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

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

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня
бакалавр
(назва освітнього ступеня)

на тему: Розробка архітектури та програмного забезпечення API платформи онлайн-навчання з використанням технології ASP.NET Core

Виконав(ла): студент(ка) IV курсу, групи СП-42
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Каленюк Д.О (прізвище та ініціали)

Керівник (підпис) Цуприк Г.Б. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю.М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М.Р. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль 2026

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

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

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М.Р.
(підпис) (прізвище та ініціали)
« 6 » квітня 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Каленюку Дмитру Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка архітектури та програмного забезпечення API платформи онлайн-навчання з використанням технології ASP.NET Core

Керівник роботи Цуприк Галина Богданівна к.т.н., доц.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 06 » квітня 2026 року №

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи Предметна область, технічне завдання, вимоги та специфікація, наукові публікації та нормативно-довідкові джерела.

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. Розділ 1. Аналіз вимог до платформи онлайн-навчання. Розділ 2. Проєктування та розробка серверної частини платформи. Розділ 3. Тестування, розгортання, та верифікація системи. Розділ. 4. Безпека життєдіяльності, основи охорони праці. Висновок.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Ілюстративні зображення, інформативні зображення для доповнення тексту, діаграми, знімки екрану з проробленою роботою, слайди презентації до захисту: титульний слайд; постановка задачі; життєвий цикл розробки; функціональні вимоги та ролі користувачів; архітектура платформи модель даних; UML-діаграма класів; сценарій оформлення підписки; реалізовані модулі API; Swagger UI і Postman; тестування API; результати тестування та Docker Compose; висновки й напрями розвитку.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	к.т.н., доцент Мариненко С.Ю. к.т.н., доцент Гурик О.Я.		
Нормоконтроль	к.т.н., доцент каф. ПІ Стоянов Ю.М.		

7. Дата видачі завдання 6 квітня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Отримання завдання	06.04 – 12.04	виконано
2.	Аналіз завдання	12.04 – 13.04	виконано
3.	Виконання розділу 1 "Аналіз вимог до платформи онлайн-навчання"	13.04 – 20.04	виконано
4.	Виконання розділу 2 "Проєктування та розробка серверної частини платформи"	20.04 – 04.05	виконано
5.	Виконання розділу 3 "Тестування, розгортання та верифікація системи"	04.05 – 16.05	виконано
6.	Виконання розділу 4 "Безпека життєдіяльності, основи охорони праці"	16.05 – 20.05	виконано
7.	Оформлення пояснювальної записки	20.05 – 27.05	виконано
8.	Оформлення графічного та презентаційного матеріалу	27.05 – 02.06	виконано
9.	Перевірка на академічний плагіат, перевірка керівником та консультантами	03.06 – 07.06	виконано
10.	Попередній захист кваліфікаційної роботи бакалавра	08.06	виконано
11.	Захист кваліфікаційної роботи бакалавра		

Студент _____ Каленюк Д.О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Цуприк Г.Б.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота на тему «Розробка архітектури та програмного забезпечення API платформи онлайн-навчання з використанням технології ASP.NET Core». Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, Тернопіль, 2026. Робота містить 72 сторінки, 18 рисунків, 1 таблицю, 4 розділи, 29 використаних джерел та 3 додатки.

Ключові слова: програмне забезпечення API платформи онлайн-навчання, платформа онлайн-навчання, серверна частина, ASP.NET Core, PostgreSQL, Keycloak, JWT, Docker, тестування, підписки, навчальний контент.

У кваліфікаційній роботі виконано проєктування та розробку програмного забезпечення API платформи онлайн-навчання EducationPlatform.

У першому розділі проаналізовано предметну область онлайн-освіти, визначено межі системи, функціональні та нефункціональні вимоги.

У другому розділі описано архітектуру програмної системи, модель даних, UML-представлення основних компонентів, механізми автентифікації, авторизації та реалізацію функціональних модулів.

У третьому розділі розглянуто тестування, розгортання в Docker Compose та верифікацію API.

У четвертому розділі наведено питання безпеки життєдіяльності та охорони праці, пов'язані з використанням і супроводом API платформи онлайн-навчання.

Практичним результатом роботи є функціональне програмне забезпечення API платформи онлайн-навчання, яке підтримує керування курсами й уроками, роботу з підписками, завантаження медіафайлів, creator-доступ, захист навчального контенту та документування API через Swagger UI і Postman.

ABSTRACT

The qualification paper on the topic “Development of the Architecture and Software API for an Online Learning Platform Using ASP.NET Core Technology”. Ivan Puluj Ternopil National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, Ternopil, 2026. The work contains 72 pages, 18 figures, 1 table, 4 chapters, 29 references, and 3 appendices.

Keywords: online learning platform, course marketplace, server side, API, ASP.NET Core, PostgreSQL, Keycloak, JWT, Docker, testing, subscriptions, creator access, learning content.

The qualification work is devoted to the design and development of the server side of EducationPlatform, an online learning platform with paid course subscriptions and paid creator access.

The first chapter analyzes the domain of online education and defines the system boundaries, functional requirements, and non-functional requirements.

The second chapter describes the architecture of the software system, the data model, UML representations of the main components, authentication and authorization mechanisms, and the implementation of functional modules.

The third chapter considers testing, deployment using Docker Compose, and API verification.

The fourth chapter presents occupational health and safety issues related to the use and maintenance of the online learning platform.

The practical result of the work is a functional server-side prototype of an online learning platform that supports course and lesson management, subscription handling, media file uploading, paid creator access, learning content protection, and API documentation through Swagger UI and Postman.

ЗМІСТ

ВСТУП

У сучасних умовах цифрової трансформації освіти зростає потреба у програмних системах, які забезпечують зручний доступ до навчальних матеріалів і підтримують взаємодію між незалежними авторами курсів та користувачами, які хочуть опановувати нові навички онлайн. Платформи онлайн-навчання працюють як цифрові майданчики: автор отримує право створювати й публікувати курси, а студенти переглядають каталог, купують доступ або оформлюють підписку та проходять уроки у власному темпі.

Актуальність теми кваліфікаційної роботи зумовлена необхідністю розроблення програмного забезпечення API платформи онлайн-навчання, яка поєднує керування курсами, публікацію уроків, завантаження медіафайлів, контроль доступу до платного контенту та підтримку комерційних сценаріїв використання. Для такої системи важливими є надійна архітектура API, чітке розмежування ролей студента, автора й адміністратора, захист ресурсів, можливість масштабування та тестування бізнес-процесів купівлі доступу до курсів і отримання creator-доступу.

Метою кваліфікаційної роботи є проєктування та розробка програмного забезпечення API платформи онлайн-навчання, що забезпечує створення та адміністрування курсів і уроків, роботу з підписками на курси, контроль доступу до навчальних матеріалів, завантаження файлів і симуляцію платіжних процесів для двох ключових сценаріїв: придбання студентом доступу до курсу та придбання користувачем можливості стати автором курсів.

Для досягнення поставленої мети необхідно розв’язати такі задачі: проаналізувати вимоги до платформи онлайн-навчання та визначити основні ролі користувачів; сформувати функціональні й нефункціональні вимоги до API; спроєктувати архітектуру програмного забезпечення API та структуру бази даних; реалізувати модулі керування курсами, уроками, підписками та доступом авторів; забезпечити завантаження і отримання медіафайлів; реалізувати імітацію платіжного підтвердження для платних підписок і доступу автора; провести

тестування основних endpoint-ів і перевірити працездатність ключових сценаріїв системи.

Об’єктом дослідження є процес розроблення програмного забезпечення API платформи онлайн-навчання для онлайн-продажу й проходження навчальних курсів. Предметом дослідження є архітектурні рішення, моделі даних, endpoint-и, механізми авторизації, підписок, creator-доступу та симуляції оплати, які використовуються під час створення API платформи онлайн-навчання.

У процесі розробки використано мову програмування C#, платформу .NET, фреймворк ASP.NET Core для побудови REST API, Entity Framework Core для роботи з даними, PostgreSQL як систему керування базами даних, механізми JWT-автентифікації та Swagger для документування і перевірки API, а також Git і GitHub для контролю версій. Для забезпечення якості реалізації застосовано автоматизовані інтеграційні тести, які перевіряють роботу основних сценаріїв системи.

Практичне значення одержаних результатів полягає у створенні працездатного програмного забезпечення API платформи онлайн-навчання, який може бути використаний як основа для подальшої розробки веб- або мобільного клієнта. Реалізована система підтримує цикл роботи marketplace-платформи: користувач може отримати creator-доступ і створювати власні курси, інші користувачі можуть оформлювати підписки на ці курси, а доступ до уроків і відеоматеріалів контролюється залежно від ролі користувача, автора курсу та статусу підписки.

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків. У першому розділі розглядаються вимоги до програмної системи. У другому розділі подано проєктування та розробку системи. У третьому розділі описано тестування, впровадження та підтримку. Четвертий розділ присвячено питанням безпеки життєдіяльності та основам охорони праці.

1 АНАЛІЗ ВИМОГ ДО ПЛАТФОРМИ ОНЛАЙН-НАВЧАННЯ

У першому розділі кваліфікаційної роботи розглянуто предметну область онлайн-освіти та визначено вимоги до серверної частини платформи онлайн-навчання marketplace-типу. Особливу увагу приділено ролям студента, автора курсу й адміністратора, сценаріям створення курсів і уроків, оформленню підписок, купівлі creator-доступу, контролю доступу до навчальних матеріалів і платіжним сценаріям, які у межах роботи реалізуються через контрольовану симуляцію оплати.

Результатом аналізу є формування функціональних і нефункціональних вимог, що визначають межі програмного забезпечення API, її основні модулі та очікувану поведінку API. Ці вимоги є основою для подальшого проєктування архітектури, структури бази даних, endpoint-ів і механізмів авторизації, які розглядаються у наступних розділах роботи.

1.1 Аналіз платформ онлайн-навчання

Онлайн-освіта є одним із напрямів цифровізації, який активно розвивається завдяки поширенню вебтехнологій, хмарної інфраструктури та дистанційних форматів навчання. У межах цієї роботи платформа онлайн-навчання розглядається не як внутрішня система навчального закладу, а як програмний майданчик для авторів і студентів: автори після отримання creator-доступу створюють курси, а студенти купують або оформлюють доступ до конкретних курсів і проходять уроки онлайн [4].

На ринку існують різні типи платформ для онлайн-навчання. Найближчою до розроблюваної системи є модель marketplace-платформи, прикладом якої є Udemy: автори розміщують власні курси, а користувачі обирають потрібний курс у каталозі, оплачують доступ і проходять уроки. Такі рішення поєднують каталог курсів, особисті кабінети, відеоматеріали, механізми монетизації та розмежування ролей між студентами, авторами й адміністраторами.

Іншу групу становлять системи керування навчанням, наприклад Moodle, які частіше використовуються навчальними закладами або організаціями для внутрішнього освітнього процесу. У цій кваліфікаційній роботі така модель не є основною, оскільки розроблювана система орієнтована на відкритіший сценарій: користувач може стати автором після активації creator-доступу, створити курс, а інші користувачі можуть придбати доступ до цього курсу.

Аналіз існуючих рішень показує, що для marketplace-платформи онлайн-навчання ключовими є каталог курсів, керування навчальними матеріалами, ролі користувачів, обмеження доступу до платного контенту, підтримка медіафайлів, механізми підписок і окремий сценарій отримання авторських прав. Водночас під час розроблення власної системи важливо не копіювати надлишковий функціонал великих платформ, а сформувати чітке ядро API, яке забезпечить базові бізнес-сценарії та залишить можливість подальшого масштабування.

Для розроблюваного програмного забезпечення API ключовим є серверний рівень, який надає клієнтським застосункам доступ до даних і бізнес-логіки. Такий підхід дозволяє відокремити користувацький інтерфейс від серверної частини та в майбутньому підключати до платформи різні типи клієнтів: вебінтерфейс, мобільний застосунок або адміністративну панель. Саме тому предметом розробки є не повноцінний вебсайт, а архітектура та програмне забезпечення API платформи онлайн-навчання.

1.2 Постановка задачі та межі програмного забезпечення

Розроблювана програмна система призначена для підтримки роботи платформи онлайн-навчання marketplace-типу, у якій користувач може придбати creator-доступ, створювати власні курси та публікувати уроки, а студенти можуть оформлювати підписки на курси інших авторів і проходити навчальні матеріали. Система повинна забезпечувати збереження структурованої інформації про курси, уроки, користувачів, підписки, статуси публікації, creator-доступ і медіафайли, а також надавати захищені HTTP endpoint-и для взаємодії з цими даними.

У межах кваліфікаційної роботи реалізується програмне забезпечення API. До меж системи входять модулі керування курсами, керування уроками, завантаження зображень і відео, оформлення підписок на курси, симуляція підтвердження оплати за підписку, придбання creator-доступу для створення курсів, авторизація користувачів і перевірка доступу до захищеного контенту. Клієнтський інтерфейс не є основним об’єктом розробки, однак API проєктується таким чином, щоб у майбутньому його можна було використати для створення повноцінного веб- або мобільного застосунку.

Основними ролями системи є студент, автор курсу та адміністратор. Студент переглядає доступні курси, оформлює підписку на обраний курс і отримує доступ до уроків після активації підписки. Автор курсу - це користувач, який отримав creator-доступ, після чого може створювати курси, додавати уроки, завантажувати навчальні матеріали та керувати контентом, який належить йому. Адміністратор має розширені права та може виконувати дії, пов’язані з підтвердженням симульованої оплати, зокрема активувати платні підписки та creator-доступ.

Окремою задачею є підтримка платних сценаріїв, характерних для marketplace-платформи. Перший сценарій стосується студента, який оформлює підписку на курс і після підтвердження отримує доступ до уроків. Другий сценарій стосується користувача, який купує creator-доступ і після активації може створювати власні курси. У межах кваліфікаційної роботи ці процеси реалізовано через симуляцію оплати: підписка або заявка на creator-доступ створюється зі статусом очікування, а після умовного підтвердження адміністратором переходить в активний стан.

1.3 Функціональні вимоги до платформи

Функціональні вимоги визначають дії, які система повинна виконувати для забезпечення основних сценаріїв роботи платформи онлайн-навчання. Насамперед система повинна надавати можливість створювати курси, отримувати список курсів, переглядати курс за ідентифікатором, оновлювати інформацію про курс і

видаляти курс. Для кожного курсу мають зберігатися назва, короткий опис, детальний опис, мова, ціна, теги, статус публікації та ідентифікатор автора.

Система повинна підтримувати роботу з уроками в межах курсу. Автор має мати змогу створювати уроки, задавати назву, короткий опис, основний текст уроку, тривалість, тип медіа та статус публікації. Користувач із відповідним доступом повинен мати можливість отримувати список уроків курсу та переглядати окремий урок за ідентифікатором. Видалення уроків має бути доступним лише користувачам, які мають право керувати відповідним навчальним контентом.

Важливою функціональною вимогою є підтримка медіафайлів. Для курсу повинна бути доступна можливість завантаження зображення попереднього перегляду, а для уроку - завантаження відеофайлу. Система повинна зберігати інформацію про тип файлу та шлях до нього, а також надавати endpoint-и для отримання зображення курсу та відео уроку. Доступ до відеоматеріалів має бути захищений, оскільки вони є частиною навчального контенту, який може бути доступний лише після оформлення підписки.

Модуль підписок повинен забезпечувати створення підписки на опублікований курс. Якщо курс є безкоштовним, підписка може одразу отримувати активний статус. Якщо курс є платним, підписка створюється зі статусом очікування оплати. До моменту активації така підписка не повинна надавати доступ до захищених уроків і відеоматеріалів. Після симульованого підтвердження платежу адміністратором підписка переходить в активний стан.

Система також повинна підтримувати окремий flow придбання доступу автора курсів. Звичайний користувач може створити заявку на купівлю creator-доступу, яка отримує статус очікування. Після підтвердження симульованої оплати адміністратором користувач набуває права створювати курси та керувати власним навчальним контентом. Це дозволяє відокремити звичайну студентську роль від ролі автора, не покладаючись лише на зовнішні ролі з системи автентифікації.

Механізм авторизації повинен розмежовувати доступ до endpoint-ів відповідно до ролі користувача та його зв’язку з ресурсом. Адміністратор має мати

повний доступ до службових операцій. Автор курсу повинен мати доступ до керування власними курсами та уроками. Студент повинен мати доступ до перегляду доступного контенту лише за умови активної підписки або якщо контент є доступним відповідно до правил системи.

1.4 Нефункціональні вимоги до програмного забезпечення

Нефункціональні вимоги визначають якісні характеристики системи, які впливають на її придатність до використання, супроводу та подальшого розвитку. Для розроблюваної платформи онлайн-навчання важливими є надійність, безпека, масштабованість, підтримуваність, тестованість і зрозумілість API для майбутніх клієнтських застосунків [4].

Вимога безпеки полягає у тому, що доступ до захищених endpoint-ів повинен надаватися лише автентифікованим користувачам. Операції створення та редагування курсів мають бути обмежені для авторів та адміністраторів. Доступ до уроків і відео повинен перевірятися на основі активної підписки, належності курсу автору або ролі адміністратора. Такий підхід зменшує ризик несанкціонованого доступу до платного навчального контенту [21].

Вимога надійності передбачає коректну обробку типових помилкових ситуацій. Якщо курс, урок, підписка або заявка на creator-доступ не знайдені, система повинна повертати відповідний HTTP-статус. Якщо користувач намагається повторно оформити вже наявну підписку або активувати вже активний доступ, система повинна повідомляти про конфлікт станів. Така поведінка робить API передбачуваним для клієнтських застосунків.

Вимога підтримуваності реалізується через поділ проєкту на окремі рівні відповідальності. HTTP endpoint-и повинні відповідати за приймання запитів і формування відповідей, application-рівень - за бізнес-логіку, data-рівень - за роботу з сутностями та базою даних, а common-рівень - за спільні моделі та перелічення. Такий поділ спрощує внесення змін, тестування окремих сценаріїв і подальше розширення системи [5, 7, 8].

Вимога тестованості полягає у тому, що основні бізнес-сценарії повинні бути перевірені автоматизованими тестами. До таких сценаріїв належать створення, отримання, оновлення та видалення курсів, створення уроків, перевірка доступу до уроків і відео, створення підписок, активація платних підписок, купівля creator-доступу та перевірка зміни прав користувача після активації такого доступу.

