

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Web додаток для організації онлайн курсів»

Виконав студент групи ПЗ-21-2

ГЕРАСИМЧУК Дмитро Олександрович

Керівник роботи:

ЄФРЕМОВ Юрій Миколайович

Рецензент:

МОРОЗОВ Андрій Васильович

Житомир – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедра інженерії
програмного забезпечення

Тетяна ВАКАЛЮК

«26» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ГЕРАСИМЧУК Дмитро Олександрович**

Керівник роботи: **ЄФРЕМОВ Юрій Миколайович**

Тема роботи: **«Web додаток для організації онлайн курсів»**,

затверджена Наказом закладу вищої освіти від **«14» лютого 2025р., №61/с**

Вихідні дані для роботи: **розробка веб-додатку для організації онлайн курсів.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів,
що їх стосуються:

Розділ	Консультант	Дата	
		Завдання видав	Завдання прийняв
1	Єфремов Ю.М.	27.02.2025р.	27.02.2025р.
2	Єфремов Ю.М.	27.02.2025р.	27.02.2025р.
3	Єфремов Ю.М.	27.02.2025р.	27.02.2025р.

РЕФЕРАТ

Випускна робота бакалавра складається з реалізації програми “Онлайн платформа з проведення онлайн курсів”, а також пояснювальної записки. Пояснювальна записка до випускної роботи містить 64 сторінки, включає 12 ілюстрації, 4 таблиці, 1 додаток та 36 використаних літературних джерел.

Основною метою цієї роботи є побудова додатку та його архітектури для онлайн платформи, який відповідатиме високим стандартам у сфері інженерії програмного забезпечення. Система підтримуватиме щонайменше дві ролі: користувача та адміністратора. Авторизація буде реалізована за допомогою JWT, з можливістю додаткового підтвердження через електронну пошту.

Адміністратор матиме доступ до функцій створення, редагування, додавання та видалення курсів і тем, а також проведення онлайн-тестування.

Користувачі зможуть проходити курси та тести, а система відстежуватиме їхній прогрес у відсотковому співвідношенні.

У процесі виконання випускної роботи було написано програмний код, за найкращими практиками у сфері ІТ. Також передбачається можливість завантаження мультимедійного контенту (зображень, відео тощо) через хмарне сховище AWS S3.

Ключові слова: C#, Postgresql, WEB, НАВЧАННЯ, .NET, React, DOCKER, VIRTUALIZATION, MVC, JS, S3, JWT, IAM.

					ІІЗ – 121 – 25 – ІІЗ				
Змн.	Арк.	№ докум.	Підпис	Дата					
Розроб.		Д. О. Герасимчук			Web додаток для організації онлайн курсів	Літ.	Арк.	Аркушів	
Керівник		Ю.М. ЄФРЕМОВ					3	64	
						Житомирська політехніка, група ІІЗ-21-2			
Рецензент		А.В.МОРОЗОВ							
Зав. каф.		Т.А. ВАКАЛЮК							
					Пояснювальна записка				

ABSTRACT

The bachelor's thesis consists of the implementation of the program “Online Platform for Conducting Online Courses” and an explanatory note. The explanatory note to the graduation thesis is 64 pages long, includes 12 illustrations, 4 tables, 1 appendix, and 36 references.

The main goal of this work is to build an application and its architecture for an online platform that will meet high standards in the field of software engineering. The system will support at least two roles: user and administrator. Authorization will be implemented with the help of JWT, with the possibility of additional confirmation via email.

The administrator will have access to the functions of creating, editing, adding and deleting courses and topics, as well as conducting online testing.

Users will be able to take courses and tests, and the system will track their progress as a percentage.

In the process of completing my graduate work, I wrote the program code according to the best IT practices. The system also provides the ability to upload multimedia content (images, videos, etc.) via AWS S3 cloud storage.

Keywords: C#, Postgresql, WEB, TRAINING, .NET, React, DOCKER, VIRTUALIZATION, MVC, JS, S3, JWT, IAM.

		Д.О. Герасимчук			ІІЗ – 121 – 25 – ІІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ	4
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	7
ВСТУП	8
РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ	10
1.1 Поставлення завдань, та їх вирішення	10
1.2 Аналіз схожого програмного забезпечення	15
1.3. Визначення архітектури програмного забезпечення	23
1.4. Обґрунтування вибору інструментальних засобів та вимоги до програмного забезпечення	24
Висновки до першого розділу	26
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА WEB ДОДАТКУ ДЛЯ ОРГАНІЗАЦІЇ ОНЛАЙН КУРСІВ	28
2.1 Проектування бази даних.....	28
2.2 Розробка архітектури додатку	30
2.3 Розробка функціональних методів для роботи програми	32
2.4 Збереження файлів.....	35
Висновки до другого розділу.....	38
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК ВИКОРИСТАННЯ ДОДАТКУ	40
3.1 Опис роботи з програмним додатком (Опис інтерфейсу)	40
3.2 Роутинг та маршрутизація	45
3.3 Робота з авторизацією (JWT).....	47
3.4 User Stories та User Flows.....	49
3.5 Заходи безпеки бази даних	51
3.6 Застосування Postman для адміністрування та тестування API.....	54
Висновки до третього розділу	59
ВИСНОВКИ.....	61

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ..... 62

ДОДАТКИ..... 65

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ПЗ - програмне забезпечення.

БД - база даних.

СУБД - система управління базами даних.

CRUD - create, read, update, delete.

SQL - Structured Query Language.

HTML - HyperText Markup Language.

CSS - Cascading Style Sheets.

JS - JavaScript.

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ПЗ	Арк.
		Ю.М. Єфремов				7
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми. Сучасний ринок освіти переживає стрімкий перехід у цифровий формат, що зумовлює зростання попиту на якісні онлайн платформи для навчання. Особливо після пандемії COVID-19 відбулося масштабне впровадження дистанційного навчання, яке виявило потребу в зручних, надійних та функціональних інструментах для організації освітнього процесу.

Веб-додатки для онлайн навчання дозволяють підвищити доступність освіти, надати можливість навчатися у зручний час, а також адаптувати контент під потреби користувача. Реалізація такого сервісу є актуальною задачею в умовах цифровізації освіти та загальної інформатизації суспільства.

Мета роботи. Метою цієї роботи є проектування та реалізація мікросервісної архітектури веб-додатку для організації онлайн курсів, який забезпечуватиме високу гнучкість, безпеку та масштабованість. Основним функціоналом є можливість створення та проходження курсів, проведення онлайн-тестування, збереження результатів та відстеження прогресу користувача.

У процесі реалізації роботи передбачено використання таких технологій, як Docker, .NET, React.js, AWS S3, JWT. Заплановане створення зручного інтерфейсу для користувача та адміністратора, а також налаштування взаємодії між сервісами через API шлюз.

Завдання роботи:

- Аналіз існуючих рішень у сфері e-learning та UI/UX для освітніх платформ;
- Розробка моделі бази даних для зберігання курсів, тем, тестів та прогресу;
- Реалізація backend частини на базі .NET Core [18] з використанням REST API;
- Розробка frontend частини з використанням React.js [5];

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		8

- Інтеграція хмарного сховища AWS S3 для зберігання мультимедійних файлів;
- Реалізація безпечної автентифікації з використанням JWT;
- Докеризація проєкту для забезпечення портативності та зручності розгортання.

Об'єктом дослідження є процес проектування та реалізації web-платформи для онлайн навчання. У фокусі дослідження - створення багатофункціональної інформаційної системи, що дозволяє ефективно організувати взаємодію між викладачами (адміністраторами платформи) та слухачами.

Предметом дослідження є розробка web-додатку для онлайн курсів із застосуванням сучасних технологій. У процесі реалізації використовуються такі технічні засоби, як .NET для серверної логіки, React.js [35] для створення динамічного інтерфейсу користувача, PostgreSQL для роботи з базами даних. AWS S3 використовується для зберігання мультимедійних матеріалів, а Docker - для контейнеризації та розгортання проєкту. Авторизація та захист даних реалізовані з використанням JWT, що забезпечує надійність та безпеку.

Такий підхід дозволяє створити високоякісний продукт, що відповідає вимогам сучасної цифрової освіти.

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		9

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ

1.1 Поставлення завдань, та їх вирішення

Метою дипломної роботи є створення програмної частини, а також проєктування та реалізація бази даних для веб-додатку, що забезпечує організацію та проходження онлайн-курсів через інтернет. Основними вимогами до системи є стабільна швидкодія, масштабованість, безпечна авторизація користувачів та зручний інтерфейс.

Однією з ключових складових проєкту є ефективне проєктування бази даних, з акцентом на забезпечення цілісності, узгодженості та безпеки даних. Швидкість обробки запитів та підтримка великої кількості одночасних користувачів є критично важливими, зважаючи на те, що система має підтримувати відстеження прогресу, виконання тестувань та збереження результатів.

Серверна частина веб-додатку реалізована за допомогою фреймворку .NET, який забезпечує стабільну роботу, хорошу масштабованість і підтримку сучасних підходів до розробки, таких як REST API, модульність і залежності через ін'єкцію. Для зручної роботи з базою даних використовується Entity Framework Core - ORM, що дозволяє працювати з об'єктами в C# та ефективно виконувати запити до реляційної бази даних.

Веб-додаток передбачає підтримку ролей адміністратора та користувача. Адміністратор має можливість створювати та редагувати курси, теми, тести, а також керувати користувачами та їх результатами. Користувачі можуть реєструватися, проходити навчальні матеріали, здавати тести, переглядати відео, отримувати оцінки та аналізувати власний прогрес.

Для автентифікації та авторизації використовується JWT (JSON Web Token), що дозволяє реалізувати безпечний та масштабований механізм контролю доступу. Всі дані про користувачів, курси та результати тестування зберігаються у захищеній базі даних із відповідним рівнем доступу.

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				10
Змн.	Арк.	№ докум.	Підпис	Дата		

Обрані технології:

У якості серверної частини обрано .NET з мовою програмування C#. Цей фреймворк забезпечує розробку надійних, масштабованих і безпечних веб-API, що відповідають сучасним вимогам до продуктивності.

Для реалізації клієнтської частини використано React.js - популярну JavaScript-бібліотеку, яка дозволяє створювати швидкий та зручний інтерфейс користувача. Завдяки архітектурі SPA (Single Page Application) взаємодія з додатком відбувається без перезавантаження сторінок. В якості вебсервера застосовано Nginx, який виконує роль зворотного проксі-сервера та відповідає за маршрутизацію запитів між фронтендом і бекендом, що сприяє ефективній роботі системи.

Для контейнеризації компонентів проєкту та створення ізольованого середовища розробки використано Docker. Це дозволяє забезпечити стабільність та уніфікованість запуску системи в різних середовищах.

Система керування базами даних – PostgreSQL [33], яка надає широкі можливості для роботи з реляційними даними та підтримує складні запити і структури. Для адміністрування бази даних використовується pgAdmin - веб-інтерфейс, який спрощує керування базою, дозволяє виконувати SQL-запити та здійснювати моніторинг її стану.

Завдання включають визначення основних функцій бази даних та аналіз вимог. У системі авторизації адміністратор школи має можливість створювати та надавати доступ до курсів. Роль "Вчитель" дозволяє створювати курси, створювати тести та визначати кількість завдань, контент завдань, їх опис та інше. Роль "Адмін" дає можливість реєструвати користувачів та редагувати всі сутності. Студент має можливість зайти на платформу, пройти тести та отримати оцінку за конкретний тест з певної теми. Вчитель може реєструватися на платформі самостійно. Школи мають свої теми, лекції та заняття. Необхідно проаналізувати способи реалізації вимог та спроектувати

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ПЗ	Арк.
		Ю.М. Єфремов				11
Змн.	Арк.	№ докум.	Підпис	Дата		

загальний алгоритм взаємодії класів між собою і елементами інтерфейсу. У результаті буде виконано проектування та реалізація системи.

Реалізувати сутності:

Клас User у C# представляє сутність користувача в системі для організації онлайн-курсів і створений для інтеграції з Entity Framework Core у середовищі ASP.NET Core [1]. Він реалізує об'єктно-реляційне відображення, дозволяючи зручно працювати з базою даних та пов'язаними сутностями.

Основою класу є набір властивостей, що описують користувача: Id - унікальний ідентифікатор, який автоматично генерується базою даних; Name - ім'я з обмеженою довжиною; Email - унікальна адреса, що також може бути використана для автентифікації; Password - хешований пароль. Властивість Role визначається через перелік UserRole і задає роль користувача в системі - студент, викладач або адміністратор. Також фіксується дата створення облікового запису у полі CreatedAt.

