого доступу | + | + | + |
| Доступ інформації на декількох мовах | + | - | + |
| Доступ до платних послуг | + | + | + |

**Таблиця складена автором на основі сайтів ІТ курсів [2;3;4]*

Варто зазначити, що у кожного курсу є автор та ним може виступати як і сама платформа, так і незалежна приватна особа. Наприклад, на сайті «Coursehunter» користувачі можуть запропонувати свій власний курс та зв’язатись з адміністраторами за допомогою пошти. У той час на сайті «Amigoscode» публікується курс незалежної приватної особи, а на платформі «Mate Academy» опубліковано курси київської школи програмування.

Дані проведеного аналізу свідчать про те, що зазначені сайти користуються популярністю завдяки змісту та наповненню сайту корисною інформацією та періодичним оновленням контенту. Всі проаналізовані сайти з навчальними IT курсами мають належний дизайн, захист від неавторизованого доступу, лаконічний та зрозумілий домен та доступ до платних послуг.

Якість продукту, представленого на сайті та комунікація з користувачами є надзвичайно важливими для ефективної роботи бізнесу та ведення прибуткової діяльності. Саме тому зазначені інтернет-ресурси мають продуману систему зв'язку з користувачем та використовують декілька мов. Багатомовність доступна на сайтах «Coursehunter» та «Mate Academy», а інформація на «Amigoscode» представлена на міжнародній мові спілкування.

Також варто звернути уваги на недоліки кожного з навчальних ресурсів та проаналізувати їх.

На нашу думку, сайт «Coursehunter» має не достатньо якісну структуру, адже інформація про курси змішується з описом книг та джерел та іншими тегами, що є невдалим рішенням, адже сильно розсіюється увага користувача.

Ще важливим аспектом є швидкість завантаження сайту. Швидке завантаження сторінок має позитивний вплив на користувацький досвід.

Дослідження компанії Kissmetrics, яка спеціалізується на веб-аналітиці, показали наявність залежності між швидкістю роботи веб-сайта і задоволеністю його відвідувачів (потенційних клієнтів) [5]:

- 47% відвідувачів очікують, що сторінки сайта завантажаться протягом 2 секунд або менше;
- 40% відвідувачів закриють вкладку, якщо сторінка буде завантажуватися більше 3 секунд;
- кожен другий користувач відчуває роздратування, якщо час очікування перевищує 3 секунди;

- 79% клієнтів, незадоволених швидкістю роботи інтернет-магазину, не зроблять в ньому повторної покупки;
- 44% онлайн-покупців розкажуть друзям і рідним про свій негативний досвід.

Ми перевірили швидкість завантаження сайту за допомогою сервісу GTmetrix [6] та виявили, що сайти «Amigoscode» та «Mate Academy» мають відносно задовільну швидкість завантаження (рейтингова оцінка «C») в порівнянні зі швидкістю завантаження «Coursehunter» (рейтингова оцінка «B» (рис. 1.1).

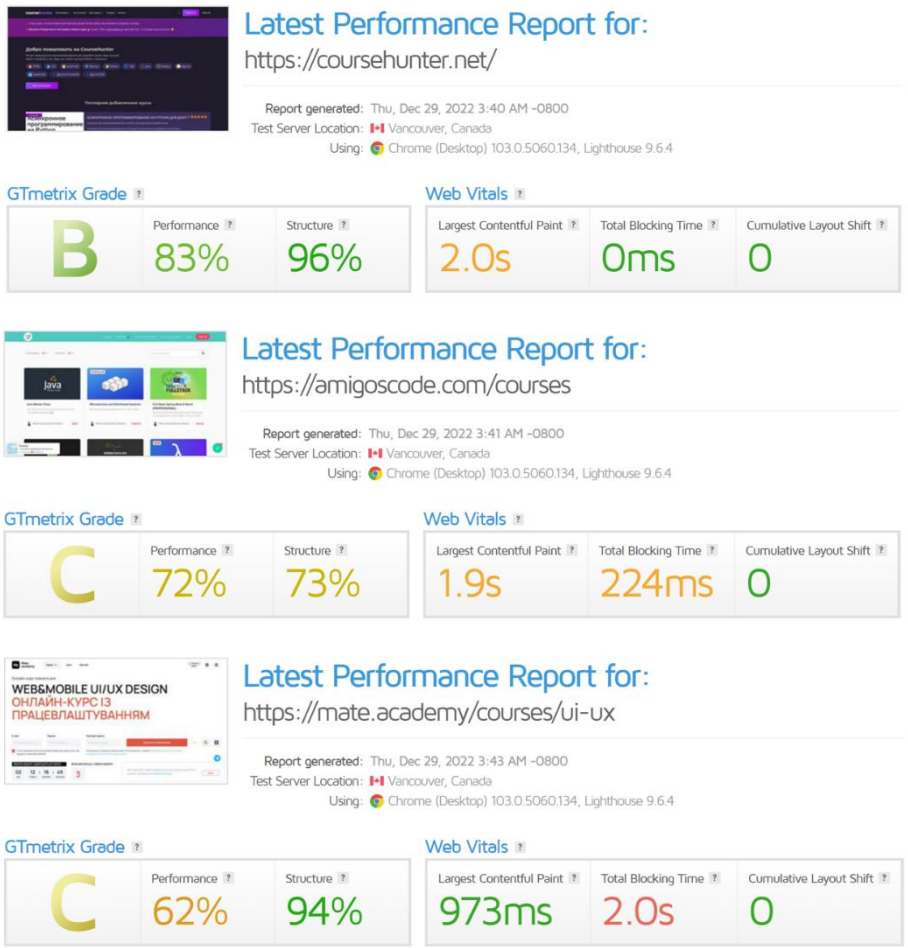


Рисунок 1.1 – Перевірка швидкості завантаження сайтів за допомогою сервісу GTmetrix

Також будь-яке онлайн навчання передбачає використання різних девайсів, зокрема смартфонів, планшетів, ноутбуків та стаціонарних комп’ютерів з різним розширенням екранів. Тому для якісного сайту важливу

роль відіграє показник адаптивності – властивості коректного відображення сайту на дисплеях різних пристроїв.

У ході роботи була перевірена адаптивність досліджуваних навчальних IT-ресурсів та виявлено, що на сайті «Amigoscode» існують деякі проблеми з адаптивністю (рис. 1.2).

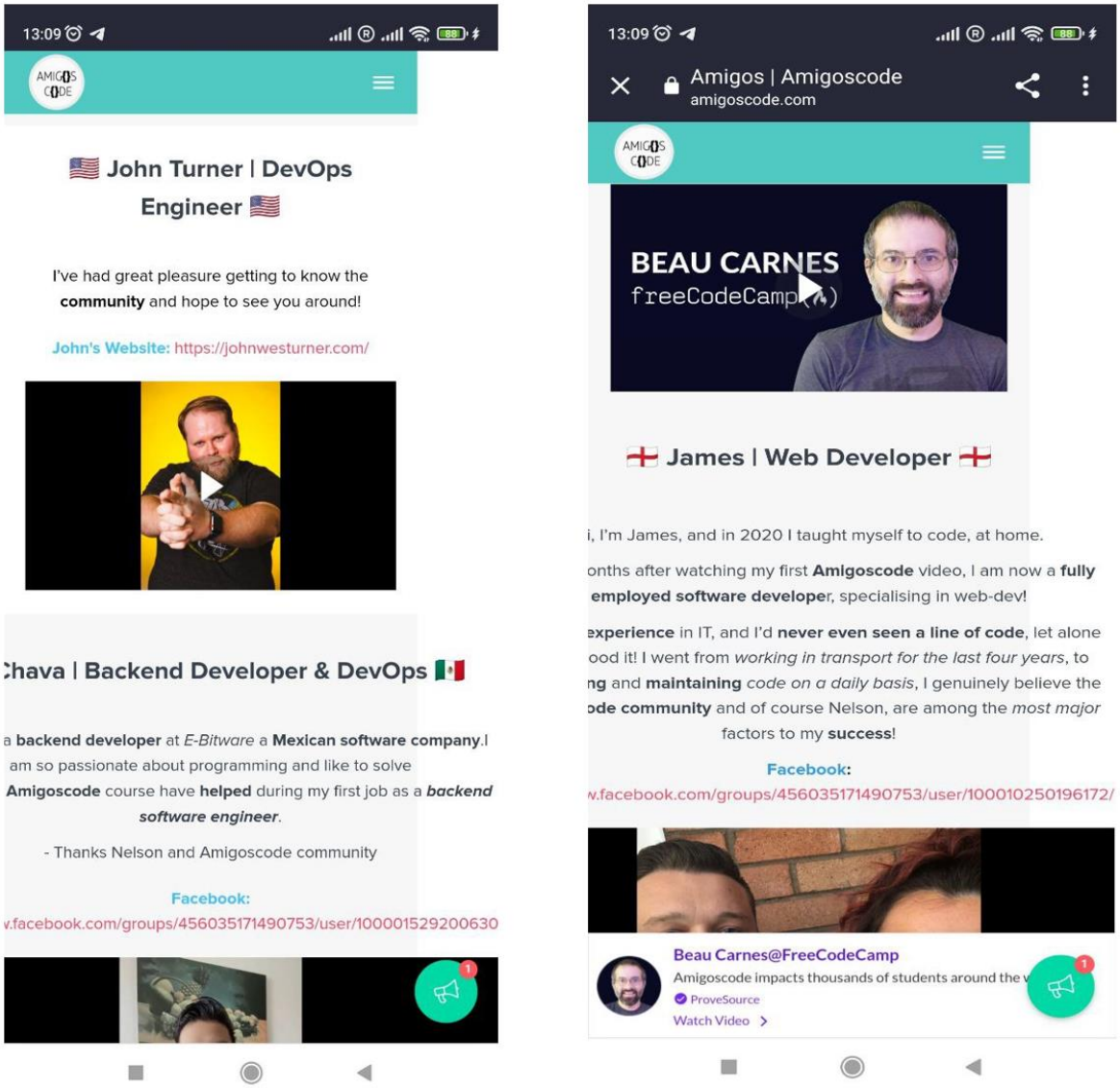


Рисунок 1.2 – Сторінки інтернет-ресурсу для IT-фахівців «Amigoscode» з порушенням адаптивності

Проблеми з адаптивністю зазначеного сайту не є критичними, але це значно спотворює комфортне сприйняття інформації користувачем та ставлення до викладеного матеріалу, зважаючи на той факт, що сайт

орієнтований на навчання студентів та фахівців в сфері інформаційних технологій, у т.ч. дизайну.

Тож ми порівняли аналоги майбутнього сайту для навчання ІТ-фахівців та виявили, що головними аспектами успішної реалізації проекту є вдала структура, періодичне та регулярне наповнення сайту актуальною та корисною інформацією та дизайн, адаптивність та швидкість завантаження сайту.

Особливу увагу слід звернути на комунікацію між адміністрацією сайту та користувачами, викладачем курсу та студентами.

Освітній сайт з курсами для ІТ-фахівців повинен забезпечити захист від неавторизованого доступу та захистити персональні дані користувачів.

Лаконічний та зрозумілий домен також відіграє вирішальну роль у просуванні сайту та створенню довіри з боку користувачів.

Гарним бонусом в реалізації проекту може стати багатомовність сайту, використання інтерактивних інструментів для додаткової комунікації тощо.

## 1.2 Постановка задачі

Отже, для виконання кваліфікаційної роботи необхідно створити сайт з ІТ-курсами, використовуючи технології сучасних фреймворків та відповідних мов програмування з метою реалізації backend та frontend частин веб-системи. Для цього необхідно виконати наступні завдання:

1. Проаналізувати можливості технологій сучасних фреймворків та мов програмування, розглянути їх модулі, бібліотеки;

2. Розглянути додаткові інструменти, які будуть використовуватися в інтеграції з фреймворками для отримання всіх необхідних можливостей для написання веб-ресурсу;

3. Створити архітектуру та дизайн додатка, для реалізації наступного функціоналу:

- аутентифікація та авторизація;
- створення користувачів із різними ролями;
- можливість додавати, видаляти, та змінювати навчальні матеріали;
- фільтрація навчальних матеріалів за категоріями тощо;
- можливість пошуку навчального матеріалу по ключовим словам.

4. Розробити програмну частину веб-орієнтованої системи керування навчальними курсами для реалізації вищезазначених вимог;

# 2 МЕТОДИКА ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ

## 2.1 Клієнтська частина веб-системи

На сьогоднішній день, зайшовши майже на будь-яки й сучасний сайт, що був створений нещодавно а/або все ще продовжує розвиватися, ми побачимо використання динамічних сторінок замість вже застарілого підходу зі статичними сторінками. Це означає, що все менше нових веб-проектів використовує підхід, коли сервер просто відправляє користувачу статичну HTML сторінку. При цьому далеко не кожна зміна внесена користувачем викликає перезавантаження сторінки браузера. Рішенням даної проблеми є такі підходи у розробці як реактивність та асинхронність, котрі використовуються майже кожним сучасним фреймворком. Особливості одного з таких фреймворків, а саме Angular, ми і розглянемо далі.

Angular – це надзвичайно потужний frontend фреймворк, побудований на мові програмування TypeScript – версії JavaScript. Платформа Angular включає [7]:

- базований на компонентах фреймворк для створення веб-додатків з можливістю масштабування;
- колекція добре інтегрованих бібліотек, які охоплюють широкий спектр функцій, включаючи маршрутизацію, керування формами, зв’язок клієнт-сервер, тощо;
- набір інструментів розробника, які допоможуть вам розробляти, тестувати й оновлювати код.

Одним із основних плюсів Angular це можливість використання платформи як і для власних pet-проектів, так і для великих програм корпоративного рівня. Найкращим є те, що Angular має вичерпну і прозору документацію, інструкції і приклади для навчальних проектів, величезну спільноту розробників, що складає понад 1,7 мільйона розробників, авторів бібліотек та різного контенту.

Отже використання Angular для написання клієнтської частини буде доцільно обраним інструментом, якого буде цілком достатньо аби реалізувати увесь необхідний функціонал.

## 2.2 Серверна частина веб-системи

Звісно ж кожен «клієнт» у будь-якому веб-додатку не може повноцінно функціонувати без серверної частини. Серверною частиною прийнято вважати ту частину додатку, що приймає певні запити від користувача, оброблює їх та формує відповідь. Найчастіше цей процес супроводжується великою кількістю додаткових операцій на стороні сервера, таких як виклики сторонніх сервісів, генерація певних подій, CRUD операції з базою даних або зовнішнім хмарним сховищем, ініціація процедур з періодичним виконанням, тощо.

Відомо, що у реальних бізнес-проектах застосунки не пишуться з використанням лише мови програмування та набором базових бібліотек. Хоч деякі з них і можуть вирішувати типові для розробників задачі, але не дивлячись на це нам би все одно часто довелося «винаходити велосипед» аби щось додати або поєднати це все у єдину екосистему, не кажучи вже про і так складну конфігурацію додатку. Саме тому у реальних проектах використовуються фреймворки, вони дають змогу «з коробки» отримати каркас бажаного додатку, поєднати усі бібліотеки і інструменти в єдину екосистему, правильним чином налаштувавши їх конфігурацію. Одним із таких фреймворків є Spring Boot.

Spring Boot – це фреймворк-каркас побудований на базі фреймворка Spring, тобто він має усі можливості Spring з певними доповненнями. Тому спершу пропонуємо розглянути можливості фреймворка Spring, а потім додаткові можливості Spring Boot.

Spring Framework пропонує функцію *ін'єкції залежностей* , яка дозволяє об'єктам визначати власні залежності, які контейнер Spring пізніше

впроваджує в них. Це дає змогу розробникам створювати модульні додатки, що складаються з слабозв'язаних компонентів, які ідеально підходять для мікросервісів і розподілених мережевих додатків [8].

Spring Framework також пропонує вбудовану підтримку типових завдань, які має виконувати програма, таких як зв'язування даних, перетворення типів, перевірка, обробка винятків, керування ресурсами та подіями, інтернаціоналізація тощо. Він інтегрується з різними технологіями Java EE, такими як RMI (Remote Method Invocation), AMQP (Advanced Message Queuing Protocol), Java Web Services та іншими. Підсумовуючи, Spring Framework надає розробникам усі інструменти та функції, необхідні для створення слабозв'язаних міжплатформних програм Java EE, які працюють у будь-якому середовищі [8].

Spring Boot — це інструмент, який робить розробку веб-додатків і мікросервісів за допомогою Spring Framework швидшою та простішою завдяки трьом основним можливостям [8]:

- автоконфігурація;
- уважний підхід до налаштування;
- можливість створення автономних додатків.

Ці функції працюють разом, щоб надати інструмент, який дозволяє налаштувати програму на основі Spring з мінімальною конфігурацією та налаштуванням [8].

Тож використання фреймворка Spring Boot у нашій роботі значно полегшить конфігурацію додатка і зменшить написання шаблонного коду, його можливостей повністю вистачить для реалізації серверної частини застосунку.

# 3 ПРОЕКТУВАННЯ ТА МОДЕЛЮВАННЯ ВЕБ-СИСТЕМИ

## 3.1 Моделювання функціональних бізнес-процесів у нотації IDEF0

Перш за все потрібно розуміти який функціонал матиме наш веб-додаток та які проблеми користувача будуть вирішуватись. Даний етап проектування є основним та надважливим, адже на базі цього будуть реалізовані необхідні методи та структури. Для цього ми використаємо нотацію IDEF0.

IDEF (Integrated Definition) — це методологія моделювання процесів, яка використовується для впровадження систем та програмного забезпечення, шляхом графічного відображення основного інструментарію системи. Ці методи використовуються для функціонального моделювання даних, симуляції, об’єктно-орієнтованого аналізу та отримання знань. Він був розроблений як стандартний метод документування та аналізу бізнес-процесів. Зараз ця методологія використовується як регламентований підхід до аналізу підприємства, охоплення моделей процесів «як є» та для моделювання діяльності в бізнес-групі. IDEF0 графічно представляє собою прямокутник зі стрілками, де прямокутник – функція, яку виконує система, а стрілки – певні активності, вони бувають чотирьох типів:

- вхідні – ставлять певне завдання;
- вихідні – відображають результат функції;
- контроль (згори донизу) – механізми управління (положення, інструкції та ін.);
- механізми (знизу нагору) – що використовується для того, щоб зробити необхідну роботу [9].

У системі, яку ми розробляємо, глобальних функцій буде дві: додавання і відображення навчальних матеріалів, котрі представлені на рисунках 3.1, 3.2 відповідно.


Рисунок 3.1 – IDEF0 діаграма функції «Додавання навчальних матеріалів»

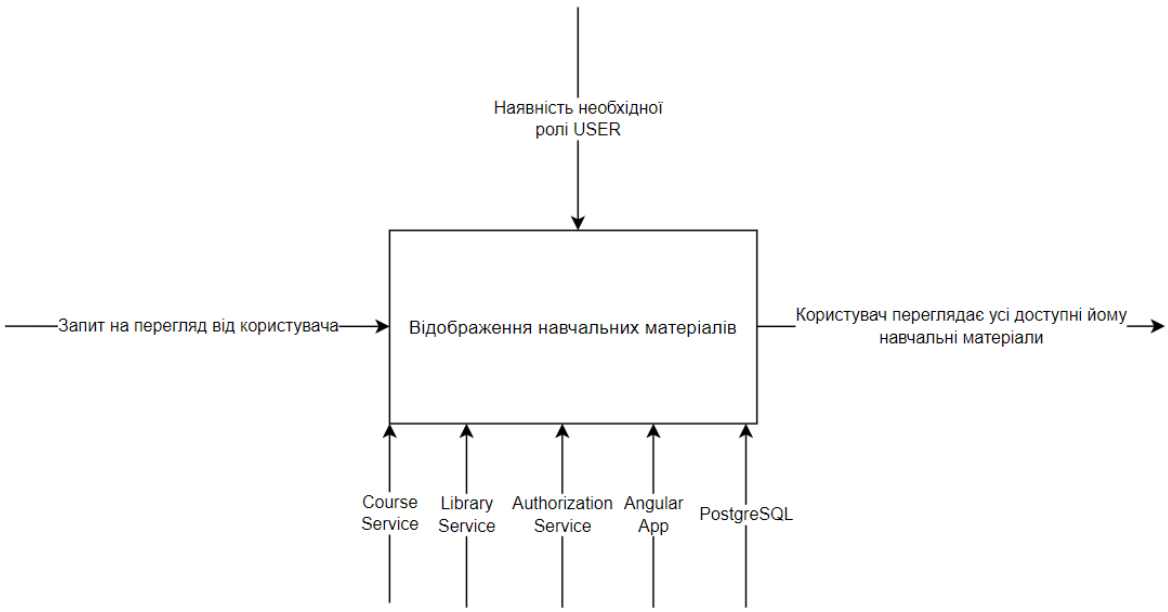


Рисунок 3.2 – IDEF0 діаграма функції «Відображення навчальних матеріалів»

## 3.2 Проектування веб-орієнтованої системи керування курсами

### 3.2.1 Діаграма прецедентів

В UML (Unified Modeling Language) діаграми прецедентів (діаграми варіантів використання) моделюють поведінку системи та допомагають охопити її вимоги. Цей підхід описує функції високого рівня та область застосування системи. Ці діаграми також ідентифікують взаємодію між системою та її акторами. Діаграма варіантів використання описує, що робить система та як актори з нею взаємодіють, але не описують те, як додаток працює всередині.