Вимога масштабованості передбачає, що API має бути спроєктоване таким чином, щоб у майбутньому можна було додати нові модулі без суттєвої перебудови системи. Потенційними напрямами розвитку є додавання повноцінного клієнтського інтерфейсу, інтеграція з реальним платіжним провайдером, розширення ролей користувачів, додавання тестів і завдань до уроків, а також побудова аналітики проходження курсів.

2 ПРОЄКТУВАННЯ ТА РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ ПЛАТФОРМИ

У другому розділі кваліфікаційної роботи розглянуто процес проєктування та розробки програмної системи платформи онлайн-навчання. На основі вимог, сформульованих у першому розділі, визначено процес розробки, архітектуру системи, структуру бази даних, UML-моделі, технологічний стек, основні класи й методи, а також інтерфейс взаємодії користувача із системою.

Оскільки розроблюване програмне забезпечення є API платформи онлайн-навчання, основний акцент зроблено на бізнес-логіці, моделі даних, автентифікації, авторизації та роботі з навчальним контентом. Клієнтський інтерфейс у межах роботи представлено через Swagger UI, Postman collection та опис майбутнього вебклієнта, який може взаємодіяти з реалізованим API.

2.1 Обґрунтування підходу до розробки

Проєктування платформи онлайн-навчання виконувалося з урахуванням того, що система повинна підтримувати декілька різних груп користувачів, роботу з навчальним контентом, обмеження доступу до платних матеріалів і можливість подальшого розширення. Тому основним завданням на етапі проєктування було не лише реалізувати окремі endpoint-и, а побудувати цілісну серверну частину відповідно до принципів Clean Architecture, у якій бізнес-логіка, доступ до даних, автентифікація та інфраструктурні компоненти мають чітко визначені межі відповідальності [5, 6].

Для контролю версій програмного забезпечення використовувалась система Git, а віддалене зберігання репозиторію та керування змінами здійснювалось за допомогою платформи GitHub. Це дозволило відстежувати історію змін, безпечно експериментувати з новим функціоналом через окремі гілки та забезпечити надійне зберігання вихідного коду проєкту.

Для розробки було обрано ітеративний підхід. Він передбачає поступове уточнення вимог, реалізацію окремих функціональних блоків, перевірку їх

працездатності та подальше розширення системи. Такий підхід є доцільним для платформи онлайн-навчання, оскільки функції курсів, уроків, підписок, завантаження медіафайлів і контролю доступу тісно пов’язані між собою, але можуть розроблятися окремими завершеними етапами.

На першому етапі було визначено базові ролі користувачів системи:

- студент переглядає доступні курси, оформлює підписку та отримує доступ до навчальних матеріалів;
- автор курсу створює та редагує курси, додає уроки, завантажує preview-зображення та відеоматеріали;
- адміністратор виконує службові операції, зокрема може підтверджувати платні підписки або надавати користувачу доступ автора;

Після визначення ролей були сформовані основні бізнес-сценарії системи: створення курсу, публікація навчальних матеріалів, створення уроку, завантаження медіафайлів, оформлення підписки, активація платного доступу, отримання захищеного уроку та активація creator-доступу. Саме ці сценарії стали основою для побудови API та структури модулів у проєкті.

Розробка виконувалася від загальної структури системи до конкретних модулів. Спочатку було сформовано solution з окремими проєктами для HTTP-рівня, прикладної логіки, доступу до даних, міграцій PostgreSQL і спільних моделей. Далі поступово реалізовувалися модулі курсів, уроків, підписок, creator-доступу, файлового сховища та інтеграції з Keycloak.

Важливою перевагою такого підходу є можливість перевіряти кожен новий сценарій окремо. Наприклад, після реалізації курсів і уроків можна було перевірити базову роботу з навчальним контентом, а після додавання підписок - перевірити обмеження доступу до матеріалів. Це зменшило ризик накопичення помилок і дозволило поступово ускладнювати систему без повної перебудови її архітектури. Послідовність ітеративного процесу проєктування та розробки системи наведено на рисунку 2.1.

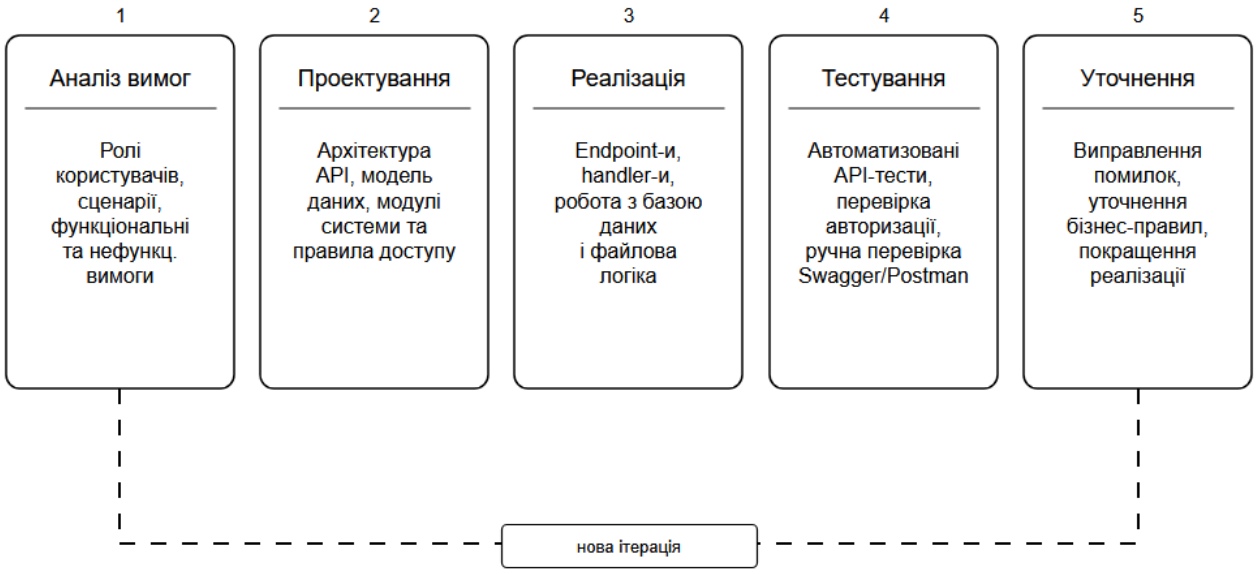

Рисунок 2.1 – Ітеративний процес проєктування та розробки системи

Наведена схема показує, що розробка виконувалася не як одноразове створення всіх модулів, а як послідовне уточнення вимог, реалізація окремих функцій і перевірка отриманого результату. Такий підхід дав змогу поступово сформувати стабільну серверну частину платформи та зменшити ризик помилок під час додавання нових бізнес-сценаріїв.

2.2 Архітектура програмного забезпечення

Архітектура EducationPlatform побудована за багаторівневим принципом із використанням ідей Clean Architecture. Такий підхід дозволяє розділити відповідальність між прийманням HTTP-запитів, виконанням бізнес-логіки, роботою з базою даних, спільними моделями та інфраструктурними сервісами. Залежності спрямовані від зовнішніх рівнів до прикладної логіки та моделей, тому код системи є більш структурованим, а зміни в інфраструктурі не повинні безпосередньо впливати на бізнес-правила [5, 6].

Проєкт AspNetCore є вхідною точкою програмного забезпечення API. Система реалізована у вигляді REST API, що взаємодіє з клієнтськими застосунками через HTTP-запити та повертає дані у форматі JSON. У ньому

налаштовується вебзастосунок, реєструються залежності, підключаються Swagger, автентифікація, авторизація, файлове сховище та endpoint-и. Саме цей рівень відповідає за взаємодію із зовнішнім клієнтом через HTTP та за перетворення запитів користувача у команди або запити прикладного рівня [10-14, 17].

Проєкт Application містить прикладну логіку системи і виступає центральним рівнем реалізації бізнес-сценаріїв відповідно до Clean Architecture. У ньому розміщені request-моделі, command-и, result-моделі та handler-и. Endpoint-и не повинні містити складної логіки самостійно: вони приймають запит, виконують базову валідацію та передають керування відповідному handler-у через MediatR. Це полегшує тестування та підтримку системи.

Проєкт Data відповідає за модель даних і містить EducationDbContext та сутності предметної області. У ньому описані курси, уроки, підписки, автори та покупки creator-доступу. Проєкт PostgreSql містить міграції Entity Framework Core і налаштування підключення до PostgreSQL. Таке розділення дозволяє відокремити опис доменної моделі від конкретного механізму застосування міграцій [15, 18].

Проєкт Common містить спільні перелічення та моделі, які використовуються різними частинами системи. До них належать статуси публікації, статуси підписок, статуси creator-доступу та типи медіаконтенту. Винесення цих типів в окремий проєкт дозволяє уникнути дублювання та підтримувати єдину систему станів у всіх модулях.

До зовнішніх та інфраструктурних компонентів системи належать:

- PostgreSQL, що використовується для зберігання структурованих даних платформи онлайн-навчання.
- Keycloak, який виконує роль сервера автентифікації та видає JWT-токени користувачам.
- Локальне файлове сховище, призначене для preview-зображень курсів і відеофайлів уроків.
- Swagger UI, який забезпечує технічний інтерфейс для перегляду та виконання API-запитів.

● Postman collection, що використовується для ручної перевірки послідовних бізнес-сценаріїв.

Типовий потік обробки запиту виглядає так: клієнт надсилає HTTP-запит до API, middleware автентифікації перевіряє JWT-токен, authorization policy визначає права користувача, endpoint формує command або request, MediatR передає його handler-у, handler працює з базою даних або файловим сховищем, після чого результат повертається клієнту у вигляді HTTP-відповіді. Загальну архітектуру програмної системи та взаємодію її основних компонентів наведено на рисунку 2.2 [16,20].

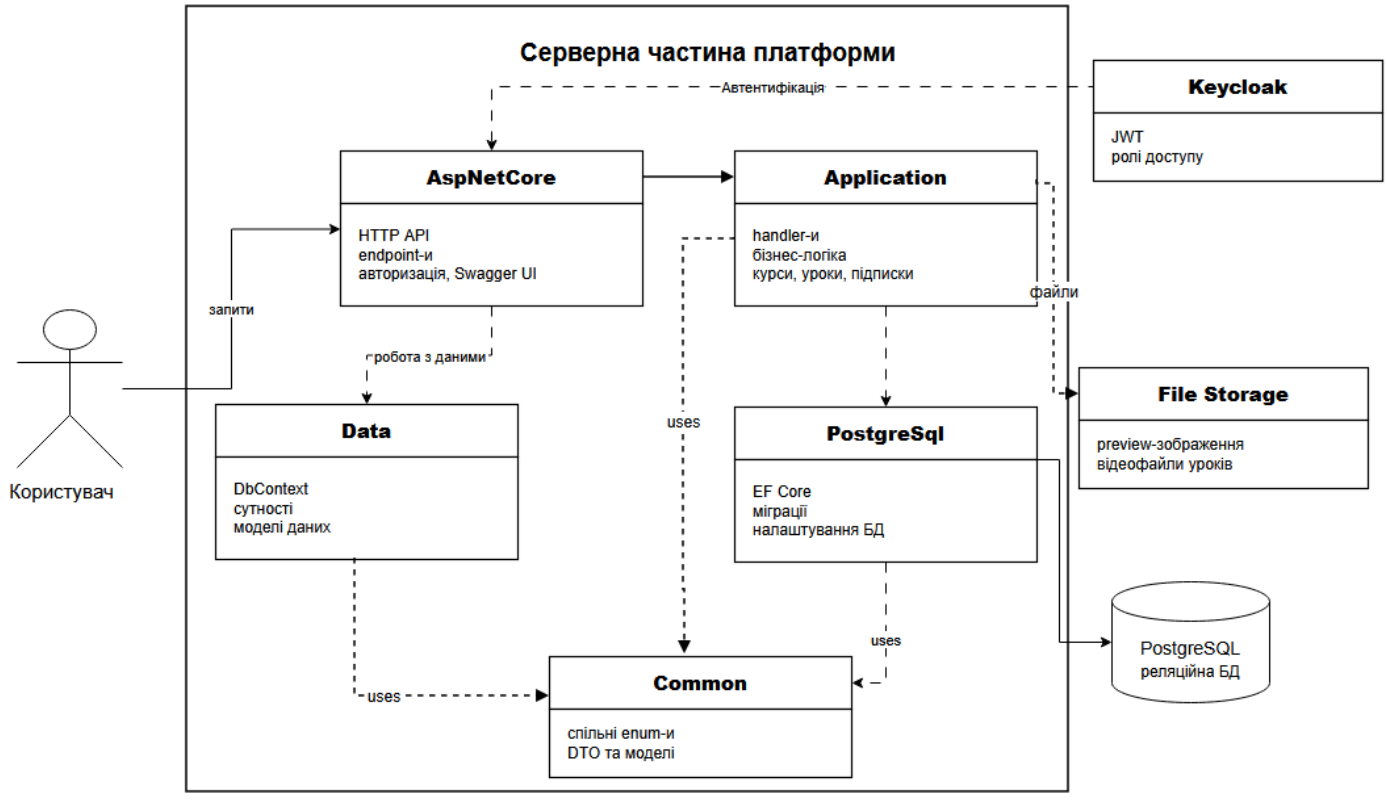

Рисунок 2.2 – Загальна архітектура розроблюваного програмного забезпечення

Зображена архітектура демонструє розподіл відповідальності між рівнями системи та показує, як HTTP-запити проходять від зовнішнього інтерфейсу до прикладної логіки й рівня доступу до даних. Завдяки такій структурі окремі

компоненти можна змінювати або розширювати без повної перебудови програмного забезпечення.

2.3 Модель даних EducationPlatform

Модель даних платформи онлайн-навчання побудована навколо навчального курсу як центрального об’єкта системи. Курс об’єднує навчальні уроки, має автора, статус публікації, ціну, описові характеристики та preview-зображення. Саме навколо курсу формується значна частина бізнес-логіки системи: оформлення підписок студентів, перевірка доступу до уроків, керування навчальним контентом, а також взаємодія з механізмами публікації та авторського доступу.

Для зберігання даних у системі використовується PostgreSQL, а взаємодія з базою даних виконується за допомогою Entity Framework Core. У програмному коді структура бази даних описана через сутності CourseEntity, LessonEntity, SubscriptionEntity, CreatorEntity та CreatorAccessPurchaseEntity. Контекст EducationDbContext містить DbSet для кожної з цих сутностей і застосовує конфігурації моделі під час формування схеми бази даних [15, 18].

Таблиця courses зберігає інформацію про навчальні курси. Вона містить ідентифікатор курсу, ідентифікатор автора, назву, короткий і повний опис, мову, статус публікації, вартість, теги, дати створення та оновлення, а також атрибути, пов’язані з preview-зображенням. Наявність статусу публікації дає змогу відокремити чернетки від курсів, доступних студентам, а також реалізувати проміжні стани життєвого циклу навчального контенту.

Таблиця lessons зберігає уроки, що належать до конкретного курсу. Для кожного уроку фіксується ідентифікатор курсу, ідентифікатор автора, назва, короткий опис, повний опис, текстовий вміст, статус публікації, тривалість, тип медіа та інформація про відеофайл. Зв’язок між таблицями courses і lessons є зв’язком один-до-багатьох: один курс може містити багато уроків, але кожен урок належить лише одному курсу. Така структура безпосередньо підтримує логіку побудови навчальної програми як набору послідовних занять.

Таблиця subscriptions описує підписки користувачів на курси. Вона містить посилання на курс і користувача, час створення підписки, ознаку безкоштовності, суму та поточний статус. Якщо курс є безкоштовним, підписка може одразу отримати статус Active. Для платного курсу підписка зазвичай створюється зі статусом Pending, а після підтвердження оплати переходить у статус Active. Такий підхід дозволяє коректно відобразити життєвий цикл придбання доступу до навчального контенту.

Таблиця creators використовується для фіксації користувачів, які мають право створювати навчальні курси. Це важливо для реалізації creator-access flow, оскільки користувач може отримати право автора не лише через роль у Keycloak, а й через локальне підтвердження в базі даних. Такий підхід забезпечує додаткову гнучкість і дає змогу відокремити зовнішню автентифікацію від внутрішньої бізнес-логіки системи.

Таблиця creator_access_purchases зберігає заявки або покупки доступу автора. У ній фіксуються користувач, момент створення заявки, сума та її поточний статус. Після активації такого запису користувач може отримати можливість створювати власні навчальні курси, що узгоджується із загальною логікою платформи.

Проєктування бази даних також враховує правила контролю доступу. Наприклад, перевірка доступу до уроку потребує використання зв’язків між lessons, courses і subscriptions, оскільки система повинна визначити, чи є користувач автором курсу, адміністратором або власником активної підписки. Модель даних не лише зберігає інформацію, а й безпосередньо підтримує реалізацію бізнес-правил захисту навчального контенту.

Структуру основних таблиць, їхні ключові атрибути та зв’язки між ними наведено на рисунку 2.3.