Для зв'язків з іншими сутностями використовуються навігаційні властивості. Викладач має колекцію курсів, які він веде (CoursesTaught), що відображає зв'язок один-до-багатьох. Студент бере участь у багатьох курсах через сутність Enrollment, що реалізує зв'язок багато-до-багатьох між користувачами й курсами. Окрім цього, студент може подавати роботи - для цього передбачено зв'язок один-до-багатьох із сутністю Submission.

У конструкторі класу ініціалізуються колекції пов'язаних об'єктів: курси, в яких користувач є викладачем, записи про участь у курсах, а також подані завдання. Усі властивості мають стандартні автосвойства з відкритими геттерами і сеттерами, що забезпечує сумісність з EF Core.

Хоча в класі відсутні визначені події життєвого циклу, такі як OnCreated чи OnSaved, за потреби вони можуть бути реалізовані через інтерсептори або логіку в методі DbContext.SaveChanges().

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		12

Таким чином, клас User є гнучкою й розширюваною моделлю для представлення учасників освітнього процесу в системі онлайн-курсів - як студентів, так і викладачів чи адміністраторів.

Клас Course у C# моделює навчальний курс у системі онлайн-освіти та призначений для роботи з Entity Framework Core. Він дозволяє зберігати й управляти інформацією про курси, викладачів, уроки та студентські підписки в реляційній базі даних.

Ключовою властивістю є Id - унікальний ідентифікатор курсу, який автоматично генерується базою даних. Поля Title та Description відповідають за назву й опис курсу й мають обмеження на обов'язковість і довжину. Через зовнішній ключ TeacherId курс пов'язується з користувачем, який є викладачем. Навігаційна властивість Teacher дозволяє безпосередньо звертатися до об'єкта викладача - це реалізація зв'язку багато-до-одного з сутністю User.

Курс має дату створення (CreatedAt), що задається у форматі DateTime, і булеве поле IsPublished, яке вказує, чи доступний курс для студентів. До курсу належать колекції Lessons - список уроків (один-до-багатьох), а також Enrollments - записи студентів, які підписалися на курс (багато-до-багатьох, через проміжну сутність Enrollment). У конструкторі класу ініціалізуються обидві колекції - уроків і підписок - щоб уникнути null-посилань під час роботи з новими об'єктами.

Клас також містить стандартні геттери та сеттери для всіх властивостей. Крім базових методів доступу до назви, опису, викладача, дати створення й статусу публікації, реалізовані методи для роботи з колекціями: додавання й видалення уроків (addLesson, removeLesson) і підписок (addEnrollment, removeEnrollment), що забезпечує гнучкість у керуванні зв'язками.

Таким чином, Course є центральною сутністю, яка пов'язує між собою користувача-викладача, навчальний контент і студентів. Його структура

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ПЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		13

дозволяє масштабувати освітню платформу, забезпечуючи підтримку зв'язків між ключовими компонентами системи.

Клас Enrollment у C# описує зв'язок між студентом і навчальним курсом у системі онлайн-освіти. Він є своєрідним "мостом", який фіксує факт реєстрації студента на курс. Цей клас призначений для використання з Entity Framework Core у середовищі ASP.NET Core [4] і дозволяє ефективно працювати з реляційною базою даних.

Основою моделі є унікальний ідентифікатор Id, який генерується автоматично, як первинний ключ. Поля StudentId і CourseId є зовнішніми ключами, що вказують відповідно на студента (користувача з роллю Student) та на курс, на який він записався. Відповідні навігаційні властивості - Student та Course - дозволяють легко звертатися до пов'язаних об'єктів. Поле EnrolledAt зберігає дату й час реєстрації та визначається як дата у відповідному форматі.

Це - проста зв'язувальна сутність, тому конструктор класу не потребує додаткової ініціалізації колекцій. Усі властивості представлені у вигляді автосвойств, що дає змогу EF Core зручно зчитувати й записувати дані без додаткового коду.

З точки зору відносин, кожен запис Enrollment пов'язаний з одним студентом і одним курсом - тобто реалізовано зв'язки типу багато-до-одного. Але загалом через цю сутність реалізується багато-до-багатьох: один студент може бути записаним на багато курсів, і один курс може мати багато студентів. Такі відносини можна конфігурувати як через анотації, так і за допомогою Fluent API.

У класі не визначено подій життєвого циклу, таких як OnCreated чи OnDeleted, однак, за потреби, цю логіку можна реалізувати через перехоплювачі (interceptors) або відповідну логіку в DbContext.

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		14

Отже, Enrollment є ключовою частиною системи онлайн-курсів - він забезпечує реєстрацію студентів, дозволяє формувати структуру підписок і підтримує гнучке керування навчальним процесом.

1.2 Аналіз схожого програмного забезпечення

Основним аналогом мого додатка є платформа Udemy. Вона представлена у веб-версії з детальною документацією. Інтерфейс, який представлено на (рис 1.1), досить зручний для користувача, проте через велику кількість функцій і налаштувань може здатися складною для новачка - так само, як і будь-який інший навчальний веб-сервіс із продаж.

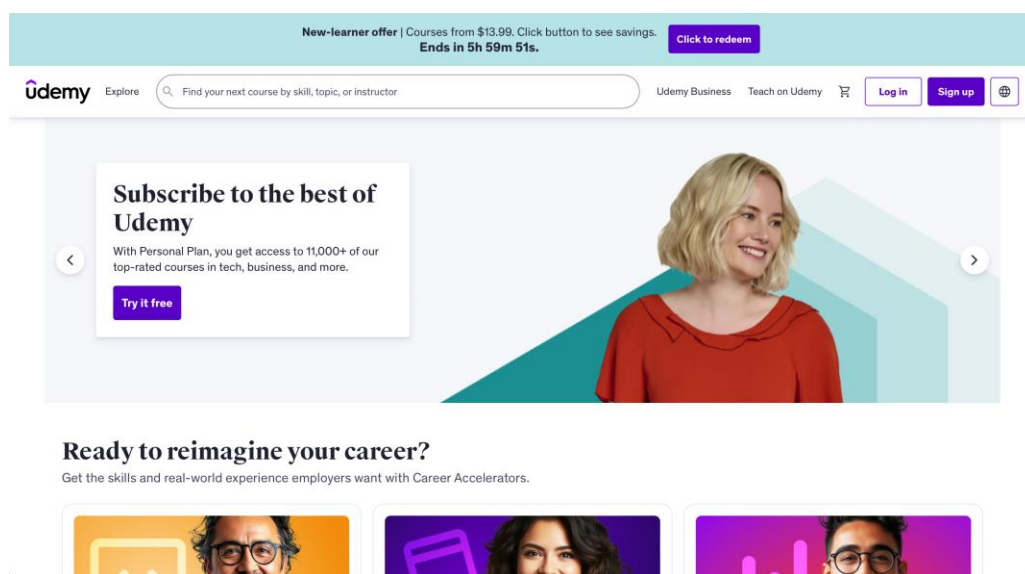


Рис.1.1 – Зовнішній вигляд Udemy

Далі розглянемо зовнішній вигляд фільтрів на сайті (рис. 1.2). Фільтри, представлені у вигляді випадаючих списків, зручні для користувачів, оскільки вони дозволяють швидко знаходити сертифікації відповідно до типу сертифікаційної програми або постачальника. Вибір постачальника (наприклад, Microsoft, Cisco, AWS) одразу обмежує список сертифікацій, що спрощує пошук.

Також можна побачити категорії, які сприяють детальнішому вибору: наприклад, по окремих сертифікаціях на основі певних тем, як-от "Microsoft Azure", що дозволяє користувачу без зайвих зусиль орієнтуватися на те, яка сертифікація їм підходить, залежно від їх цілей і кар'єрних планів.

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІПЗ	Арк.
		Ю.М. Єфремов				15
Змн.	Арк.	№ докум.	Підпис	Дата		

Цей підхід дозволяє швидко і зручно обирати сертифікації на основі специфічних вимог, що є важливим для користувачів, які шукають сертифікацію, яка відповідає їх потребам і рівню знань.

Explore by Goal	Popular Issuers	AZ-900: Microsoft Azure Fundamentals
Launch a new career >	Amazon Web Services (AWS) Certifications >	AZ-104: Microsoft Azure Administrator Associate
Certification preparation >	Microsoft Certifications >	AZ-204: Microsoft Azure Developer Associate
Development >	Cisco Certifications >	PL-300: Microsoft Power BI Data Analyst Associate
Business >	CompTIA Certifications >	AI-900: Microsoft Azure AI Fundamentals
Finance & Accounting >	Project Management Institute (PMI) Certifications >	PL-900: Microsoft Power Platform Fundamentals
IT & Software >	Google Cloud Certifications >	AZ-500: Microsoft Azure Security Engineer Associate
Office Productivity >	Six Sigma Certifications >	DP-900: Microsoft Azure Data Fundamentals
Personal Development >	Oracle Certifications >	DP-203: Microsoft Azure Data Engineer Associate
Design >	Popular Subjects	AZ-400: Microsoft DevOps Engineer Expert
Marketing >	Cloud Certification >	AZ-305: Microsoft Azure Solutions Architect Expert
Lifestyle >	Networking Certification >	
Photography & Video >	Cybersecurity Certification >	
Health & Fitness >	Project Management Certification >	
Music >	DevOps Certification >	
Teaching & Academics >		

Рис.1.2 – Зовнішній вигляд фільтрів сайту

Платформи онлайн-курсів мають низку важливих переваг, які роблять їх привабливими для широкої аудиторії користувачів. По-перше, це велика різноманітність навчального контенту: користувачі мають доступ до понад 210 000 курсів у різних категоріях - від програмування до особистісного розвитку, згідно з даними Upskillwise. Завдяки тому, що курси створюють незалежні інструктори, платформи часто охоплюють навіть нішеві теми.

Другою важливою перевагою є доступне ціноутворення. Під час регулярних розпродажів вартість багатьох курсів становить лише \$10–20, що значно вигідніше порівняно з традиційною офлайн-освітою (за даними uхsel.com). До того ж, наявність 30-денної гарантії повернення коштів зменшує фінансові ризики для студентів, а завдяки вебверсії навчатися можна з будь-якого місця.

Ще однією вагомою перевагою є безстроковий доступ до матеріалів. Після придбання курс залишається доступним у будь-який час, незалежно від дати покупки (Wall Street Survivor), а в мобільному застосунку передбачено можливість завантажувати відео для перегляду в офлайн-режимі.

Крім того, користувачі відзначають зручний інтерфейс і функціональні можливості плеєра. Зокрема, функції регулювання швидкості відтворення, додавання закладок, субтитри та інтуїтивна навігація по лекціях роблять процес навчання комфортнішим і ефективнішим, про що свідчать огляди на edukatico.org. До того ж, платформи характеризуються мінімальним рівнем технічних збоїв.

Останнім, але не менш важливим аспектом є постійні оновлення та підтримка. Інструктори мають можливість доповнювати свої курси новими матеріалами, адаптуючи їх до актуальних вимог, а також активно взаємодіють зі слухачами через форуми, відповідаючи на запитання та коментарі. Це створює динамічне навчальне середовище, що підтримує мотивацію та залученість користувачів.

Попри численні переваги, онлайн-платформи курсів мають і свої недоліки, які варто враховувати при виборі формату навчання. Однією з головних проблем є нерівномірна якість контенту. Оскільки курси можуть створювати всі охочі, без ретельної модерації, часто трапляються матеріали низької якості або поверхневого змісту (Wall Street Survivor). Крім того, відсутність офіційної акредитації призводить до того, що багато роботодавців не визнають сертифікати, наприклад, з платформи Udemy.

Ще одним суттєвим недоліком є агресивна політика знижок. Постійні «розпродажі» із зниженням цін на 90% створюють враження штучного завищення початкової вартості курсів (Reddit). Це не лише формує недовіру у студентів, а й шкодить самим авторам курсів, які через часті акції отримують значно менший прибуток (Great Work Life).

Також слід звернути увагу на домінування відеоформату. Більшість курсів подаються виключно у вигляді відеолекцій, що не завжди супроводжується достатньою кількістю практичних завдань або інтерактивних елементів (Reddit). Такий підхід часто призводить до явища,

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ПЗ	Арк.
		Ю.М. Єфремов				17
Змн.	Арк.	№ докум.	Підпис	Дата		

відомого як tutorial hell, коли учень споживає великий обсяг інформації, переглядаючи відео, але не закріплює знання на практиці.