Діаграми варіантів використання ілюструють і визначають контекст і вимоги всієї системи або важливих її частин. Ви можете змоделювати складну логіку за допомогою однієї діаграми варіантів використання або створити багато діаграм для моделювання компонентів системи. Зазвичай дана візуалізація створюється на ранніх етапах проекту та посилаєтеся на них протягом усього процесу розробки.

Діаграми прецедентів корисні в таких ситуаціях:

- перед стартом проекту, для розуміння набору бізнес-процесів та їх учасників;
- в процесі розробки, для візуалізації вимог та побажань до функціоналу системи;
- на етапах аналізу та проектування, для розуміння набору класів, котрі мають бути в системі [10].

У контексті нашої системи будуть присутні два актори у ролі споживачів: «Адміністратор» і звичайний «Користувач», а також два актори у ролі постачальників: «PostgreSQL» і «AWS S3». Їх взаємодія представлена на рисунку 3.3.

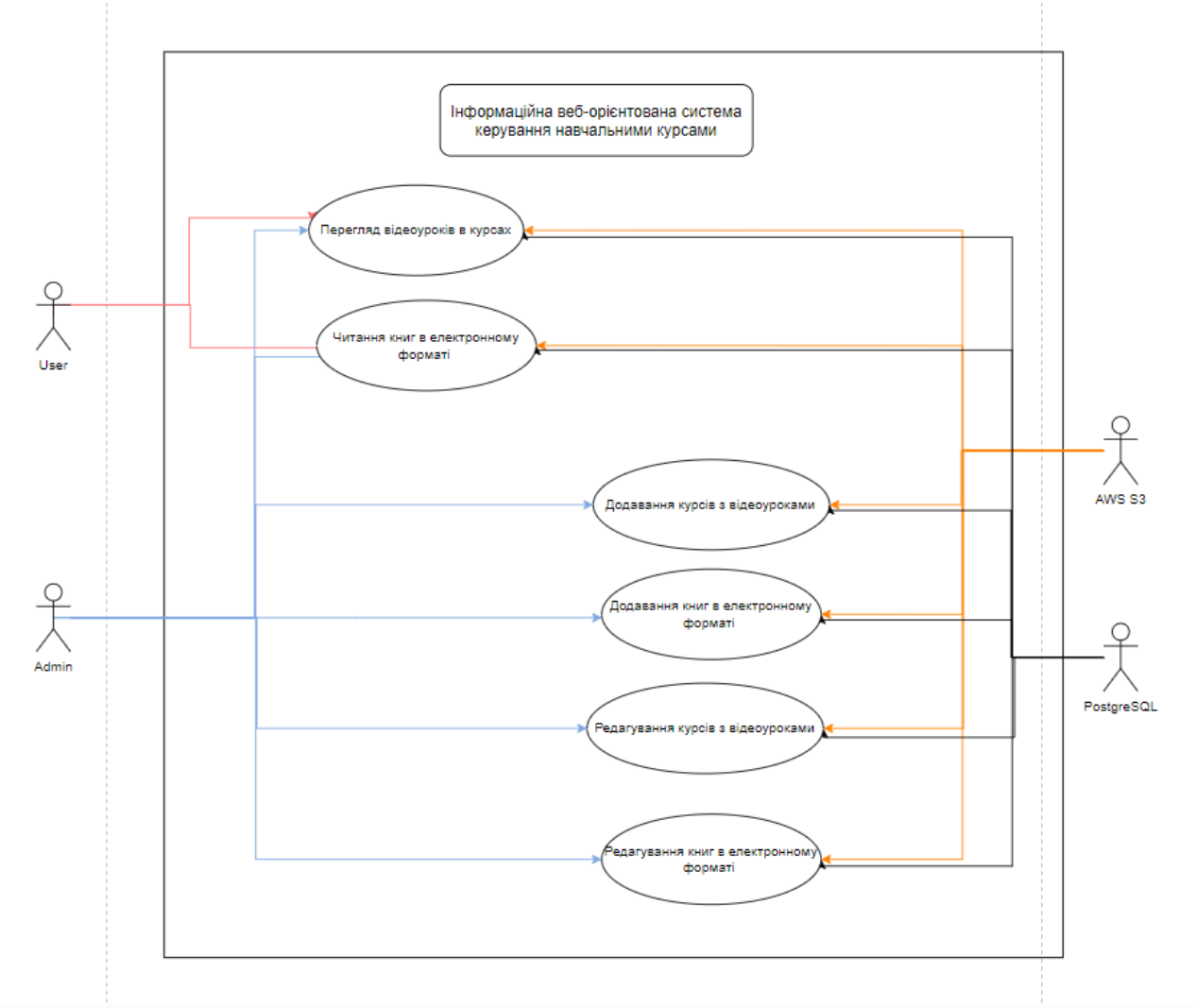


Рисунок 3.3 – Діаграма прецедентів

### 3.2.2 Діаграма розгортання

З кожною діаграмою ми будемо збільшувати рівень деталізації і зменшувати рівень абстракції аби якомога детальніше описати нашу інформаційну систему. До цього ми описали діаграму прецедентів у котрій наш додаток був представлений у вигляді єдиного великого компонента «Інформаційна веб-орієнтована система керування навчальними курсами». А зараз ми деталізуємо його до рівня сервісів, бази даних, способу їх комунікації та розгортання.

Дана UML діаграма використовується для зображення зв’язків між програмним і апаратним компонентами системи, а також для того, щоб

показати, як додаток розгортається у фізичному чи віртуальному середовищі. Це допомагає візуалізувати розподіл програмних компонентів між різними апаратними вузлами, такими як сервери, клієнти та бази даних.

Окрім візуалізації фізичних компонентів системи, діаграми розгортання також можуть відображати розподіл і встановлення програмних служб у мережі. Вони надають спосіб візуалізації зв’язків між різними службами та вузлами, на яких вони розгорнуті, наприклад веб-серверами і базами даних. Використовуючи діаграми розгортання для моделювання сервісу, ми можемо краще зрозуміти, як наші сервіси розподіляються та як вони взаємодіють один з одним, а також визначати потенційні проблеми та оптимізувати стратегії комунікації [11].

При розробці веб-орієнтованої системи було виділено 3 основних backend сервіси:

- Course Service – його задачею є управління курсами (створення нових, редагування вже існуючих та видалення непотрібних;
- Library Service – задача аналогічна до Course Service-у за тим винятком, що даний сервіс буде управляти книгами;
- Authorization Service – сервіс, котрий забезпечує можливість реєстрації і авторизації в додатку, перевіряє токени сторонніх сервісів.

В якості ж frontend сервісу буди виступати один Angular додаток, задачею якого є взаємодія з клієнтом через його браузер і комунікація з сервісами backand-у та storage-у.

У якості storage-сервісу був обраний хмарний сервіс – S3 компанії «Amazon».

У якості СКБД була обрана PostgreSQL.

Після розподілення інформаційної системи на відповідні сервіси необхідно спроектувати їх комунікацію і відповідній інфраструктурі розгортання. Для використаємо контейнеризацію за допомогою Docker. Для кожного проекту буде створено Dockerfile для побудови «знімку» додатку.

Після виділення усіх необхідних вузлів можемо візуалізувати отриманий результат за допомогою діаграми розгортання. На рисунку 3.4 зображено отриманий результат.

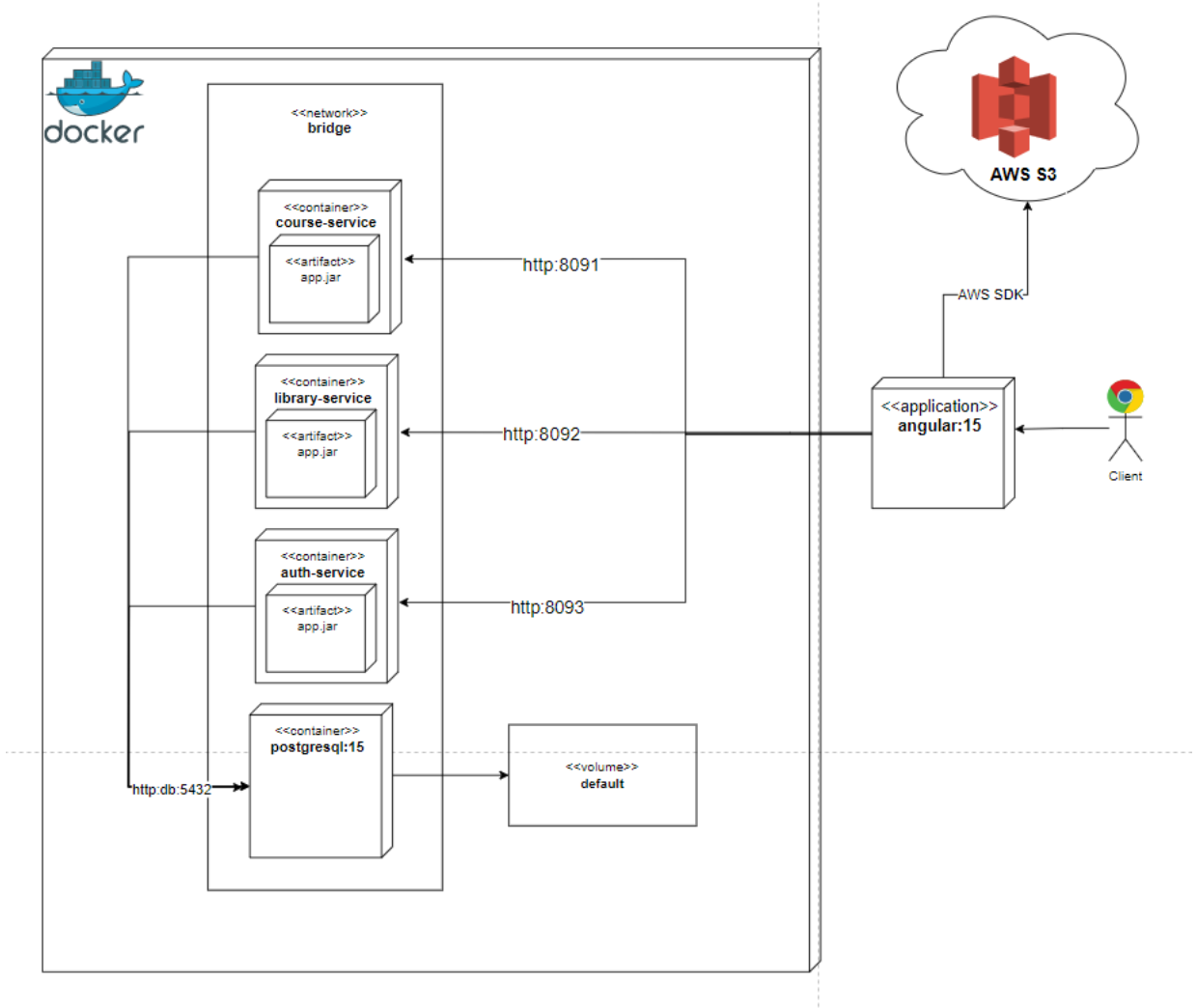


Рисунок 3.4 – Діаграма розгортання

### 3.2.3 Діаграма класів

На даному етапі ми визначили функціональні вимоги до нашого додатку, обробили сценарії експлуатації веб-ресурсу, а також розбили систему на сервіси і організували між ними зв’язок. Тепер нам необхідно заглибитись в кожен із сервісів

Діаграма класів зображує статичний вигляд програми. Він представляє типи об'єктів, що знаходяться в системі, і зв'язки між ними. Клас складається

зі своїх об’єктів, а також може успадковуватись від інших класів. Діаграма класів використовується для візуалізації, опису, документування різних аспектів системи, а також для створення виконуваного програмного коду.

Дана візуалізація найчастіше використовуються в моделюванні об'єктно-орієнтованих систем, оскільки вони є єдиними діаграмами UML, які можуть бути відображені безпосередньо за допомогою об'єктно-орієнтованих мов.

Діаграми класів для кожного сервісу системи можна побачити на рисунках [3.5 - 3.7].

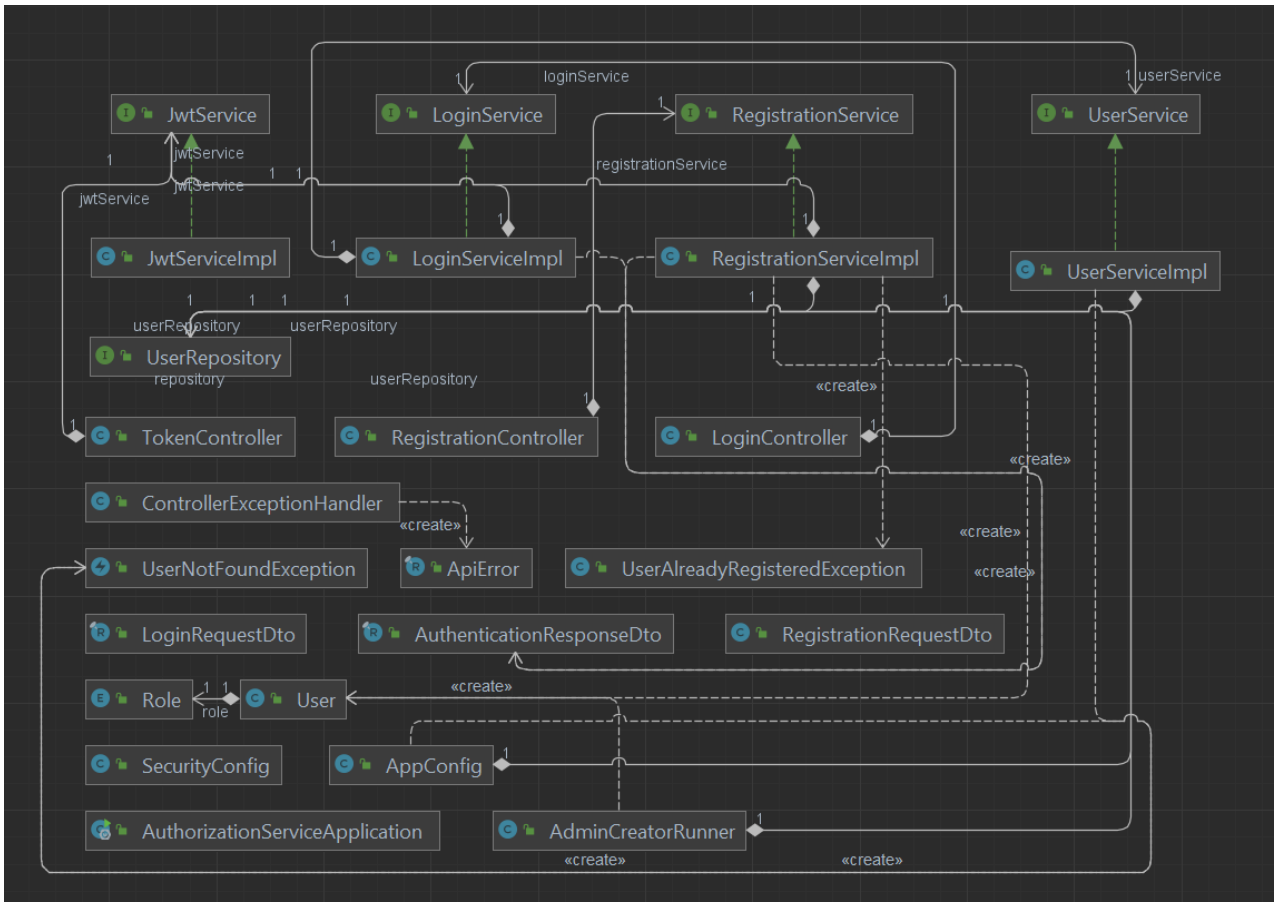


Рисунок 3.5 – Діаграма класів для Authorization Service

Діаграма класів показує класи, функції та зв’язки, щоб дати загальне уявлення про програмну систему та допомогти в розробці програмного забезпечення [12].