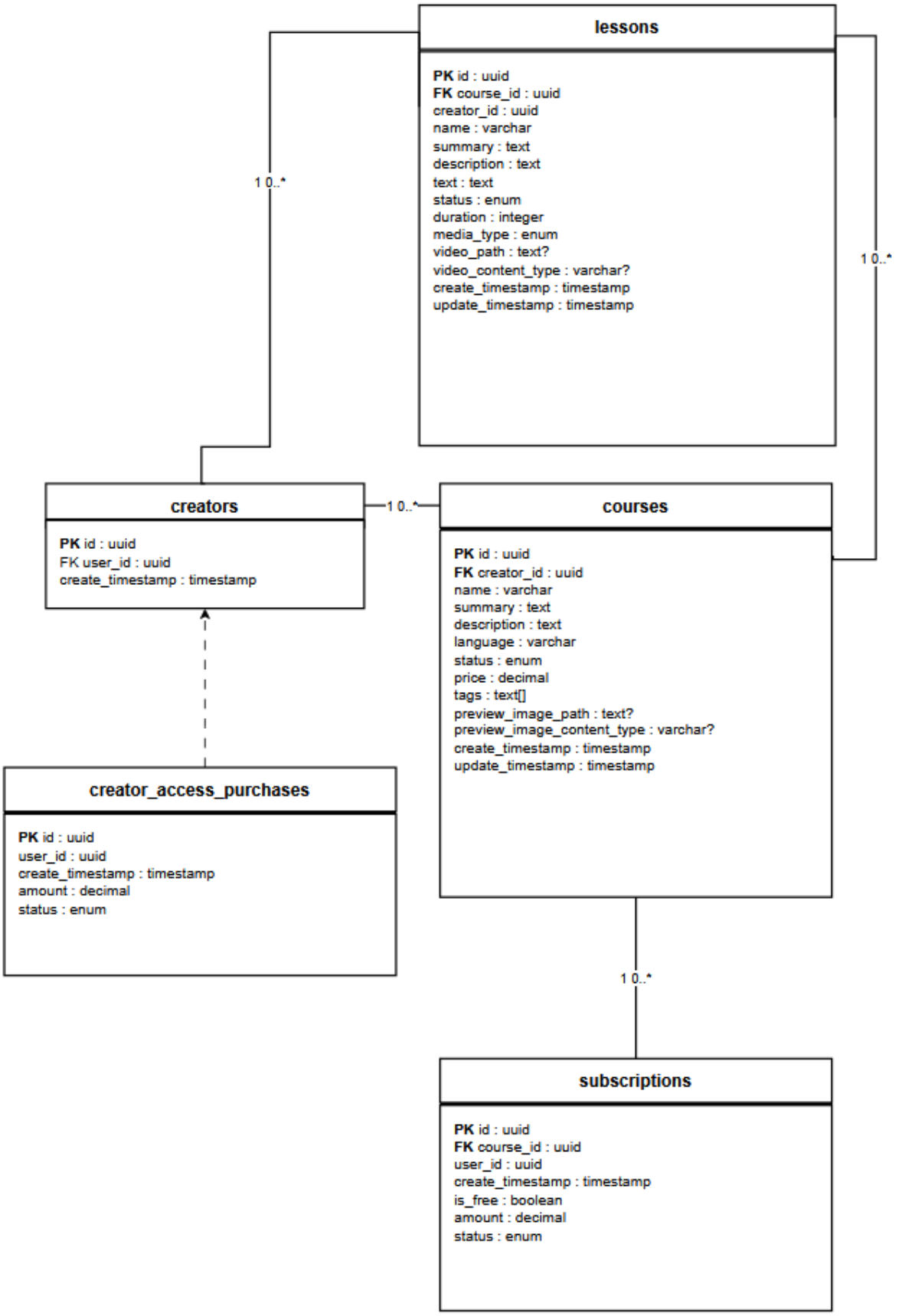

Рисунок 2.3 – ER-діаграма бази даних платформи онлайн-навчання

ER-діаграма уточнює, які сутності є основою зберігання даних у системі та як вони пов’язані між собою. Вона також показує, що база даних підтримує не лише навчальний контент, а й комерційні сценарії платформи, зокрема підписки та отримання creator-доступу.

2.4 UML-моделювання модулів і механізмів доступу

Для більш повного опису програмної структури системи використано UML-діаграму класів. На відміну від ER-діаграми, яка відображає таблиці бази даних, їхні поля та зв’язки на рівні схеми зберігання даних, UML-діаграма класів показує, як основні об’єкти предметної області представлені у програмному коді. Це дозволяє уточнити склад сутностей, їхні властивості, навігаційні зв’язки та роль контексту бази даних у роботі серверної частини платформи.

На UML-діаграмі центральним класом є CourseEntity, який представляє навчальний курс. Він містить ідентифікатор курсу, ідентифікатор автора, назву, короткий опис, повний опис, мову, статус публікації та ціну. Також у класі визначено навігаційні властивості Lessons і Subscriptions, які відображають зв’язок курсу з уроками та підписками користувачів. Це означає, що один курс може містити багато уроків і мати багато підписок.

Клас LessonEntity описує окремий урок у межах навчального курсу. Він містить ідентифікатор уроку, ідентифікатор курсу, ідентифікатор автора, назву, короткий опис, основний текстовий вміст, статус публікації, тривалість, тип медіа та шлях до відеофайлу. Зв’язок між CourseEntity і LessonEntity відображає структуру навчального матеріалу: один курс може складатися з багатьох уроків, але кожен урок належить лише одному курсу.

Клас SubscriptionEntity використовується для опису підписки користувача на курс. Він містить ідентифікатор підписки, ідентифікатор користувача, ідентифікатор курсу, дату створення, ознаку безкоштовної підписки, суму та статус. Наявність властивості Course дозволяє отримати інформацію про курс, до

якого належить підписка. Саме ця сутність використовується під час перевірки доступу користувача до навчального контенту.

Окремо на діаграмі показано клас CreatorEntity, який описує користувача, що має право створювати навчальні курси. Він містить ідентифікатор запису, ідентифікатор користувача, біографію, досвід, напрями експертизи та мови. Поле UserId у цьому класі не є посиланням на локальну таблицю користувачів, оскільки облікові записи користувачів зберігаються в Keycloak. У локальній базі даних зберігається лише ідентифікатор користувача, необхідний для зв’язку бізнес-даних із конкретним обліковим записом.

Клас CreatorAccessPurchaseEntity відповідає за збереження заявки або покупки доступу автора. Він містить ідентифікатор запису, ідентифікатор користувача, дату створення, суму та статус. Після активації такого запису користувач може отримати creator-доступ, що на діаграмі показано залежністю між CreatorAccessPurchaseEntity та CreatorEntity. Такий підхід дозволяє відокремити процес придбання доступу автора від фактичного надання користувачеві прав на створення курсів.

Важливе місце на діаграмі займає клас EducationDbContext, який є контекстом бази даних Entity Framework Core. Він містить набори сутностей Courses, Lessons, Subscriptions, Creators та CreatorAccessPurchases. Через ці властивості серверна частина отримує доступ до відповідних таблиць бази даних, виконує читання, створення, оновлення та видалення записів. Метод OnModelCreating використовується для налаштування моделі даних, зв’язків між сутностями та правил формування схеми бази даних.

Пунктирні залежності з позначенням «use» показують, що EducationDbContext використовує відповідні сутності як частину моделі даних. Це означає, що класи предметної області не існують ізольовано, а об’єднані в єдину модель доступу до даних. Суцільні зв’язки між сутностями відображають основні навігаційні відношення: курс пов’язаний з уроками та підписками, а також має автора, який визначається через відповідний ідентифікатор користувача.

Таким чином, UML-діаграма класів демонструє програмне представлення основних об’єктів системи онлайн-навчання. Вона доповнює ER-діаграму, оскільки показує не лише структуру збереження даних, а й те, як ці дані представлені у вигляді класів, властивостей, навігаційних зв’язків і контексту Entity Framework Core. Узагальнену UML-діаграму основних класів системи наведено на рисунку 2.4.

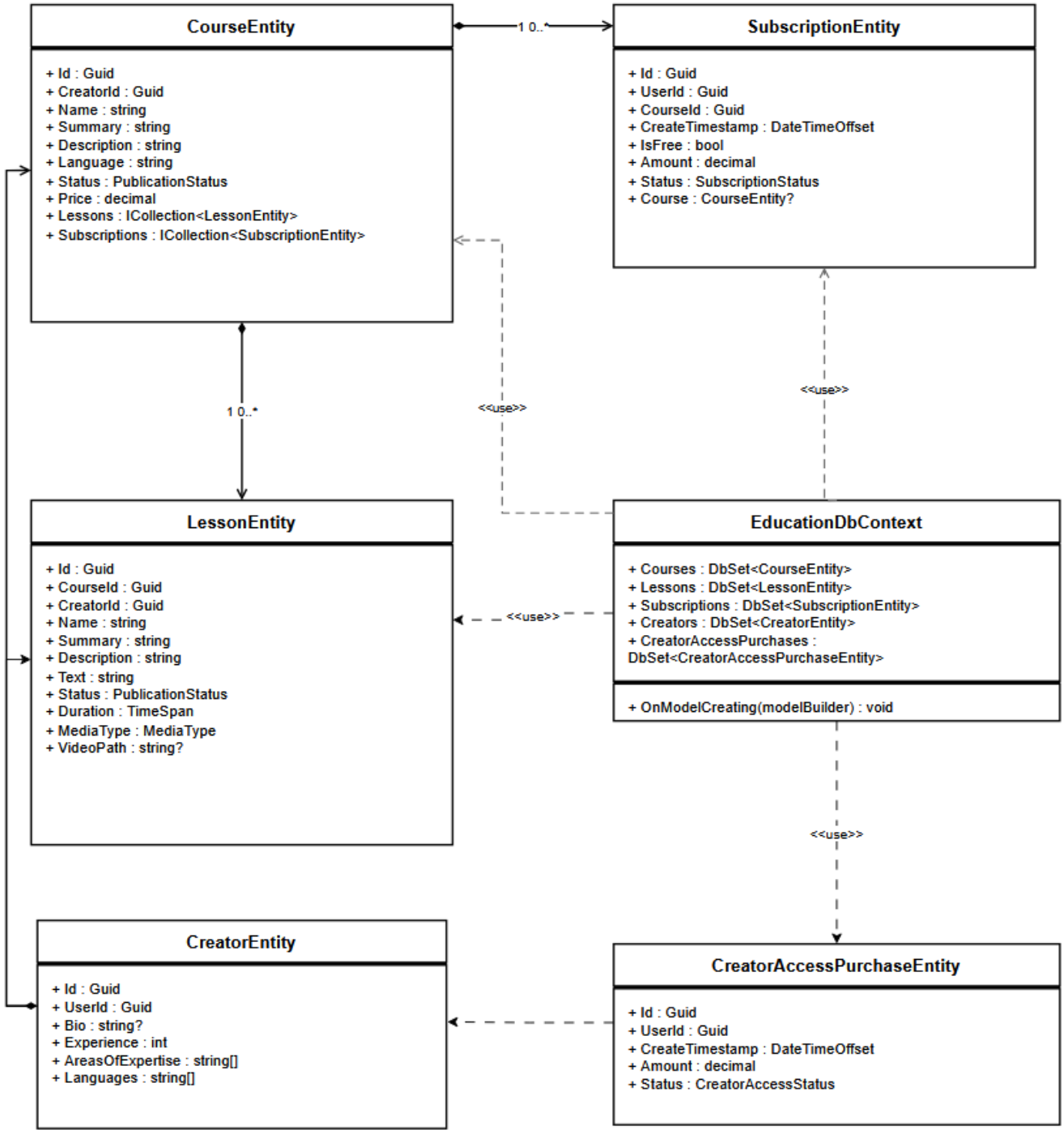

Рисунок 2.4 – UML-діаграма основних класів системи

Окрему увагу при проєктуванні системи приділено механізмам автентифікації та авторизації. У системі використовується Keycloak як зовнішній сервер ідентифікації. Користувач проходить автентифікацію в Keycloak, після чого отримує JWT-токен. API перевіряє токен за допомогою механізму JWT Bearer authentication, звіряє issuer, audience та інші параметри токена, а також перетворює ролі з Keycloak у role claims середовища ASP.NET Core [16, 20].

Авторизація реалізована через політики доступу. У системі визначено політики CreatorOnly, AdminOnly та StudentOrCreatorOnly. Політика CreatorOnly застосовується до операцій, пов’язаних зі створенням і редагуванням навчального контенту. AdminOnly використовується для службових і адміністративних операцій. Політика StudentOrCreatorOnly доцільна в тих сценаріях, де доступ повинен бути дозволений студенту або автору залежно від конкретного контексту виконання операції.

Для реалізації creator-доступу використовується власний authorization requirement, який перевіряє не лише роль користувача, а й наявність запису в таблиці creators. Це дозволяє підтримати ситуацію, коли користувач отримує право автора після активації покупки creator-доступу, навіть якщо роль creator не була вручну призначена в Keycloak. Такий механізм робить систему більш гнучкою та узгоджує зовнішню модель ролей із внутрішнім станом платформи.

Перевірка доступу до уроків базується на поєднанні ролей і стану підписок. Користувач може переглядати захищений урок, якщо він є адміністратором, автором відповідного курсу або має активну підписку на курс. Якщо підписка має статус Pending або взагалі відсутня, система не відкриває захищений навчальний матеріал. Таким чином, контроль доступу реалізується як поєднання механізмів автентифікації, авторизаційних політик та перевірки актуального стану предметних сутностей.

Для відображення взаємодії користувачів із системою доцільно використати діаграму варіантів використання. Вона дозволяє показати, які саме функції доступні основним акторам системи - студенту, автору курсу та адміністратору. Студент може переглядати каталог курсів, оформлювати підписку та переглядати

доступні уроки. Автор курсу працює зі створенням і редагуванням курсів, додаванням уроків і завантаженням медіафайлів. Адміністратор бере участь у службових сценаріях, пов’язаних з активацією підписок та наданням creator-доступу.

Відповідну діаграму варіантів використання наведено на рисунку 2.5.

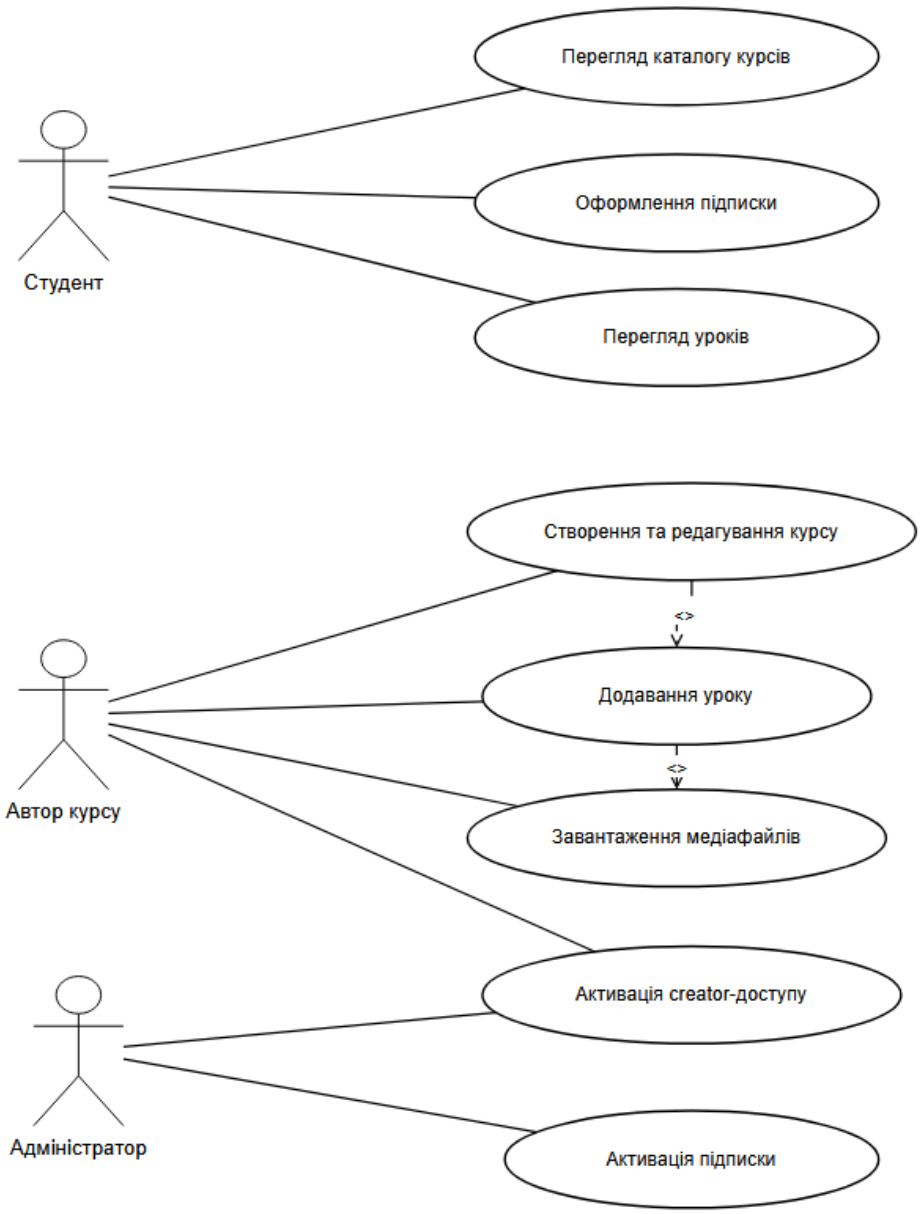

Рисунок 2.5 – Діаграма варіантів використання системи

Діаграма варіантів використання узагальнює основні дії студента, автора курсу та адміністратора в межах платформи. Вона підтверджує, що

функціональність системи охоплює як роботу з навчальним контентом, так і контроль доступу до платних можливостей.

2.5 Реалізація модулів курсів, уроків і підписок

Функціональна частина системи реалізована у вигляді окремих модулів, кожен з яких відповідає за певний набір бізнес-сценаріїв. Такий підхід спрощує підтримку коду, оскільки логіка курсів, уроків, підписок і creator-доступу не змішується в одному великому контролері. У проєкті використано endpoint-класи, request/command-моделі та handler-и прикладного рівня [7].

Модуль курсів забезпечує створення, перегляд, оновлення, видалення курсів і роботу з preview-зображенням. Для цього реалізовано endpoint-и AddCourseEndpoint, GetCoursesEndpoint, GetCourseByIdEndpoint, UpdateCourseEndpoint, DeleteCourseEndpoint, UploadCoursePreviewImageEndpoint та GetCoursePreviewImageEndpoint. На прикладному рівні відповідні handler-и виконують перевірки, створюють або оновлюють CourseEntity та зберігають зміни в базі даних.

Модуль уроків відповідає за створення уроків, отримання списку уроків курсу, перегляд окремого уроку, видалення уроку та роботу з відеофайлами. Уроки пов’язані з курсами, тому під час виконання операцій система перевіряє існування курсу, автора та право користувача змінювати або переглядати навчальний матеріал.

Особливо важливими є сценарії отримання уроків і відео. Вони не повинні повертати захищений контент будь-якому автентифікованому користувачу. Handler-и перевіряють, чи є користувач адміністратором, автором курсу або власником активної підписки. Тільки після цього система повертає текст уроку або шлях до відеоматеріалу.

Модуль підписок реалізує сценарій отримання студентом доступу до курсу. AddSubscriptionHandler перевіряє існування курсу, статус публікації, заборону підписки на власний курс і відсутність дубліката підписки. Якщо курс

безкоштовний, підписка активується одразу. Якщо курс платний, створюється запис зі статусом Pending, який надалі може бути активований через окремий endpoint.