Ще один недолік полягає у відсутності універсальної моделі підписки. Для більшості курсів користувачеві потрібно сплачувати окремо, що може бути незручно або фінансово обтяжливо, особливо якщо студент хоче пройти кілька курсів одночасно. Хоча існує Udemу Pro, вона охоплює лише обмежену кількість тем і не є загальнодоступною (classpert.com).

Нарешті, варто відзначити обмежену взаємодію між студентами. У більшості випадків платформи не мають вбудованих інструментів для організації групових обговорень або проведення живих вебінарів, обмежуючись лише простими чат-форумами. Це знижує рівень залученості та соціального аспекту навчання, який є важливим у традиційних або більш інтерактивних форматах. Немає вбудованих інструментів для групових дискусій чи живих вебінарів (крім чат-форумів).

Таким чином, Udemу - це економічно вигідне та зручне рішення для самостійного навчання з величезним вибором курсів, проте варто уважно оцінювати якість інструктора й бути готовим до поверхневого формату матеріалів.

Аналіз сайту Moodle

Переваги Moodle

Відкритий безкоштовний код Поширюється за ліцензією GNU GPL, тому нема ліцензійних витрат на використання та модифікацію. Кошти можна спрямувати на розвиток контенту й підтримку, а не на оплату софту.

Масштабна екосистема плагінів і налаштувань Понад 1 500 плагінів для розширення функціоналу (від відстеження відвідувань до медіа-інструментів). Можливість створювати власні теми, інтегрувати зовнішні сервіси (автентифікація, звіти) і формувати унікальне середовище навчання.

Жива спільнота та докладна документація Активні форуми, вебінари, Q&A-зустрічі й офіційні гіді допомагають швидко вирішувати проблеми.

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		18

Глобальні конференції MoodleMoot обмінюють практиками та напрацюваннями.

Повний набір освітніх інструментів Створення курсів, тести, завдання, форуми, спільні проєкти, електронні журнали оцінювання. Вбудована аналітика та звіти для відстеження успішності учнів і ефективності курсів.

Мобільність та офлайн-доступ Офіційний мобільний додаток Moodle Mobile дозволяє працювати з курсами навіть без інтернету: переглядати матеріали, здавати завдання, спілкуватися в дискусіях.

Відповідність стандартам e-learning Підтримка SCORM 1.2, IMS Common Cartridge, LTI-інтеграцій для імпорту/експорту контенту й підключення зовнішніх інструментів (BigBlueButton, Zoom).

Недоліки Moodle

Крута крива навчання; Висока гнучкість ускладнює початкове налаштування й освоєння; потрібні технічні знання для адміністрування. Адміністраторам і викладачам доводиться витратити час на навчання роботі з усіма функціями.

Застарілий і перевантажений інтерфейс; Стандартна тема виглядає громіздко: багато меню й налаштувань, що ускладнює навігацію без кастомізації. Користувачі зазначають, що без доопрацювань UI здається заплутаним.

Витрати на підтримку та хостинг Як self-hosted рішення потребує власного серверу, оновлень, бекапів, безпеки - це навантажує ІТ-відділ і бюджет . Сумісність плагінів і оновлення платформи часто потребують втручання технічних спеціалістів.

Обмежена аналітика «з коробки» Для глибшого трекінгу (час на сторінці, детальна активність) потрібні додаткові плагіни чи зовнішні ВІ-інструменти.

Проблеми продуктивності при навантаженні На великих інституціях чи в курсах із багатим медіаконтентом спостерігаються уповільнення без потужної

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		19

multi-server архітектури . Потрібні додаткові налаштування БД і серверів для масштабування.

Приховані витрати на кастомізацію. Хоча софт безкоштовний, реальні впровадження вимагають витрат на преміум-плагіни, розробку тем, хостинг та техпідтримку.

Інтерфейс Moodle є функціональним, але може виглядати перевантаженим без кастомізації, що ускладнює навігацію для нових користувачів (рис. 1.3).

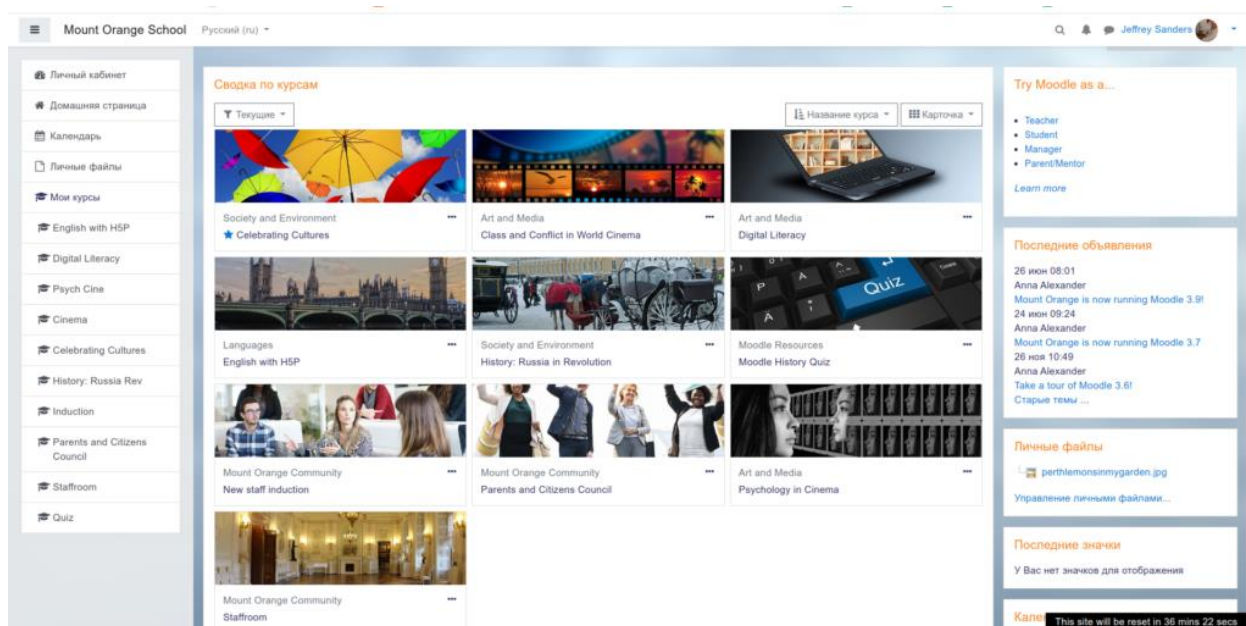


Рис.1.3 – Інтерфейс сайту Moodle

Moodle ідеально підходить для організацій із власними ІТ-ресурсами, що потребують гнучкої та масштабованої LMS. Проте школам чи компаніям без технічної підтримки варто оцінити витрати часу та зусиль на адміністрування або розглянути керовані (SaaS) альтернативи.

Порівняльний аналіз сайтів Moodle та Udemu щодо функціональності та можливостей хостингу.

Moodle - це відкрита платформа для управління навчанням, яку можна розгорнути на власних серверах або в хмарі, що дає повний контроль над налаштуваннями та безпекою. Вона підходить для університетів, шкіл і корпоративного навчання. Udemu, в свою чергу, є SaaS-платформою, де

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ПЗ	Арк.
		Ю.М. Єфремов				20
Змн.	Арк.	№ докум.	Підпис	Дата		

інструктори можуть публікувати курси без потреби в хостингу, орієнтуючись на індивідуальних користувачів.

Порівняння функцій Moodle та Udemu, наведене в таблиці, демонструє перевагу Moodle у гнучкості налаштувань і інтеграцій, тоді як Udemu забезпечує простоту використання і мінімальні технічні вимоги, що робить її ідеальним вибором для тих, хто шукає швидкий і зручний спосіб створення та публікації курсів (табл. 1.1).

Таблиця 1.1

Порівняння функцій Moodle та Udemu

Функція	Moodle	Udemu
Course Creation	Розширений редактор курсів, модулі, плагіни (1 500+ плагінів)	Проста публікація відео-лекцій, тести, Q&A
Кастомізація UI	Повний доступ до коду, теми, CSS, API	Обмежені можливості брендингу; є Trademark-правила
Мультимедіа та плагіни	Підтримка SCORM, LTI, інтеграція BigBlueButton, Zoom	Вбудовані плеєри, субтитри, офлайн-завантаження
Мобільний доступ	Moodle Mobile з офлайн-режимом	Android/iOS-додатки, offline-watch, Q&A, practice tests
Аналітика та звіти	Базова аналітика «з коробки», розширюється плагінами	Прогрес-трекінг, completion certificates, базові звіти
Автентифікація та ролі	Гнучкі ролі, права доступу, LDAP/SAML, OAuth	JWT-подібна сесійна авторизація, email-верифікація

Модель монетизації, існують різні моделі монетизації та витрат, пов'язані з використанням платформ для онлайн-освіти, і кожна з них має свої особливості. Наприклад, Moodle є безкоштовним програмним забезпеченням із відкритим кодом, що робить його доступним для широкого кола користувачів. Однак, незважаючи на відсутність плати за саму платформу, використання Moodle передбачає певні витрати. Зокрема, користувач повинен самостійно забезпечити хостинг, технічну підтримку, можливу інтеграцію преміум-плагінів, а також розробку та налаштування системи відповідно до потреб навчального процесу (Research.com).

Інший підхід демонструє платформа Udey, яка працює за моделлю разової оплати за кожен курс. Завдяки регулярним знижкам вартість більшості курсів становить лише 10–20 доларів США, що робить їх фінансово доступними для широкої аудиторії. До того ж, Udey пропонує 30-денну гарантію повернення коштів, що мінімізує ризики для користувача (Upskillwise). Для корпоративних клієнтів платформа має окрему пропозицію - Udey Business, яка надає доступ до великої бібліотеки курсів на умовах передплати, забезпечуючи зручність та гнучкість для командного навчання в бізнес-середовищі.

Moodle підходить для освітніх закладів і корпоративного навчання, де є ІТ-команда для адміністрування та потреба в інтеграції з SIS/LDAP. Вона надає гнучкість у налаштуванні та підтримує глибоку інтеграцію з іншими системами. Udey, в свою чергу, орієнтований на індивідуальних інструкторів і студентів, які шукають швидкий доступ до курсів без технічних бар'єрів, що робить платформу зручною для тих, хто хоче зосередитися на навчанні, а не на технічному обслуговуванні.

Отже, вибір Moodle доцільний, якщо необхідна глибока кастомізація, відповідність стандартам SCORM/LTI, а також якщо ви готові вкладатися в інфраструктуру та підтримку. У свою чергу, Udey є оптимальним варіантом

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		22

для швидкого виходу на ринок, мінімальних технічних зусиль та орієнтації на широкий ринок кінцевих користувачів.

1.3. Визначення архітектури програмного забезпечення

Для розробки веб-додатку для організації онлайн-курсів було обрано клієнт-серверну архітектуру, де клієнтська частина взаємодіє з сервером для обробки даних, збереження результатів і забезпечення взаємодії з користувачем. Цей підхід забезпечує масштабованість і модульність системи, дозволяючи легко додавати нові функціональні можливості без значних змін у існуючій структурі (рис. 1.4).

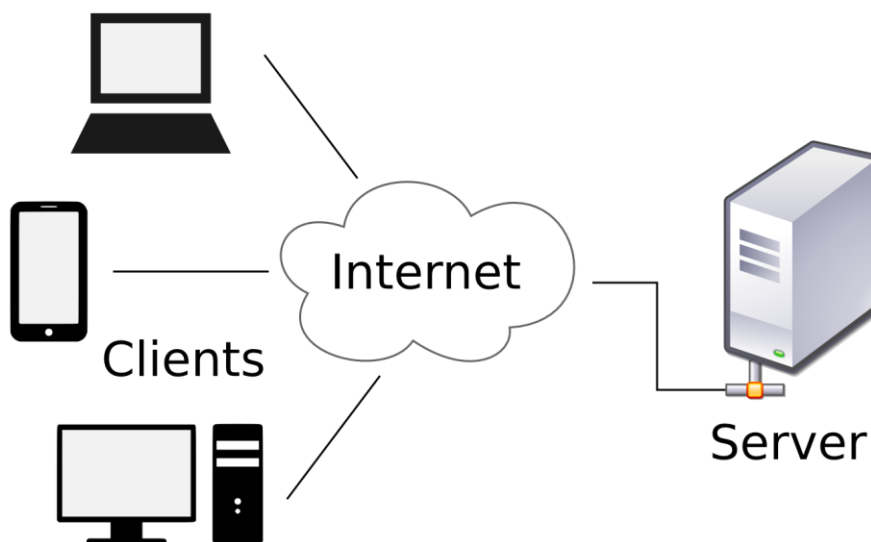


Рис. 1.4 - Схематичне зображення клієнтсько-серверної архітектури

Клієнтська частина реалізована за допомогою React.js [5], що дозволяє створювати швидкий і зручний інтерфейс для взаємодії з користувачем. Вона відповідає за показ курсу, перегляд тестів, надання оцінок і відображення результатів у реальному часі. Веб-додаток використовує архітектуру SPA (Single Page Application), що дозволяє уникнути перезавантаження сторінок і покращує користувацький досвід.