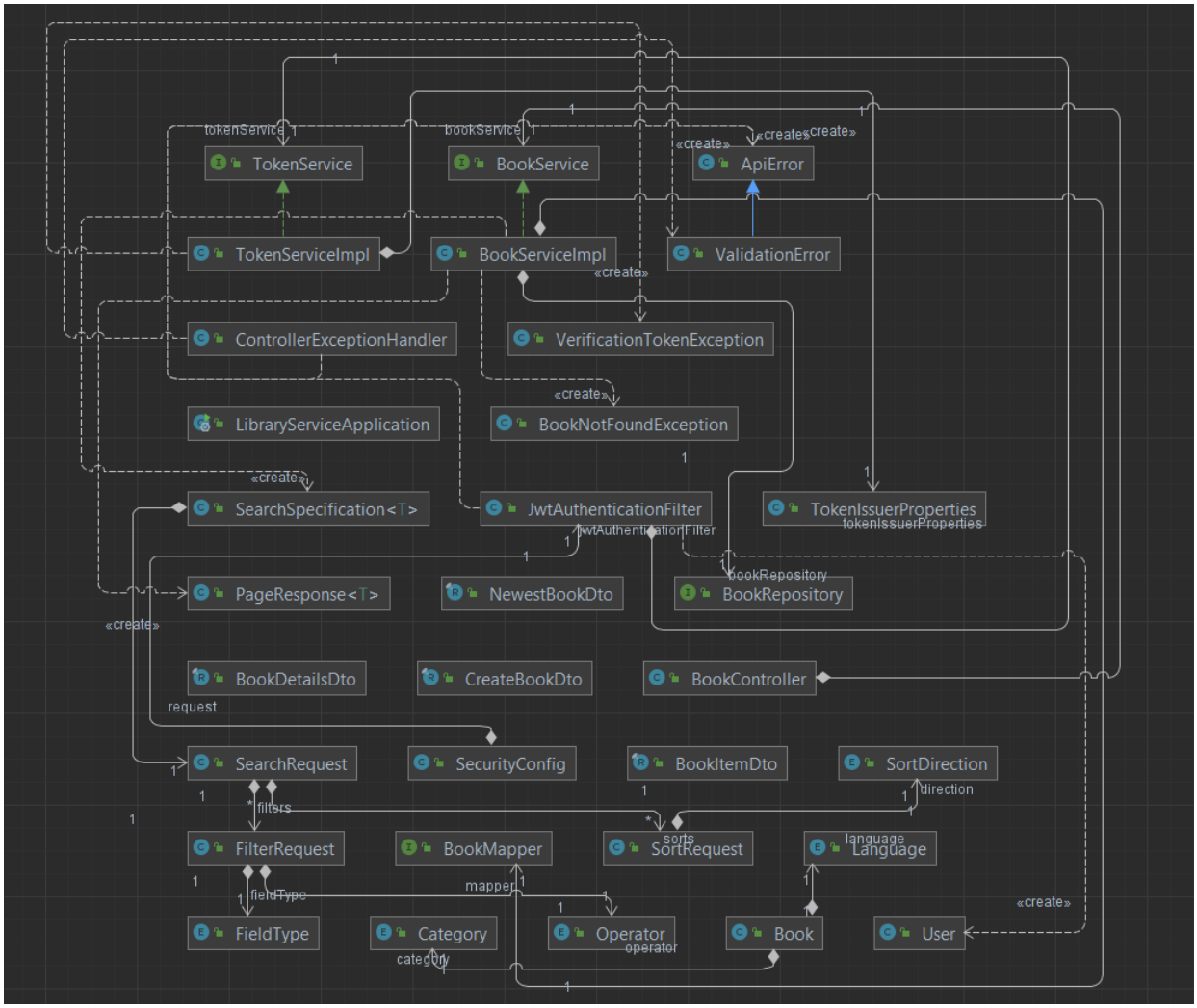


Рисунок 3.6 – Діаграма класів для Library Service

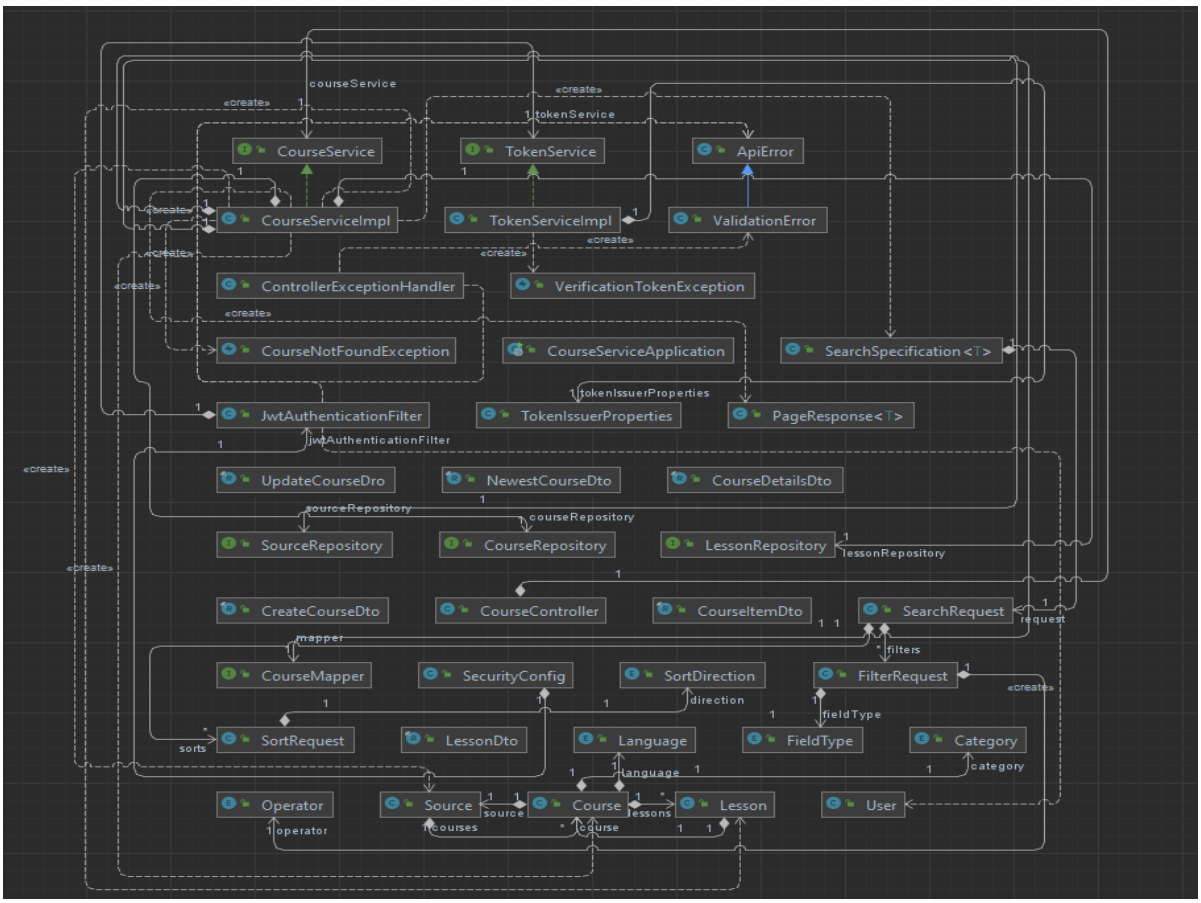


Рисунок 3.7 – Діаграма класів для Course Service

### 3.2.4 Діаграма послідовності

Діаграма послідовності — це один із багатьох типів діаграм взаємодії системи, які використовуються UML для візуального представлення взаємодії між об’єктами, які знаходяться у системі. Зокрема, діаграми послідовності забезпечують уявлення про порядок, у якому відбуваються ці взаємодії, через зображення окремих об’єктів і повідомлень між ними. За допомогою належної нотації можна використовувати діаграми послідовності на кількох етапах процесу розробки, щоб визначити час і порядок взаємодії.

Існує багато компонентів діаграми послідовності, яка складається з різних стандартизованих розмірів і позначень. Двома основними вимірами осі є об’єкти та час. Об’єкти, які проходять поперек горизонтальної осі, показують різні елементи, навколо яких обертається взаємодія. Вертикальна вісь часу

вказує на послідовність взаємодій, зображених на діаграмі, однак зауважте, що вона не вказує на тривалість цих взаємодій [13].

На даному етапі ми розглянемо дві основних послідовності, які існують в нашому додатку: створення та перегляд курсів, але спочатку розглянемо процес авторизації і аутентифікації користувачів. Він присутній в будь-якій взаємодії користувача з додатком, а тому має знаходитись в кожній описаній послідовності. Аби не перевантажувати кожну діаграму авторизація і аутентифікація була винесена в окрему візуалізацію на рисунку 3.8.

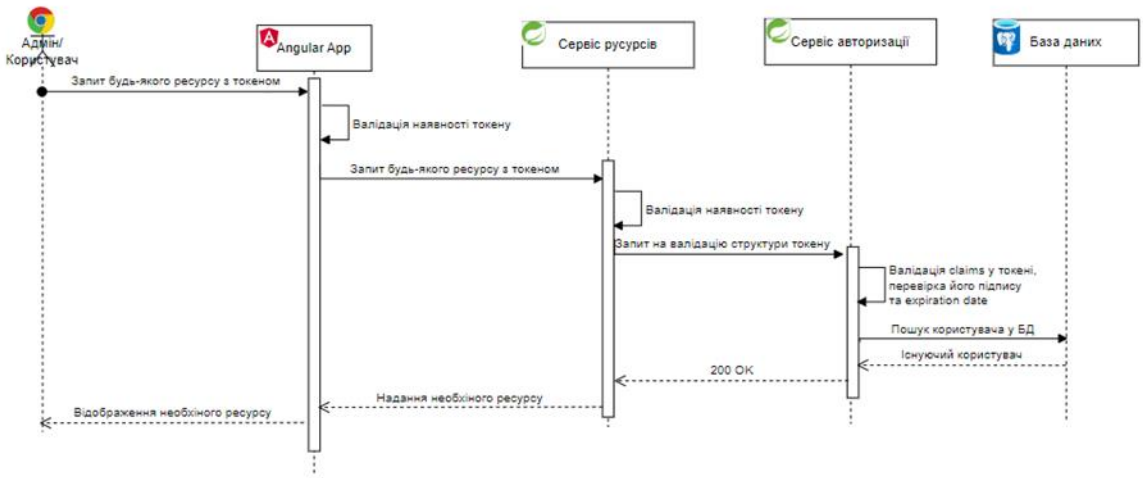


Рисунок 3.8 – Діаграма послідовності для процесу авторизації і аутентифікації

Отже, процес розпочинається, коли юзер намагається перейти на будь-яку сторінку нашого додатку. У цей момент клієнт, а саме його браузер, ініціює запит на frontend частину нашої веб-системи. Першим кроком відбувається перевірка наявності токена, якщо його немає – система перенаправить користувача на сторінку входу. У випадку, коли токен присутній – буде виконаний запит необхідної інформації на backend сервіс. В кожному ресурсному сервісі буде описаний фільтр, який перехоплює HTTP запит і перевіряє наявність токену в Authorization заголовку, при його присутності відбувається більш детальна перевірка за допомогою сервісу авторизації. Він в свою чергу декодує токен і перевіряє усі три його частини:

header, payload, signature. Кожен з них несе необхідну для сервісу інформацію, аби однозначно ідентифікувати користувача. Останнім кроком є перевірка існування людини в базі даних.

Тож перейдемо до основних послідовностей і почнемо зі створення курсу з додаванням відеоуроків. Графічна візуалізація показана на рисунку 3.9. Процес розпочинається з адміністратора, котрий, як ми вже пам’ятаємо, успішно пройшов попередній крок аутентифікації і заповнив форму додавання курсу. Першим цей запит обробляє frontend частина нашого додатку і проводить перевірку заповнених даних. Якщо щось було попущено – система повідомить і призупинить подальший процес до моменту дозаповнення всіх даних. Якщо валідація успішно пройдена angular сервіс починає завантаження логотипа і відеоуроків в хмарне сховище. Далі відбувається запит на backend сервіс, що відповідає за роботу з курсами. Там відбувається повторна валідація даних. Варто відмітити, що подвійна перевірка на frontend та backend частинах є best-practice розробки веб-систем. Адже клієнт у сервісу може змінюватися і при відсутності на новому клієнті перевірки, дані можуть стати неконсистентними, що в подальшому призведе до некоректної роботи додатку, або повного виходу його з ладу. Наступним кроком є збереження курсу в базу даних.

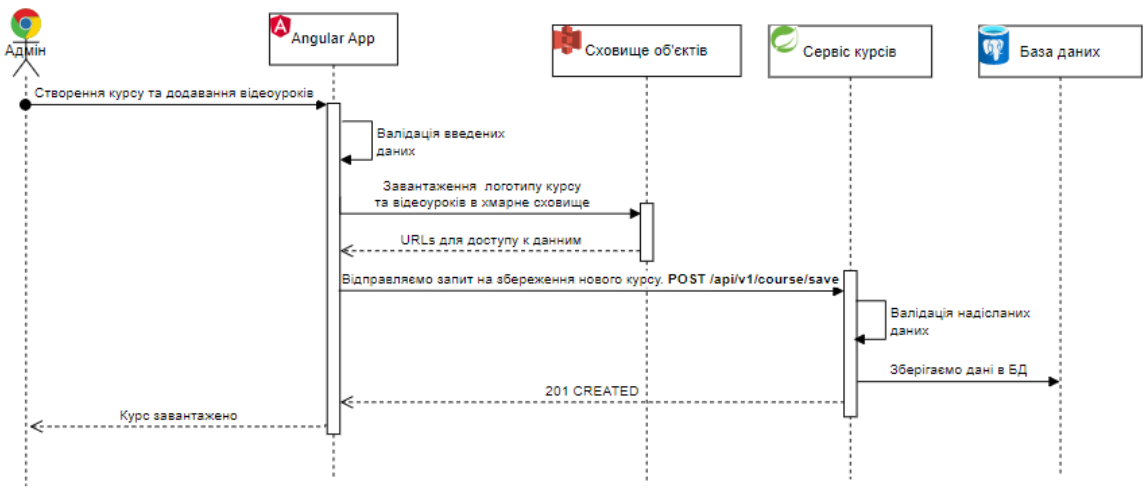


Рисунок 3.9 – Діаграма послідовності для процесу створення курсу

Другою основною послідовністю є перегляд існуючого курсу користувачем. Графічна візуалізація показана на рисунку 3.10. Процес розпочинається з користувача, котрий перейшов на відповідну сторінку для перегляду. Першим запит обробляє frontend частина нашого додатку, вона визначає ідентифікатор курсу деталі якого потрібно відобразити й робить запит на backend. Сервіс курсів перевіряє наявність курсу в базі даних. У разі його відсутності буду повернути помилка. Варто зазначити, що всі помилки в кожному сервісі мають однакову структуру відповіді. Це допомагає уніфікувати їх відображення користувачу в процесі експлуатації системи. Коли курс був знайдений backend повертає усю необхідну інформацію, включно з набором URL адрес для завантаження логотипу і відеоуроків з хмарного сховища, що і відбувається наступним кроком. Усе це дає змогу відобразити користувачеві повну інформацію про курс.

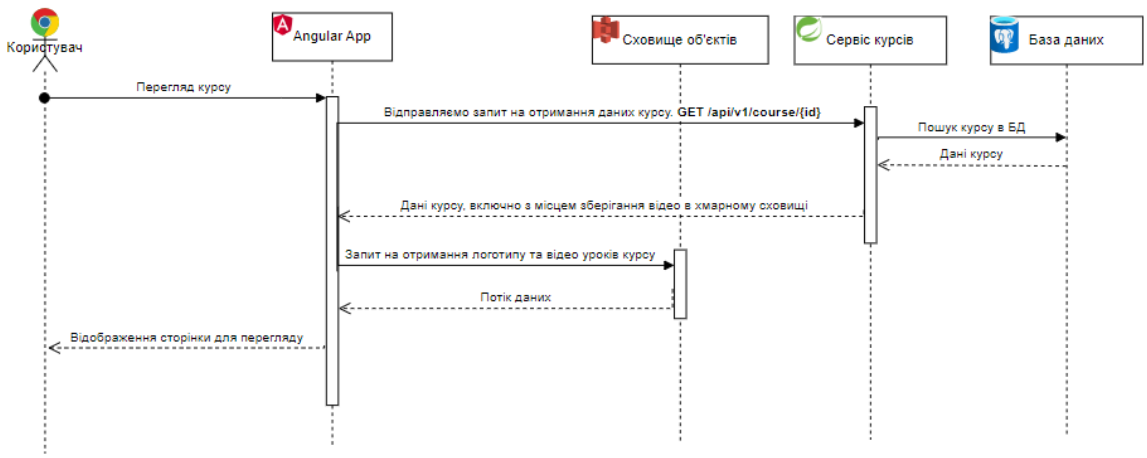


Рисунок 3.10 – Діаграма послідовності для процесу перегляду курсу

### 3.3 Проектування схеми бази даних

Проектування схеми бази даних є критично важливим кроком у розробці будь-якого програмного додатку, який передбачає використання сховища даних. Добре розроблена схема э надважливою для ефективного зберігання,

пошуку та керування даними. Будь-які недоліки в конструкції схеми можуть мати серйозні наслідки для продуктивності програми, її масштабованості та зручності обслуговування.

Перш ніж приступати до розробки нашого додатку, важливо витратити достатньо часу на розробку схеми бази даних. Це передбачає ідентифікацію сутностей і зв’язків, які необхідно змоделювати, визначення таблиць і їхніх атрибутів, а також встановлення обмежень і правил, які керують даними.

Застосовуючи якісний підхід до розробки схеми, ми уникаємо багатьох типових проблем в подальшому і переконуємося, що отримана база даних є надійною, масштабованою та простою в обслуговуванні. Основними перевагами в якісно спроектованій схемі є:

- ефективне зберігання та пошук даних: добре розроблена схема гарантує, що дані зберігаються ефективним і організованим способом, полегшуючи їх пошук і маніпулювання.
- покращена продуктивність: правильно розроблена схема може покращити продуктивність програми, зменшивши затримку та покращивши час відповіді.
- більша масштабованість: масштабована схема може рости та розвиватися разом із розвитком програми, гарантуючи, що вона зможе й надалі обробляти зростаючі обсяги даних і трафіку.
- просте технічне обслуговування: добре розроблену схему легше підтримувати та змінювати, зменшуючи час і зусилля, необхідні для внесення змін до бази даних з часом.

У нашій системі ми будемо використовувати СКБД (систему керування базами даних) PostgreSQL. Дана система є оптимальним вибором, зважаючи на ряд причин.

По-перше, PostgreSQL — це СКБД з відкритим кодом, яка широко використовується та високо цінується за свою надійність, масштабованість і

безпеку. Перевагами є потужні функції, а також підтримка розширених типів даних, індексування та оптимізацію запитів.

По-друге, PostgreSQL легко налаштовується та розширюється, що дозволяє адаптувати базу даних до наших конкретних потреб.

Загалом PostgreSQL є чудовим вибором для тих, кому потрібна надійна та масштабована система керування базами даних, яка може обробляти великі обсяги даних і складні запити. Її потужні функції, гнучкість і простота використання роблять PostgreSQL ідеальним вибором для широкого діапазону додатків, від невеликих проектів до великих систем корпоративного рівня [14].

Отже, наша веб-орієнтована система побудована на трьох backend сервісах, для кожного з них буде розроблена окрема незалежна схема даних, яка відповідає сутностям і даним якими оперує сервіс.

Зоною відповідальності для Library Service є керування літературою, а основною сутністю є книга. Тому для зберігання необхідної інформації була сформована таблиця «book» – рисунок 3.11. В якості первинного ключа слугує поле «id» у форматі UUID. Даний підхід дає нам впевненість у відсутності дублікатів. Основними полями для опису книги слугують наступні поля: заголовок, автор, опис, URL посилання на обкладинку книги, URL посилання на PDF файл, мова, категорія.

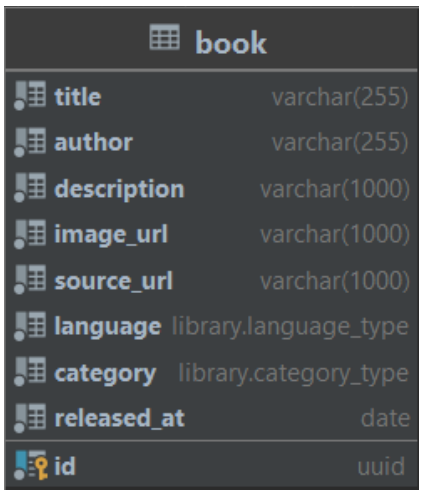


Рисунок 3.11 – Схема бази даних для Library Service

Зоною відповідальності для Course Service є керування курсами, а основною сутністю є курс у котрого є набір уроків і певне джерело (автор, команда або компанія, що створила курс). Тому для зберігання необхідної інформації були сформовані таблиці «course», «lesson», «source» – рисунок 3.12. Оскільки один курс може мати декілька відеоуроків, ми використали відношення таблиць один до багатьох. А також, кожного курсу є певне джерело. В свою чергу джерело може створювати декілька курсів, тому тут також було використано відношення один до багатьох.

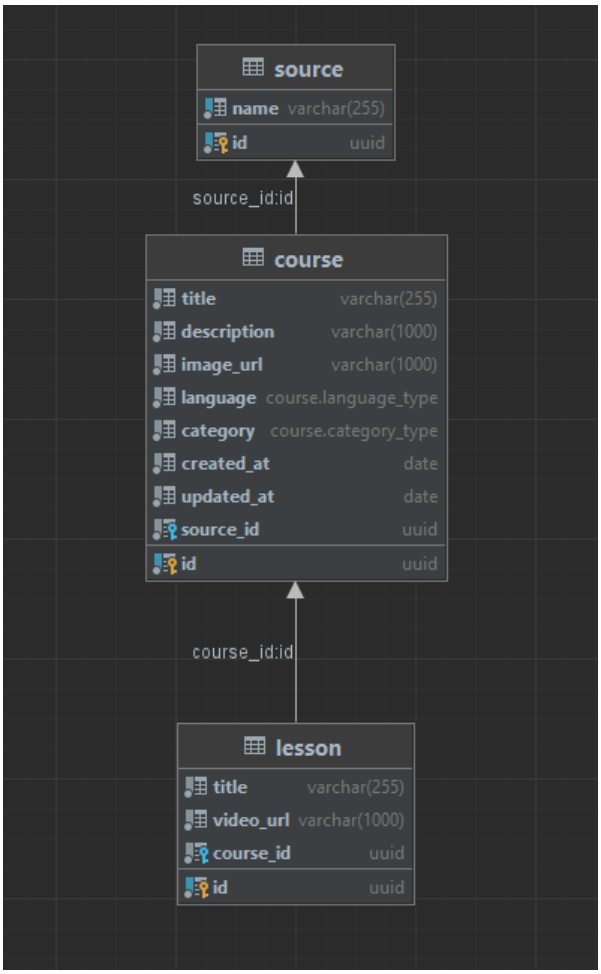


Рисунок 3.12 – Схема бази даних для Course Service

Зоною відповідальності для Authorization Service є керування користувачами, їх реєстрація і подальша аутентифікація. Основною сутністю є користувач у котрого є електронна адреса, пароль і роль для авторизації і

розмежування доступу до конкретних ресурсів, а також поля, що його характеризують: ім’я та прізвище. Тому для зберігання необхідної інформації була сформована таблиця «user» – рисунок 3.13.

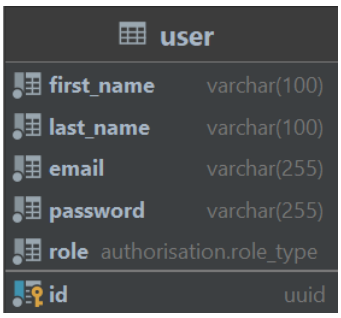


Рисунок 3.13 – Схема бази даних для Course Service

Варто окремо виділити дві таблиці котрі будуть присутні у схемах для кожного сервісу, тому і винесені окремо. Насправді вони не є частиною логіки, яка закладена у нашу систему. Ці таблиці є згенерованими бібліотекою для міграції – «Liquibase» і необхідні для коректного функціонування і контролю за міграціями.  Тому для зберігання необхідної інформації була сформовані таблиці «databasechangelog» і «databasechangeloglock» – рисунок 3.14.