Модуль creator-доступу реалізує можливість отримання користувачем права створювати курси. AddCreatorAccessPurchaseHandler створює заявку на придбання creator-доступу, а ActivateCreatorAccessPurchaseHandler переводить її в активний стан і додає користувача до таблиці creators. Це імітує процес платного підключення автора до платформи.

Для валідації вхідних даних використовується FluentValidation. Валідатори перевіряють коректність запитів на створення та оновлення курсів і уроків. Це дозволяє відокремити перевірку форми запиту від бізнес-логіки handler-ів і робить API більш передбачуваним для клієнтського застосунку. На рисунку 2.6 наведено діаграму послідовності оформлення платної підписки та її подальшої активації адміністратором.

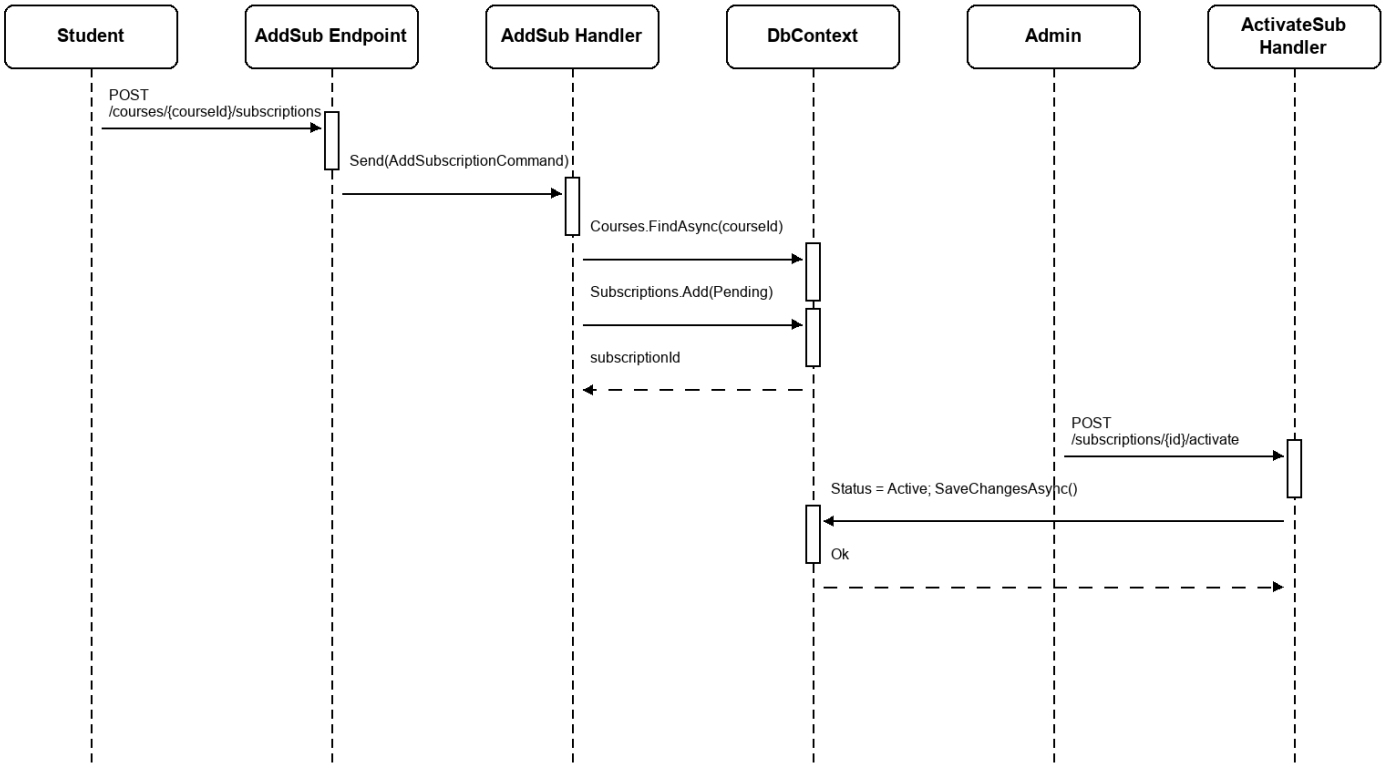

Рисунок 2.6 – Послідовність створення та активації підписки

Наведена послідовність показує, як взаємодіють користувач, API та база даних під час оформлення доступу до курсу. Такий сценарій є важливим для системи, оскільки саме він поєднує навчальну логіку платформи з моделлю платного доступу до контенту.

2.6 Зберігання медіафайлів та інтеграція сервісів

Окрім структурованих даних, платформа онлайн-навчання повинна працювати з медіафайлами, зокрема preview-зображеннями курсів і відеоматеріалами уроків. Зберігати такі файли безпосередньо в реляційній базі даних недоцільно, оскільки це ускладнює роботу з великими обсягами бінарних даних, збільшує розмір бази та погіршує зручність обслуговування системи. Тому в серверній частині реалізовано окремий механізм файлового сховища, а в PostgreSQL зберігаються лише службові метадані: відносний шлях до файлу та його content type.

Для роботи з файлами використано інтерфейс IFileStorage та його реалізацію LocalFileStorage. Інтерфейс визначає базові операції, необхідні для роботи з медіафайлами: збереження файлу, отримання абсолютного шляху до нього та видалення. Такий підхід дозволяє відокремити бізнес-логіку від конкретного способу фізичного зберігання файлів. У поточній реалізації файли зберігаються локально, однак за потреби реалізацію можна замінити, наприклад, на хмарне сховище без суттєвої зміни прикладної логіки.

Реалізація LocalFileStorage зберігає файли в каталозі storage всередині кореневого каталогу API. Під час збереження формується унікальне ім’я файлу на основі Guid, що зменшує ризик конфлікту назв і дозволяє уникнути перезапису вже наявних файлів. У базі даних при цьому зберігається не сам файл, а лише шлях до нього та тип вмісту, наприклад image/png, image/jpeg або video/mp4.

Окрема перевірка в LocalFileStorage гарантує, що сформований абсолютний шлях не виходить за межі кореневого каталогу сховища. Це важливо з погляду

безпеки, оскільки API приймає файли від користувачів, а некоректно сформований шлях не повинен дозволяти доступ до довільних файлів файлової системи. Таким чином, механізм файлового сховища не лише забезпечує збереження медіаконтенту, а й враховує базові вимоги до безпечної роботи з файлами.

Завантаження preview-зображення курсу та відео уроку реалізовано через окремі endpoint-и. Endpoint приймає HTTP-запит типу multipart/form-data, отримує файл, передає його до файлового сховища, а після успішного збереження оновлює відповідну сутність у базі даних. Для курсу зберігається інформація про preview-зображення, а для уроку - інформація про відеофайл. Під час оновлення медіафайлу старий файл може бути видалений зі сховища, щоб уникнути накопичення зайвих даних.

Перевірку роботи механізму завантаження медіафайлів було виконано за допомогою Postman або Swagger UI. У запиті передається файл у форматі multipart/form-data, після чого API зберігає його у файловому сховищі та оновлює метадані у відповідній таблиці бази даних. Приклад виконання запиту на завантаження та отримання медіафайлу через API наведено на рисунках 2.7 - 2.8.

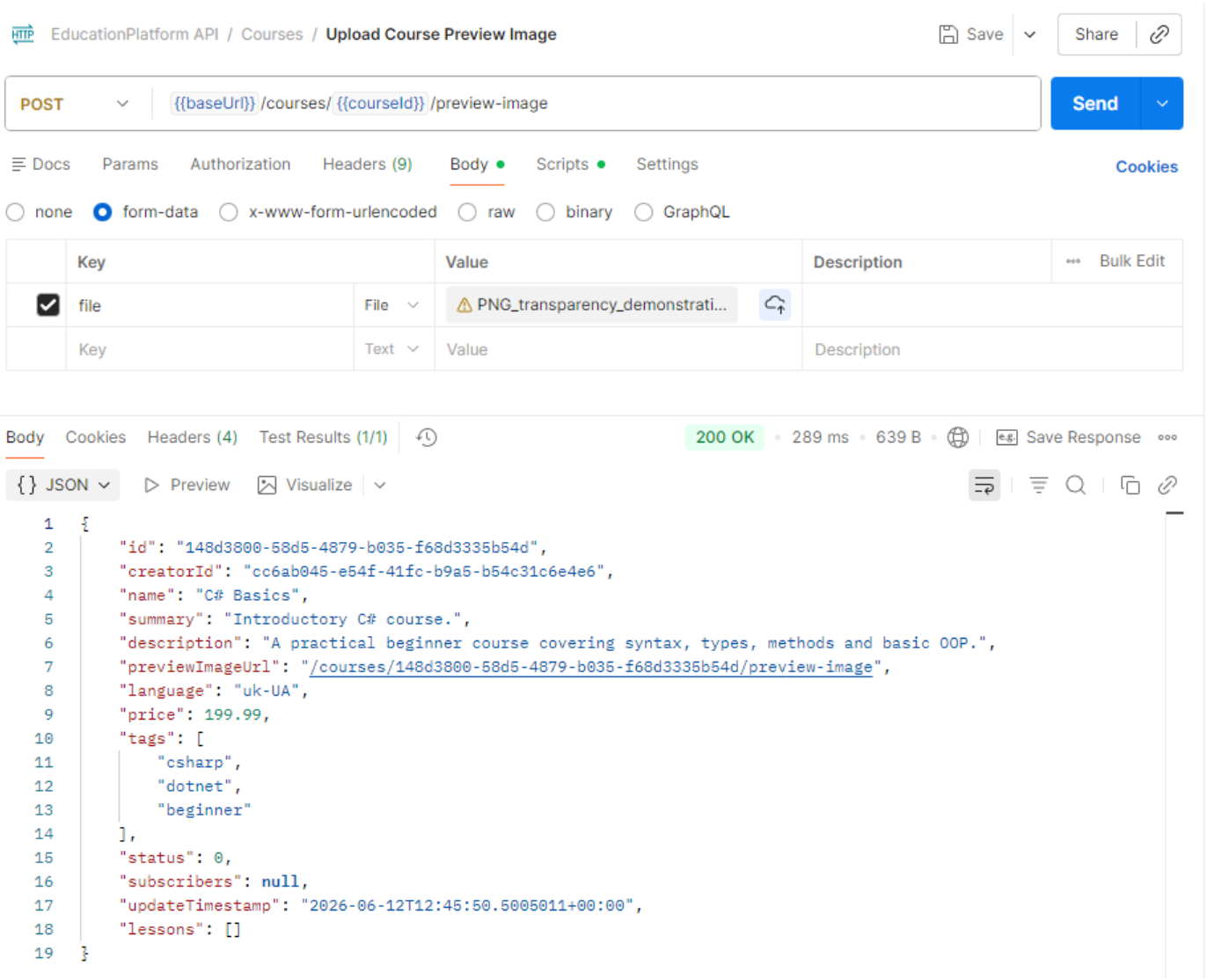

Рисунок 2.7 – Завантаження медіафайлу через API

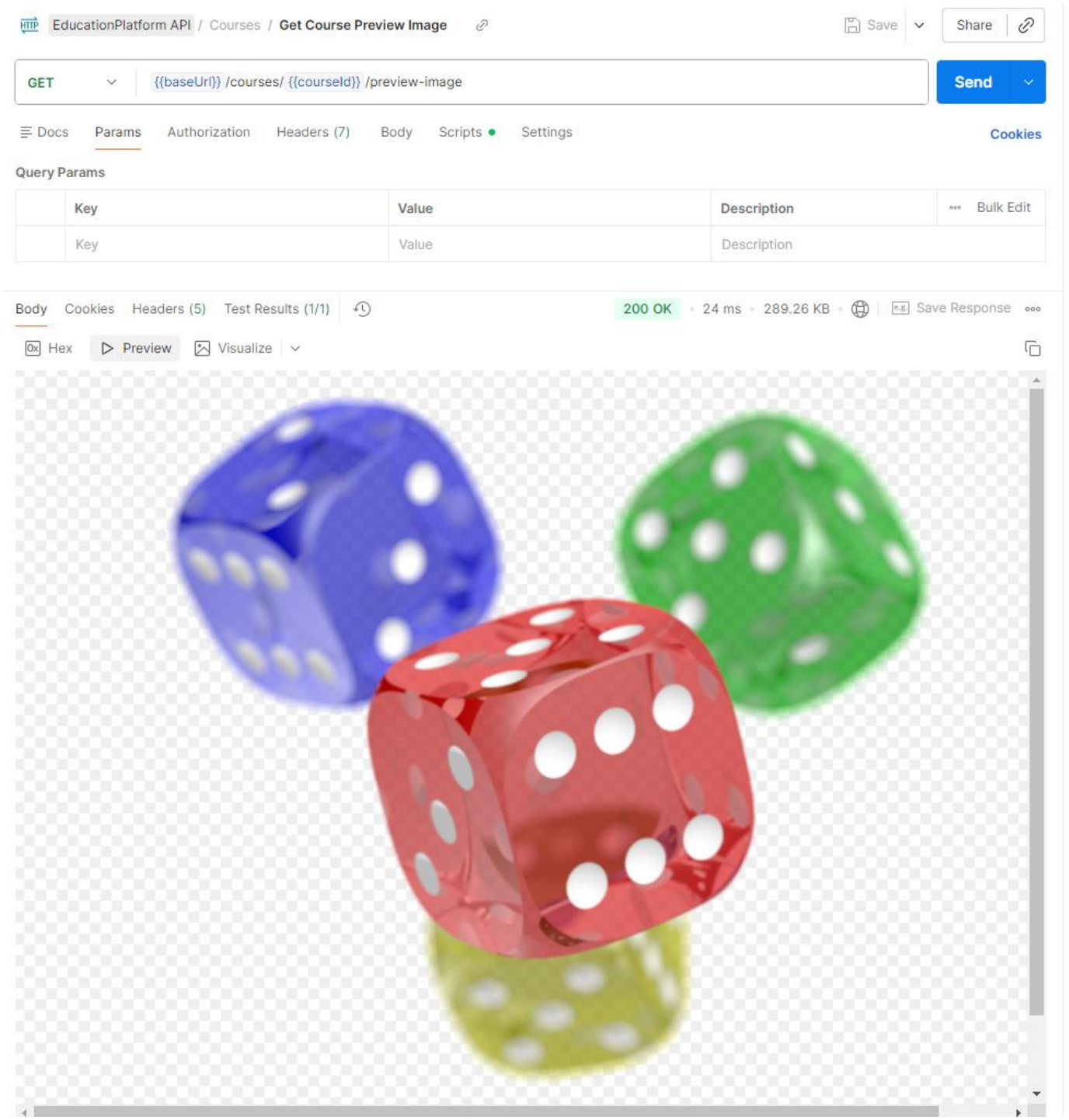

Рисунок 2.8 – Отримання медіафайлу через API

Інфраструктура запуску системи організована за допомогою Docker Compose. У compose-конфігурації описано основні сервіси, необхідні для роботи платформи: education-api, education-postgres та education-keycloak. Сервіс education-api відповідає за роботу програмного забезпечення API, education-postgres

використовується як реляційна база даних, а education-keycloak забезпечує автентифікацію користувачів і керування ролями.

PostgreSQL використовується для зберігання даних платформи онлайн-навчання, зокрема курсів, уроків, підписок і заявок на отримання creator-доступу. Keycloak запускається як окремий інфраструктурний компонент і відповідає за облікові записи користувачів, ролі та видачу JWT-токенів. API отримує налаштування Authority, MetadataAddress, Issuer та Audience через конфігурацію. У Docker-середовищі MetadataAddress спрямований на внутрішню адресу контейнера Keycloak, а Issuer налаштовується так, щоб відповідати токенам, які отримує клієнт.

Під час запуску API також застосовуються міграції Entity Framework Core, що дозволяє автоматично привести структуру бази даних до актуального стану. Це спрощує розгортання системи, оскільки після запуску контейнерів середовище одразу готове до перевірки основних функцій платформи.

Такий спосіб організації інфраструктури робить систему відтворюваною. Для запуску демонстраційного середовища достатньо виконати команду docker compose up --build, після чого будуть підняті API, PostgreSQL і Keycloak. Це спрощує перевірку роботи системи на іншому комп’ютері та зменшує залежність від локальних налаштувань розробника. Приклад запущених контейнерів системи в Docker Desktop наведено на рисунку 2.9.

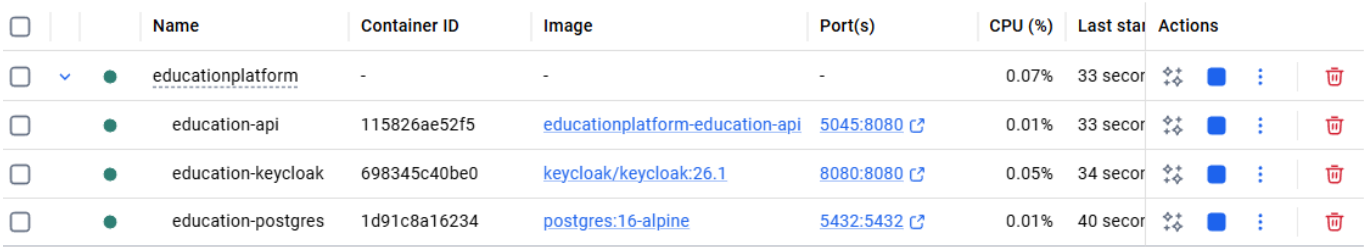

Рисунок 2.9 – Docker Desktop із запущеними контейнерами.

Результат запуску контейнерів підтверджує, що всі основні інфраструктурні компоненти системи можуть працювати разом у локальному середовищі. Це спрощує демонстрацію роботи платформи, повторне розгортання та перевірку взаємодії API з PostgreSQL і Keycloak.

2.7 Документування API та взаємодія через Swagger і Postman

Оскільки результатом розробки є серверна частина платформи онлайн-навчання, основним інтерфейсом взаємодії з системою є HTTP API. Для зручності перевірки та демонстрації endpoint-ів у проєкті підключено Swagger UI. Він автоматично формує вебсторінку з описом маршрутів, HTTP-методів, параметрів запитів, схем відповідей і механізму авторизації [17].

Swagger UI дозволяє виконувати запити без написання окремого клієнтського застосунку. Через нього можна перевірити створення курсу, отримання списку курсів, додавання уроку, оформлення підписки, активацію підписки, завантаження preview-зображення та відеофайлу. Для захищених endpoint-ів передбачено Bearer-авторизацію, де користувач може передати JWT-токен, отриманий з Keycloak.