Серверна частина реалізована за допомогою фреймворку .NET, що забезпечує стабільну і масштабовану роботу з базою даних, обробку запитів через REST API та взаємодію з клієнтською частиною. База даних, збереження

даних користувачів, курсів і тестів, реалізована за допомогою PostgreSQL, що гарантує швидку обробку запитів і забезпечує високий рівень безпеки. Для автентифікації користувачів використовується JWT (JSON Web Token), що дозволяє ефективно й безпечно здійснювати контроль доступу до різних частин системи. Всі дані користувачів, курси та результати тестування зберігаються в захищеній базі даних із відповідними рівнями доступу, що забезпечує конфіденційність і цілісність інформації. Для контейнеризації компонентів застосовано Docker, що дозволяє створювати ізольоване середовище розробки та забезпечує стабільність роботи додатку в різних середовищах. Це дозволяє знизити ризики, пов'язані з конфігураціями серверів, і підвищити ефективність розробки.

Завдяки такій архітектурі, система має високу масштабованість, що дозволяє додавати нові модулі та можливості без значних змін у коді, що полегшує підтримку та розвиток проекту.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до програмного забезпечення

Для розробки веб-додатку онлайн-освіти було обрано інструменти, які відповідають вимогам щодо продуктивності, масштабованості та зручності в обслуговуванні. Вибір технологій був обумовлений необхідністю забезпечити ефективне управління курсами, тестами та збереженням результатів.

Для серверної частини обрано фреймворк .NET, оскільки він забезпечує високу продуктивність, стабільність і масштабованість, а також підтримує сучасні підходи до розробки, такі як REST API та ін'єкцію залежностей. Для роботи з базою даних використовується PostgreSQL, що є надійною реляційною системою управління базами даних, здатною ефективно виконувати складні запити та підтримувати великі обсяги даних.

Клієнтська частина побудована на React.js [7], що дозволяє створювати швидкі та зручні інтерфейси користувача. Завдяки архітектурі SPA (Single

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		24

Page Application), взаємодія з додатком відбувається без перезавантаження сторінок, що підвищує ефективність роботи користувачів.

Для контейнеризації було обрано Docker, що дозволяє створити ізольоване середовище для розробки та забезпечити стабільну роботу програми в різних середовищах.

Порівняння баз даних для використання у проєкті показує переваги PostgreSQL перед іншими системами управління базами даних, такими як MongoDB чи MariaDB, завдяки своїй здатності підтримувати складні запити, забезпечувати надійність та продуктивність при великих обсягах даних, що є важливими для системи онлайн-освіти (табл. 1.2).

Таблиця 1.2

Порівняння баз даних для використання в проєкті

Критерій	MongoDB	PostgreSQL	MariaDB
Тип СУБД	Документна (NoSQL), зберігає JSON-документи	Реляційна (SQL), підтримує JSONB для неструктурованих даних	Реляційна (SQL), підтримка JSON для зберігання
Масштабованість	Горизонтальна (для неструктурованих даних)	Горизонтальна і вертикальна, підтримка шардінгу	Вертикальна, з можливістю шардінгу
Продуктивність на складних запитах	Висока для неструктурованих даних	Висока для аналітичних запитів	Висока для CRUD-операцій
Робота з великими обсягами даних	Оптимізована для великих неструктурованих обсягів	Підтримує великі обсяги структурованих даних	Добре масштабується при помірних навантаженнях
Безпека даних	Додаткова конфігурація для безпеки	Розширене шифрування та аудит	Високий рівень безпеки, підтримка доступу
Резервне копіювання та відновлення	Потребує зовнішніх інструментів	Вбудоване резервне копіювання і реплікація	Підтримка бекапів та реплікації

Вибір PostgreSQL для нашого проєкту був обумовлений його здатністю ефективно обробляти складні запити, підтримувати великі обсяги даних та забезпечувати високу продуктивність для транзакційних операцій. Це робить його ідеальним вибором для платформи онлайн-освіти, де критично важливо зберігати дані студентів, курси та результати тестів.

Висновки до першого розділу

У першому розділі було розглянуто загальну концепцію, цільове призначення та функціональні особливості системи, що розробляється. Основна ідея полягає у створенні платформи, яка поєднує класичний функціонал LMS - керування курсами, ролями, тестуванням та відстеженням прогресу - з можливістю реалізації курсів у форматі маркетплейсу. Такий підхід дозволяє одночасно задовольнити потреби викладачів і адміністраторів у гнучкому інструменті для організації навчального процесу, а також надати студентам зрозумілий та інтуїтивно зручний інтерфейс для взаємодії з навчальним контентом.

З технічної точки зору, система базується на сучасній архітектурі. Фронтенд реалізовано у вигляді SPA-додатка з використанням React [8] або Next.js, а бекенд може бути реалізований на базі .NET Core або Node.js з використанням REST або GraphQL API. Для зберігання структурованих даних обрано PostgreSQL [12] як надійну реляційну базу даних. Зберігання мультимедійного контенту реалізується через AWS S3 з розповсюдженням через CloudFront, що забезпечує швидке завантаження і глобальну доступність. Інфраструктура підтримує сучасні практики розгортання за допомогою Docker і Kubernetes, що дає можливість легко масштабувати систему та оновлювати її без простоїв.

Особливу увагу в архітектурі приділено безпеці та масштабованості. Аутентифікація та авторизація користувачів реалізовані через JSON Web Tokens із підтримкою refresh-токенів, що забезпечує безстейтову архітектуру. Додатково застосовується email-верифікація для підтвердження облікових

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ПЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		26

записів, що підвищує рівень довіри та захисту. У результаті це дозволяє системі ефективно працювати як у відкритому доступі, так і в рамках корпоративних середовищ із підвищеними вимогами до контролю доступу.

У підсумку, платформа вдало поєднує переваги таких рішень, як Moodle - з його високим рівнем кастомізації, та Udemu - з простотою запуску й бізнес-орієнтованістю. При цьому вона зберігає гнучкість у налаштуваннях, підтримку SCORM/LTI, аналітичні інструменти та інші розширені можливості. Завдяки гібридній бізнес-моделі, яка передбачає підписки, разові платежі та корпоративні ліцензії, а також можливість запуску фріміум-доступу, система має потенціал стати конкурентоспроможною як серед open-source LMS, так і серед комерційних маркетплейсів.

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		27

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА WEB ДОДАТКУ ДЛЯ ОРГАНІЗАЦІЇ ОНЛАЙН КУРСІВ

2.1 Проектування бази даних

Проектування бази даних для веб-додатку з для організації онлайн курсів включає кілька етапів, які відображають структуру та взаємозв'язки сутностей (рис. 2.1).

Аналіз вимог: на першому етапі проводиться аналіз вимог до системи. Визначаються основні об'єкти, з якими буде взаємодіяти система, а також визначаються зв'язки між ними.

Створення сутностей та атрибутів: на цьому етапі кожному об'єкту, визначеному на першому етапі, призначаються атрибути. Наприклад, у сутності "Lessons" можуть бути атрибути "ID", "Name", "Tests" і т. д. Сутності розроблені в попередньому тексті слугують основою для створення таблиць бази даних.

Визначення первинних та зовнішніх ключів: Визначаються первинні ключі для кожної таблиці, що є унікальним ідентифікатором запису. Зовнішні ключі вказують на зв'язки між таблицями, утворюючи таким чином взаємозв'язану базу даних.

Нормалізація бази даних: на цьому етапі проводиться процес нормалізації для зменшення дублювання даних та підтримки цілісності бази. Це включає у себе розбиття таблиць на менші, але більш зв'язані частини.

Розробка запитів та зберігання процедур: Створюються SQL-запити для вибірки, оновлення, вставки та видалення даних. Зберігальні процедури використовуються для обробки складних операцій та забезпечення ефективності бази даних.

Оптимізація та індексація: проводиться оптимізація бази даних для покращення швидкодії та ефективності. Це включає у себе створення індексів на основі часто використовуваних запитів.

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		28

Тестування та валідація: проводиться тестування бази даних на відповідність вимогам та валідацію на наявність можливих помилок або конфліктів в даних.

Впровадження та моніторинг: на останньому етапі база даних впроваджується в роботу. Важливо ведення моніторингу та підтримка бази даних в актуальному стані, враховуючи зміни та розвиток сервісу з проведення онлайн курсів.

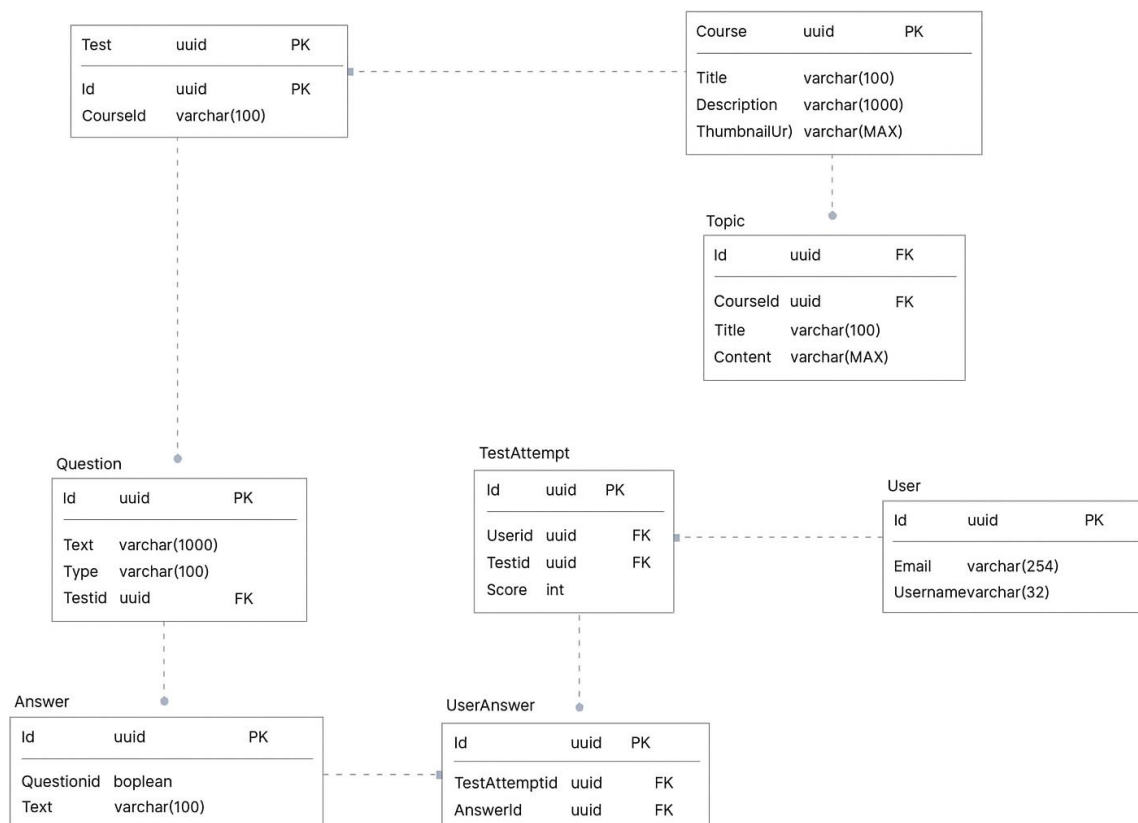


Рис.2.1 – Діаграма сутностей та зв'язків

В процесі проектування БД налічувала в собі 8 таблиць, включаючи проміжні таблиці потрібні для реалізації зв'язку багато до багатьох.