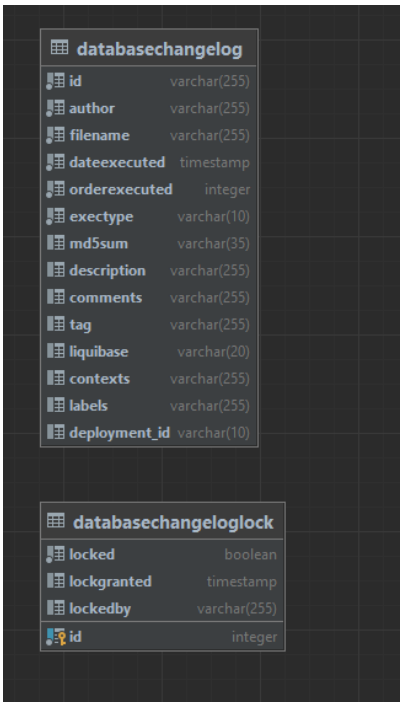


Рисунок 3.14 – Схема допоміжних таблиць для бібліотеки Liquibase

# 4 РОЗРОБКА ВЕБ-СИСТЕМИ

## 4.1 Налаштування сховища даних

Перед програмною реалізацією, необхідно створити і сконфігурувати хмарне сховище, яке буде використовуватися для зберігання відеоуроків, книг та логотипів. Процеси реєстрації, ввімкнення подвійної аутентифікації, підв'язка карти будуть пропущені, адже прямого впливу на розробку не мають.

З метою конфігурації хмарного сховища, в роботі була використана покрокова інструкція, вказана на офіційному сайті компанії Amazon [19].

Після успішно пройдених етапів входу, нас направить на сторінку «Console Home» де представлений доступ до всіх продуктів AWS, але нас цікавить тільки сервіс під назвою «S3». Для зручності ми можемо додавати його у закладки. Вигляд сторінки та сервісу, а також приклад додавання його до закладок представлений на рисунку 4.1.

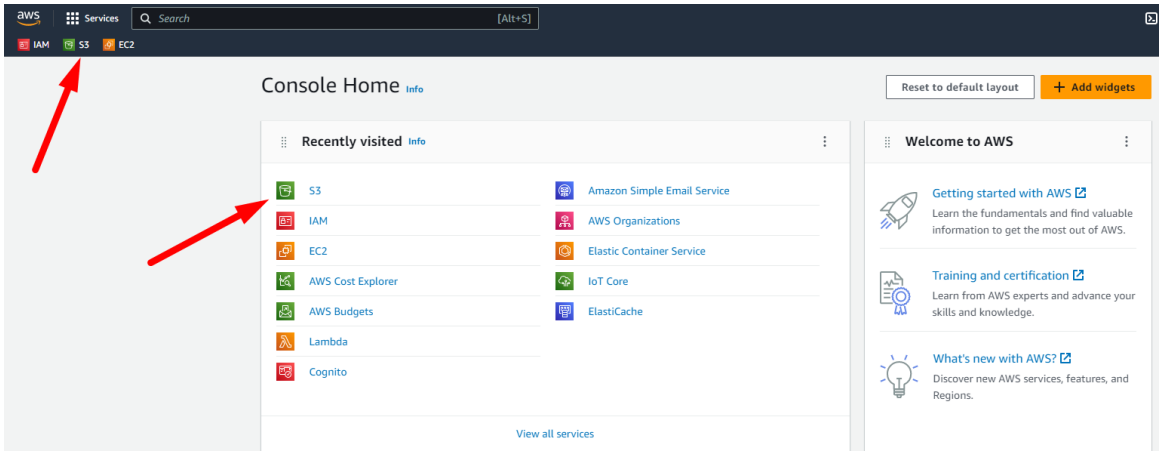


Рисунок 4.1 – Сторінка «Console Home» в AWS

Наступник кроком буде перехід у сервіс S3 та створення так званої корзини. Це аналог каталогу, якій існую в рамках сервісів сховища даних. Основною різницею між ними є те, що корзина має велику кількість додаткових налаштувань, а також існує ліміт у сто корзин на один аккаунт, його звісно ж можна збільшити за додаткову плату. У невеликих проектах зазвичай використовують одну корзину, а її вже розбивають на каталоги в

залежності від необхідності. Приклад створення корзини для нашого веб-додатку можна побачити на рисунках – [4.2 - 4.4].

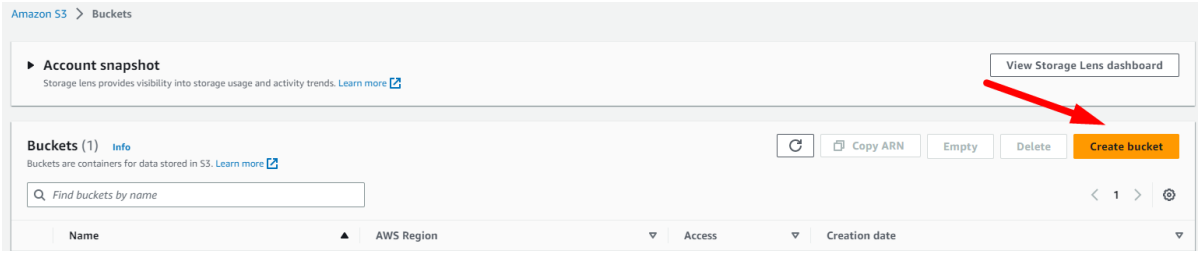


Рисунок 4.2 – Процес створення корзини. Крок 1. Перехід в необхідний розділ

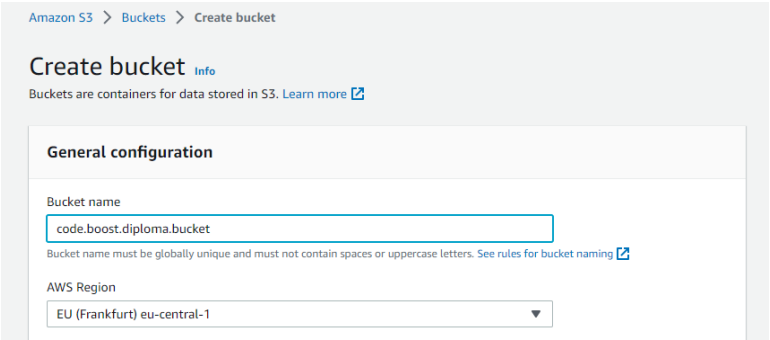


Рисунок 4.3 – Процес створення корзини. Крок 2. Заповнення інформації про корзину

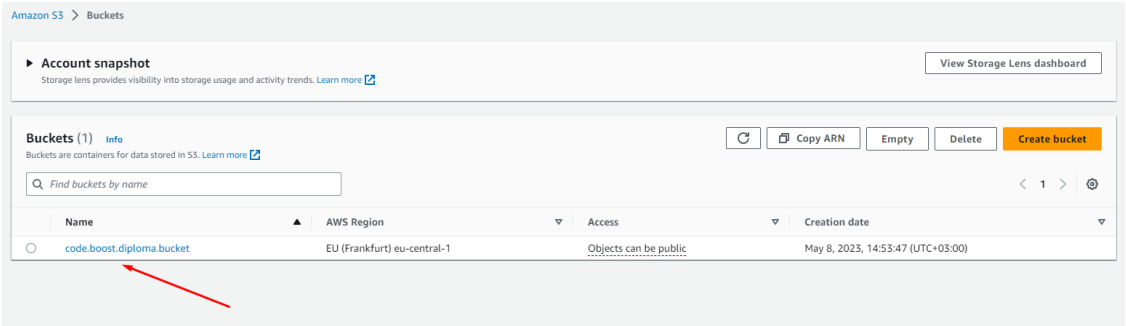


Рисунок 4.4 – Процес створення корзини. Крок 3. Створена корзина відображається в загальному списку

Далі необхідно створити в корзині додаткові каталоги для конкретних задач. У нашій системі основним функціоналом є керування книгами і курсами, тому створимо два каталоги: «course» та «book», рисунок – 4.5.

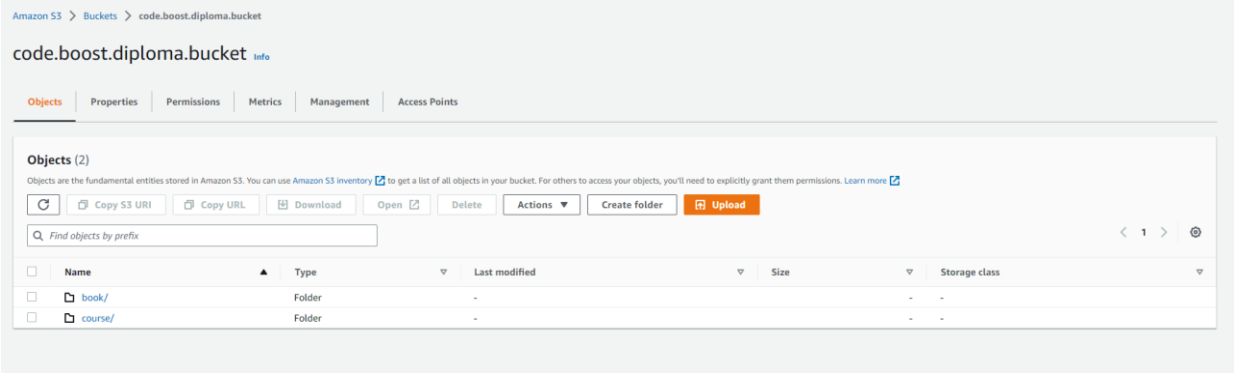


Рисунок 4.5 – Створені каталоги в корзині

Подібна структура сховища дозволить зберігати курси і книги ізольовано один від одного. Варто зазначити, що для кращої структуризації всередині каталогів «course» та «book» для кожної книги і курсу будуть програмно створюватися відповідні каталоги з назвою курсу чи книги.

### 4.2 Розробка серверної частини додатку

Задля реалізації серверної частини нашого веб-додатку використовується мова програмування Java у комбінації з фреймворком Spring Boot. Причини вибору та переваги описані в пункті 2.2.

Демонстрація розробки і конфігурації буде відбуватися на прикладі Course сервісу, адже усього наша система містить три мікросервіси, які дуже схожі у своїй конфігурації і структурі. Єдина різниця між ними це саме бізнес-логіка і сфера відповідальності.

Першим кроком буде створення Spring Boot проекту. Для цього при створенні проекту необхідно вказати параметри конфігурації. З метою виконання даного завдання було використано Spring Initializer [15]. Значення параметрів і причини вибору описані в таблиці 4.1.

Таблиця 4.1 – Список обраних параметрів конфігурації в Spring Boot та причини їх вибору

| Назва параметру | Обраний варіант | Причина(и) вибору |
|---|---|---|
| Система управління проектом | Maven | - Вичерпна документація<br>- Власний репозиторій |
| Версія Spring Boot | 3.0.6 | На момент написання роботи це найновіша стабільна версія. |
| Версія мови програмування Java | 17 | На момент написання роботи це найновіша LTS (Long Time Support) версія Java. |
| Тип пакування проекту | Jar | Best practice у даному типі додатків. |

**Створено на основі джерел [16, 17, 18]*

Після створення і відкриття проекту у нашій IDE ми можемо побачити базову структуру проекту – рисунок 4.6.

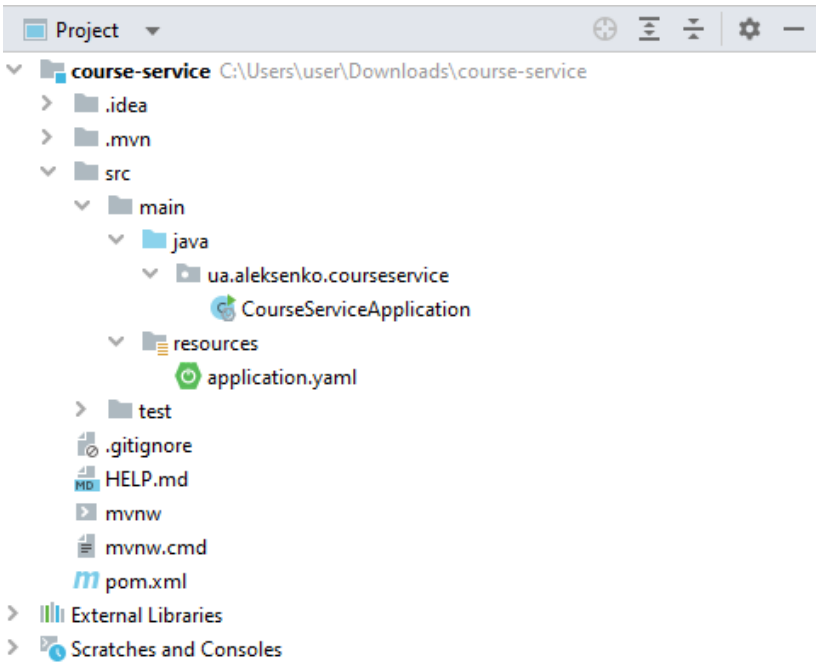


Рисунок 4.6 – Структура новоствореного проекту Spring Boot

Перед додаванням певного функціоналу ми маємо прописати залежності до нашого застосунку. Це дозволить нам розширити базовий функціонал фреймворку. Основними з імпортованих залежностей є:

- PostgreSQL Driver – для підключення до бази даних;
- Spring Data JPA – включає в себе імпорт Hibernate, специфікації JPA та функціоналу для створення Repository;
- Spring Web – для комунікації з клієнтом через REST API;
- Spring Security – для проведення авторизації і аутентифікації користувача за допомогою його токена;
- Java JWT – бібліотека для створення/декодування токена;
- Liquibase – для створення і запуску SQL міграцій.
- Lombok – допомагає генерувати конструктори, методи доступу до данних.
- Mapstruct – для переведення об’єктів бази даних у об’єкти DTO.

Повний список див. Додаток А.

Після додавання необхідних бібліотек необхідно описати параметри для їх коректної роботи:

- URL для підключення до бази даних;
- шлях до Liquibase каталогу з міграціями;
- порт серверу;
- налаштування JPA.

Повний список див. Додаток Б.

Далі, при кожному старті додатку, бібліотека Liquibase буде запускати файл описаний нами в попередньому кроці задля створення схеми даних. Для цього додаємо у відповідний каталог файл зі створенням схеми в базі даних.

Зміст файлу див. Додаток В.

Коли додаток успішно запускається і створюється схема в БД, то можемо переходити до додавання основної бізнес логіки, у нашому застосунку для її реалізації використовується шаблон Controller-Service-Repository.

Repository – рівень, є абстракцією для доступу до бази даних і використовується для CRUD (Create, Read, Update, Delete) операцій над об’єктами.

Controller – рівень, який служить у якості приймача усіх вхідних HTTP запитів від клієнта, відповідає за їх перевірку, а також за формування відповіді на запит клієнта. На цьому ж рівні відбувається перевірка прав на доступ до даного ресурсу. Якщо все гаразд і запит може бути оброблений, то він викликає відповідний метод рівня Service-у.

Service – рівень, який зберігає в собі основну бізнес-логіку додатку. Окрім цього саме на ньому відбувається переведення об’єктів із вигляду DTO в об’єкти бази даних і навпаки.

Першим кроком в імплементації даного шаблону є опис сутностей, які будуть використовуватися усіма рівнями. Веб-додаток має чотири основних груп об’єктів:

- DTO – представлення запитів і відповідей при HTTP комунікації;
- entity – представлення об’єктів в БД;
- error – представлення помилок, які будуть повернуті клієнту;
- exception – представлення помилок в рамках сервісу;
- specification – представлення специфікації для пошуку курсів.

Повний список класів можна побачити на рисунку 4.7.

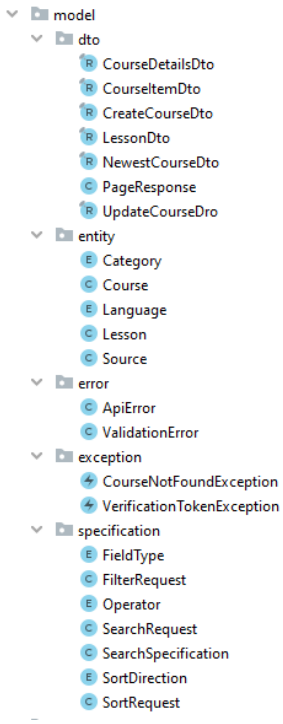


Рисунок 4.7 – Структура класів-моделей в проекті

Коли модель застосунку описана, можна переходити до реалізації рівня Repository. У веб-додатку представлені три інтерфейси для роботи з кожної з entity. Структура відображена на рисунку 4.8. Зміст файлів див. Додаток Г.

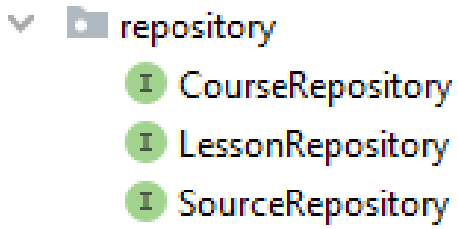


Рисунок 4.8 – Структура інтерфейсів Repository в проекті

Наступним кроком буде реалізація рівня Service. У даних класах ми будемо реалізовувати основну бізнес-логіку. Саме через цей рівень ми повинні звертатися до рівня Repository для отримання або редагування даних. В нашій інформаційній системі присутні два сервіси, перший відповідальний за бізнес-логіку по управлінню курсами, а інший відповідає за перевірку токена користувача при запиті на наш застосунок. Структура відображена на рисунку 4.9. Зміст файлів див. Додаток Д.

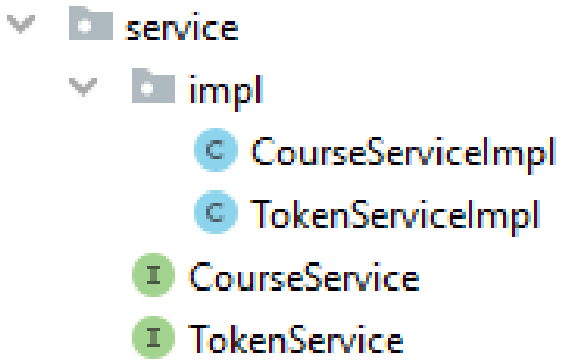


Рисунок 4.9 – Структура класів Service в проекті

Останнім кроком буде реалізація рівня Controller. У веб-додатку представлений один контролер у якому наявні вісім endpoints для різних типів запитів. Крім того для всіх класів даного рівня необхідно створити один глобальний handler помилок, він допоможе конвертувати помилку на рівні

додатка і перевести її у зрозумілу відповідь від контролера в форматі JSON. Структура відображена на рисунку 4.10. Зміст файлів див. Додаток Е.

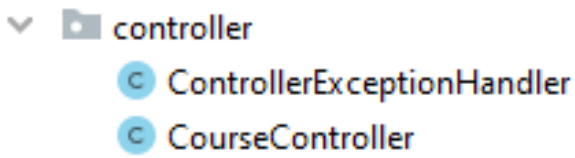


Рисунок 4.10 – Структура класів Controller в проекті

Далі опишемо інтерфейс для бібліотеки Mapstruct. Структура відображена на рисунку 4.11. Зміст файлу див. Додаток Ж.


Рисунок 4.11 – Структура інтерфейсів Mapper в проекті

Останнім етапом буде додавання класів конфігурації Spring Security. Вони будуть представлені трьома класами:

- SecurityConfig – клас для описання поведінки і типу захисту в нашій системі.
- JwtAuthenticationFilter – клас для перехвату запитів. Потрібен для обробки запитів до того, як вони потрапляють в контролер. Основної ціллю є перевірка наявності токена і його валідація через запит на сервіс авторизації.
- User – клас для представлення вже авторизованого користувача в системі. Це не одне й те саме, що сутність в БД сервісу авторизації. Даний користувач існує для внутрішнього використання в самому фреймворку.

Структура відображена на рисунку 4.12. Зміст файлів див. Додаток И.


Рисунок 4.12 – Структура класів конфігурації Spring Security

### 4.3 Розробка клієнтської частини додатку

У нашій системі клієнтська частина реалізована за допомогою фреймворку Angular і мови програмування TypeScript. Причини вибору та переваги описані в пункті 2.2.

Першим кроком ми створюємо новий проект за допомогою команди в CMD ***ng new code-boost-fe*** [20]. Результат генерації проекту зображений на рисунку 4.13.