Додатково підготовлено Postman collection яку показано на рисунку 2.10. Вона корисна для демонстрації послідовних бізнес-сценаріїв, оскільки дозволяє зберігати змінні baseUrl, courseId, lessonId, subscriptionId, keycloakUrl та інші параметри. Це зручно у випадках, коли результат одного запиту використовується в наступному запиті, наприклад під час створення курсу, додавання уроку та оформлення підписки на цей курс.

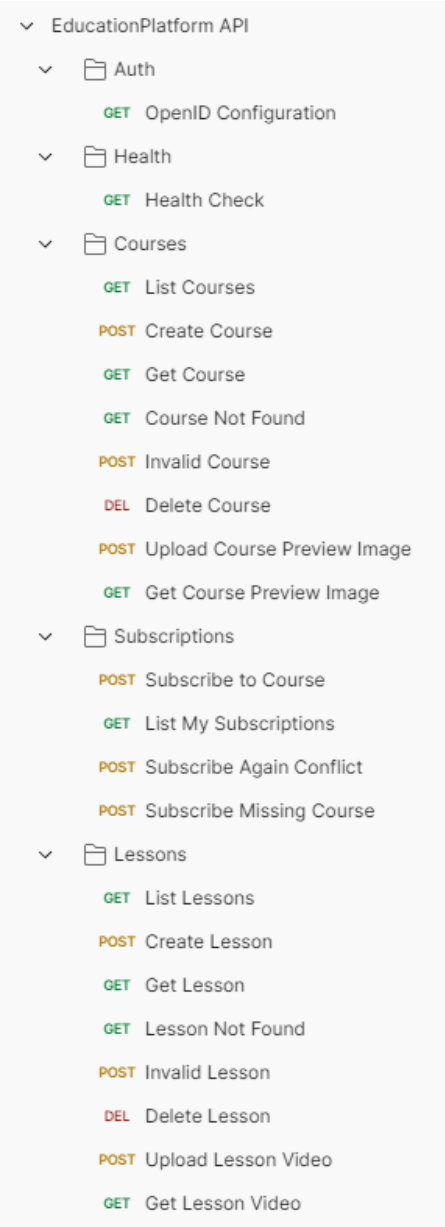

Рисунок 2.10 – Postman collection для перевірки роботи API

Postman також зручний для роботи з OAuth 2.0 Authorization Code with PKCE. Такий підхід ближчий до реальної взаємодії клієнтського застосунку з Keycloak, оскільки токен отримується не вручну, а через налаштований механізм авторизації. Це дозволяє перевіряти API в умовах, наближених до майбутнього frontend-застосунку [20].

Майбутній користувацький інтерфейс може бути реалізований як окремий вебзастосунок. Для студента він повинен містити каталог курсів, сторінку курсу, список уроків, перегляд навчального матеріалу та сторінку підписок. Для автора потрібні сторінки створення й редагування курсів, керування уроками та

завантаження медіафайлів. Для адміністратора потрібні сторінки підтвердження платних операцій.

Наявність документованого API дає змогу розробляти frontend незалежно від серверної частини. Клієнтський застосунок буде надсилати HTTP-запити до API, отримувати JSON-відповіді, відображати дані користувачу та передавати JWT-токен у заголовку Authorization для захищених операцій. Такий поділ відповідає сучасному підходу до розробки вебсистем.

Таким чином, у межах цієї роботи інтерфейси взаємодії з системою представлені на двох рівнях: технічному та проєктному. Технічний рівень забезпечують Swagger UI і Postman, які дозволяють перевірити реалізоване API без окремого frontend-застосунку. На рисунках 2.11-2.12 наведено скріншоти роботи програми: UI з переліком endpoint-ів, а також приклад виконання HTTP-запиту до API.

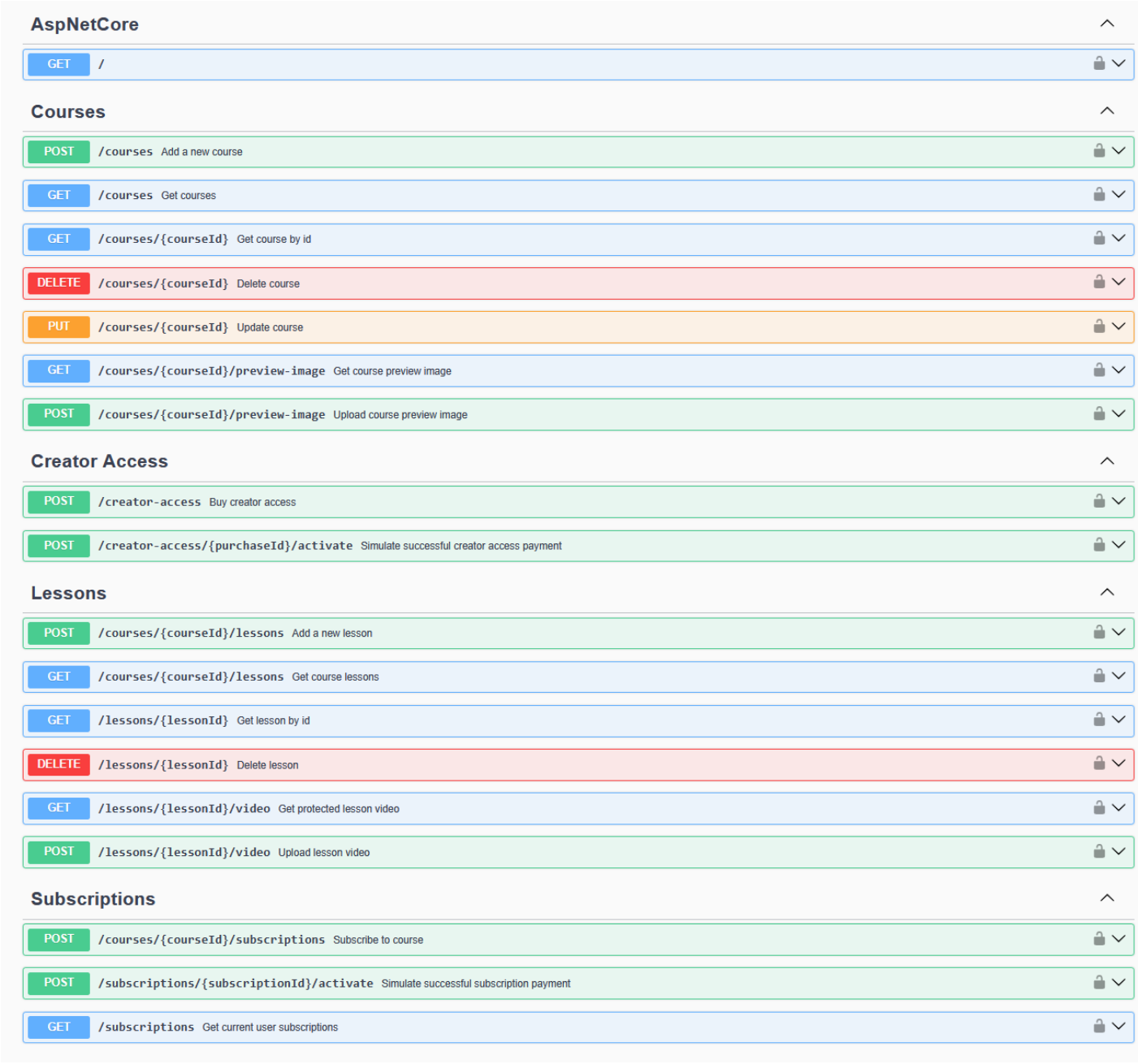

Рисунок 2.11 – Інтерфейс Swagger UI з переліком endpoint-ів API

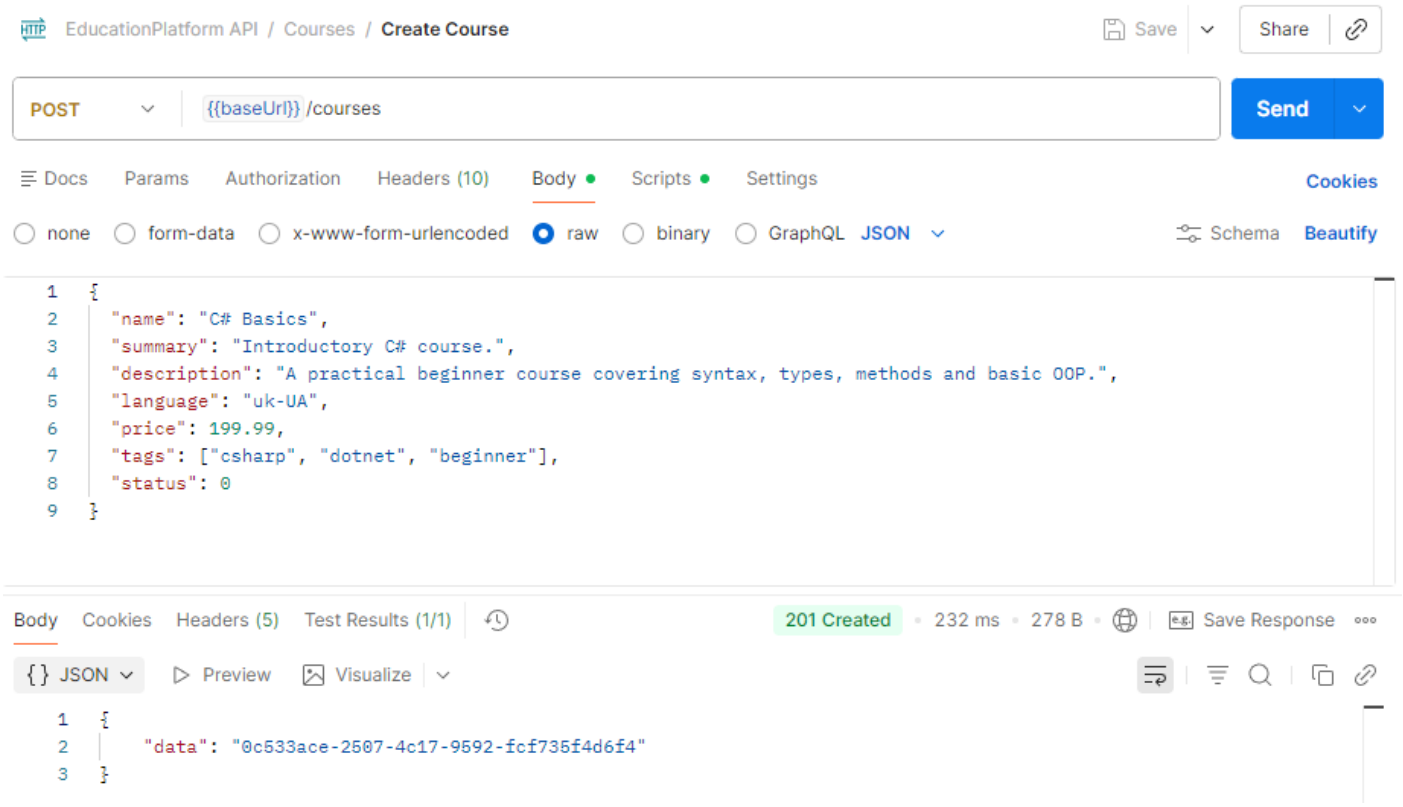

Рисунок 2.12 – Приклад виконання API-запиту у Postman

У другому розділі було розглянуто проєктування архітектури та розробку серверної частини платформи онлайн-навчання. Обґрунтовано вибір ітеративного підходу до розробки, описано багаторівневу архітектуру системи з урахуванням принципів Clean Architecture, модель даних, UML-представлення основних класів, механізми автентифікації та авторизації, функціональні модулі, файлове сховище, інфраструктуру запуску та засоби документування API.

Запропонована архітектура забезпечує розділення відповідальності між HTTP-рівнем, прикладною логікою, доступом до даних, спільними моделями та зовнішніми сервісами. Така організація відповідає ідеї Clean Architecture: бізнес-сценарії зосереджені в Application, а AspNetCore, PostgreSql, PostgreSQL, Keycloak і файлове сховище виконують роль зовнішніх або інфраструктурних компонентів. Використання ASP.NET Core, MediatR, Entity Framework Core, PostgreSQL, Keycloak, Docker Compose, Swagger UI та Postman створює основу для підтримуваної, тестованої та придатної до розширення програмної системи [5, 6, 13, 15, 18-20].

3 ТЕСТУВАННЯ, РОЗГОРТАННЯ ТА ВЕРИФІКАЦІЯ СИСТЕМИ

Третій розділ присвячено перевірці працездатності платформи онлайн-навчання, підготовці системи до запуску та визначенню способів її подальшої підтримки. Для серверної частини програмної системи важливо перевірити не лише правильність окремих методів, а й повний шлях обробки HTTP-запиту: від отримання запиту клієнта до зміни стану бази даних і формування відповіді API.

Оскільки EducationPlatform містить модулі курсів, уроків, підписок, creator-доступу, файлового сховища та авторизації через Keycloak, тестування повинно охоплювати як функціональні сценарії, так і обмеження доступу. Окрему увагу приділено перевірці ситуацій, у яких система повинна відмовити користувачу: відсутність автентифікації, спроба підписатися на власний курс, повторне створення підписки, звернення до неіснуючого ресурсу або доступ до захищеного уроку без активної підписки.

3.1 Тестування програмного забезпечення

Тестування програмної системи виконувалося на рівні API та прикладної логіки. Основним інструментом автоматизованої перевірки є набір тестів у проєкті AspNetCore.Tests. Тести написані з використанням NUnit, Microsoft.AspNetCore.Mvc.Testing та Entity Framework Core InMemory provider. Така комбінація дозволяє запускати API у тестовому середовищі та виконувати HTTP-запити без підключення до реальної PostgreSQL-бази [13, 15].

Для запуску тестового екземпляра застосунку використовується WebApplicationFactory. У проєкті створено клас EducationPlatformApiFactory, який налаштовує середовище Development, підміняє реальну базу даних на InMemory-базу, вимикає автоматичне застосування міграцій і за потреби вмикає development authentication. Завдяки цьому тести можуть швидко створювати ізольоване середовище для кожного сценарію.

Development authentication використовується лише для тестування та локальної перевірки. Він дозволяє передавати ідентифікатор користувача і ролі через службові HTTP-заголовки. У продукційному режимі система використовує JWT-токени, отримані з Keycloak. Такий поділ дає змогу тестувати бізнес-логіку авторизації без необхідності піднімати Keycloak для кожного автоматизованого тесту.

Автоматизовані тести охоплюють такі основні напрями перевірки:

- модуль курсів, зокрема створення, перегляд, оновлення, видалення та роботу з preview-зображеннями;
- модуль уроків, зокрема створення уроків, отримання навчального матеріалу та завантаження відеофайлів;
- модуль підписок, включно зі створенням, повторною підпискою, активацією та перевіркою доступу;
- модуль creator-доступу, який перевіряє отримання користувачем прав автора після активації заявки;
- системні сценарії, зокрема доступність Swagger UI та заборону доступу без автентифікації;

Ручне тестування може виконуватися за допомогою Swagger UI та Postman collection. Swagger UI зручний для швидкої перевірки окремих endpoint-ів, а Postman дозволяє демонструвати послідовні бізнес-сценарії з використанням змінних середовища. Такі інструменти доповнюють автоматизовані тести і використовуються для перевірки системи в Docker-середовищі. Загальну схему процесу тестування API наведено на рисунку 3.1 [17].

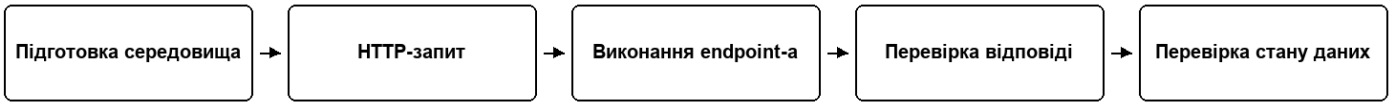

Рисунок 3.1 – Загальна схема процесу тестування API

Схема тестування показує, що перевірка системи охоплює як автоматизовані тести, так і ручні сценарії через зовнішні інструменти. Такий підхід дозволяє оцінити не лише коректність окремих handler-ів, а й поведінку API з позиції реального клієнта.

3.1.1 План тестування та бізнес-сценарії

План тестування серверної частини платформи онлайн-навчання побудовано таким чином, щоб перевірити основні ризики, які можуть виникати під час роботи API. До таких ризиків належать некоректна робота CRUD-операцій, помилки у зв’язках між курсами та уроками, неправильна активація підписок, некоректне надання creator-доступу, проблеми із завантаженням файлів, помилки у правилах авторизації та некоректна взаємодія API з базою даних.

У межах тестування було передбачено поєднання ручної перевірки та автоматизованих тестів. Ручна перевірка виконувалася за допомогою Postman або Swagger UI і використовувалася для демонстрації роботи API в реальному середовищі запуску. Автоматизовані тести були реалізовані в окремому тестовому проєкті AspNetCore.Tests з використанням фреймворку NUnit. Такий підхід дозволяє перевіряти систему як з позиції зовнішнього користувача API, так і на рівні повторюваних автоматизованих сценаріїв [13, 17].

Функціональне тестування спрямоване на перевірку того, чи реалізовані endpoint-и виконують очікувані бізнес-операції. Для модуля курсів перевіряється створення, отримання, оновлення, видалення та робота з preview-зображенням. Для модуля уроків перевіряється створення уроку, отримання списку уроків, отримання уроку за ідентифікатором, видалення уроку та завантаження відеофайлу. Для модуля підписок перевіряється створення підписки, отримання власних підписок і активація підписки адміністратором. Окремо перевіряється creator-access flow, який відповідає за надання користувачеві права створювати курси.

Інтеграційне тестування перевіряє взаємодію між HTTP-рівнем, прикладною логікою та рівнем даних. У таких тестах клієнт надсилає HTTP-запит до тестового

екземпляра API, endpoint передає керування відповідному handler-у, handler працює з EducationDbContext, а тест перевіряє HTTP-статус, зміст відповіді та зміну стану системи. Такий підхід дозволяє виявляти помилки, які не завжди видно під час ізольованої перевірки окремого класу, наприклад неправильну прив'язку маршруту, помилки в middleware авторизації, некоректну серіалізацію відповіді або неправильну роботу з тестовою базою даних [13, 15].