Система побудована на п'яти ключових сутностях – Course, User, Organization, Lesson і Enrollment – що забезпечує мінімально необхідну, але повністю функціональну базу даних для платформи онлайн-курсів. Відмова від окремої сутності «проміжний прогрес» і використання localStorage на стороні клієнта для тимчасового збереження поточного стану проходження

уроків спрощує модель даних і усуває потребу в додатковій серверній логіці для чернеток чи тимчасових сесій.

База даних дозволяє: виконувати CRUD-операції над курсами, користувачами, організаціями, уроками та записами про запис на курс; автоматично обчислювати відсоток завершення курсу на основі пройдених уроків і тестів; застосовувати довільний набір фільтрів до списків курсів та записів (за статусом, датою реєстрації, рейтингом тощо).

Для контролю доступу застосовано патерн RBAC: ролі зберігаються в полі ROLES таблиці User і перевіряються в кожному запиті за допомогою JWT-токена. Це забезпечує чітке розмежування прав:

У системі передбачено три основні ролі користувачів. Роль Admin надає повний доступ до всіх ресурсів платформи. Користувач з роллю Instructor має можливість створювати та редагувати курси й уроки, а також переглядати прогрес студентів. Роль Student дозволяє записуватись на курси, проходити уроки та переглядати власний прогрес. У моделі даних реалізовано кілька типів зв'язків між сутностями. Один-до-багатьох (one-to-many) зв'язок існує між Organization і Course, оскільки одна організація може створювати кілька курсів. Такий самий тип зв'язку встановлено між Course і Lesson - курс складається з кількох уроків. Водночас зв'язок між Enrollment і Course/User є багато-до-одного (many-to-one), адже кожен запис на курс належить конкретному курсу та конкретному користувачу. Для категоризації курсів також використано зв'язок багато-до-багатьох (many-to-many) між Course і Tag, реалізований через проміжну таблицю course_tags, що дозволяє здійснювати гнучкий пошук та фільтрацію.

2.2 Розробка архітектури додатку

В цьому проекті використано систему Docker для контейнеризації всіх мікросервісів. Docker надає можливість ізоляції додатків в окремі контейнери, що забезпечує простоту розгортання та масштабування, а також покращує безпеку і стабільність додатку. Файл docker-compose.yaml описує всі

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		30

налаштування кожного контейнера, включаючи сервіси, мережі та змінні середовища.

API - сервіс API побудований на основі C# [3] і працює у контейнері. Контейнер включений у приватну мережу, що забезпечує ізоляцію від інших мереж.

Frontend - сервіс Frontend базується на JS і використовується для обробки користувацького інтерфейсу. Він запускається за допомогою команди `npm start` і налаштований для автоматичного перезапуску. Для забезпечення гарячого перезавантаження, налаштовано змінні середовища, такі як `CHOKIDAR_USEPOLLING`.

За замовчуванням, Chokidar використовує механізми файлової системи для спостереження за змінами (такі як `inotify` на Linux, `fsevents` на macOS). Ці механізми забезпечують низьку затримку і ефективне використання ресурсів. Проте в деяких випадках, особливо на мережевих файлових системах або у середовищах з поганою підтримкою таких механізмів (наприклад, в Docker-контейнерах, що працюють на Windows), спостереження може не працювати належним чином. Саме тому рекомендовано використовувати `linux` дестрибутиви, для запуску цього проєкт.

Nginx використовується як зворотний проксі-сервер для розподілу запитів між фронтенд та бекенд сервісами. Він також забезпечує захищених з'єднань. Nginx налаштований на автоматичний перезапуск та використовує сертифікати SSL для шифрування даних.

Контейнер PostgreSQL забезпечує розвертання бази даних для додатку. Він налаштований для автоматичного перезапуску і зберігає дані у вказаному об'ємі. Параметри підключення до бази даних передаються через змінні середовища.

PgAdmin надає зручний веб-інтерфейс для адміністрування бази даних PostgreSQL. Він налаштований для роботи з контейнером PostgreSQL і доступний на порту 8080.

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		31

2.3 Розробка функціональних методів для роботи програми

У проєкті використовується шаблон CQRS (Command Query Responsibility Segregation) разом із бібліотекою MediatR, яка реалізує шаблон "Посередник" для обробки запитів та команд. Сутність RegisterUserCommandHandler - це обробник, який відповідає за логіку реєстрації нового користувача. Вона реалізує інтерфейс IRequestHandler, в якому обробляється команда RegisterUserCommand і повертається результат у вигляді DTO UserResponse.

Частина коду можна побачити в додатках.

Клас є sealed, тобто його не можна наслідувати, що гарантує стабільність поведінки та захищає від небажаних змін логіки в похідних класах.

У реалізації відповідного функціоналу використовуються основні залежності, які впроваджуються через конструктор. Зокрема, інтерфейс IUserRepository забезпечує доступ до інформації про користувачів у базі даних. Через IUnitOfWork реалізується шаблон одиниці роботи, який відповідає за атомарне збереження змін у системі. Сервіс IIdentityService використовується для створення нового облікового запису користувача у внутрішній системі аутентифікації. Додатково застосовуються мапери: один з них трансформує сутність користувача у формат DTO для формування відповіді, інший - перетворює користувача на внутрішню модель, необхідну для подальшої ідентифікації в системі.

У методі обробки запиту спочатку виконується перевірка, чи існує в системі користувач із вказаною електронною поштою. Якщо такий користувач уже зареєстрований, система повертає відповідну помилку, яка сигналізує про наявність облікового запису з такою адресою. Якщо ж користувача не існує, створюється новий об'єкт із унікальним ідентифікатором, ім'ям користувача та електронною поштою.

Далі створений користувач трансформується у внутрішню модель, якій призначається роль "користувач". Після цього новий об'єкт додається до

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		32

репозиторію, а також реєструється у внутрішній системі ідентифікації за допомогою сервісу аутентифікації. Завершальним кроком є збереження змін у базі даних. У разі успішного завершення операції система формує та повертає об'єкт відповіді, що містить основну інформацію про створеного користувача.

Крім базової реалізації, у RegisterUserCommandHandler також можуть бути реалізовані додаткові функції безпеки та перевірки валідності. Наприклад, до реєстраційного процесу може бути інтегрована валідація пароля, перевірка на дотримання політик складності, або навіть перевірка електронної пошти через зовнішній сервіс. Такі перевірки можуть виконуватись до створення користувача у системі аутентифікації, що підвищує надійність та безпеку процесу.

Крім того, система може підтримувати механізм надсилання підтвердження реєстрації на електронну пошту, що забезпечить додатковий рівень верифікації та запобіжить створенню фейкових акаунтів. Для цього в обробнику може бути викликаний відповідний EmailService, який після реєстрації користувача ініціює надсилання листа з підтвердженням.

Архітектурно важливо, що реєстрація реалізована як окрема команда в рамках шаблону CQRS, що дозволяє чітко розділяти операції на command (зміна стану системи) і query (запит даних). Це не лише підвищує зрозумілість коду, а й відкриває можливості для масштабування, наприклад, при переході до мікросервісної архітектури або впровадженні асинхронних обробників через черги повідомлень.

Використання MediatR забезпечує централізацію логіки обробки запитів і команд, що дозволяє уникнути жорсткої зв'язаності між компонентами. Це, у свою чергу, дає змогу легко тестувати окремі обробники в ізоляції, використовуючи підставні реалізації залежностей (mock або stub об'єкти).

Крім того, впровадження шаблону Unit of Work через IUnitOfWork дозволяє групувати декілька змін в рамках однієї транзакції, забезпечуючи атомарність і консистентність даних. Якщо на будь-якому етапі реєстрації

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		33

виникає помилка — наприклад, не вдалося створити обліковий запис в системі ідентифікації — всі зміни можна відкликати, гарантуючи цілісність даних у системі. Завдяки цьому система лишається легко підтримуваною та гнучкою до змін.

Таким чином, ця частина проєкту реалізує реєстрацію користувача, об'єднуючи доменну логіку, збереження у базу даних, створення облікового запису в системі ідентифікації та повернення відповідного результату. Такий підхід дозволяє чітко розділити відповідальності, спрощує тестування і розширення системи.

У межах дипломного проєкту функціональні методи реалізовано з використанням архітектурного підходу CQRS (Command Query Responsibility Segregation) у поєднанні з бібліотекою MediatR, яка впроваджує шаблон "Посередник" (Mediator Pattern). Такий підхід дозволяє розділити логіку читання та модифікації даних, зменшуючи зв'язаність між компонентами системи та спрощуючи масштабування і підтримку коду. Команди відповідають за зміну стану системи (створення, оновлення, видалення), а запити — лише за отримання даних без побічних ефектів. Прикладом реалізації функціонального методу є обробник RegisterUserCommandHandler, який відповідає за процес реєстрації нового користувача. Цей клас реалізує інтерфейс IRequestHandler<RegisterUserCommand, UserResponse>, що визначає метод Handle, у якому безпосередньо реалізована вся логіка обробки команди. Клас є позначеним як sealed, що означає заборону на наслідування, а отже — гарантію стабільної поведінки та захист від небажаних змін у спадкоємцях.

У методі Handle спочатку виконується перевірка на існування користувача з вказаною електронною поштою. Якщо такий запис уже є в базі, обробник повертає відповідну бізнес-помилку. У протилежному випадку створюється новий користувач за допомогою фабричного методу, якому передається GUID, ім'я та email. Після цього об'єкт мапиться у модель

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		34

ApplicationUser, якій присвоюється роль "користувач". Далі користувач додається до репозиторію, а також реєструється в системі аутентифікації через IdentityService. Зміни зберігаються за допомогою SaveChangesAsync, і у випадку успішного завершення операції повертається DTO з основною інформацією про створеного користувача. Крім базової логіки, обробник також може включати додаткову функціональність — наприклад, валідацію пароля, перевірку політик безпеки чи ініціацію надсилання електронного листа для підтвердження реєстрації. Для цього може бути інтегрований окремий сервіс — наприклад, IEmailService, який після успішної реєстрації надсилає лист із посиланням для верифікації. Такий підхід сприяє розділенню відповідальностей, що є одним із принципів чистої архітектури. Реєстрація користувача реалізована у вигляді окремої команди, а всі супутні сервіси чітко інкапсують свою логіку. Це забезпечує високу тестованість обробника: кожен компонент можна протестувати окремо, використовуючи підставні реалізації залежностей (моки чи стаби). Використання IUnitOfWork дозволяє об'єднати всі дії в межах однієї транзакції. У разі виникнення помилки (наприклад, якщо не вдалося створити обліковий запис у системі ідентифікації), усі попередні дії можна відкликати, що гарантує консистентність і надійність системи. Отже, реалізація функціональних методів на базі CQRS, MediatR і принципів чистої архітектури дозволяє створити гнучку, розширювану та безпечну систему, яка готова до подальшого масштабування — зокрема, у напрямку мікросервісної архітектури, асинхронних обробників або інтеграції з чергами повідомлень (наприклад, RabbitMQ чи Azure Service Bus).

2.4 Збереження файлів

За для збереження файлів, було використано Amazon Simple Storage Service (S3). Це сторонній third-party service, який надає послуги збереження даних. Аналогами цього сервісу є Google Cloud та Firebase. Але на S3 можна

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		35

розвертати повноцінні проекти, а налаштування збереження даних проходить досить просто.

Спочатку необхідно створити IAM користувача, це умовний користувач, якому надаються права доступу до кошику де зберігається наші дані. Цей користувач працює автоматично за певними правилами та дозволами. Аналогію можна провести як користувач на рівні бази даних, йому так само можна надавати привілеї, права та можливості змінювати, видаляти чи додавати дані.

Після чого необхідно створити кошик, та надати права на редагування цього кошика, нашому користувачеві IAM. В цьому проекті було надано повний доступ до кошика, цьому користувачу. Самі правила даня даних до покупця, вже були реалізовані в самописному сервісі AWSS3Service.

При конфігурування кошику, ми можемо вказати місце розташування серверу, на якому будуть зберігатися ці зображення.

Amazon Simple Storage Service (S3) є одним з найпопулярніших сервісів для зберігання файлів в хмарі. Він забезпечує високу надійність, масштабованість та безпеку зберігання даних. Зберігання фотографій за допомогою S3 має кілька переваг: Масштабованість: S3 дозволяє зберігати будь-яку кількість даних, що дуже важливо для додатків, які працюють з великою кількістю зображень. Безпека: S3 підтримує різні механізми шифрування, як на стороні клієнта, так і на стороні сервера, що забезпечує високий рівень безпеки даних. Доступність: S3 гарантує високу доступність даних, що робить його ідеальним для зберігання важливих файлів, таких як фотографії.