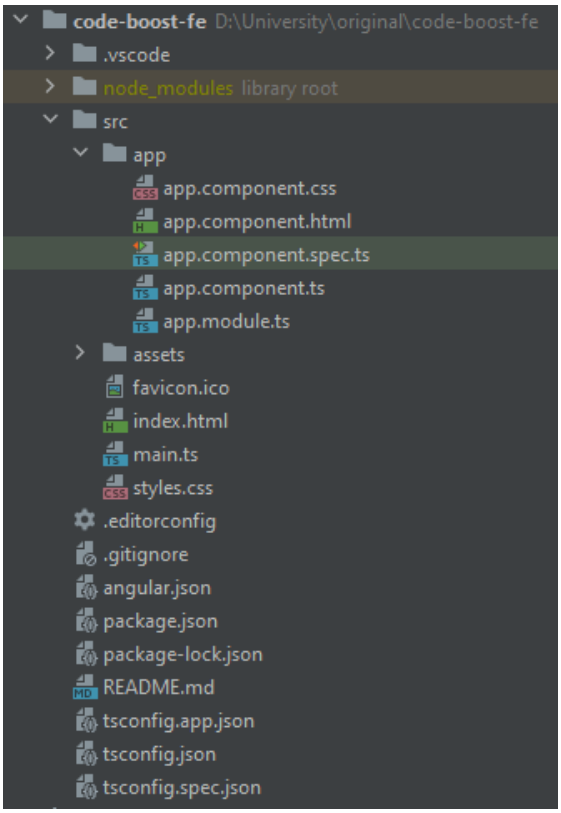


Рисунок 4.13 – Структура новоствореного проекту Angular

Структура Angular проекту складається з наступних елементів:

- головний компонент (app.component) – модуль, який є точкою входу в додаток. Він відповідає за зберігання та перехід між не головними компонентами;
- не головні компоненти – аналоги app.component, але може існувати у будь-якій кількості. Кожен не головний компонент є відображенням певної сторінки веб сайту. Усі компоненти складаються з трьох основних файлів (може бути більше при додаванні тестів):
    1. .css – файл зі стилями;
    2. .html – основний файл розмітки. Він має доступ до файлів .css і .ts для імпорту стилів і зв'язку з моделями даних. Відображається при демонстрації компонента;
    3. .ts – файл з бізнес-логікою компонента. Він відповідає за HTTP запити на backend, перехід між компонентами, може викликати і об’єднувати результати із різних сервісів для відображення на сторінці.
- сервіси – класи, котрі відповідають за певний функціонал у системі. Їх може використовувати будь-який компонент або інший сервіс;
- перехоплювач (Interceptor) – може виконувати різні типи задач, але у випадку нашого додатку перед відправленням запиту на backend перехоплює запит і підкладає туди токен авторизації;
- моделі – класи з набором полів для відправлення/отримання запитів;
- активи (Assets) – загальна назва для додаткових картинок, стилів, скриптів тощо.

Першим кроком створимо і опишемо список майбутніх сторінок. Для цього використаємо команду генерації нового компоненту: ***ng generate component*** **[component-name]** ***--skip-tests=true***

Результат генерації набору сторінок зображений на рисунку 4.14.

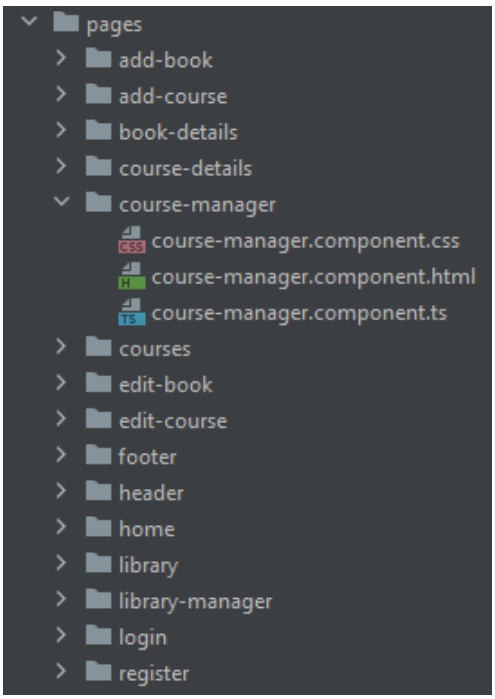


Рисунок 4.14 – Структура згенерованих компонентів

Наступним кроком опишемо сутності для комунікацією з серверною частиною через REST API. Для цього опишемо .ts класи з необхідними полями для кожного запиту. Результат створення класів зображений на рисунку 4.15. Лістинг програмного коду класу моделі Course Details див. Додаток К.

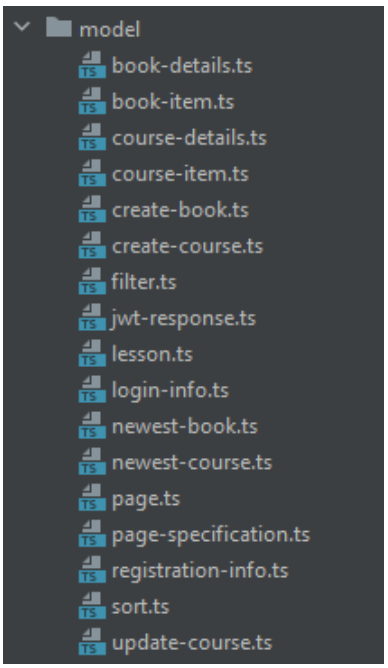


Рисунок 4.15 – Структура класів моделі

Після створення класів моделі необхідно створити сервіси, що будуть з їх допомогою виконувати HTTP запити на серверну частину додатку, додавати/видаляти токен з локального сховища, робити запити на хмарне сховище тощо. Результат створення сервісів зображений на рисунку 4.16. Лістинг програмного коду Course Service для отримання/збереження даних на серверній частині про курси див. Додаток Л.

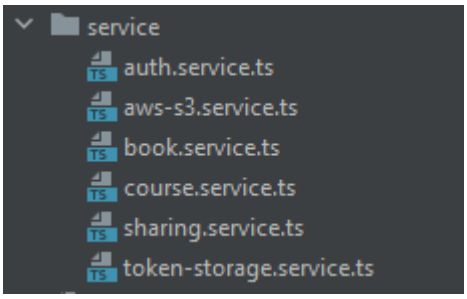


Рисунок 4.16 – Структура сервісів

Також важливим етапом реалізації інтеграції з серверною частиною, без якого кожен запит сервісів буде повертати помилку – це реалізація класу-перехоплювача, задля додавання токену авторизації, при будь-якому запиті на backend. Результат створення класу зображений на рисунку 4.17. Лістинг програмного коду класу-перехоплювача див. Додаток М.

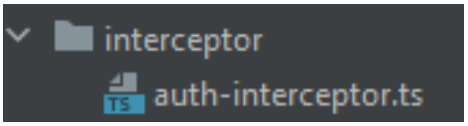


Рисунок 4.17 – Структура перехоплювача

# 5 ВИКОРИСТАННЯ ВЕБ-СИСТЕМИ

## 5.1 Процес взаємодії з системою у ролі адміністратора

Першим кроком для використання системи адміністратором є сторінка авторизації, рисунок – 5.1. Форма з’являється якщо користувач переходить на сайт без персонального токену. Логін і пароль адміністратора ми встановлюємо при першому запуску додатку.

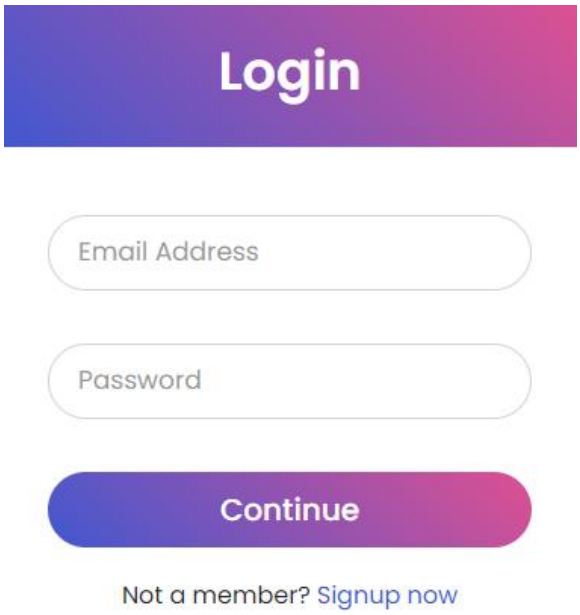


Рисунок 5.1 – Вигляд веб-сайту після його першого запуску

Після успішної авторизації нас переводить на сторінку управління курсами, рисунок – 5.2. Тут нам доступний наступний функціонал:

- пошук курсу по назві або її частині;
- фільтрація по категоріям і мові;
- кнопка для скидання фільтрів;
- навігація між сторінками;
- розділи у верхньому меню, щоб переключитися на управління книгами або виконати вихід з акаунта;
- кнопка додавання нового курсу.

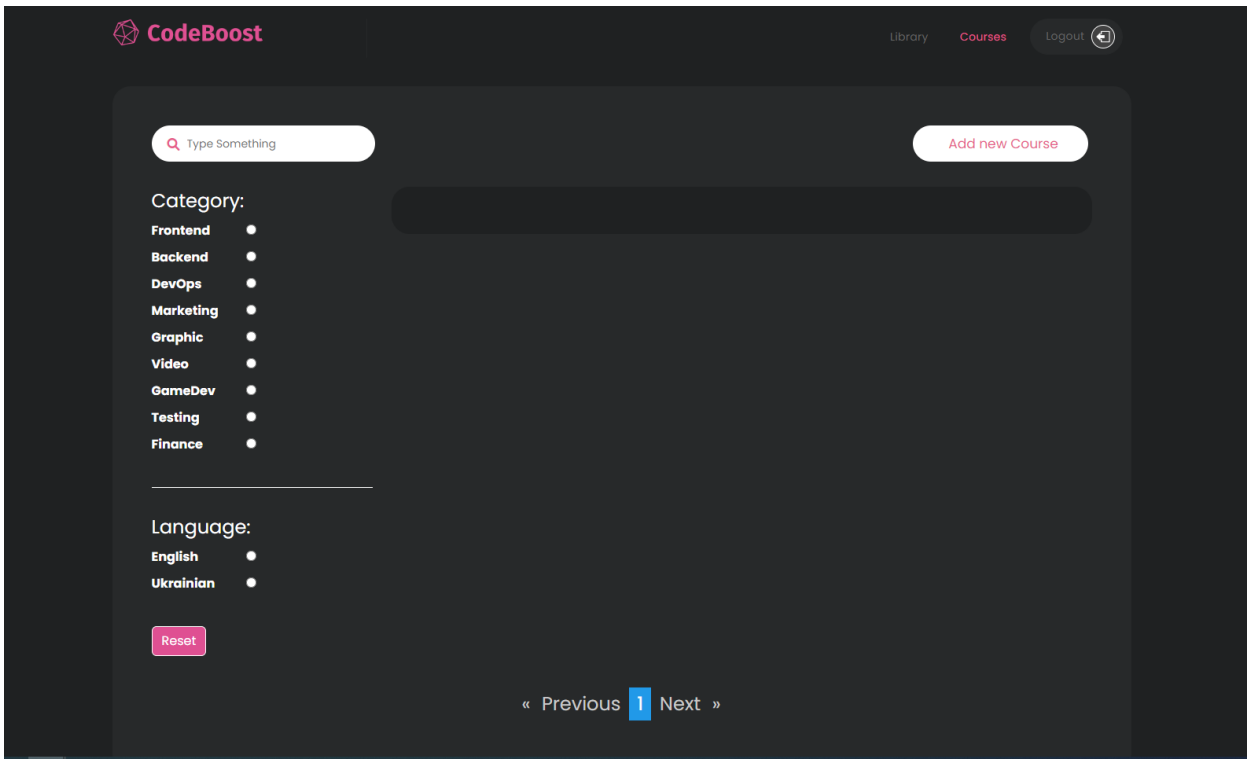


Рисунок 5.2 – Вигляд сторінки управління курсами, коли в системі відсутні курси

Аби розпочати роботу з курсами, необхідно їх додати. Для цього натискаємо на кнопку «Add new Course» у верхньому правому кутку. Нас переводить на сторінку для додавання курсів із відповідною формою – рисунок – 5.3. Заповнюємо усі поля, першим файлом – завантажуємо логотип курсу, а другим – завантажуємо один або більше файлів з відеоуроками.

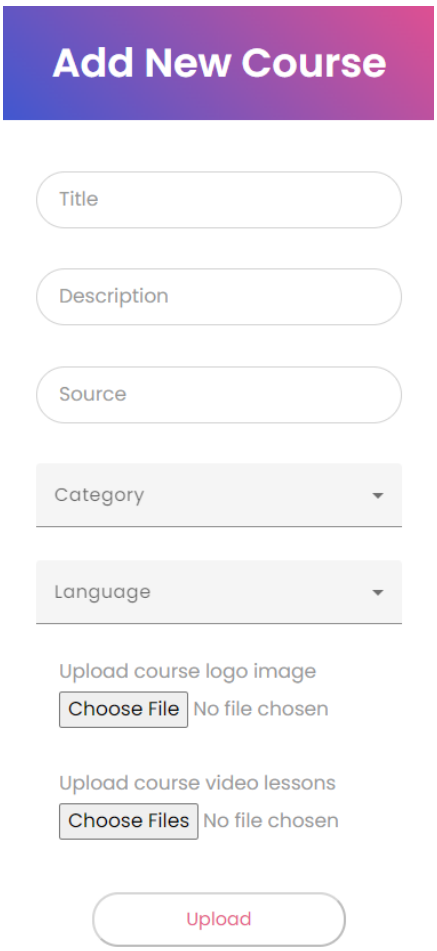


Рисунок 5.3 – Вигляд сторінки по додаванню курсу

Після завершення заповлення форми – натискаємо кнопку «Upload» і очікуємо завершення загрузки. Після успішного завершення ми будем перенаправлені на сторінку управління курсами, але тепер у списку ми побачимо щойно доданий курс, рисунок – 5.4. Аби перевірити чи все вірно було завантажено – натиснемо на єлемент курсу в списку і переглянемо сторінку його деталей, рисунок – 5.5. Тут присутня повна інформація про курс: його назва, опис, джерело, категорія, кількість уроків, дата останнього оновлення, мова курсу та плеєр для перегляду уроків.

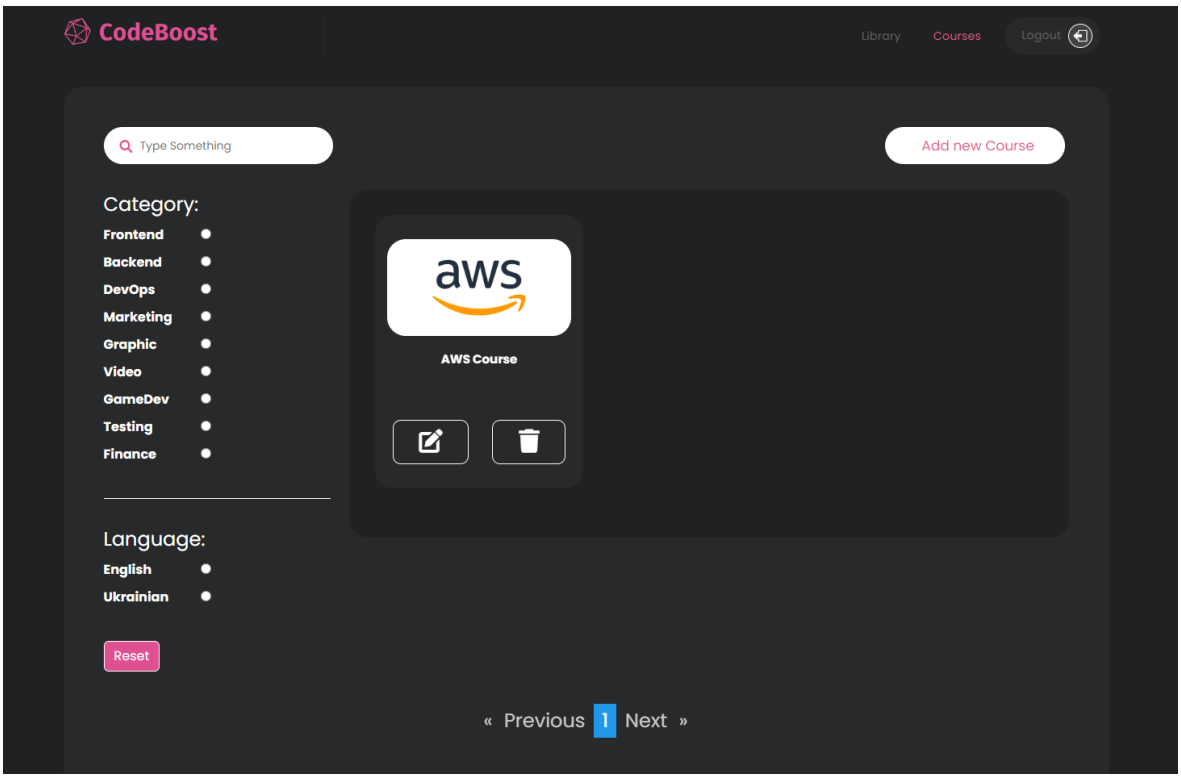


Рисунок 5.4 – Вигляд сторінки управління курсами, коли в системі присутні курси

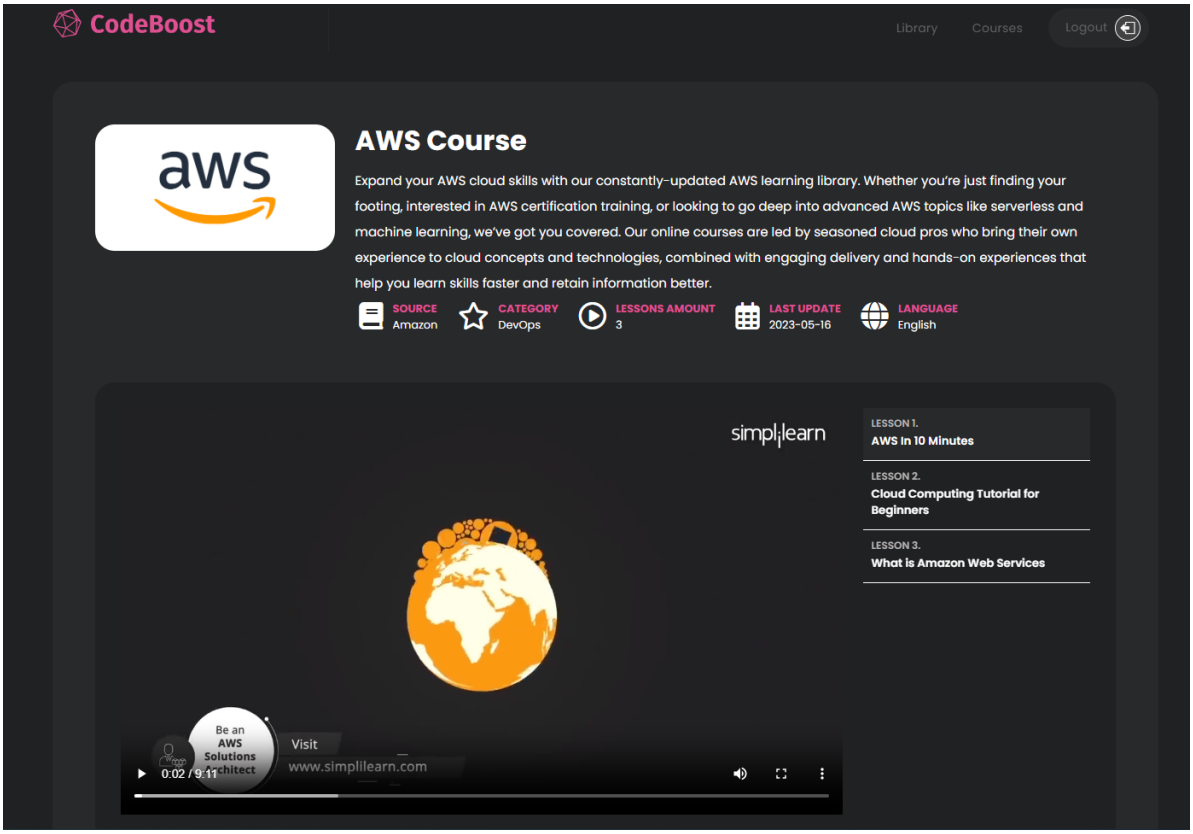


Рисунок 5.5 – Вигляд сторінки перегляду деталей курсу

Наступним кроком розглянемо можливості взаємодії зі створеним курсом, а саме процес видалення і редагування. На сторінці управління курсами під логотипом курсу можемо побачити дві кнопки:

- редагування – перенаправляє на сторінку редагування з предзаповненою формою, у якій ми можемо редагувати усі поля, додавати/перейменовувати/видаляти відеоуроки, завантажувати новий логотип тощо, рисунок – 5.6. Після натискання кнопки «Update» всі оновлені дані з форми будуть збережені в базі даних і нас переведе на сторінку управління курсами;
- видалення – показує pop-up для підтвердження видалення, рисунок – 5.7. Після підтвердження курс буде остаточно видалений.