Автоматизовані тести реалізовано з використанням NUnit, який застосовується для опису тестових класів, тестових методів, підготовки тестових даних та перевірки очікуваних результатів. NUnit дозволяє формалізувати тестові сценарії у вигляді повторюваних перевірок, які можна запускати з середовища розробки або через командний рядок. У тестах перевіряються HTTP-статуси відповідей, коректність створення та зміни сутностей, правила авторизації, негативні сценарії та робота endpoint-ів із файлами.

Тестування авторизації перевіряє, чи система правильно обмежує доступ до захищених операцій. Наприклад, користувач без автентифікації не повинен отримувати доступ до захищених endpoint-ів, студент без creator-доступу не повинен створювати курс, а користувач без активної підписки не повинен переглядати захищений урок або відеоматеріал. Для таких сценаріїв важливо перевіряти не лише факт помилки, а й правильний HTTP-статус, наприклад Unauthorized, Forbidden, NotFound, BadRequest або Conflict залежно від конкретної ситуації [16, 20, 21].

Тестування негативних сценаріїв необхідне для перевірки стабільності API. Система повинна коректно реагувати на неіснуючі ідентифікатори, повторні запити, некоректні дії користувача та порушення бізнес-правил. Наприклад, повторне створення підписки на той самий курс повинно повертати Conflict, спроба підписатися на власний курс - BadRequest, а спроба отримати неіснуючий курс або урок - відповідний статус помилки. Такі перевірки дозволяють переконатися, що API не лише працює у позитивних сценаріях, а й передбачувано поводиться при помилкових діях користувача.

Окремим видом перевірки є тестування роботи з файлами. Для цього перевіряються endpoint-и, які приймають multipart/form-data запити для завантаження preview-зображень курсів і відеофайлів уроків. Під час такого тестування перевіряється, чи API приймає файл, передає його до файлового сховища, оновлює відповідні метадані в базі даних і дозволяє отримати файл через окремий endpoint. Це важливо, оскільки робота з файлами поєднує HTTP-рівень, файлову систему та базу даних.

План тестування також передбачає ручну перевірку розгортання системи в Docker Compose. Для цього запускаються контейнери API, PostgreSQL та Keycloak, після чого через Swagger UI або Postman перевіряється доступність API, отримання JWT-токена, виконання основних запитів і робота endpoint-ів із файлами. Ручне тестування дозволяє показати, що система працює не лише в тестовому середовищі, а й у реальному запущеному середовищі з базою даних, авторизацією та зовнішнім сервером ідентифікації [19, 20].

Для демонстрації ручної перевірки було використано Postman collection з підготовленими наборами запитів до модулів курсів, уроків, підписок і creator-доступу. Колекція містить окремі групи запитів для автентифікації, перевірки доступності API, роботи з курсами, уроками, медіафайлами, підписками та адміністративними діями. Такий набір запитів дає змогу послідовно перевірити отримання токена, створення курсу, додавання уроку, завантаження медіафайлів, оформлення підписки та перевірку доступу до захищених ресурсів. Структуру Postman collection для ручного тестування API наведено на рисунку 3.2.

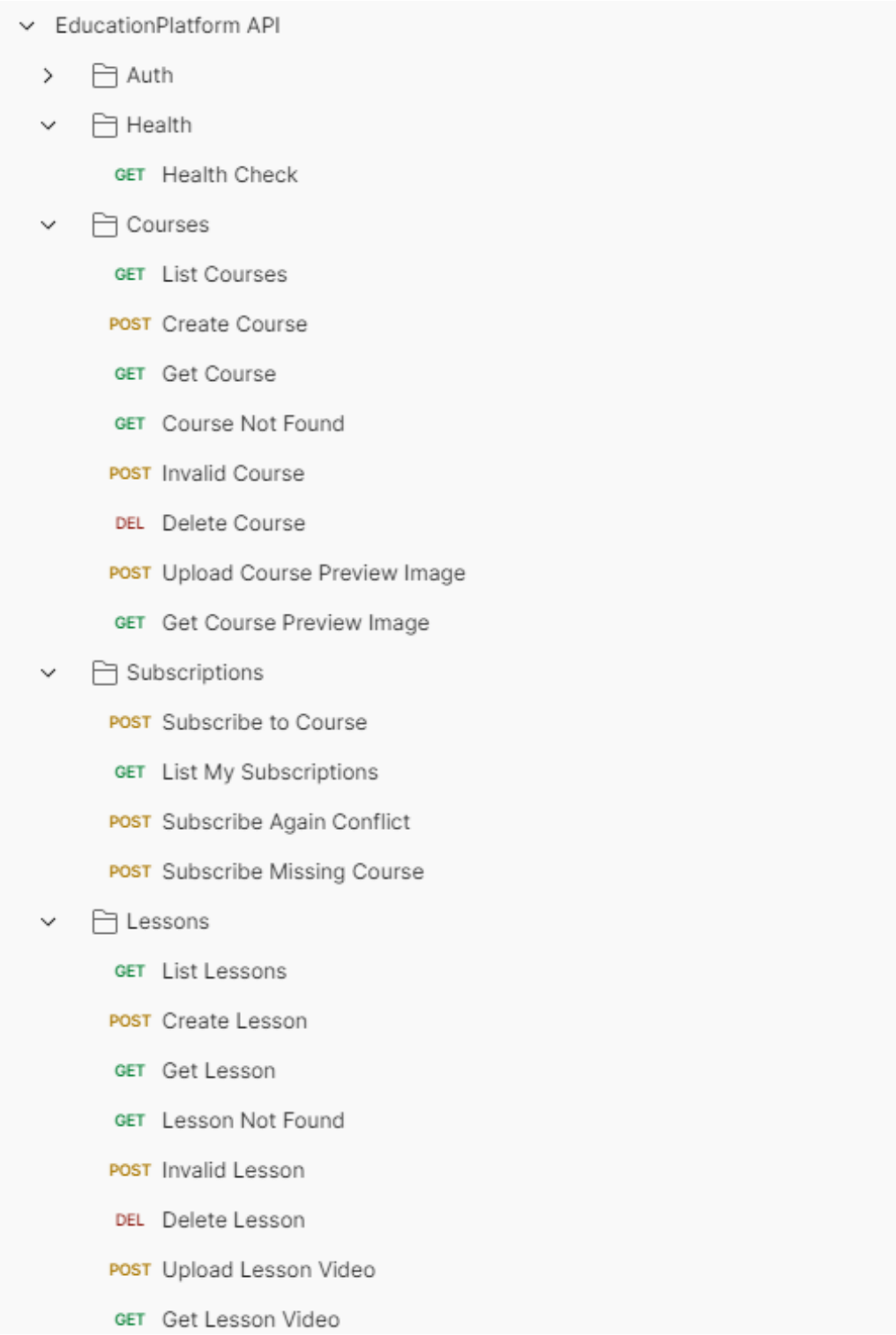

Рисунок 3.2 – Виконання ручних тестових запитів до API у Postman

Для автоматизованої перевірки в рішенні створено окремий тестовий проєкт AspNetCore.Tests. Його структура містить каталоги Courses, Lessons, Subscriptions, CreatorAccess, System та TestSupport. Такий поділ відповідає основним функціональним модулям системи та спрощує підтримку тестів, оскільки кожна

група перевірок зосереджена навколо окремої частини API. Наприклад, тести для курсів зібрані в каталозі Courses, тести для уроків - у каталозі Lessons, а допоміжні класи для підготовки тестового середовища розміщені в TestSupport. Структуру тестового проєкту показано на рисунку 3.3.

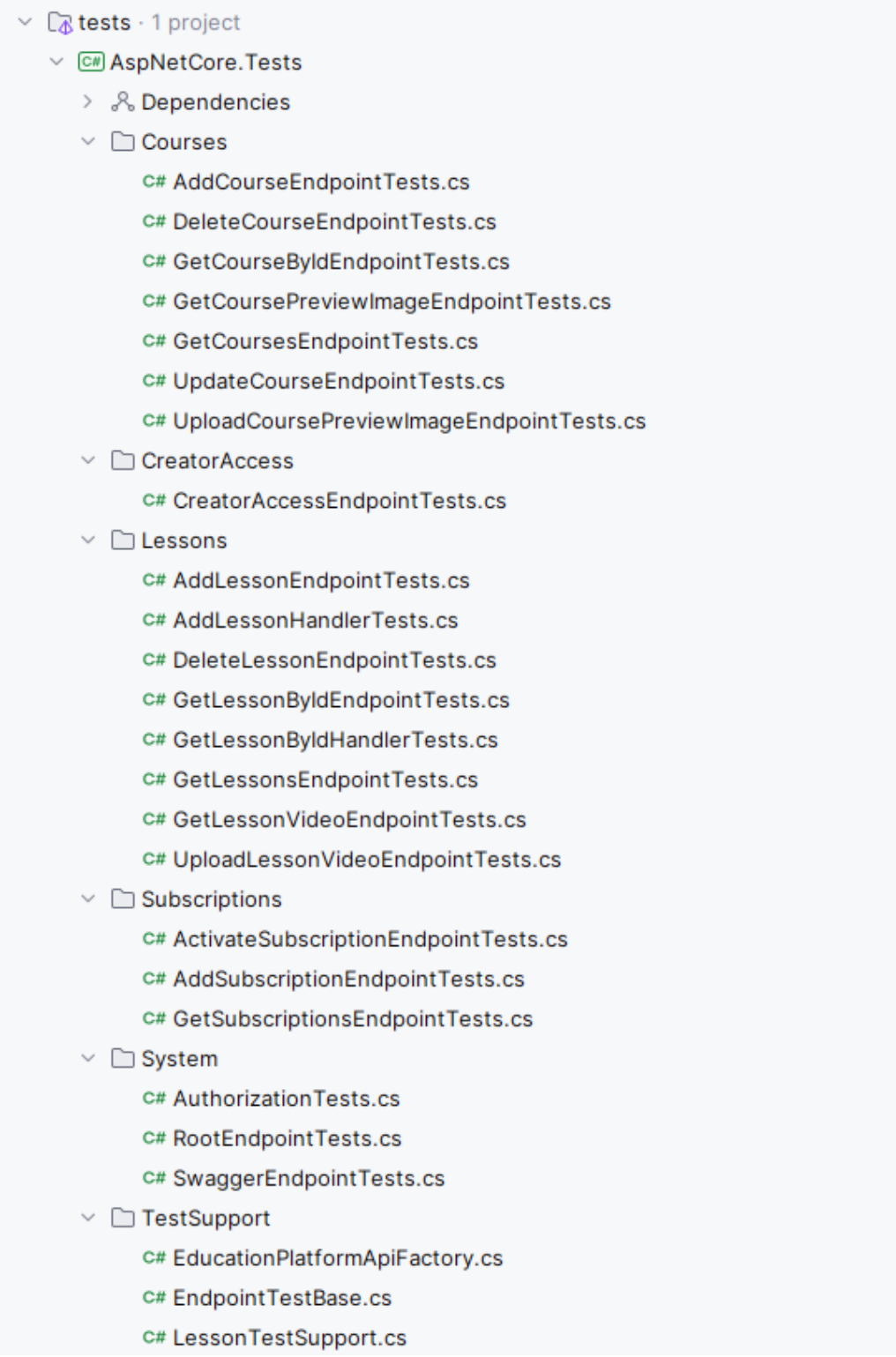

Рисунок 3.3 – Структура проєкту автоматизованих NUnit-тестів AspNetCore.Tests

Автоматизовані NUnit-тести можуть запускатися безпосередньо з IDE або через командний рядок. Для запуску тестового проєкту використовується команда dotnet test. Під час запуску .NET Test SDK знаходить NUnit-тести в проєкті AspNetCore.Tests, виконує їх і формує результат проходження. У середовищі розробки або в консолі відображається кількість успішних, невдалих і пропущених тестів. Такий результат є підтвердженням того, що основні сценарії роботи API були перевірені автоматизовано. Приклад результату виконання тестів наведено на рисунку 3.4.

Рисунок 3.4 – Результат виконання автоматизованих NUnit-тестів серверної частини

Таким чином, план тестування поєднує ручну перевірку через Postman або Swagger UI та автоматизовані NUnit-тести з проєкту AspNetCore.Tests. Ручні перевірки дозволяють продемонструвати роботу системи в реальному середовищі запуску, а автоматизовані тести забезпечують повторювану перевірку основних endpoint-ів і бізнес-сценаріїв після внесення змін у код.

3.1.2 Автоматизовані та ручні тестові сценарії

Тестові сценарії розроблялися на основі основних бізнес-процесів платформи онлайн-навчання та реалізовувалися у вигляді автоматизованих тестів з

використанням NUnit. Для кожного сценарію визначалися передумови, дія користувача, очікуваний HTTP-статус і очікувана зміна стану системи. Такий підхід дозволяє перевірити не лише технічну доступність endpoint-а, а й відповідність поведінки API вимогам предметної області.

Основою для побудови сценаріїв стали функціональні можливості API: керування курсами, керування уроками, оформлення та активація підписок, надання creator-доступу, перевірка авторизації та робота з медіафайлами. Для кожного модуля було підготовлено як позитивні, так і негативні сценарії. Позитивні сценарії підтверджують, що система виконує очікувану операцію за правильних умов, а негативні - що API коректно реагує на помилки, відсутність прав доступу або порушення бізнес-правил.

У NUnit-тестах кожен тестовий метод відповідає окремому сценарію або групі близьких сценаріїв. Для перевірки очікуваного результату використовуються assertion-перевірки, які порівнюють фактичний HTTP-статус, дані відповіді та стан сутностей із очікуваними значеннями. Завдяки цьому тест не лише виконує запит до API, а й підтверджує, що система змінила або не змінила свій стан відповідно до очікуваної бізнес-логіки.

Для сценаріїв курсів перевіряється створення курсу користувачем, який має права автора, отримання списку курсів, перегляд окремого курсу, редагування, видалення та завантаження preview-зображення. У тестовому проєкті такі перевірки винесені в окремі файли, зокрема AddCourseEndpointTests, GetCoursesEndpointTests, GetCourseByIdEndpointTests, UpdateCourseEndpointTests, DeleteCourseEndpointTests, UploadCoursePreviewImageEndpointTests та GetCoursePreviewImageEndpointTests. Такий поділ дозволяє швидко знайти тест для конкретного endpoint-а та спрощує супровід тестового коду.

Для сценаріїв уроків перевіряється створення уроку в межах курсу, отримання уроку за ідентифікатором, отримання списку уроків курсу, видалення уроку, завантаження відеофайлу та отримання відео. Окремо перевіряються ситуації, коли курс або урок не існує, а також випадки, коли користувач не має прав на зміну навчального матеріалу. Відповідні перевірки реалізовані в тестах

AddLessonEndpointTests, GetLessonsEndpointTests, GetLessonByIdEndpointTests, DeleteLessonEndpointTests, UploadLessonVideoEndpointTests та GetLessonVideoEndpointTests.

Для сценаріїв підписок перевіряється створення підписки на безкоштовний і платний курс, повторне створення підписки, заборона підписки на власний курс і активація платної підписки. Якщо курс безкоштовний, підписка може одразу переходити в активний стан. Якщо курс платний, підписка створюється в очікуваному стані та потребує подальшої активації. Такі сценарії дозволяють перевірити правильність бізнес-логіки, пов’язаної з доступом студента до навчального контенту.

Для creator-access flow перевіряється окремий процес отримання користувачем права створювати курси. Спочатку користувач без creator-доступу не повинен мати можливості створювати курс. Далі він створює заявку на отримання creator-доступу. Після цього адміністратор активує заявку, і користувач отримує можливість виконувати операції автора курсу. Такий сценарій є важливим, оскільки він поєднує авторизацію, роботу з базою даних і зміну стану користувача в межах бізнес-логіки платформи.

Окрему групу становлять системні тести. Вони перевіряють базову доступність API, роботу кореневого endpoint-а, доступність Swagger-документації та правила авторизації. Такі тести не перевіряють окремий бізнес-процес, але є важливими для підтвердження того, що серверна частина коректно запускається, приймає запити та правильно обробляє базові сценарії доступу [17].

Для спрощення написання тестів у проєкті передбачено допоміжні класи в каталозі TestSupport. Зокрема, там розміщуються класи для створення тестового екземпляра API, базової логіки endpoint-тестів і допоміжних методів для підготовки тестових даних. Це дозволяє не дублювати однаковий код у кожному тесті та забезпечує однаковий підхід до створення HTTP-клієнта, підготовки користувачів, курсів, уроків і підписок.

Ручні сценарії в Postman використовуються для перевірки тих самих бізнес-процесів, але вже в демонстраційному середовищі. Наприклад, через Postman

можна послідовно отримати JWT-токен, створити курс, додати урок, завантажити preview-зображення або відеофайл, оформити підписку та перевірити доступ до навчального матеріалу. Такий підхід є зручним для демонстрації, оскільки дозволяє показати не лише результат автоматизованого тесту, а й реальну відповідь API у форматі JSON, HTTP-статус і передані параметри запиту.

Більшість наведених сценаріїв реалізовано у вигляді NUnit-тестів у проєкті AspNetCore.Tests, а окремі сценарії додатково перевірялися вручну через Postman або Swagger UI. Узагальнений перелік основних тестових сценаріїв програмної системи наведено в таблиці 3.1.

Таблиця 3.1 – Основні тестові сценарії програмної системи

№	Сценарій	Передумова	Очікуваний результат	Тип перевірки
1	Створення курсу	Користувач має права автора	API повертає Created, створюється CourseEntity	Позитивна
2	Отримання списку курсів	У системі існують курси	API повертає список курсів	Функціональна
3	Створення уроку	Існує курс, користувач є автором	API повертає Created, урок прив’язаний до курсу	Позитивна
4	Завантаження preview-зображення	Існує курс, користувач має права автора	Файл зберігається, у базі оновлюються метадані	Файлова
5	Завантаження відео уроку	Існує урок, користувач має права автора	Відеофайл зберігається, шлях доступний через API	Файлова
6	Підписка на курс	Курс опублікований, користувач не є автором	Створюється підписка зі статусом Active або Pending	Позитивна
7	Повторна підписка	Підписка на курс уже існує	API повертає Conflict	Негативна
8	Підписка на власний курс	Користувач є автором курсу	API повертає BadRequest	Негативна
9	Доступ до уроку без підписки	Користувач не автор і не має Active-підписки	Доступ до захищеного матеріалу заборонено	Авторизація
10	Активація підписки	Існує підписка зі статусом Pending	Після активації користувач отримує доступ до курсу	Інтеграційна
11	Активація creator-доступу	Існує заявка зі статусом Pending	Після активації користувач може створювати курси	Інтеграційна
12	Запит без автентифікації	JWT-токен відсутній	API повертає Unauthorized	Безпека

Продовження таблиці 3.1

№	Сценарій	Передумова	Очікуваний результат	Тип перевірки
13	Запит із недостатніми правами	Користувач не має потрібної ролі або доступу	API повертає Forbidden або інший відповідний статус	Авторизація
14	Отримання неіснуючого ресурсу	Передано неіснуючий ідентифікатор	API повертає помилку, ресурс не створюється і не змінюється	Негативна

Наведені сценарії є основою для автоматизованих і ручних перевірок. Частина з них реалізована у вигляді NUnit-тестів у проєкті AspNetCore.Tests, а частина виконується через Postman або Swagger UI під час демонстрації роботи системи. Такий поділ є практичним, оскільки автоматизовані NUnit-тести швидко перевіряють стабільність коду після змін, а ручні сценарії дозволяють показати роботу системи в реальному середовищі запуску.