Щоб зберегти файл на S3 за допомогою C#, необхідно використовувати AWS SDK для C# [29] (лістинг 2.1).

```
var request = new PutObjectRequest { BucketName = _s3Settings.BucketName, Key = fileName,
InputStream = file, ContentType = contentType, Metadata = { ["file-name"] = fileName,}};
var response = await _s3Client.PutObjectAsync(request);
```

Лістинг 2.1 – Присвоєння змінної для бакета

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		36

Окрім базових функцій завантаження та зчитування файлів, у проєкті також реалізовано отримання публічного посилання на збережене зображення, яке дозволяє користувачу переглядати файл напряму з браузера. Для цього використовується генерація signed URL з обмеженим часом дії, що забезпечує контрольований доступ до файлів без потреби робити кошик повністю публічним. Також було враховано можливість очищення кошика від застарілих файлів, шляхом встановлення правил життєвого циклу (Lifecycle Rules), які автоматично видаляють або архівують файли після певного періоду зберігання. Це дозволяє оптимізувати витрати на інфраструктуру та уникнути накопичення неактуального контенту. З технічного боку, всі взаємодії з Amazon S3 інкапсульовані у спеціалізований сервіс AWSS3Service, який приховує внутрішню реалізацію API та забезпечує централізований доступ до операцій над файлами. Такий підхід полегшує тестування та повторне використання логіки збереження в інших частинах проєкту.

Для взаємодії з Amazon S3 у проєкті було використано AWS SDK for .NET, який надає зручний інтерфейс для здійснення всіх основних операцій: завантаження файлів, отримання, видалення та керування об'єктами в кошику. SDK повністю інтегрується з платформою .NET та дозволяє працювати з асинхронними методами, що є важливою перевагою для побудови масштабованих веб-додатків. Налаштування SDK передбачає створення клієнта AmazonS3Client, якому передаються облікові дані IAM-користувача та регіон розміщення S3-бакету. Операція завантаження файлу реалізується за допомогою класу PutObjectRequest. У цьому запиті вказується назва кошика (BucketName), унікальний ключ об'єкта (Key), сам файл у вигляді потоку (InputStream), а також його MIME-тип (ContentType). Додатково можна передати метадані — наприклад, оригінальну назву файлу. Метод PutObjectAsync викликається асинхронно та повертає відповідь із результатом операції. Це забезпечує неблокуючу роботу сервера навіть при завантаженні великих файлів. Для отримання файлів із S3 використовується метод

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ПЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		37

GetObjectAsync, який дозволяє або завантажити файл на сервер, або напряму передати потік клієнту через API. У деяких випадках більш зручним є створення signed URL — спеціального посилання з обмеженим строком дії, яке дозволяє безпечно надати користувачу прямий доступ до зображення або документа без автентифікації. Такий підхід особливо ефективний для збереження пропускну здатності сервера та зменшення часу відповіді. Крім базових операцій, SDK також підтримує видалення файлів (DeleteObjectAsync), перевірку наявності (DoesS3ObjectExistAsync через AWS Extensions), а також налаштування життєвого циклу об'єктів безпосередньо з коду. Завдяки використанню AWS SDK, проєкт отримав гнучкий і розширюваний механізм роботи з файлами, який легко адаптується під майбутні потреби — наприклад, додавання підтримки мініатюр зображень або логування всіх операцій із файлами для аудиту.

Висновки до другого розділу

У ході реалізації функціональності реєстрації користувача та збереження файлів у хмарному середовищі було застосовано сучасні підходи до проєктування програмного забезпечення, зокрема принципи чистої архітектури (Clean Architecture), шаблони DDD (Domain-Driven Design), а також інтеграцію з зовнішніми хмарними сервісами для забезпечення масштабованості й надійності.

Метод Handle команди RegisterUserCommand реалізує бізнес-логіку реєстрації, перевіряючи наявність користувача з вказаною електронною адресою, створюючи доменну модель User і трансформуючи її в модель ApplicationUser для збереження через репозиторій і шаблон одиниці роботи (Unit of Work). Такий підхід забезпечує модульність, повторне використання компонентів і спрощує тестування.

Окремо варто зазначити застосування двох Mapper-компонентів для перетворення між доменною моделлю користувача і DTO-відповіддю, а також для інтеграції з внутрішньою моделлю автентифікації. Це дозволяє уникнути

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		38

прямої залежності між шарами і дотримуватись принципу єдиного обов'язку кожного компонента. Завдяки застосуванню інтерфейсів (IUserRepository, IUnitOfWork, IIdentityService) досягається інверсія залежностей, що робить систему більш гнучкою до змін і полегшує модульне тестування.

Окрім логіки роботи з користувачами, важливою складовою проєкту є збереження файлів за допомогою стороннього хмарного сервісу Amazon Simple Storage Service (S3). Для інтеграції з цим сервісом було реалізовано окремий сервіс AWSS3Service, який абстрагує логіку взаємодії з хмарним сховищем. Перед цим було виконано налаштування облікового запису IAM (Identity and Access Management), створено кошик (bucket) і надано необхідні дозволи для взаємодії з ним. Такий підхід дозволяє централізовано управляти файлами, що особливо актуально у випадках зберігання великої кількості мультимедійних об'єктів, зокрема фотографій користувачів.

Amazon S3 надає низку переваг, серед яких масштабованість, що забезпечує можливість зберігання необмеженої кількості даних, безпеку, підтримуючи шифрування на рівні сервера та клієнта, а також високу доступність, що включає стійкість до збоїв і постійний доступ до даних. Застосування AWS SDK для C# дозволило налаштувати завантаження файлів безпосередньо з бекенду, що спрощує інтеграцію й покращує досвід кінцевого користувача.

У підсумку, можна стверджувати, що реалізована функціональність реєстрації користувача разом із системою зберігання файлів відповідає сучасним стандартам розробки програмного забезпечення. Вона забезпечує чітке розділення відповідальностей, високу розширюваність, підтримуваність та безпечність. Це свідчить про готовність розробленого рішення до реального використання у промислових умовах, де вимоги до якості, стабільності та безпеки є особливо критичними.

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		39

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК ВИКОРИСТАННЯ ДОДАТКУ

3.1 Опис роботи з програмним додатком (Опис інтерфейсу)

На зображеннях представлено інтерфейси сторінок реєстрації та авторизації веб-платформи «Courses Platform». Нижче наведено докладний опис та аналіз реалізації кожної з цих сторінок у контексті розробки повноцінної системи управління курсами, яка є частиною дипломного проєкту.

Після переходу на сайт користувач потрапляє на сторінку авторизації, де може увійти до системи за допомогою email та пароля. Для зручності реалізовано функцію відновлення пароля та реєстрації нових користувачів. Сторінка авторизації має простий і зручний інтерфейс, що підтримує єдиний стиль дизайну платформи. Інтерфейс сторінки авторизації виглядає наступним чином (рис. 3.1).

Рис.3.1 – Сторінка авторизації

На зображенні представлено сторінку входу до системи, де користувач вводить свою електронну пошту та пароль. Також є кнопка для відновлення пароля і посилання для переходу до сторінки реєстрації нового користувача. Після успішної авторизації користувач потрапляє до свого особистого кабінету.

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ПЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		40

Для нових користувачів передбачено сторінку реєстрації, де вони можуть створити обліковий запис. Інтерфейс реєстрації простий і інтуїтивно зрозумілий, з полями для введення імені, email, пароля та підтвердження пароля. Сторінка реєстрації виглядає наступним чином (рис. 3.2).

Рис.3.2 – Сторінка реєстрації

На зображенні показано сторінку створення нового облікового запису - один із ключових етапів забезпечення доступу до функціоналу вебплатформи. Вона має простий і інтуїтивно зрозумілий інтерфейс із такими полями для вводу: Ім'я користувача - текстове поле, яке дозволяє користувачу ввести унікальний ідентифікатор. Електронна пошта - поле для введення email-адреси, яка використовується для автентифікації та подальшої комунікації. Пароль та Підтвердження пароля - два поля з прихованим введенням, що дозволяють користувачу встановити надійний пароль і уникнути помилок завдяки повторному введенню.

Після заповнення форми натискання на кнопку Створити аккаунт ініціює валідацію введених даних, а також подальше створення облікового запису в базі даних за умови відповідності вимогам.

Важливою особливістю є посилання Увійти, що дозволяє швидко перейти на сторінку авторизації, якщо користувач уже має обліковий запис.

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		41

Обидві форми розміщені по центру сторінки та оформлені у вигляді білого блоку з округлими краями, що візуально виокремлює їх з фону й створює враження чистоти та простоти інтерфейсу.

Верхнє меню містить логотип платформи «Курси» і навігаційні посилання Головна, Курси та Увійти, що сприяє простій навігації по сайту.

Форми створено з урахуванням принципів UX/UI-дизайну, що робить їх зручними для користувачів з різними технічними навичками.

Сторінки логіну та реєстрації, представлені на зображеннях, є технічно і візуально реалізованими відповідно до сучасних стандартів веброзробки. Вони відіграють критичну роль у забезпеченні безпеки та керування доступом до платформи курсів. Інтуїтивно зрозумілий інтерфейс, адаптивна верстка, коректна логіка переходів і навігації сприяють покращенню користувацького досвіду та є прикладом ефективної реалізації механізму автентифікації в рамках дипломного проекту.

Перед вами сторінка, яка демонструє доступні курси на платформі онлайн-навчання. Вона виконує роль каталогу, де користувачі можуть переглядати всі курси, що доступні для проходження, а також отримати коротку інформацію про кожен з них. Кожна картка курсу містить назву, опис та дату початку навчання, що дозволяє легко зорієнтуватися в пропозиціях. Система також підтримує пагінацію, що дозволяє зручно переглядати великі обсяги інформації, а навігація між основними розділами платформи сприяє зручному користуванню. Завдяки такому інтерфейсу, користувачі мають можливість швидко знайти потрібний курс і дізнатися про нього більше, що значно покращує досвід використання платформи. Сторінка виглядає наступним чином (рис. 3.3).

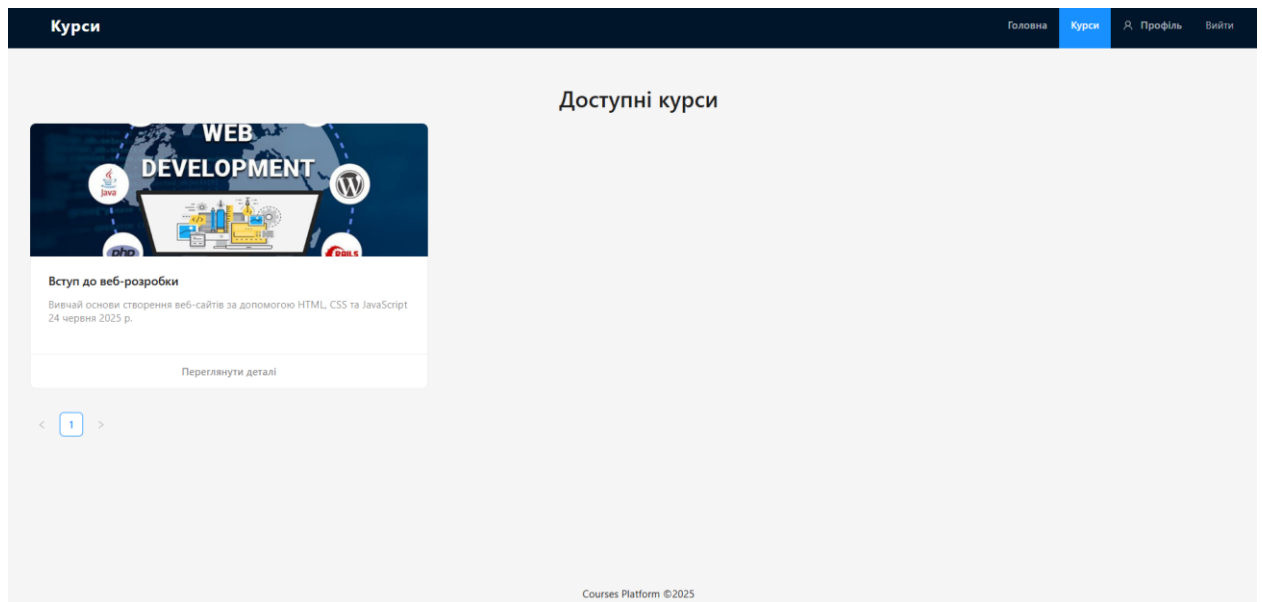


Рис.3.3 – Сторінка курсів

Сторінка "Доступні курси" виконує функцію каталогу доступних курсів на платформі онлайн-навчання, дозволяючи користувачеві ознайомитися з пропозицією освітніх програм. У верхній частині сторінки розміщено заголовок "Доступні курси", який чітко позначає зміст сторінки - перелік курсів, що відкриті для проходження. Основною одиницею інтерфейсу є картка курсу, яка подає стислу, але інформативну інформацію про кожен навчальний модуль. Зокрема, на прикладі курсу "Вступ до веб-розробки", картка містить його назву, короткий опис, що пояснює, що курс охоплює основи створення вебсайтів з використанням HTML, CSS і JavaScript, а також дату початку навчання - 24 травня 2025 року. Для переходу до більш детальної інформації про курс передбачено кнопку Переглянути деталі. Якщо кількість курсів перевищує межі однієї сторінки, система підтримує пагінацію, яка реалізується через навігаційні кнопки (у поточній версії доступна лише одна сторінка). Додатково реалізована загальна навігація, яка дозволяє користувачу швидко переміщатися між основними розділами платформи, такими як Головна, Курси, Профіль та Вийти.