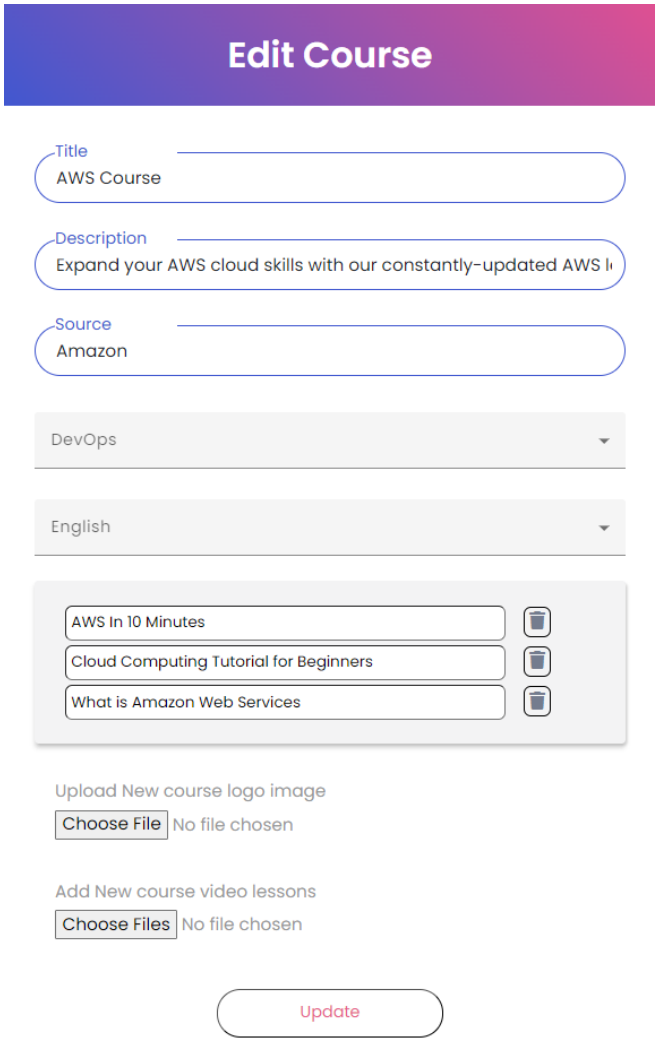


Рисунок 5.6 – Вигляд  сторінки для редагування курсу

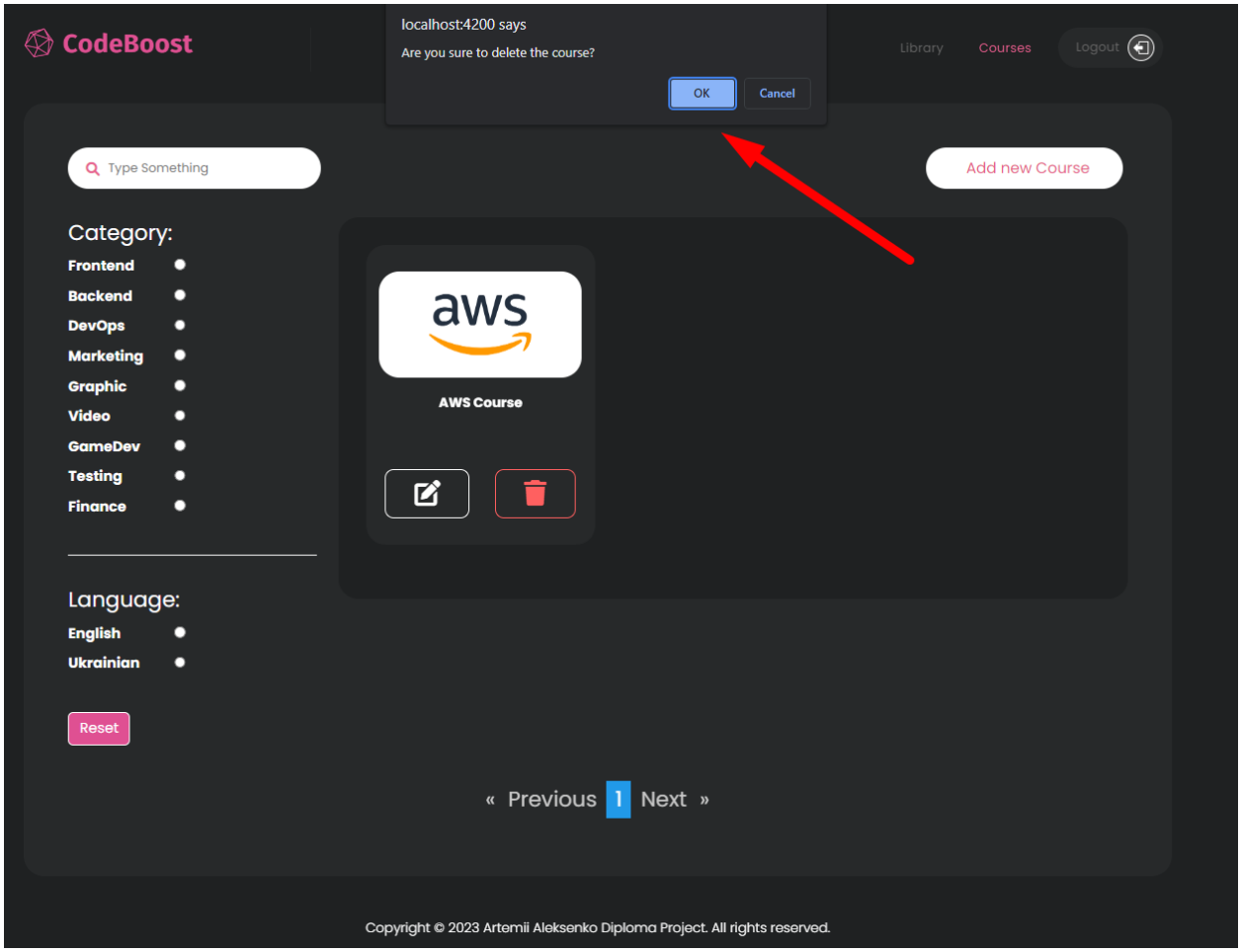


Рисунок 5.7 – Pop-up для підтвердження видалення курсу

До можливостей адміністратора також входить додавання/редагування/видалення наукової літератури. Для цього переходимо у вкладку «Library» у верхньому меню. Нас переведе на сторінку керування книгами, дуже схожу на сторінку керування курсами, за винятком того, що кнопка додавання називається «Add new Book». Після натискання на неї ми перейдемо на сторінку додавання нової книги, рисунок – 5.8, де необхідно заповнити інформацію про книгу, завантажити логотип (обкладинку) і завантажити .pdf файл зі змістом книги. Після натискання кнопки «Upload» книга буде завантажена і нас переведе на сторінку управління книгами, рисунок – 5.9. Аби перевірити чи все вірно було завантажено – натиснемо на єлемент книги в списку і переглянемо сторінку її деталей, рисунки – 5.10. Тут

присутня повна інформація про книгу: її назва, опис, автор, категорія, дата релізу, мова та PDF reader для ознайомлення зі змістом книги, рисунок – 5.11.

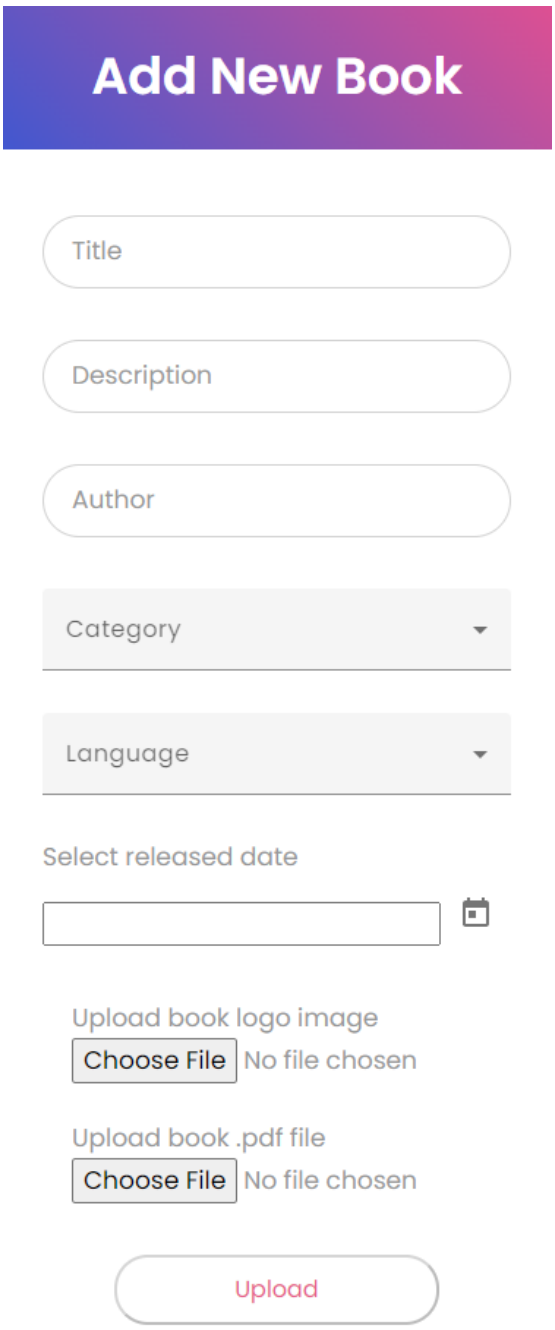


Рисунок 5.8 – Вигляд сторінки по додаванню книги

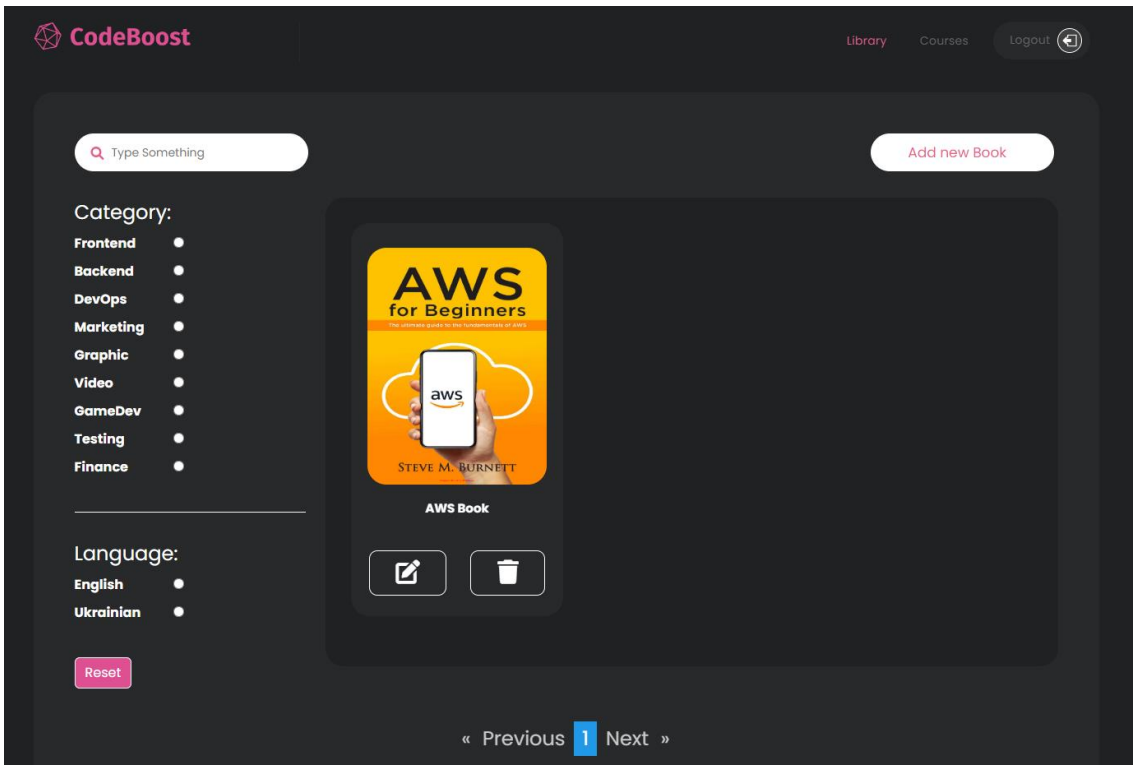


Рисунок 5.9 – Вигляд сторінки управління книгами, коли в системі присутні книги

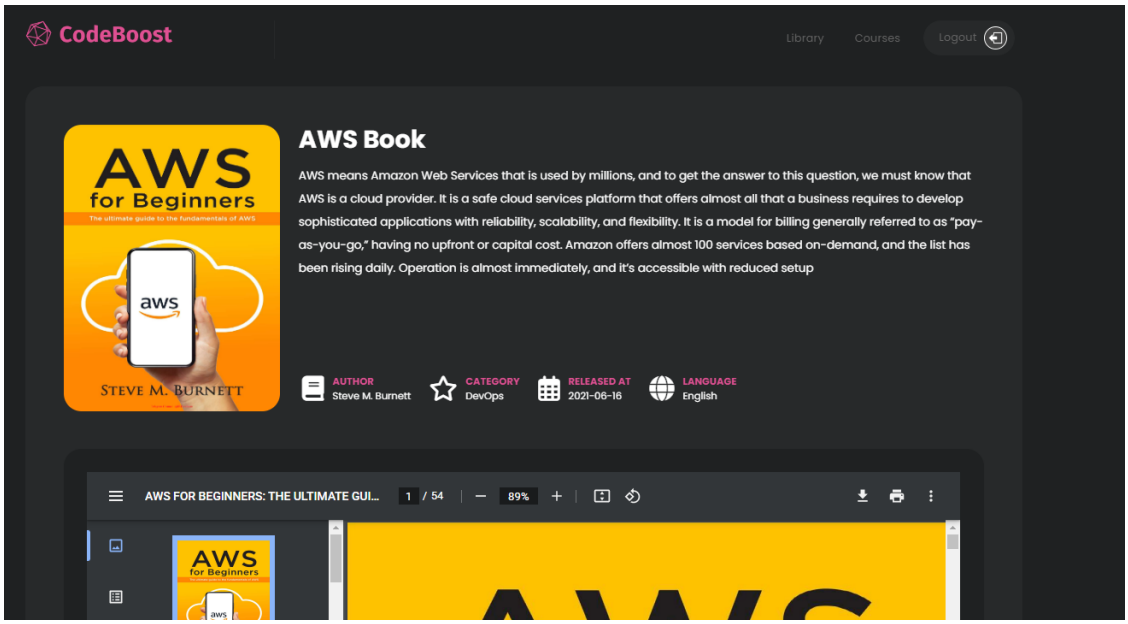


Рисунок 5.10 – Вигляд сторінки перегляду деталей книги. Загальна інформація про книгу


Рисунок 5.11 – Вигляд сторінки перегляду деталей книги. PDF reader для читання книги

За аналогією з управлінням курсами у нас є можливість видалити і редагувати книгу в наші системі.

## 5.2 Процес взаємодії з системою у ролі користувача

У нового користувача процес взаємодії з веб-додатком розпочинається з етапу реєстрації. Для цього необхідно натиснути посилання «Signup now» внизу форми авторизації, показаної у попередньому розділі. Після цього користувача буде переведено на сторінку реєстрації, рисунок – 5.12.

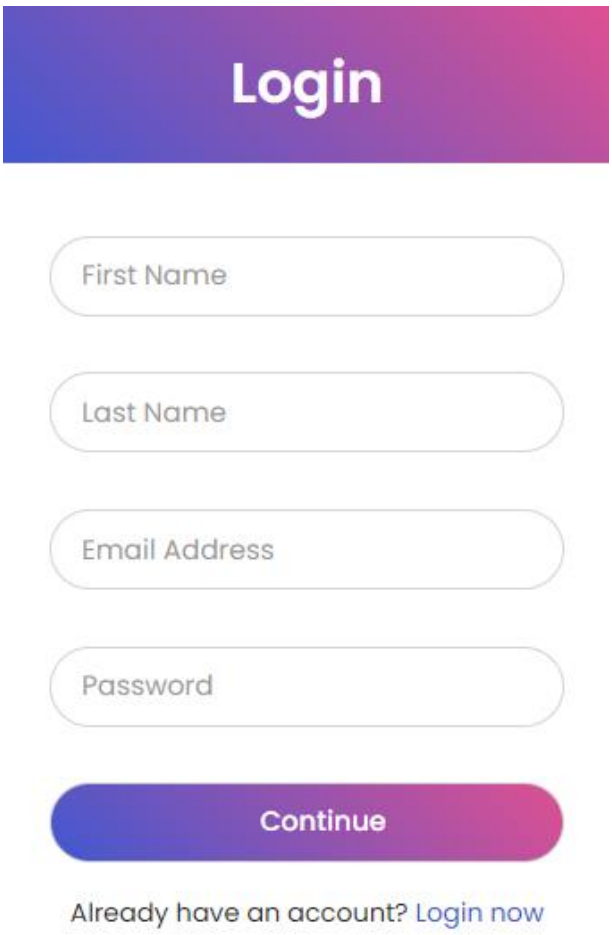


Рисунок 5.12 – Вигляд сторінки реєстрації

Після введення коректних даних, користувач натискає кнопку «Continue» і його переводить на домашню сторінку, рисунок – 5.13. Домашня сторінка складається з трьох основних частин:

- вітальний плакат, рисунок – 5.13;
- найновіші курси – 6 курсів, які біло додані останніми, рисунок – 5.14;
- найновіші книги – 6 книг, у яких найновіша дата релізу, рисунок – 5.15.

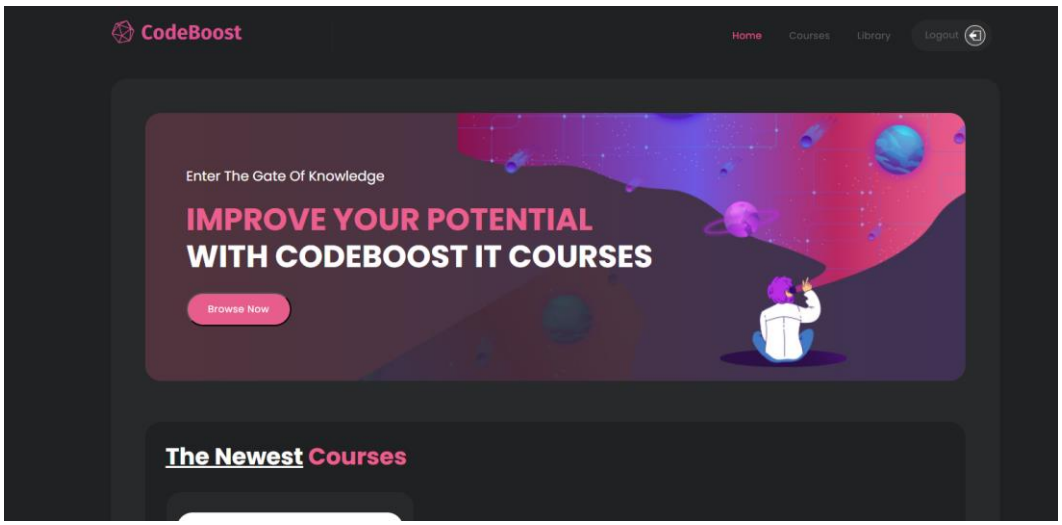


Рисунок 5.13 – Вигляд домашньої сторінки. Вітальний плакат

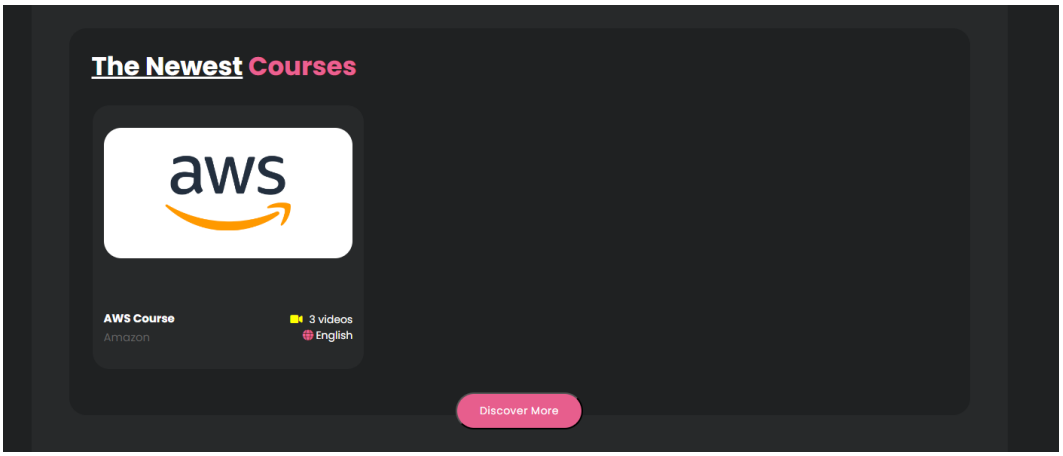


Рисунок 5.14 – Домашня сторінка. Найновіші курси

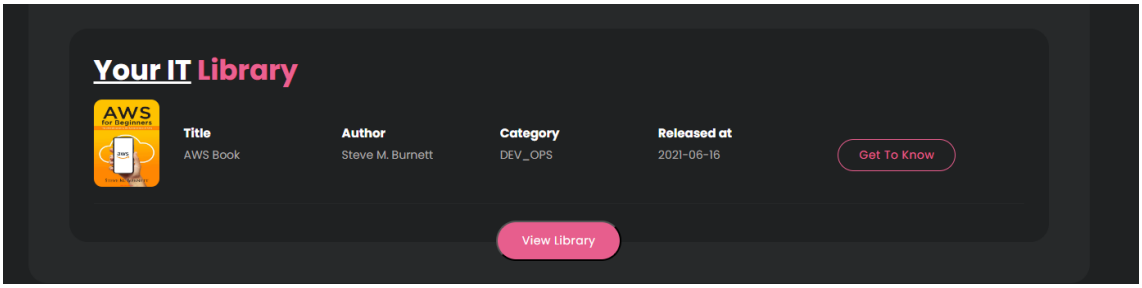


Рисунок 5.15 – Домашня сторінка. Найновіші книги

Наступним кроком користувач може дослідити повний список курсів чи книг. Це можна зробити через пункти в верхньому меню: «Courses», «Library»,

або використовуючи кнопки на домашній сторінці: «Browse Now», «Discover More», «View Library».