Таким чином, процес тестування охоплює як внутрішню перевірку програмної реалізації, так і зовнішню перевірку API з позиції клієнта. NUnit використовується для автоматизованої перевірки endpoint-ів і бізнес-логіки, тоді як Postman-сценарії демонструють фактичну взаємодію користувача з API через HTTP-запити.

3.2 Розгортання в Docker Compose та системні вимоги

Розгортання програмного забезпечення API організовано за допомогою Docker Compose. Такий підхід дозволяє запускати API, PostgreSQL та Keycloak як узгоджений набір контейнерів. Це спрощує підготовку середовища, оскільки користувачу не потрібно окремо встановлювати та налаштовувати сервер бази даних або сервер автентифікації [19].

Для API створено Dockerfile з багатостадійною збіркою. На першому етапі використовується образ .NET SDK, у якому виконується відновлення залежностей і публікація проєкту AspNetCore у Release-конфігурації. На другому етапі використовується runtime-образ ASP.NET Core, у який копіюється результат

публікації. Такий підхід зменшує розмір фінального контейнера та відокремлює середовище збірки від середовища виконання.

Compose-конфігурація містить три основні сервіси: education-api, education-postgres та education-keycloak. Контейнер PostgreSQL створює бази даних education і keycloak, контейнер Keycloak використовує PostgreSQL як сховище своїх службових даних, а контейнер API підключається до PostgreSQL і Keycloak через змінні середовища. API доступне на порту 5045, Keycloak - на порту 8080, PostgreSQL - на порту 5432.

Під час запуску API автоматично застосовує міграції Entity Framework Core. Це дозволяє створити або оновити структуру бази даних без ручного виконання SQL-скриптів. Для першого запуску достатньо виконати команду docker compose up --build. Після успішного запуску Swagger UI доступний за адресою http://localhost:5045/swagger [15, 17, 19]. Результат запуску контейнерів у середовищі розробки наведено на рисунку 3.5. На ньому показано, що Docker Compose успішно підняв усі необхідні компоненти програмної системи: education-api, education-keycloak та education-postgres. Також видно, що загальний стан середовища визначено як Running (3/3), а контейнер PostgreSQL має статус Healthy, що підтверджує готовність бази даних до роботи.

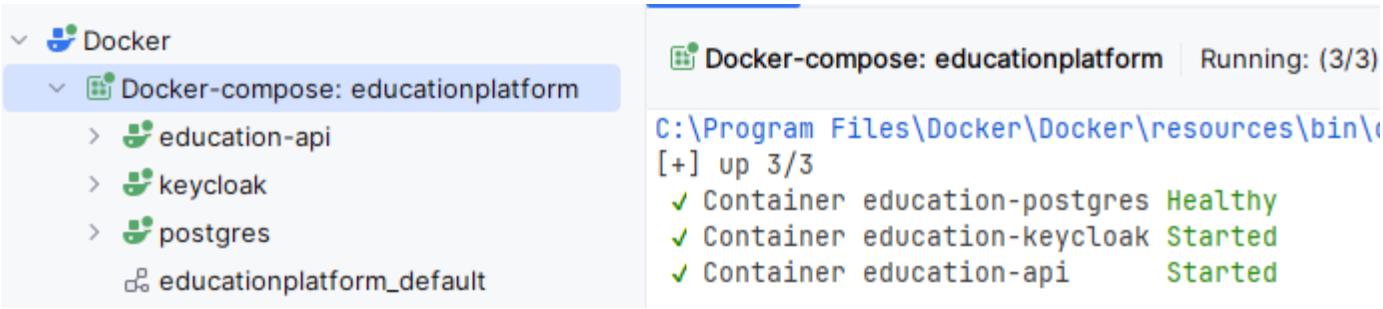

Рисунок 3.5 – Запущені контейнери програмної системи

Для локального запуску та перевірки програмної системи потрібні такі компоненти:

- Docker Desktop для запуску контейнерів API, PostgreSQL та Keycloak.

● Вільні порти 5045, 8080 і 5432 для доступу до API, сервера автентифікації та бази даних.

● .NET SDK 10 для локальної розробки та запуску автоматизованих тестів без Docker.

● Браузер, Swagger UI або Postman для ручної перевірки endpoint-ів програмної системи.

Підтримка системи передбачає оновлення міграцій бази даних, розширення тестового набору, контроль конфігурації Keycloak, очищення або перенесення файлового сховища та моніторинг помилок API. Завдяки контейнеризації система може бути перенесена на інше середовище з мінімальними змінами конфігурації.

3.3 Верифікація роботи системи

Верифікація програмного забезпечення API полягає у перевірці того, що реалізована система відповідає визначеним функціональним і нефункціональним вимогам. Для EducationPlatform верифікація охоплює перевірку доступності API, коректності роботи основних бізнес-сценаріїв, правильності обмеження доступу, працездатності Docker-середовища та можливості ручної взаємодії через Swagger UI або Postman [4].

Функціональна верифікація підтверджує, що користувач з відповідними правами може створити курс, додати урок, завантажити медіафайл, оформити підписку та отримати доступ до навчального матеріалу. Також перевіряється, що адміністратор може активувати платну підписку або creator-доступ, після чого стан відповідного запису змінюється на Active.

Верифікація безпеки підтверджує, що API не відкриває захищені ресурси без автентифікації та належних прав. JWT-токен перевіряється через налаштування Keycloak, ролі користувача перетворюються у claims, а authorization policies визначають доступ до операцій. Для сценаріїв доступу до уроків додатково перевіряється наявність активної підписки або право автора курсу [16, 20, 21].

Верифікація розгортання виконується після запуску Docker Compose. Перевіряється, що контейнер PostgreSQL проходить healthcheck, Keycloak стартує та доступний на порту 8080, API запускається на порту 5045, застосовує міграції та відкриває Swagger UI. Після цього через Postman або Swagger виконуються контрольні запити до основних endpoint-ів.

Результати автоматизованих тестів і ручної перевірки дозволяють зробити висновок, що програмне забезпечення API платформи онлайн-навчання реалізує основні вимоги до керування курсами, уроками, підписками, creator-доступом, медіафайлами та захищеним доступом до навчального контенту. Система має відтворюване середовище запуску та може бути розширена клієнтським вебінтерфейсом. Як приклад практичної верифікації роботи API варто навести скріншот успішного виконання контрольного запиту в Postman із HTTP-статусом відповіді та JSON-результатом (рисунок 3.6).

EducationPlatform API - Run results

Ran today at 06:21:31 PM · View all runs

Source	Environment	Iterations	Duration	All tests	Errors	Avg. Resp. Time
Runner	none	1	2s 588ms	26	0	10 ms

RUN SUMMARY

	1
GET OpenID Configuration	1 | 0
GET Health Check	1 | 0
GET List Courses	1 | 0
POST Create Course	2 | 0
GET Get Course	1 | 0
GET Course Not Found	1 | 0
POST Invalid Course	1 | 0
POST Upload Course Preview Image	1 | 0
GET Get Course Preview Image	1 | 0
POST Subscribe to Course	2 | 0
GET List My Subscriptions	2 | 0
POST Subscribe Again Conflict	1 | 0
POST Subscribe Missing Course	1 | 0
GET List Lessons	1 | 0
POST Create Lesson	2 | 0
GET Get Lesson	1 | 0
GET Lesson Not Found	1 | 0
POST Invalid Lesson	1 | 0
POST Upload Lesson Video	1 | 0
GET Get Lesson Video	1 | 0
DELETE Delete Lesson	1 | 0
DELETE Delete Course	1 | 0

Рисунок 3.6 – Успішне виконання контрольного запиту до API у Postman

У третьому розділі розглянуто тестування, розгортання та верифікацію програмного забезпечення API. Описано автоматизоване тестування API, план тестування, основні тестові сценарії, контейнеризацію за допомогою Docker, системні вимоги та порядок перевірки працездатності. Запропонований підхід дозволяє контролювати якість реалізації та забезпечує підготовку системи до демонстрації й подальшого розвитку.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

У межах кваліфікаційної роботи розробляється серверна частина платформи онлайн-навчання, яка може використовуватися студентами, авторами курсів та адміністраторами під час дистанційного або змішаного навчання. Тому питання безпеки життєдіяльності та охорони праці доцільно розглядати у двох пов’язаних площинах: безпечна організація навчальної діяльності користувачів цифрової платформи та безпечні умови праці фахівців, які розробляють, тестують і супроводжують програмну систему [22].

Особливістю таких систем є тривала робота з комп’ютерною технікою, мережевими сервісами, обліковими записами та цифровими навчальними матеріалами. Неправильна організація робочого місця, надмірне зорове навантаження, порушення режиму праці та відпочинку, а також ігнорування правил кібергігієни можуть негативно впливати як на фізичний стан користувачів, так і на безпечність освітнього процесу.

4.1 Безпечне використання платформи онлайн-навчання

Онлайн-навчання стало одним із поширених способів самостійного здобуття нових знань і професійних навичок. Платформа онлайн-навчання забезпечує доступ до каталогів курсів, уроків, відеоматеріалів і підписок, однак ефективність такого доступу залежить не лише від функціональності програмної системи, а й від безпечної організації роботи користувачів, які навчаються або створюють курси за комп’ютером.

Під безпекою життєдіяльності в контексті використання платформи онлайн-навчання слід розуміти комплекс організаційних, технічних і поведінкових заходів, спрямованих на збереження здоров’я користувача, запобігання перевтомі, зниження ризиків під час роботи з комп’ютерною технікою та забезпечення стабільного доступу до навчальних матеріалів. Для студентів це означає правильну

організацію робочого місця й раціональний режим навчання, а для авторів курсів - відповідальну роботу з контентом, файлами та власним обліковим записом.

Під час тривалого перегляду уроків, роботи з текстовими матеріалами або виконання практичних завдань виникає навантаження на зір, опорно-руховий апарат і нервову систему. Щоб зменшити негативний вплив, робоче місце користувача повинно бути достатньо освітленим, екран монітора має розташовуватися на зручній відстані, а поза під час роботи не повинна створювати надмірного напруження для спини, шиї та рук [23-25].

Рекомендовано організовувати роботу з платформою так, щоб навчальні або авторські сесії чергувалися з короткими перервами. Після тривалої роботи за комп’ютером доцільно робити перерви для відпочинку очей, зміни положення тіла та легкої рухової активності. Це важливо як для студентів, які переглядають відеолекції й текстові матеріали, так і для авторів, які готують уроки, завантажують медіафайли та перевіряють вміст курсу [22, 23].

Платформа онлайн-навчання також повинна сприяти безпечній поведінці користувача в цифровому середовищі. Оскільки система використовує автентифікацію через Keycloak і JWT-токени, користувачі повинні уважно ставитися до захисту власних облікових записів. Необхідно використовувати надійні паролі, не передавати токени або облікові дані іншим особам, завершувати сеанс роботи на спільних комп’ютерах і не відкривати підозрілі посилання, пов’язані з купівлею доступу, підписками або creator-доступом [21, 22].

Для студентів важливим є збереження доступності придбаних навчальних матеріалів у разі технічних збоїв або нестабільного інтернет-з’єднання. Якщо користувач працює з відеоуроками або завантаженими матеріалами, він повинен враховувати можливість втрати з’єднання, обмеження швидкості мережі або тимчасової недоступності сервісів. З боку програмного забезпечення API це враховується через поділ системи на API, базу даних, файлове сховище та сервіс автентифікації, що спрощує діагностику й відновлення роботи окремих компонентів.

Окремим аспектом є безпечне використання навчальних відеоматеріалів. Тривалий безперервний перегляд відео може сприяти перевтомі, тому навчальні матеріали бажано структурувати на окремі уроки з чіткою тривалістю. У реалізованій системі урок має поле тривалості, тип медіа та текстовий опис, що дозволяє надалі будувати інтерфейс, у якому користувач бачитиме обсяг матеріалу й зможе планувати навчання без надмірного навантаження.

Для авторів курсів безпека життєдіяльності також пов’язана з відповідальною підготовкою матеріалів. Оскільки автор отримує можливість створювати курси після активації creator-доступу, він повинен уникати розміщення шкідливих файлів, некоректних посилань або матеріалів, які можуть порушувати правила користування платформою. Наявність ролей, підписок і creator-доступу дозволяє обмежити створення навчального контенту лише тими користувачами, які мають відповідні права.

У разі використання платформи онлайн-навчання як відкритого сервісу необхідно передбачати зрозумілі правила користування цифровими можливостями системи. Студенти повинні знати, як безпечно організувати робоче місце, як захищати обліковий запис і як діяти у випадку проблем із доступом до придбаного курсу. Автори курсів повинні розуміти вимоги до завантажуваних матеріалів, відповідальність за власний контент і правила використання creator-доступу.

Таким чином, безпечна організація роботи з платформою онлайн-навчання поєднує технічні та організаційні заходи. З технічного боку система повинна забезпечувати контроль доступу, стабільну роботу API, захист навчального контенту та коректну обробку сценаріїв підписок і creator-доступу. З організаційного боку користувачі повинні дотримуватися правил роботи з комп’ютерною технікою, підтримувати здоровий режим навчання або підготовки курсів та відповідально користуватися власними обліковими записами.

4.2 Охорона праці під час розробки та супроводу

Розробка серверної частини платформи онлайн-навчання виконується переважно за персональним комп’ютером із використанням середовища розробки, браузера, терміналу, Docker Desktop, системи керування базами даних, Swagger UI та Postman. Така діяльність належить до розумової праці з високою концентрацією уваги та тривалим використанням засобів обчислювальної техніки, тому потребує дотримання вимог охорони праці [23, 25].

Робоче місце розробника повинно бути організоване так, щоб забезпечити зручну позу, достатнє освітлення, безпечне розміщення обладнання та мінімізацію факторів, які призводять до перевтоми. Монітор має бути розташований перед користувачем на зручній відстані, верхня межа екрана бажано на рівні очей або трохи нижче, клавіатура і миша повинні дозволяти працювати без надмірного напруження кистей [24, 25].

Освітлення робочого місця має бути рівномірним і не створювати відблисків на екрані. Недостатнє освітлення або різкий контраст між екраном і навколишнім середовищем підвищує навантаження на зір. Під час роботи з кодом, документацією та тестовими сценаріями розробник тривалий час фокусує погляд на дрібному тексті, тому правильне освітлення є важливою умовою профілактики зорової втоми [24].

Під час розробки та тестування API значну частину часу займає робота з середовищем програмування, логами, консоллю, Docker-контейнерами та браузером. Для запобігання перевтомі необхідно дотримуватися таких правил:

- регулярно робити короткі перерви після тривалої роботи за комп’ютером;
- змінювати положення тіла та виконувати легкі вправи для очей, шиї та спини;
- не працювати тривалий час у незручній позі або за недостатнього освітлення;
- планувати складні завдання так, щоб уникати надмірної концентрації без відпочинку;

Важливим фактором охорони праці є електробезпека. Комп’ютерна техніка, мережеве обладнання та периферійні пристрої повинні бути справними, підключеними до безпечної електромережі та розміщеними так, щоб кабелі не створювали ризику спотикання або пошкодження. Не допускається використання пошкоджених кабелів, нестабільних подовжувачів або обладнання з ознаками перегріву [22].

Під час роботи з Docker і локальними сервісами розробник запускає декілька контейнерів, зокрема API, PostgreSQL та Keycloak. Хоча це програмні компоненти, їх робота впливає на навантаження комп’ютера. Надмірне використання ресурсів може призводити до перегріву пристрою, підвищеного шуму системи охолодження та уповільнення роботи. Тому під час тестування слід контролювати стан системи, завершувати непотрібні контейнери та уникати запуску зайвих служб.

Охорона праці розробника також включає психологічні аспекти. Робота з помилками, тестами, конфігураціями автентифікації та контейнерним середовищем часто потребує високої концентрації. Щоб зменшити стрес і ризик помилок, доцільно використовувати поетапну перевірку змін, автоматизовані тести, резервні копії документів і контроль версій. У межах цієї роботи такі підходи застосовуються через тестовий проєкт AspNetCore.Tests, Docker Compose та окреме збереження робочих артефактів дипломної записки [22].

Під час супроводу серверної частини важливо дотримуватися правил інформаційної безпеки, оскільки помилки в роботі з конфігурацією можуть впливати на доступ користувачів до навчальних матеріалів. Паролі, токени, параметри підключення до бази даних і налаштування Keycloak не повинні передаватися стороннім особам або зберігатися у відкритому вигляді в непередбачених місцях. Для демонстраційного середовища допускаються спрощені налаштування, але для реального впровадження необхідно використовувати захищені секрети та HTTPS.

У разі виникнення аварійної ситуації, наприклад запаху гару, перегріву пристрою, пошкодження кабелю або нестабільної роботи електромережі, роботу необхідно припинити, за можливості безпечно вимкнути обладнання та повідомити

відповідальну особу. Не слід самостійно ремонтувати електрообладнання без відповідної кваліфікації. У навчальній або офісній лабораторії такі дії повинні виконуватися відповідно до локальних інструкцій з охорони праці.

Таким чином, охорона праці під час розробки та супроводу серверної частини платформи онлайн-навчання передбачає правильну організацію робочого місця, дотримання режиму праці й відпочинку, контроль електробезпеки, обережну роботу з локальною інфраструктурою та відповідальне поводження з конфігураційними даними. Дотримання цих вимог зменшує ризики для здоров’я розробника та сприяє стабільній роботі над програмною системою.