Перед вами сторінка, яка детально описує курс "Вступ до веб-розробки" на платформі онлайн-навчання. Ця сторінка є важливим елементом взаємодії користувача з навчальним контентом, оскільки вона містить всю необхідну

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		43

інформацію про курс, його зміст і доступні можливості для старту навчання. Інтерфейс дозволяє користувачеві легко зрозуміти, що саме він буде вивчати, а також дає змогу одразу розпочати курс або ознайомитися з його структурою. Сторінка виглядає наступним чином (рис. 3.4).

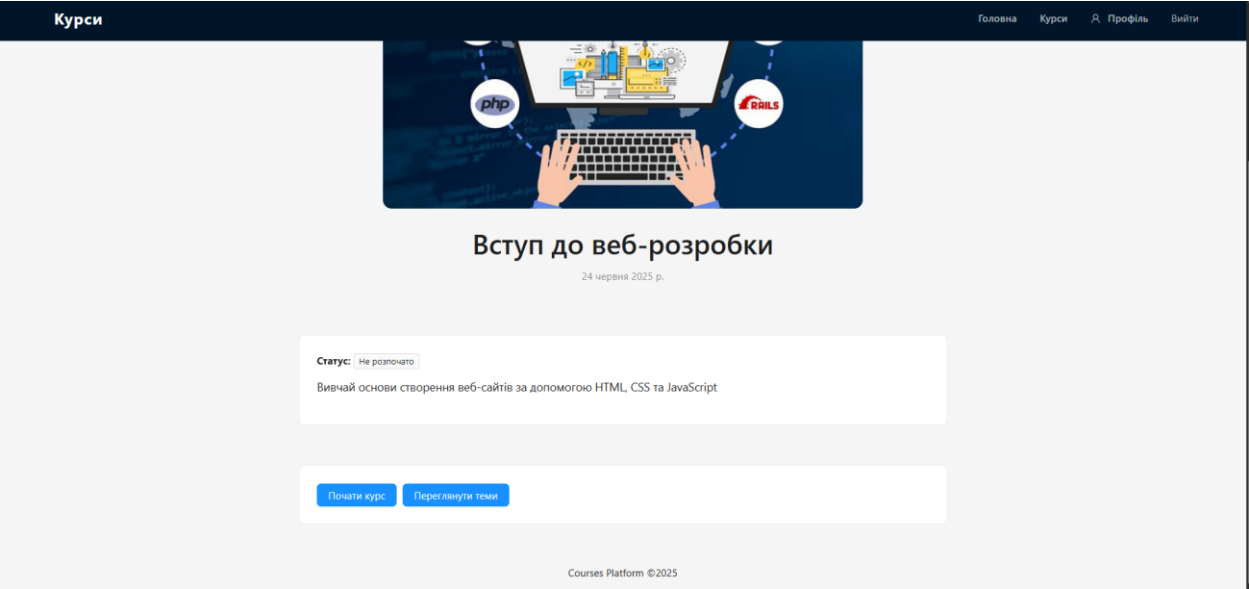


Рис.3.4 – Сторінка деталей курсу

Сторінка деталей курсу "Вступ до веб-розробки" є ключовим елементом платформи онлайн-навчання, що реалізує базову логіку взаємодії користувача з освітнім контентом. Вона містить банер із тематичним зображенням, пов'язаним із веброзробкою, що візуально підсилює ідентичність курсу. Основний заголовок сторінки включає назву курсу та дату його початку - 25 травня 2025 року, а також інформує користувача про поточний статус навчання – не розпочато, тобто курс ще не розпочато. Під заголовком розміщено стислий опис змісту курсу, який охоплює базові технології фронтенду: HTML, CSS і JavaScript. На сторінці також передбачено дві основні кнопки: Почати курс, що дозволяє користувачеві одразу розпочати навчання, та Переглянути теми, яка відкриває структуру й перелік тем курсу. Таким чином, ця сторінка відіграє роль інтерактивного вузла, через який користувач може не лише ознайомитись із курсом, а й ініціювати його проходження або попередньо вивчити навчальний план.

3.2 Роутинг та маршрутизація

Перед вами таблиця, яка демонструє роутинг та маршрутизацію для API запитів, що стосуються статей на платформі. Вона описує, як здійснюється обробка запитів для отримання списку статей, перегляду деталей статей, створення нових статей, редагування і видалення. Ця таблиця важлива для налаштування взаємодії між фронтендом і бекендом при роботі з контентом на платформі (табл. 3.1).

Таблиця 3.1

api/articles

HTTP	Route	React Route	Дія
GET	/api/articles	/articles	Перелік статей
GET	/api/articles/:articleId	/articles/:id	Деталі статті
POST	/api/articles	-	Створення статті (з форми)
PUT	/api/articles	-	Редагування статті
DELETE	/api/articles/:articleId	-	Видалення статті

Перед вами таблиця, що містить роутинг та маршрутизацію для запитів, що стосуються користувачів на платформі. Вона описує маршрути для отримання списку користувачів, перегляду детальної інформації про користувачів, підтвердження пошти, а також для реєстрації, входу, оновлення профілю та видалення профілю. Ці маршрути важливі для забезпечення основних операцій з користувачами на платформі, таких як реєстрація нових користувачів, автентифікація, а також управління профілями користувачів. Крім того, реалізовано підтвердження електронної пошти, що є необхідним для активації акаунтів користувачів. Всі ці функціональні можливості дають

користувачам можливість ефективно взаємодіяти з платформою, зберігаючи при цьому належний рівень безпеки та доступу. (табл. 3.2).

Таблиця 3.2

api/users

HTTP	Route	React Route	Дія
GET	/api/users	/articles	Перелік користувачів
GET	/api/users/:userId	/articles/:id	Деталі користувача
GET	/api/users/confirm-email	/confirm-email?userId=...	Підтвердження пошти
POST	/api/users/register	/register	Реєстрація
POST	/api/users/login	/login	Вхід
PUT	/api/users	-	Оновлення профілю
DELETE	/api/users/:userId	-	Видалення профілю

У проєкті роутинг на бекенді реалізовано відповідно до принципів REST-архітектури та структуровано за ресурсами, зокрема articles та users. Для кожного ресурсу визначено набір маршрутів, що відповідають основним HTTP-методам: GET (отримання даних), POST (створення), PUT (оновлення) та DELETE (видалення). Така структура забезпечує чітку організацію API, зрозумілу логіку обробки запитів та спрощує підтримку.

На фронтенді за допомогою бібліотеки React Router відтворено аналогічну структуру маршрутів. Компоненти інтерфейсу логічно розділено на окремі сторінки, відповідно до REST-ресурсів. Наприклад, маршрути /articles, /articles/:id, /users/:id/edit відповідають сторінкам перегляду, редагування або створення сутностей.

Уніфікована структура маршрутів на бекенді та фронтенді сприяє масштабованості, полегшує розробку нових функцій і підвищує зрозумілість архітектури застосунку.

3.3 Робота з авторизацією (JWT)

При логіні (/api/users/login) отримується token. Він зберігається у localStorage або sessionStorage. Для захищених маршрутів - дадаються хедери Authorization: Bearer {token}. axios.defaults.headers.common['Authorization'] = `Bearer \${token}`;

У проєкті для автентифікації та авторизації користувачів використовується механізм JWT (JSON Web Token). Після успішного входу користувача сервер формує токен, який містить закодовану інформацію про користувача, зокрема його ідентифікатор та роль. Цей токен підписується за допомогою секретного ключа, що гарантує цілісність та достовірність даних.

Згенерований JWT передається клієнту, який зберігає його, як правило, у локальному сховищі браузера. При кожному подальшому запиті до захищених маршрутів клієнт надсилає токен у заголовок Authorization. Таким чином, сервер може ідентифікувати користувача без необхідності зберігати сесію, що забезпечує безстейтовість взаємодії.

На сервері токен проходить перевірку підпису та строку дії. У разі успішної валідації сервер отримує доступ до вкладеної інформації й надає або забороняє доступ до ресурсів згідно з роллю користувача. У випадку недійсного або відсутнього токена запит відхиляється з відповідним статусом помилки.

Застосування JWT забезпечує надійний та масштабований підхід до керування доступом у клієнт-серверній архітектурі. Цей механізм добре підходить для SPA-додатків, де важлива швидкість взаємодії, відсутність збереження стану на сервері та гнучке управління правами доступу.

Для генерації JWT на сервері використовується спеціальний компонент, який формує токен на основі об'єкта користувача. Зазвичай цей процес

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		47

включає створення так званого payload — це набір інформації, який потрібно передати в токені, зокрема `userId`, `email`, `role`, дата створення і строк дії. Цей об'єкт кодується у форматі JSON, після чого підписується криптографічно за допомогою секретного ключа або ключової пари (у разі використання алгоритмів типу RS256). Підпис гарантує, що дані не були змінені сторонньою особою після створення.

Метод формування токена зазвичай реалізується через бібліотеки типу `System.IdentityModel.Tokens.Jwt` у .NET. У ньому створюється об'єкт `JwtSecurityToken`, в який передаються `claims` (вимоги), термін дії токена та параметри підпису. Потім цей токен серіалізується у строку (рядок), яка і передається клієнту як результат авторизації. У структурі токена можна умовно виділити три частини: `header` (алгоритм і тип токена), `payload` (дані) та `signature` (підпис).

На стороні клієнта, після отримання токена (під час логіну), він зберігається або у `localStorage`, або в `sessionStorage`, залежно від логіки застосунку. Збереження у `localStorage` дозволяє зберегти токен навіть після перезавантаження сторінки, однак варто враховувати потенційні ризики XSS-атак. Для підвищення безпеки, особливо в чутливих додатках, рекомендується зберігати токени у `httpOnly` куках, які недоступні з JavaScript. У поточному проєкті використовується `localStorage`.

При зверненні до захищених маршрутів клієнт автоматично додає токен у заголовок `Authorization` кожного запиту. Це реалізується через глобальну конфігурацію HTTP-клієнта (наприклад, через `axios.defaults.headers.common['Authorization'] = Bearer {token}`), або локально — безпосередньо в кожному запиті. Таким чином, сервер отримує кожен запит вже з потрібною інформацією для авторизації користувача.

На сервері реалізовано механізм `middleware`-автентифікації, який перехоплює вхідні HTTP-запити, перевіряє наявність і коректність токена. Якщо токен є, підпис дійсний, і термін дії не минув — дані користувача

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		48

втягаються з ClaimsPrincipal і стають доступними в контролерах. Далі система може приймати рішення про доступ користувача до певних ресурсів на основі ролі, яка була вказана при формуванні токена. Якщо перевірка не проходить, користувачу повертається статус 401 Unauthorized або 403 Forbidden.

Завдяки використанню JWT, архітектура системи лишається безстейтовою (stateless), оскільки сервер не зберігає інформації про сесію між запитами. Це дозволяє масштабувати систему, використовуючи балансування навантаження або мікросервісну структуру, де токен може бути валідований у будь-якому сервісі без централізованого сховища сесій. Таким чином, проєкт отримує не лише безпечну, а й гнучку модель керування автентифікацією та авторизацією користувачів.