В залежності від навчального матеріалу, користувач буде направлений або на сторінку списку курсів, рисунок – 5.16, або на сторінку списку книг, рисунок – 5.17.

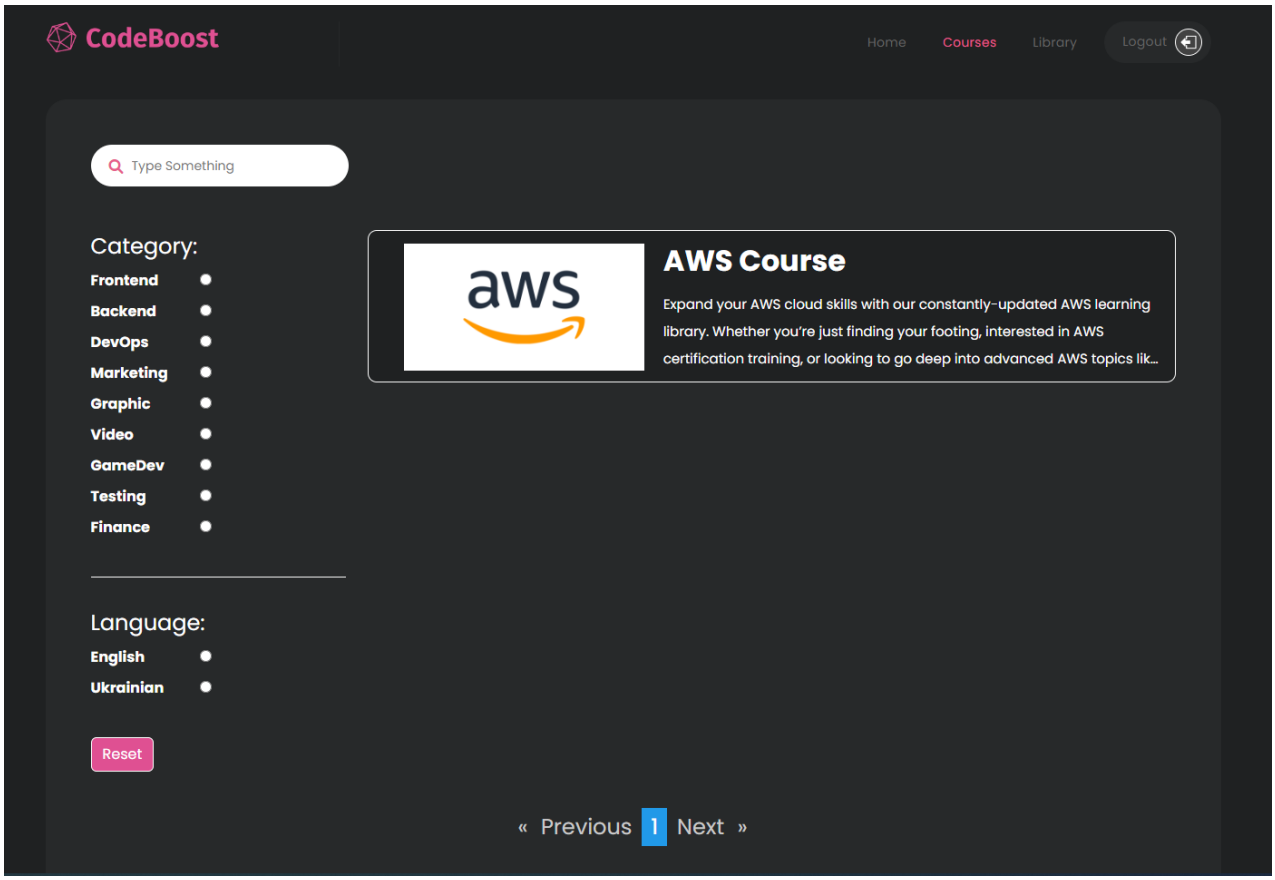


Рисунок 5.16 – Вигляд сторінки із загальним списком курсів

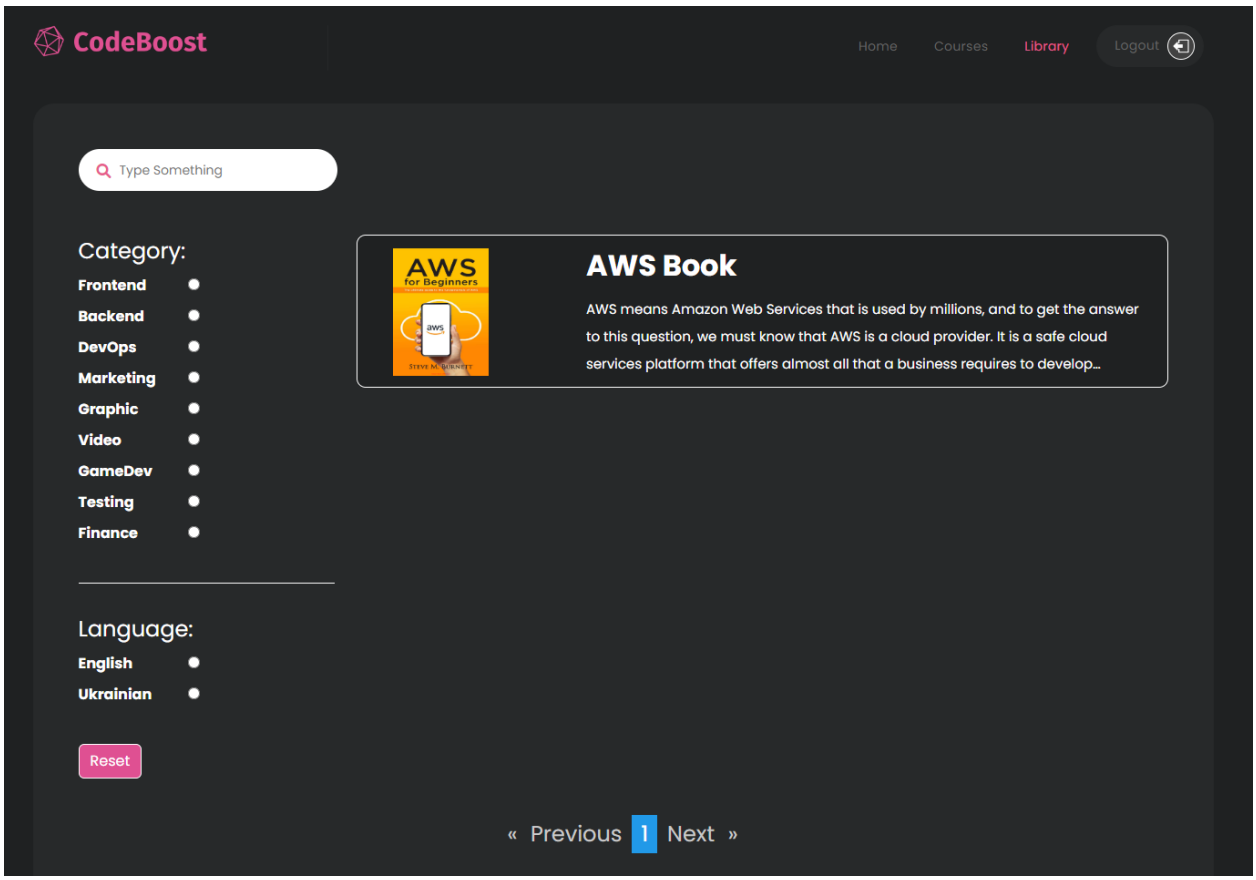


Рисунок 5.17 – Вигляд сторінки із загальним списком книг

Коли користувач знаходить бажаний курс чи книгу він може натиснути на необхідний елемент і перейти на сторінки деталей, які є аналогічними до сторінок деталей адміністратора.

# ВИСНОВКИ

У процесі виконання бакалаврської кваліфікаційної роботи було виконано літературний огляд існуючих аналогів веб-орієнтованих систем керування навчанням та визначено основні переваги та недоліки. Дані проведеного аналізу дозволили сформувати основні вимоги до розробки веб-орієнтованої системи керування онлайн курсами, а саме: можливість додавати, видаляти та редагувати навчальні матеріали, виконувати їх фільтрацію. Також було виділено, що важливим аспектом при створені навчальної веб-системи є корисність викладеного матеріалу, вдало обрана структура та дизайн веб-системи, захист від неавторизованого доступу та багатомовність матеріалів.

На етапі вибору інструментів розробки ми зупинились на фреймворках Angular та Spring Boot та мовах програмування Java та TypeScript.

Обрані фреймворки та мови програмування дозволяють найліпшим чином реалізувати розробку веб-системи, з урахуванням поставлених функціональних вимог, що включають реєстрацію нового користувача, аутентифікацію та авторизацію, розбиття на ролі, надання можливостей додавати, видаляти, та змінювати курси чи інші навчальні матеріали, переглядати курси за категоріями та шукати навчальний курс по ключовим словам.

В процесі роботи було спроектовано та змодельовано функціональні бізнес-процеси у нотації IDEF0. Також для проектування інформаційної системи були використані наступні діаграми:

- діаграма прецедентів;
- діаграма розгортання;
- діаграма класів;
- діаграма послідовності.

Для зберігання інформації про курси або інші навчальні метеріали була спроектована схема бази даних для кожного backend сервісу;

Було налаштовано хмарне сховище даних для зберігання навчальних матеріалів (відеоуроки, книги, логотипи).

В процесі реалізації були створені серверна і клієнтська частини додатку, які представлені одним сервісом для функціонування frontend частини і трьома сервісами для функціонування backend частини.

Результатом роботи є інформаційна веб-система керування навчальними курсами, яка повністю відповідає поставленому завданню, є стійкою до великих навантажень зі здатністю до масштабування.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. 180 тисяч нових робочих місць: як змінилася українська IT-індустрія за 5 років? - Офіс ефективного регулювання. Офіс ефективного регулювання. URL: https://brdo.com.ua/news/180-tysyach-novyh-robochyh-mists-yak-zminylasya-ukrayinska-it-industriya-za-5-rokiv/ (дата звернення: 29.12.2022).
2. Відеокурси для розробників!. *CourseHunter*. URL: https://coursehunter.net/ (дата звернення: 29.12.2022).
3. Mama Samba Braima N. Amigoscode. Amigoscode | Amigoscode. URL: https://amigoscode.com/courses (date of access: 29.12.2022).
4. IT Курси Програмування онлайн з працевлаштуванням у Києві і інших містах Навчаємо з нуля | Mate academy. Mate academy. URL: https://mate.academy/ (дата звернення: 29.12.2022).
5. Як перевірити швидкість завантаження сайту: онлайн-сервіси – Lemarbet. Lemarbet | Створення та розвиток інтернет-магазинів. URL: https://lemarbet.com/ua/razvitie-internet-magazina/kak-proverit-skorost-zagruzki-sajta-onlajn-servisy/ (дата звернення: 29.12.2022).
6. GTmetrix | Website Performance Testing and Monitoring. GTmetrix | Website Performance Testing and Monitoring. URL: https://gtmetrix.com/ (date of access: 29.12.2022).
7. Angular. Angular. URL: https://angular.io/guide/what-is-angular (date of access: 29.12.2022).
8. What is Java Spring Boot? | IBM. IBM - Deutschland | IBM. URL: https://www.ibm.com/topics/java-spring-boot#:~:text=Java%20Spring%20Boot%20(Spring%20Boot,ability%20to%20create%20standalone%20applications (date of access: 29.12.2022).

9. What is IDEF - definition, methods, and benefits - edraw. [OFFICIAL] Edraw Software: Unlock Diagram Possibilities. URL: https://www.edrawsoft.com/what-is-idef.html (date of access: 16.05.2023).
10. IBM documentation. IBM - Deutschland | IBM. URL: https://www.ibm.com/docs/en/rational-soft-arch/9.6.1?topic=diagrams-use-case (date of access: 16.05.2023).
11. UML deployment diagrams overview, common types of deployment diagrams - manifestation diagram, specification and instance level deployment diagram. Unified Modeling Language (UML) description, UML diagram examples, tutorials and reference for all types of UML diagrams - use case diagrams, class, package, component, composite structure diagrams, deployments, activities, interactions, profiles, etc. URL: https://www.uml-diagrams.org/deployment-diagrams-overview.html (date of access: 16.05.2023).
12. UML Class Diagram - Javatpoint. www.javatpoint.com. URL: https://www.javatpoint.com/uml-class-diagram (date of access: 16.05.2023).
13. Wisbey O. What is a sequence diagram?. Software Quality. URL: https://www.techtarget.com/searchsoftwarequality/definition/sequence-diagram (date of access: 16.05.2023).
14. PostgreSQL: About. PostgreSQL: The world's most advanced open source database. URL: https://www.postgresql.org/about/ (date of access: 16.05.2023).
15. Spring Initializer. URL: https://start.spring.io/ (date of access: 16.05.2023).
16. Getting Started. Spring | Home. URL: https://docs.spring.io/spring-boot/docs/current/reference/html/getting-started.html (date of access: 16.05.2023).
17. JDK 17 Release Notes, Important Changes, and Information. Oracle | Cloud Applications and Cloud Platform. URL:

https://www.oracle.com/java/technologies/javase/17-relnote-issues.html (date of access: 16.05.2023).

18. Maven Features. Apache Maven Project. URL: https://maven.apache.org/maven-features.html (date of access: 16.05.2023).
19. Getting started with Amazon S3 - Amazon Simple Storage Service. URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/GetStartedWithS3.html (date of access: 16.05.2023).
20. Create a new project. Angular. URL: https://angular.io/tutorial/tour-of-heroes/toh-pt0 (date of access: 16.05.2023).

# ДОДАТОК А

## pom.xml файл із залежностями Course Service

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.0.5</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>ua.aleksenko.courseservice</groupId>
    <artifactId>course-service</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>course-service</name>
    <description>course-service</description>

    <properties>
        <java.version>17</java.version>
        <checkstyle-maven-plugin.version>3.2.1</checkstyle-maven-
plugin.version>
        <preliquibase.version>1.3.0</preliquibase.version>
        <mapstruct.version>1.5.3.Final</mapstruct.version>
        <lombok-mapstruct-binding.version>0.2.0</lombok-mapstruct-
binding.version>
        <hypersistence-utils.version>3.3.1</hypersistence-utils.version>
       <java-jwt.version>4.4.0</java-jwt.version>
    </properties>

    <dependencies>
        <!--Spring-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
       <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-security</artifactId>
       </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <!--Database-->
        <dependency>
            <groupId>org.liquibase</groupId>
            <artifactId>liquibase-core</artifactId>
        </dependency>
        <dependency>
            <groupId>net.lbruun.springboot</groupId>
            <artifactId>preliquibase-spring-boot-starter</artifactId>
            <version>${preliquibase.version}</version>
```

```
        </dependency>
        <dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>io.hypersistence</groupId>
            <artifactId>hypersistence-utils-hibernate-60</artifactId>
            <version>${hypersistence-utils.version}</version>
        </dependency>
        <!--Other-->
       <dependency>
          <groupId>io.projectreactor</groupId>
          <artifactId>reactor-core</artifactId>
          <version>3.4.26</version>
       </dependency>
       <dependency>
          <groupId>org.springframework</groupId>
          <artifactId>spring-webflux</artifactId>
          <version>5.3.25</version>
       </dependency>
       <dependency>
          <groupId>io.projectreactor.netty</groupId>
          <artifactId>reactor-netty-http</artifactId>
       </dependency>
       <dependency>
          <groupId>com.auth0</groupId>
          <artifactId>java-jwt</artifactId>
          <version>${java-jwt.version}</version>
       </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.mapstruct</groupId>
            <artifactId>mapstruct</artifactId>
            <version>${mapstruct.version}</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>${maven-compiler-plugin.version}</version>
                <configuration>
                    <source>${java.version}</source>
```

```
                    <target>${java.version}</target>
                    <annotationProcessorPaths>
                        <path>
                            <groupId>org.mapstruct</groupId>
                            <artifactId>mapstruct-processor</artifactId>
                            <version>${mapstruct.version}</version>
                        </path>
                        <path>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                            <version>${lombok.version}</version>
                        </path>
                        <path>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok-mapstruct-binding</artifactId>
                            <version>${lombok-mapstruct-
binding.version}</version>
                        </path>
                    </annotationProcessorPaths>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-checkstyle-plugin</artifactId>
                <version>${checkstyle-maven-plugin.version}</version>
                <configuration>

<configLocation>config/checkstyle/checkstyle.xml</configLocation>
                    <sourceDirectories>

<sourceDirectory>${project.build.sourceDirectory}</sourceDirectory>

<sourceDirectory>${project.build.testSourceDirectory}</sourceDirectory>
                    </sourceDirectories>
                </configuration>
                <executions>
                    <execution>
                        <phase>install</phase>
                        <goals>
                            <goal>check</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

# ДОДАТОК Б

application.yaml файл із глобальною конфігурацією для Course Service

```yaml
server:
  port: ${SERVER_PORT:8090}
spring:
  jpa:
    open-in-view: false
    database-platform: org.hibernate.dialect.PostgreSQLDialect
    generate-ddl: false
    hibernate:
      ddl-auto: validate
    show-sql: ${SHOW_SQL:false}
    database: postgresql
    properties:
      hibernate:
        format_sql: true
        generate_statistics: false
        jdbc:
          time_zone: UTC
        default_schema: course
      jdbc:
        batch_size: 100
      order_inserts: true
      order_updates: true
  datasource:
    url:
jdbc:postgresql://${DB_HOST:localhost}:${DB_PORT:5432}/${DB_NAME:postgres}
    password: ${DB_PASSWORD:password}
    username: ${DB_USERNAME:postgres}
    driver-class-name: org.postgresql.Driver
  liquibase:
    change-log: classpath:db/changelog/db.changelog-master.yaml
    default-schema: course
    liquibase-schema: course
token-issuer:
  base-url: ${TOKEN_ISSUER_BASE_URL:http://localhost:8092}
  token-validation-endpoint:
${TOKEN_VALIDATION_ENDPOINT:/api/v1/token/validate}
```

# ДОДАТОК В

001_init-db-schema.sql файл із SQL скриптом для створення схеми в БД

```
-- liquibase formatted sql
-- changeset artemii.aleksenko:001_init-db-schema.sql
CREATE TYPE language_type AS ENUM ('ENGLISH', 'UKRAINIAN');
-- rollback DROP TYPE language_type;

CREATE TYPE category_type AS ENUM ('FRONTEND', 'BACKEND', 'DEV_OPS',
'MARKETING', 'GRAPHIC', 'VIDEO', 'GAME_DEV', 'TESTING', 'FINANCE');
-- rollback DROP TYPE language_type;

CREATE CAST (character varying AS language_type) WITH INOUT AS IMPLICIT;
CREATE CAST (character varying AS category_type) WITH INOUT AS IMPLICIT;

CREATE TABLE "source"
(
    "id"          UUID PRIMARY KEY,
    "name"        VARCHAR(255) UNIQUE NOT NULL
);
-- rollback DROP TABLE source;

CREATE TABLE "course"
(
    "id"             UUID PRIMARY KEY,
    "title"          VARCHAR(255)  NOT NULL,
    "description"    VARCHAR(1000) NOT NULL,
    "image_url"      VARCHAR(1000) NOT NULL,
    "language"       language_type NOT NULL,
    "category"       category_type NOT NULL,
    "created_at"     date     NOT NULL,
    "updated_at"     date     NOT NULL,
    "source_id"      UUID          NOT NULL,
    CONSTRAINT "FK_course.source_id" FOREIGN KEY ("source_id") REFERENCES
"source" ("id") ON DELETE CASCADE
);
-- rollback DROP TABLE course;

CREATE TABLE "lesson"
(
    "id"        UUID PRIMARY KEY,
    "title"     VARCHAR(255)  NOT NULL,
    "video_url" VARCHAR(1000) NOT NULL,
    "course_id" UUID          NOT NULL,
    CONSTRAINT "FK_lesson.course_id" FOREIGN KEY ("course_id") REFERENCES
"course" ("id")
);
-- rollback DROP TABLE lesson;
```