У четвертому розділі розглянуто питання безпеки життєдіяльності та охорони праці, пов’язані з використанням і розробкою платформи онлайн-навчання. Перше питання стосується безпечної організації дистанційного навчання для користувачів системи, друге - вимог охорони праці під час розробки, тестування та супроводу серверної частини. Обидва питання мають практичне значення, оскільки програмна система використовується в цифровому освітньому середовищі та потребує безпечної організації роботи з комп’ютерною технікою.

ВИСНОВКИ

У кваліфікаційній роботі було виконано аналіз, проєктування, розробку та перевірку програмного забезпечення API платформи онлайн-навчання EducationPlatform. Метою роботи було створення програмного забезпечення API для marketplace-платформи, яка забезпечує керування курсами, уроками, підписками, медіаматеріалами, creator-доступом і доступом користувачів до захищеного навчального контенту.

У першому розділі було проаналізовано предметну область і визначено основні вимоги до програмної системи. Було встановлено, що платформа онлайн-навчання повинна підтримувати різні ролі користувачів, зокрема студента, автора курсу та адміністратора, а також забезпечувати створення курсів після активації creator-доступу, публікацію уроків, оформлення підписок на курси і контроль доступу до навчальних матеріалів.

У другому розділі було спроєктовано архітектуру програмної системи та описано реалізацію її основних компонентів. Система побудована на основі ASP.NET Core і має багаторівневу структуру з окремими проєктами для HTTP-рівня, прикладної логіки, моделі даних, міграцій PostgreSQL і спільних моделей. Такий підхід відповідає принципам Clean Architecture, забезпечує розділення відповідальності та полегшує подальшу підтримку проєкту.

У межах розробки було реалізовано модулі курсів, уроків, підписок, creator-доступу, локального файлового сховища та інтеграції з Keycloak. Для автентифікації використовується JWT Bearer authentication, а для авторизації - політики доступу та перевірка ролей користувача. Це дозволяє захищати операції створення навчального контенту, надавати право автора лише після активації creator-доступу та обмежувати доступ до уроків лише для користувачів з відповідними правами.

У третьому розділі було розглянуто тестування, розгортання та верифікацію системи. Автоматизовані тести реалізовано з використанням NUnit, WebApplicationFactory та Entity Framework Core InMemory provider. Тестовий набір

перевіряє основні endpoint-и, бізнес-сценарії, помилкові випадки та правила авторизації. Фактичний запуск тестів підтвердив працездатність поточної реалізації: усі 35 тестів завершилися успішно.

Для розгортання системи підготовлено Dockerfile і docker-compose конфігурацію. Контейнерне середовище включає API, PostgreSQL та Keycloak, що робить запуск системи відтворюваним і зручним для демонстрації. Swagger UI та Postman collection використовуються як технічні інтерфейси для ручної перевірки API та демонстрації послідовних бізнес-сценаріїв.

У четвертому розділі було розглянуто питання безпеки життєдіяльності та охорони праці, пов’язані з використанням і розробкою платформи онлайн-навчання. Окрему увагу приділено безпечній роботі студентів і авторів курсів із цифровим сервісом, захисту облікових записів, раціональному режиму роботи за комп’ютером, електробезпеці та умовам праці розробника під час супроводу серверної частини.

Отже, у результаті виконання кваліфікаційної роботи було створено функціональний серверний прототип платформи онлайн-навчання marketplace-типу, який відповідає поставленим вимогам і може бути використаний як основа для подальшого розвитку. Подальше вдосконалення системи може включати розробку повноцінного frontend-застосунку, розширення адміністративних можливостей, інтеграцію з платіжними сервісами, додавання інструментів для авторів курсів та перенесення файлового сховища у хмарну інфраструктуру.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для здобувачів першого (бакалаврського) рівня вищої освіти за спеціальністю 121 «Інженерія програмного забезпечення» / уклад. кафедра програмної інженерії ТНТУ імені Івана Пулюя. Тернопіль, 2024. 46 с.
2. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. Київ : ДП «УкрНДНЦ», 2016. 31 с.
3. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Київ : ДП «УкрНДНЦ», 2016. 20 с.
4. Guide to the Software Engineering Body of Knowledge (SWEBOK Guide). Version 4.0 / ed. H. Washizaki. IEEE Computer Society, 2024. 411 p.
5. Bass L., Clements P., Kazman R. Software Architecture in Practice. 4th ed. Boston : Addison-Wesley Professional, 2021. 464 p.
6. Martin R. C. Clean Architecture: A Craftsman’s Guide to Software Structure and Design. Boston : Prentice Hall, 2017. 432 p.
7. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley Professional, 2002. 560 p.
8. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston : Addison-Wesley Professional, 2003. 560 p.
9. Олянін, Д., Цуприк, Г. (2025) Transformer Neural Networks in Industry 4.0 / Д. Олянін, Г. Цуприк, Т. Говорущенко, О. Багрій-Заяць, І. Андрущак // Computer Information Technologies in Industry 4.0: proceedings of the 3rd International Workshop (CITI-2025), Ternopil, Ukraine, 11–12 June 2025. – Ternopil : Ternopil Ivan Puluj National Technical University, 2025 (Scopus) https://ceur-ws.org/Vol-4057/
10. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : doctoral dissertation. University of California, Irvine, 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (дата звернення: 31.05.2026).

11. Pautasso C., Zimmermann O., Leymann F. RESTful Web Services vs. “Big” Web Services: Making the Right Architectural Decision. Proceedings of the 17th International Conference on World Wide Web. 2008. P. 805-814. DOI: 10.1145/1367497.1367606.
12. Masse M. REST API Design Rulebook. Sebastopol : O’Reilly Media, 2011. 114 p.
13. Microsoft. ASP.NET Core documentation. URL: https://learn.microsoft.com/en-us/aspnet/core/ (дата звернення: 31.05.2026).
14. Microsoft. Minimal APIs overview. URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/minimal-apis (дата звернення: 23.05.2026).
15. Microsoft. Entity Framework Core documentation. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 23.05.2026).
16. Microsoft. Authentication and authorization in ASP.NET Core. URL: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/ (дата звернення: 23.05.2026).
17. Swashbuckle.AspNetCore documentation. URL: https://github.com/domaindrivendev/Swashbuckle.AspNetCore (дата звернення: 23.05.2026).
18. PostgreSQL Global Development Group. PostgreSQL 16 Documentation. URL: https://www.postgresql.org/docs/16/ (дата звернення: 24.05.2026).
19. Docker Inc. Docker Compose documentation. URL: https://docs.docker.com/compose/ (дата звернення: 24.05.2026).
20. Keycloak. Server Administration Guide. URL: https://www.keycloak.org/docs/latest/server_admin/ (дата звернення: 20.05.2026).
21. OWASP Foundation. OWASP API Security - 2023. URL: https://owasp.org/API-Security/editions/2023/en/0x00-header/ (дата звернення: 31.05.2026).
22. Безпека життєдіяльності та охорона праці : підруч. / В.В. Сокуренко, О.М. Бандурка та ін. – Харків : ХНУВС, 2021. 308 с.
23. Жидецький В.Ц. Охорона праці користувачів комп’ютерів : підручник. Львів : Афіша, 2020. 176 с.

24. ДСТУ 8604:2015 Дизайн і ергономіка. Робоче місце для виконання робіт у положенні сидячи. Загальні ергономічні вимоги.
25. НПАОП 0.00-7.15-18. Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями : наказ від 14.02.2018 № 207.
26. Tsupryk, H., Olianin, D. (2025). Vydobuvannia danyh z tekstu vykorystovuiuchy transformerni neironni merezhi [Data extraction from text using Transformer Neural Networks]. Information Technology: Computer Science, Software Engineering and Cyber Security, 125–130, DOI: https://doi.org/10.32782/IT/2025-2-13
27. ОЛЯНІН D., & ЦУПРИК Н. (2025). Огляд ролі трансформерних нейроних мереж у видобувані інформації із неструктурованих даних. Measuring and computing devices in technological processes, 82(2), 360–364. https://doi.org/10.31891/2219-9365-2025-82-52
28. Tsupryk H. LLM-based Extraction from Resumes / D. Olianin, H. Tsupryk // Advanced Technologies in Scientific Research: collection of scientific papers with proceedings of the 1st International Scientific and Practical Conference, Rotterdam, Netherlands, 20–22 August 2025. – International Scientific Unity, 2025. – 72-76
29. Yaroslav Kotov, Evhenia Yavorska, Halyna Tsupryk, Róża Dzierżak 1 , Oleksandr Reshetnik, Viktoriia Bokovets (2025) Evaluating interoperability and data quality in FHIR-based AI assessment pipelines. Proc. SPIE 14009, Photonics Applications in Astronomy, Communications, Industry, and High Energy Physics Experiments 2025, 140091F (30 December 2025) https://doi.org/10.1117/12.3100561

ДОДАТКИ

Додаток А. Тези конференції

IX Міжнародна студентська науково - технічна конференція
"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"

УДК 004.42
Каленюк Д. – ст. гр. СП-42
Тернопільський національний технічний університет імені Івана Пулюя

ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ API ПЛАТФОРМИ ДИСТАНЦІЙНОГО НАВЧАННЯ З ВИКОРИСТАННЯМ СУЧАСНИХ IT-ТЕХНОЛОГІЙ

Науковий керівник: кандидат технічних наук, доцент Цуприк Г.Б.

Kaleniuk D.
Ternopil Ivan Puluj National Technical Univercity

DESIGN AND IMPLEMENTATION OF AN API FOR A DISTANCE LEARNING PLATFORM USING MODERN IT TECHNOLOGIES

Supervisor: PhD, Associate Professor H. B. Tsupryk

Ключові слова: API, програмна архітектура, веб-сервіси.
Keywords: API, software architecture, web services.

Сучасні платформи дистанційного навчання є складними інформаційними системами, що забезпечують взаємодію між користувачами, навчальними ресурсами та сервісами оцінювання знань. Ключову роль у таких системах відіграють програмні інтерфейси прикладного програмування (API), які реалізують доступ до функціональних можливостей платформи та забезпечують інтеграцію з зовнішніми сервісами. Зі зростанням навантаження та вимог до безпеки виникає потреба у побудові масштабованої, гнучкої та захищеної архітектури.

У сучасних підходах до розробки API значна увага приділяється архітектурним патернам, що дозволяють розділити відповідальності між компонентами системи. Зокрема, Clean Architecture забезпечує незалежність бізнес-логіки від інфраструктурних деталей, що підвищує тестованість та підтримуваність системи [1]. Додатково використання патерну CQRS (Command Query Responsibility Segregation) дозволяє розділити операції читання та запису даних, оптимізуючи продуктивність та спрощуючи масштабування системи [2].

У рамках роботи запропоновано архітектуру API платформи дистанційного навчання, яка включає рівні представлення (Web API), прикладний (Application), доменний (Domain) та інфраструктурний (Infrastructure). Взаємодія між шарами здійснюється через інтерфейси та медіатор-патерн, що забезпечує слабке зв'язування компонентів. Для реалізації API використано REST-архітектурний стиль, який дозволяє стандартизувати обмін даними між клієнтами та сервером [3].

Практична реалізація виконана з використанням платформи ASP.NET Core, що забезпечує високу продуктивність і підтримку сучасних веб-технологій. Для доступу до даних застосовано Entity Framework Core разом із системою керування базами даних PostgreSQL. Для реалізації безпечної автентифікації та авторизації використано Keycloak, що підтримує протокол OAuth 2.0 та дозволяє централізовано керувати користувачами і ролями.

Важливою складовою розробки API є забезпечення якості програмного забезпечення шляхом тестування. Для перевірки коректності роботи системи застосовуються різні рівні тестування, зокрема модульне та інтеграційне. Модульні

тести дозволяють перевіряти окремі компоненти бізнес-логіки ізольовано від зовнішніх залежностей, тоді як інтеграційні тести забезпечують перевірку взаємодії між шарами системи та коректність роботи API-ендпоінтів. Використання принципів Clean Architecture та інверсії залежностей дозволяє ефективно застосовувати мок-об'єкти, що спрощує автоматизоване тестування без необхідності підключення реальної бази даних. Це підвищує швидкість виконання тестів і надійність програмного забезпечення.

Експериментальна перевірка показала, що застосування CQRS у поєднанні з багаторівневою архітектурою дозволяє підвищити продуктивність системи при обробці запитів та зменшити навантаження на базу даних. Використання зовнішнього сервісу ідентифікації забезпечує підвищений рівень безпеки та спрощує інтеграцію з іншими системами. Запропонована архітектура характеризується високою модульністю, що дозволяє легко розширювати функціонал платформи, зокрема шляхом додавання нових освітніх сервісів або аналітичних модулів.

Таким чином, розробка архітектури API платформи дистанційного навчання з використанням сучасних підходів, таких як Clean Architecture, CQRS та OAuth 2.0, дозволяє створити масштабоване, безпечне та ефективне програмне забезпечення. Подальші дослідження можуть бути спрямовані на впровадження мікросервісної архітектури, оптимізацію продуктивності та використання інтелектуальних алгоритмів для персоналізації навчального процесу.

Література:

1. Robert C. Martin. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston: Prentice Hall, 2017. 432 p.

2. Greg Young. Domain-Driven Design and Command Query Responsibility Segregation (CQRS). — Електронний ресурс. — Режим доступу: https://cqrs.files.wordpress.com/2010/11/cqrs_documents.pdf (дата звернення: 10.04.2026)

3. Fielding R. T. Architectural Styles and the Design of Network-Based Software Architectures: Doctoral dissertation. University of California, Irvine, 2000. 162 p.

4. Internet Engineering Task Force. OAuth 2.0 Authorization Framework (RFC 6749), 2012. 76 p.

5. Олянін, Д., Цуприк, Г. (2025) Transformer Neural Networks in Industry 4.0 / Д. Олянін, Г. Цуприк, Т. Говорущенко, О. Багрій-Заяць, І. Андрущак // Computer Information Technologies in Industry 4.0: proceedings of the 3rd International Workshop (CITI-2025), Ternopil, Ukraine, 11–12 June 2025. – Ternopil : Ternopil Ivan Puluj National Technical University, 2025 (Scopus) https://ceur-ws.org/Vol-4057/

6. Tsupryk, H., Olianin, D. (2025). Vydobuvannia danyh z tekstu vykorystovuiuchy transformerni neironni merezhi [Data extraction from text using Transformer Neural Networks]. Information Technology: Computer Science, Software Engineering and Cyber Security, 125–130, DOI: https://doi.org/10.32782/IT/2025-2-13

7. ОЛЯНІН D., & ЦУПРИК Н. (2025). Огляд ролі трансформерних нейроних мереж у видобувані інформації із неструктурованих даних. Measuring and computing devices in technological processes, 82(2), 360–364. https://doi.org/10.31891/2219-9365-2025-82-52

Додаток Б. Код бізнес-логіки

Лістинг Б.1 - Handler створення курсу

```
using Application.Modules.Courses.Models;
using Application.Modules.Courses.Requests;
using Data;
using Data.Entities;
using MediatR;

namespace Application.Modules.Courses.Handlers;

internal sealed class AddCourseHandler(EducationDbContext dbContext)
    : IRequestHandler<AddCourseCommand, AddCourseResponse>
{
    public async Task<AddCourseResponse> Handle(AddCourseCommand request,
CancellationToken cancellationToken)
    {
        var timestamp = DateTimeOffset.UtcNow;
        var entity = new CourseEntity
        {
            Id = Guid.NewGuid(),
            CreatorId = request.CreatorId,
            Name = request.Name,
            Summary = request.Summary,
            Description = request.Description,
            Language = request.Language,
            Status = request.Status,
            Price = request.Price,
            Tags = request.Tags.ToArray(),
            CreateTimestamp = timestamp,
            UpdateTimestamp = timestamp,
        };

        dbContext.Courses.Add(entity);
        await dbContext.SaveChangesAsync(cancellationToken);

        return new AddCourseResponse(entity.Id);
    }
}
```

Лістинг Б.2 - Handler отримання уроку з перевіркою доступу

```
using Application.Modules.Lessons.Models;
using Application.Modules.Lessons.Requests;
using Common.Models;
using Data;
using MediatR;
using Microsoft.EntityFrameworkCore;

namespace Application.Modules.Lessons.Handlers;

internal sealed class GetLessonByIdHandler(EducationDbContext dbContext)
    : IRequestHandler<GetLessonByIdRequest, GetLessonByIdResult>
{
    public async Task<GetLessonByIdResult> Handle(GetLessonByIdRequest request,
CancellationToken cancellationToken)
    {
        var lesson = await dbContext.Lessons
            .AsNoTracking()
            .Where(x => x.Id == request.LessonId)
```

```
            .Select(x => new
            {
                Entity = x,
                CourseCreatorId = x.Course == null ? (Guid?)null :
x.Course.CreatorId,
            })
            .FirstOrDefaultAsync(cancellationToken);

        if (lesson is null || lesson.Entity.CourseId is null ||
lesson.CourseCreatorId is null)
            return new GetLessonByIdResult(LessonAccessStatus.NotFound);

        if (!request.CanManageAllLessons &&
            lesson.CourseCreatorId != request.UserId &&
            !await HasActiveSubscription(lesson.Entity.CourseId.Value,
request.UserId, cancellationToken))
        {
            return new GetLessonByIdResult(LessonAccessStatus.Forbidden);
        }

        return new GetLessonByIdResult(
            LessonAccessStatus.Success,
            Map(lesson.Entity));
    }

    private async Task<bool> HasActiveSubscription(
        Guid courseId,
        Guid userId,
        CancellationToken cancellationToken)
    {
        return await dbContext.Subscriptions
            .AsNoTracking()
            .AnyAsync(
                x => x.CourseId == courseId &&
                     x.UserId == userId &&
                     x.Status == SubscriptionStatus.Active,
                cancellationToken);
    }

    private static Lesson Map(Data.Entities.LessonEntity entity)
    {
        return new Lesson
        {
            Id = entity.Id,
            CourseId = entity.CourseId,
            Name = entity.Name,
            Summary = entity.Summary,
            Description = entity.Description,
            Text = entity.Text,
            Status = entity.Status,
            Duration = entity.Duration,
            MediaType = entity.MediaType,
            VideoUrl = entity.VideoPath == null ? null :
$"/lessons/{entity.Id}/video",
            CreateTimestamp = entity.CreateTimestamp,
        };
    }
}
```

Додаток В. Репозиторій GitHub

https://github.com/daffy24/2025-2026-KRB-SE-42-Dmytro-KALENIUK