# ДОДАТОК Г

Файли Repository рівня в проекті

## CourseRepository.java

```
@Repository
public interface CourseRepository extends JpaRepository<Course, UUID>,
JpaSpecificationExecutor<Course> {

   List<Course> findFirst6ByOrderByCreatedAtDesc();
}
```

## LessonRepository.java

```
@Repository
public interface LessonRepository extends JpaRepository<Lesson, UUID> {

}
```

## SourceRepository.java

```
@Repository
public interface SourceRepository extends JpaRepository<Source, UUID> {

   Optional<Source> findByNameIgnoreCase(String name);
}
```

# ДОДАТОК Д

Файли Service рівня в проекті

## CourseService.java

```
public interface CourseService {

   PageResponse<CourseItemDto> searchCourses(SearchRequest request);

   CourseDetailsDto getCourseDetails(UUID id);

   List<NewestCourseDto> getNewestCourses();

   List<String> getAllCategories();

   List<String> getAllLanguages();

   void deleteCourse(UUID id);

   void create(CreateCourseDto dto);

   void update(UpdateCourseDro dto);
}
```

## TokenService.java

```
public interface TokenService {

    void validateExpirationDate(String token);

    void validateToken(String token);
}
```

## CourseServiceImpl.java

```
@Service
@RequiredArgsConstructor
@Slf4j
public class CourseServiceImpl implements CourseService {

   private final CourseRepository courseRepository;
   private final SourceRepository sourceRepository;
   private final LessonRepository lessonRepository;
   private final CourseMapper mapper;

   @Override
   @Transactional(readOnly = true)
   public PageResponse<CourseItemDto> searchCourses(SearchRequest request) {
      log.info("Start executing method searchCourses: {}", request);
      request.setPage(request.getPage() - 1);
      SearchSpecification<Course> specification = new
SearchSpecification<>(request);
      Pageable pageable = SearchSpecification.getPageable(request.getPage(),
request.getSize());
      Page<Course> coursesPage = courseRepository.findAll(specification,
pageable);
      List<CourseItemDto> employeesShortInfoDto =
```

```
mapper.toCourseItemDtos(coursesPage.getContent());
      PageResponse<CourseItemDto> pageResponse = new
PageResponse<>(employeesShortInfoDto, coursesPage);
      pageResponse.setPageNumber(pageResponse.getPageNumber() + 1);
      log.info("Finish executing method searchCourses with result: {}",
pageResponse);
      return pageResponse;
   }

   @Override
   @Transactional(readOnly = true)
   public CourseDetailsDto getCourseDetails(UUID id) {
      log.info("Start executing method getCourseDetails for course: {}", id);
      Course course = courseRepository.findById(id)
            .orElseThrow(() -> new CourseNotFoundException(id));
      CourseDetailsDto courseDetailsDto = mapper.toCourseDetailsDto(course);
      log.info("Finish executing method getCourseDetails with result: {}",
courseDetailsDto);
      return courseDetailsDto;
   }

   @Override
   @Transactional(readOnly = true)
   public List<NewestCourseDto> getNewestCourses() {
      log.info("Start executing method getNewestCourses");
      List<Course> first6NewestCourses =
courseRepository.findFirst6ByOrderByCreatedAtDesc();
      List<NewestCourseDto> first6NewestCoursesDto =
mapper.toNewestCourseDtos(first6NewestCourses);
      log.info("Finish executing method getNewestCourses with result: {}",
first6NewestCoursesDto);
      return first6NewestCoursesDto;
   }

   @Override
   public List<String> getAllCategories() {
      log.info("Start executing method getAllCategories");
      List<String> allCategories = Category.getAllCategories();
      log.info("Finish executing method getAllCategories with result: {}",
allCategories);
      return allCategories;
   }

   @Override
   public List<String> getAllLanguages() {
      log.info("Start executing method getAllLanguages.");
      List<String> allLanguages = Language.getAllLanguages();
      log.info("Finish executing method getAllLanguages with result: {}",
allLanguages);
      return allLanguages;
   }

   @Override
   @Transactional
   public void deleteCourse(UUID id) {
      log.info("Start executing method deleteCourse for course: {}", id);
      Course course = courseRepository.findById(id)
            .orElseThrow(() -> new CourseNotFoundException(id));
      courseRepository.delete(course);
      log.info("Finish executing method deleteCourse.");
   }

   @Override
```

```
    @Transactional
    public void create(CreateCourseDto dto) {
        log.info("Start adding new course: {}", dto);
        Source source = sourceRepository.findByNameIgnoreCase(dto.source())
                .orElseGet(() -> sourceRepository.save(new
Source(dto.source())));

        Course course = courseRepository.save(new Course(dto.title(),
                dto.description(),
                dto.imageUrl(),
                Language.findValueByName(dto.language()),
                Category.findValueByName(dto.category()),
                source));

        List<Lesson> lessons =
lessonRepository.saveAll(dto.videoUrls().stream()
                .map(videoUrl -> new Lesson(getLessonTitle(videoUrl), videoUrl,
course))
                .toList());
    }

    @Override
    @Transactional
    public void update(UpdateCourseDro dto) {
        log.info("Start updating course: {}", dto.title());
        Source source = sourceRepository.findByNameIgnoreCase(dto.source())
                .orElseGet(() -> sourceRepository.save(new
Source(dto.source())));
        Course course = courseRepository.findById(dto.id())
                .orElseThrow(() -> new CourseNotFoundException(dto.id()));
        course.setTitle(dto.title());
        course.setDescription(dto.description());
        course.setSource(source);
        course.setLanguage(Language.findValueByName(dto.language()));
        course.setCategory(Category.findValueByName(dto.category()));
        course.setImageUrl(dto.imageUrl());

        courseRepository.saveAndFlush(course);

        dto.lessonsToDelete().removeIf(Objects::isNull);
        lessonRepository.deleteAllById(dto.lessonsToDelete());

        dto.editedLessons().forEach(editedLesson -> {
                Optional<Lesson> lessonOptional =
lessonRepository.findById(editedLesson.id());
                if (lessonOptional.isPresent() &&
Objects.nonNull(editedLesson.newTitle())) {
                    Lesson lesson = lessonOptional.get();
                    lesson.setTitle(editedLesson.newTitle());
                    lessonRepository.saveAndFlush(lesson);
                }
            }
        );
        if (Objects.nonNull(dto.newLessons())) {
            lessonRepository.saveAll(dto.newLessons().stream()
                    .map(videoUrl -> new Lesson(getLessonTitle(videoUrl),
videoUrl, course))
                    .toList());
        }
    }

    private String getLessonTitle(String videoUrl) {
        return videoUrl.substring(videoUrl.lastIndexOf('/') + 1,
```

```
videoUrl.lastIndexOf('.'));
    }
}
```

## TokenServiceImpl.java

```
@Service
@Slf4j
@RequiredArgsConstructor
public class TokenServiceImpl implements TokenService {

    private final TokenIssuerProperties tokenIssuerProperties;

    @Override
    public void validateExpirationDate(String token) {
        log.info("Start validation token expiration date");

        Instant expiresAt = JWT.decode(token).getExpiresAt().toInstant();
        if (expiresAt.isBefore(Instant.now())) {
            throw new TokenExpiredException("Token has been expired",
expiresAt);
        }
        log.info("Finish validation token expiration date");
    }

    @Override
    public void validateToken(String token) {
        log.info("Start validation token signature");
        WebClient.create(tokenIssuerProperties.getBaseUrl())
                .get()
                .uri(uriBuilder ->
uriBuilder.path(tokenIssuerProperties.getTokenValidationEndpoint())
                        .queryParam("token", token)
                        .build())
                .retrieve()
                .onStatus(HttpStatus::is2xxSuccessful, response -> {
                    log.info("Validation passed, response: {}", response);
                    return Mono.empty();
                })
                .onStatus(HttpStatus::isError,
                        response -> {
                            if
(HttpStatus.UNAUTHORIZED.equals(response.statusCode())) {
                                Mono<ApiError> bodyToMono =
response.bodyToMono(ApiError.class);
                                return bodyToMono.flatMap(
                                        apiError -> Mono.error(
                                                new
VerificationTokenException(apiError.getMessage())
                                        )
                                );
                            } else {
                                return Mono.error(
                                        new RuntimeException("Error while verifying
token signature")
                                );
                            }
                        }).bodyToMono(Void.class)
                .block();
        log.info("Finish validation token signature");
    }
}
```

# ДОДАТОК Е

Файли Controller рівня в проекті

## ControllerExceptionHandler.java

```
@RestControllerAdvice
@Slf4j
public class ControllerExceptionHandler {

    private static final String CONSTRAINT_VIOLATION_MESSAGE = "Validation
failed";

    @ExceptionHandler(ConstraintViolationException.class)
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    public ValidationError
handleConstraintViolationException(ConstraintViolationException e) {
        Map<String, String> errors =
e.getConstraintViolations().stream().collect(
                Collectors.toMap(constraintViolation ->

StreamSupport.stream(constraintViolation.getPropertyPath().spliterator(),
false)
                                        .reduce((first, second) -> second)

.map(Path.Node::getName).orElse(null),
                        ConstraintViolation::getMessage));
        log.error("ConstraintViolationException handled due to errors: {}",
errors);
        return new ValidationError(HttpStatus.BAD_REQUEST.value(),
CONSTRAINT_VIOLATION_MESSAGE, errors);
    }

    @ExceptionHandler(MethodArgumentNotValidException.class)
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    public ValidationError
handleMethodArgumentNotValidException(MethodArgumentNotValidException e) {
        Map<String, String> errors = e.getFieldErrors().stream()
                .collect(Collectors.toMap(
                        FieldError::getField,
                        o ->
Optional.ofNullable(o.getDefaultMessage()).orElse("Message is unavailable")
                ));
        log.error("MethodArgumentNotValidException handled due to errors:
{}", errors);
        return new ValidationError(HttpStatus.BAD_REQUEST.value(),
CONSTRAINT_VIOLATION_MESSAGE, errors);
    }

    @ExceptionHandler(CourseNotFoundException.class)
    @ResponseStatus(HttpStatus.NOT_FOUND)
    public ApiError handleCourseNotFoundException(CourseNotFoundException e)
{
        log.error("CourseNotFoundException handled with error: {}",
e.getMessage());
        return new ApiError(HttpStatus.NOT_FOUND.value(), e.getMessage());
    }

   @ExceptionHandler(AccessDeniedException.class)
   @ResponseStatus(HttpStatus.FORBIDDEN)
```

```
    public ApiError handleAccessDeniedException(AccessDeniedException e) {
        log.error("AccessDeniedException handled with error: ", e);
        return new ApiError(HttpStatus.FORBIDDEN.value(), e.getMessage());
    }

     @ExceptionHandler(RuntimeException.class)
     @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
     public ApiError handleAllOtherException(RuntimeException e) {
         log.error("RuntimeException handled with error: ", e);
         return new ApiError(HttpStatus.INTERNAL_SERVER_ERROR.value(),
e.getMessage());
     }
}
```

## CourseController.java

```
@RestController
@RequiredArgsConstructor
@RequestMapping("/api/v1/course")
public class CourseController {

    private final CourseService courseService;

    @PreAuthorize("hasAnyAuthority('USER', 'ADMIN')")
    @PostMapping("/search")
    public PageResponse<CourseItemDto> searchCourses(@RequestBody
SearchRequest request) {
        return courseService.searchCourses(request);
    }

    @PreAuthorize("hasAnyAuthority('USER', 'ADMIN')")
    @GetMapping("/{id}")
    public CourseDetailsDto getCourseDetails(@PathVariable UUID id) {
        return courseService.getCourseDetails(id);
    }

    @PreAuthorize("hasAnyAuthority('USER', 'ADMIN')")
    @GetMapping("/newest")
    public List<NewestCourseDto> getNewestCourses() {
        return courseService.getNewestCourses();
    }

    @PreAuthorize("hasAnyAuthority('USER', 'ADMIN')")
    @GetMapping("/categories")
    public List<String> getAllCategories() {
        return courseService.getAllCategories();
    }

    @PreAuthorize("hasAnyAuthority('USER', 'ADMIN')")
    @GetMapping("/languages")
    public List<String> getAllLanguages() {
        return courseService.getAllLanguages();
    }

    @PreAuthorize("hasAuthority('ADMIN')")
    @DeleteMapping("/delete/{id}")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void deleteCourse(@PathVariable UUID id) {
        courseService.deleteCourse(id);
    }
```

```
    @PreAuthorize("hasAuthority('ADMIN')")
    @PostMapping("/create")
    @ResponseStatus(HttpStatus.CREATED)
    public void createCourse(@RequestBody CreateCourseDto dto) {
        courseService.create(dto);
    }

    @PreAuthorize("hasAuthority('ADMIN')")
    @PutMapping("/update")
    @ResponseStatus(HttpStatus.OK)
    public void updateCourse(@RequestBody UpdateCourseDro dto) {
        courseService.update(dto);
    }
}
```

# ДОДАТОК Ж

Інтерфейс бібліотеки Mapstruct в проекті

## CourseMapper.java

```
@Mapper(componentModel = "spring",
      unmappedTargetPolicy = ReportingPolicy.IGNORE)
public interface CourseMapper {

   CourseItemDto toDto(Course course);

   List<CourseItemDto> toCourseItemDtos(List<Course> courses);

   @Mapping(target = "language", source = "language.name")
   @Mapping(target = "category", source = "category.name")
   @Mapping(target = "source", source = "source.name")
   CourseDetailsDto toCourseDetailsDto(Course course);

   List<NewestCourseDto> toNewestCourseDtos(List<Course> newestCourse);

   @Mapping(target = "language", source = "language.name")
   @Mapping(target = "source", source = "source.name")
   @Mapping(target = "lessonsAmount",
         expression = """
               java(
                   newestCourse.getLessons().size()
               )
               """
   )
   NewestCourseDto toNewestCourseDto(Course newestCourse);
}
```

# ДОДАТОК И

Файли для конфігурації Spring Security

## SecurityConfig.java

```
@Configuration
@EnableWebSecurity
@EnableMethodSecurity
@RequiredArgsConstructor
public class SecurityConfig {

   private final JwtAuthenticationFilter jwtAuthenticationFilter;

   @Bean
   public SecurityFilterChain filterChain(HttpSecurity http) throws Exception
{
      http.csrf().disable()
            .cors().configurationSource(corsConfigurationSource());
      http.formLogin().disable()
            .httpBasic().disable()
            .sessionManagement(sm -> sm.sessionCreationPolicy(STATELESS))
            .addFilterBefore(jwtAuthenticationFilter,
UsernamePasswordAuthenticationFilter.class)
            .authorizeHttpRequests()
            .requestMatchers("/**")
            .permitAll();
      return http.build();
   }

   @Bean
   CorsConfigurationSource corsConfigurationSource() {
      CorsConfiguration configuration = new CorsConfiguration();
      configuration.setAllowedOriginPatterns(List.of("*"));
      configuration.setAllowedMethods(List.of("*"));
      configuration.setAllowedHeaders(List.of("*"));
      configuration.setAllowCredentials(true);
      configuration.setExposedHeaders(List.of("*"));
      UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();
      source.registerCorsConfiguration("/**", configuration);
      return source;
   }
}
```

## JwtAuthenticationFilter.java

```
@Slf4j
@Component
@RequiredArgsConstructor
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    private static final int TOKEN_START_INDEX = 7;

    private final TokenService tokenService;
    private final ObjectMapper objectMapper;

    @Override
    protected void doFilterInternal(@NonNull HttpServletRequest request,
```

```
                                        @NonNull HttpServletResponse response,
                                        @NonNull FilterChain filterChain) throws
ServletException, IOException {
        String authHeader = request.getHeader(AUTHORIZATION);

        if (Objects.isNull(authHeader) ||
!authHeader.trim().startsWith("Bearer")) {
            filterChain.doFilter(request, response);
            return;
        }

        String token = authHeader.trim().substring(TOKEN_START_INDEX);

        try {
            tokenService.validateExpirationDate(token);
            tokenService.validateToken(token);
        } catch (VerificationTokenException | TokenExpiredException |
JWTDecodeException e1) {
           log.error(e1.getMessage());
           response.setStatus(UNAUTHORIZED.value());
            response.getWriter().write(convertErrorToJson(UNAUTHORIZED,
e1.getMessage()));
            return;
        } catch (Exception e2) {
         log.error(e2.getMessage());
            response.setStatus(INTERNAL_SERVER_ERROR.value());

response.getWriter().write(convertErrorToJson(INTERNAL_SERVER_ERROR,
e2.getMessage()));
            return;
        }

        User user = new User(token);

        UsernamePasswordAuthenticationToken authToken = new
UsernamePasswordAuthenticationToken(
                user, null, user.getAuthorities()
        );
        authToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
        SecurityContextHolder.getContext().setAuthentication(authToken);
        filterChain.doFilter(request, response);
    }

    private String convertErrorToJson(HttpStatus status, String errorMessage)
throws JsonProcessingException {
        log.error("Validation token error: {}", errorMessage);
        return objectMapper.writeValueAsString(new ApiError(status.value(),
errorMessage));
    }
}
```

## User.java

```
@Getter
public class User implements UserDetails {

    private final String email;

    private final List<SimpleGrantedAuthority> authorities = new
ArrayList<>();
```

```
    public User(String token) {
        Map<String, Claim> claims = JWT.decode(token).getClaims();
        this.email = claims.get("sub").asString();
        setUpAuthorities(claims);
    }


    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return authorities;
    }

    @Override
    public String getPassword() {
        return null;
    }

    @Override
    public String getUsername() {
        return email;
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }

    private void setUpAuthorities(Map<String, Claim> claims) {
        String role = claims.get("role").asString();
      authorities.add(new SimpleGrantedAuthority(role));
    }
}
```

# ДОДАТОК К

Файли моделі для отримання деталей курсу

## course-details.ts

```
import {Lesson} from "./lesson";

export interface CourseDetails {
  title: string;
  description: string;
  imageUrl: string;
  language: string;
  category: string;
  source: string;
  createdAt: Date;
  updatedAt: Date;
  lessons: Lesson[];
}
```

## lesson.ts

```
export interface Lesson {
  id: string;
  title: string;
  videoUrl: string;
  newTitle: string;
}
```

# ДОДАТОК Л

Файл сервісу для керування курсами

## course.service.ts

```
@Injectable({providedIn: 'root'})
export class CourseService {
  constructor(private http: HttpClient) { }

  baseServerUrl = 'http://localhost:8090';

  getCourseDetails(courseId: string) {
    return
this.http.get<CourseDetails>(`${this.baseServerUrl}/api/v1/course/${courseId}
`);
  }

  getNewestCourses() {
    return
this.http.get<NewestCourse[]>(`${this.baseServerUrl}/api/v1/course/newest`);
  }

  public searchCourses(pageSpecification: PageSpecification):
Observable<Page<CourseItem>> {
    return
this.http.post<Page<CourseItem>>(`${this.baseServerUrl}/api/v1/course/search`
, pageSpecification);
  }

  public getAllCategories(): Observable<string[]> {
    return
this.http.get<string[]>(`${this.baseServerUrl}/api/v1/course/categories`);
  }

  public getAllLanguages(): Observable<string[]> {
    return
this.http.get<string[]>(`${this.baseServerUrl}/api/v1/course/languages`);
  }


  deleteCourse(courseId: string): Observable<void> {
    return
this.http.delete<void>(`${this.baseServerUrl}/api/v1/course/delete/${courseId
}`);
  }

  createCourse(newCourse: CreateCourse): Observable<void> {
    return this.http.post<void>(`${this.baseServerUrl}/api/v1/course/create`,
newCourse);
  }

  updateCourse(updatedCourse: UpdateCourse) {
    return this.http.put<void>(`${this.baseServerUrl}/api/v1/course/update`,
updatedCourse);
  }
}
```

# ДОДАТОК М

Файл класу-перехоплювача

## auth-interceptor.ts

```
const TOKEN_HEADER_KEY = 'Authorization';

@Injectable()
export class AuthInterceptor implements HttpInterceptor {

  constructor(private token: TokenStorageService) {
  }

  intercept(req: HttpRequest<any>, next: HttpHandler) {
    let authReq = req;
    const token = this.token.getToken();
    if (token != null) {
      authReq = req.clone({headers: req.headers.set(TOKEN_HEADER_KEY, 'Bearer
' + token)});
    }
    return next.handle(authReq);
  }
}

export const httpInterceptorProviders = [
  {provide: HTTP_INTERCEPTORS, useClass: AuthInterceptor, multi: true}
];
```