3.4 User Stories та User Flows

У цьому підрозділі розглядаються основні сценарії взаємодії користувача з платформою, а саме реєстрація, управління статтями та профілями користувачів. Це дозволяє краще зрозуміти, як користувачі взаємодіють із системою та якими функціями вони користуються для досягнення своїх цілей. Описані потоки роботи надають чітке уявлення про послідовність дій користувачів у різних сценаріях (рис. 3.5).

User Stories описують основні функції, які користувачі хочуть мати на платформі для ефективної взаємодії з контентом. Як користувач, я хочу переглядати список статей, щоб ознайомитися з доступними матеріалами. Я також хочу мати можливість натискати на конкретну статтю, щоб переглянути її повний зміст. Як автор, я хочу мати можливість створювати нову статтю через форму, щоб поділитися інформацією, а також редагувати та видаляти вже опубліковану статтю для оновлення або видалення старої інформації. Як відвідувач, я хочу зареєструватися, щоб створити свій профіль, і потім увійти в систему, щоб отримати доступ до приватних функцій. Після реєстрації я хочу підтвердити свою електронну адресу для завершення реєстрації. Я також хочу

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		49

переглядати свій профіль або профілі інших користувачів, а як адміністратор, хочу бачити список користувачів для ефективного управління платформою. Я хочу мати можливість оновити свій профіль або видалити його, якщо я вирішу вийти з платформи.

User Flow описує конкретну послідовність дій, які виконує користувач для виконання тих чи інших операцій на платформі. При реєстрації користувач заходить на сторінку /register, вводить свої дані, після чого отримує лист для підтвердження на свою електронну пошту. Після підтвердження переходить на сторінку /login, де вводить email і пароль для входу. Після успішного входу користувача редіректить на сторінку /articles. Для управління статтями користувач переходить на /articles, обирає статтю та переглядає її деталі. Якщо користувач хоче створити нову статтю, він натискає на кнопку для створення статті, що відкриває форму для введення контенту. Для редагування статті він натискає кнопку "Редагувати", а для видалення статті - кнопку "Видалити". Для управління профілем користувач може переглядати профілі інших користувачів на /api/users, натискати на профіль для перегляду деталей, редагувати свій профіль або видаляти його.

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ПЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		50

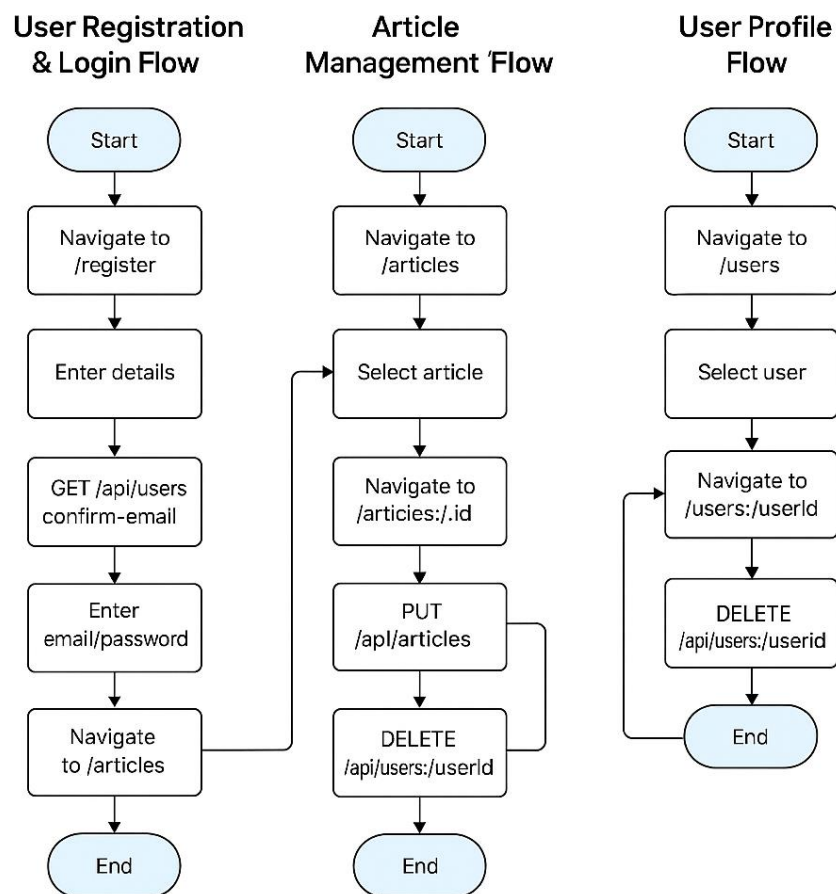


Рис.3.5 – User flow

На зображенні представлено графічне зображення потоків користувача, що демонструють кроки, які здійснюють користувачі при реєстрації, управлінні статтями та профілями. Це дозволяє наочно побачити, як реалізовані різні функціональні можливості платформи.

3.5 Заходи безпеки бази даних

Під час роботи з базою даних PostgreSQL [15] через pgAdmin було реалізовано кілька заходів безпеки, які сприяють захисту даних та зниженню ризику несанкціонованого доступу. Серед них: хешування паролів, використання UUID як унікального ідентифікатора, а також налаштування ролей та привілеїв для кожного користувача бази даних.

Використання Docker та pgAdmin

Docker забезпечує ефективне середовище для розробки, дозволяючи швидко розгортати контейнери PostgreSQL та pgAdmin, ізоляцію сервісів, та легке управління залежностями.

PostgreSQL вирізняється високою надійністю та масштабованістю, оскільки підтримує ACID-транзакції, реплікацію та здатен ефективно працювати з великими обсягами даних. Гнучка система ролей дозволяє точно налаштовувати права доступу на рівні користувачів, таблиць і навіть цілих схем. Крім того, PostgreSQL [22] підтримує зберігання структурованих даних у форматі JSONB, роботу з розширеннями та використання тригерів, що робить його ідеальним для складних сценаріїв обробки даних.

PgAdmin, у свою чергу, надає зручний веб-інтерфейс для керування PostgreSQL-базами даних. Він дозволяє легко управляти ролями, схемами та іншими аспектами бази через інтуїтивно зрозуміле графічне середовище. Крім того, pgAdmin підтримує виконання SQL-запитів, створення резервних копій і базову аналітику - все це доступне буквально в кілька кліків.

Всі параметри описані в Docker [36] Compose конфігурації для PostgreSQL [11] та pgAdmin.

Опис параметрів:

Контейнер postgres запускає сервер бази даних PostgreSQL і використовує змінні оточення для створення бази даних, користувача та пароля. Поруч із ним працює контейнер pgadmin, у якому запущено pgAdmin 4 - інтерфейс для керування базою, доступний за адресою <http://localhost:5050>.

Для збереження даних бази між перезапусками використовується том (volume). Обидва контейнери підключені до спільної мережі, у межах якої їм вручну призначені статичні IP-адреси для зручнішого з'єднання між сервісами.

Переваги використання Docker [21]:

Використання Docker у процесі розробки та розгортання програмних рішень забезпечує низку суттєвих переваг, які значно спрощують як технічну реалізацію, так і підтримку системи. Однією з ключових переваг є

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		52

автоматизація розгортання - вся необхідна інфраструктура описується у конфігураційному файлі `'docker-compose.yml'`. Це дозволяє запускати всю систему в кілька команд без потреби у складному ручному налаштуванні кожного компонента окремо, що особливо корисно при розгортанні на нових серверах або при CI/CD-процесах.

Ще однією важливою перевагою є уніфікація середовищ. Docker дозволяє створити стандартне, ідентичне середовище для всіх членів команди, незалежно від їхньої операційної системи чи локальних налаштувань. Завдяки цьому усуваються типові проблеми, пов'язані з фразою "у мене на комп'ютері працює", що суттєво зменшує витрати часу на виправлення несумісностей між середовищами.

Крім того, Docker забезпечує безпечну ізоляцію сервісів. Кожен компонент системи - база даних, бекенд, фронтенд тощо - працює у своєму окремому контейнері, незалежно від інших. Такий підхід не лише підвищує безпеку, але й дозволяє легше виявляти та локалізувати проблеми, оскільки помилка в одному сервісі не впливає безпосередньо на інші частини системи.

Окремо варто відзначити зручність моніторингу та тестування. Docker дозволяє легко інтегрувати допоміжні інструменти, такі як pgAdmin для роботи з базою даних, або сервіси логування й відлагодження. Це сприяє оперативному виявленню та усуненню помилок у середовищі, що відтворює продуктивну інфраструктуру. У підсумку, використання Docker [10] значно покращує стабільність, масштабованість і контрольованість усієї системи.

У рамках розробки рішень на платформі .NET використання Docker є надзвичайно ефективним інструментом, який істотно спрощує роботу над проектом - як під час розробки, так і на етапі розгортання. Однією з головних переваг є автоматизація налаштування всієї інфраструктури. Завдяки файлу `docker-compose.yml` розробник може визначити конфігурацію всіх сервісів - наприклад, ASP.NET Core API [14], базу даних PostgreSQL, Redis, RabbitMQ або інші компоненти. Це дозволяє запустити повноцінне середовище однією

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		53

командою `docker compose up`, без необхідності локально встановлювати кожен сервіс вручну. Такий підхід ідеально підходить для CI/CD-процесів та забезпечує швидке розгортання на staging або production-серверах.

Ще один важливий аспект - уніфікація середовища розробки. Завдяки Docker команда, яка працює з проєктом .NET, отримує гарантовано однакове середовище, незалежно від операційної системи (Windows, Linux або macOS). Наприклад, якщо додаток використовує Entity Framework Core [2], Nginx-проксі або сторонні інструменти (як-от pgAdmin чи MailHog), вони можуть бути включені до docker-compose-конфігурації, і розробники не матимуть проблем з їхнім налаштуванням - усе вже готове до використання.

Крім того, контейнеризація сервісів забезпечує безпечну ізоляцію. ASP.NET Core API [24], база даних і front-end застосунок можуть працювати у власних контейнерах, не заважаючи одне одному. Це дозволяє швидко ідентифікувати джерело проблем - наприклад, якщо запит до API не працює, можна перевірити окремо лог файли лише цього сервісу, не зачіпаючи інші.

Для повсякденної роботи з .NET-проєктом [30] Docker [9] також надає гнучкі можливості для моніторингу, тестування та відлагодження. Наприклад, можна інтегрувати Elastic Stack або Prometheus для збору метрик, запускати тести в ізольованому середовищі, а також легко вмикати допоміжні сервіси, як-от локальний SMTP для перевірки email-функціональності.

У підсумку, використання Docker [27] у проєктах на .NET значно підвищує стабільність і масштабованість рішень, забезпечуючи при цьому контрольоване середовище, що легко переноситься, розгортається та супроводжується.

3.6 Застосування Postman для адміністрування та тестування API

Перед вами зображення документації для роутів API, створене за допомогою Postman. Це наочно демонструє структуру колекції запитів для тестування API, зокрема для реєстрації користувача через POST метод. Інтерфейс Postman дозволяє тестувати API-ендпоінти та перевіряти їхню

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ПЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		54

правильність у реальному часі. Документація містить всю необхідну інформацію про запити, заголовки та методи. Зображення виглядає наступним чином (рис. 3.6).

```
{
  "info": {
    "_postman_id": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
    "name": "Courses API",
    "description": "Collection for testing Courses API endpoints",
    "schema": "https://schema.getpostman.com/json/collection/v2.1.0/collection.json"
  },
  "item": [
    {
      "name": "User",
      "item": [
        {
          "name": "Register",
          "request": {
            "method": "POST",
            "header": [
              {
                "key": "Content-Type",
                "value": "application/json"
              }
            ]
          }
        }
      ]
    }
  ]
}
```

Рис.3.6 – Документація роутів Postman

У процесі розробки сучасних вебзастосунків особливу увагу приділяють розробці та підтримці API, які виступають посередниками між клієнтською частиною та сервером. Одним із найбільш зручних інструментів для взаємодії з API є Postman - багатофункціональне середовище для тестування та документування RESTful [28] сервісів. У рамках цього дослідження Postman активно використовувався як засіб для перевірки правильності реалізації маршрутизованих запитів, зокрема при взаємодії з модулями, що обробляють статті, користувачів та питання.

Завдяки наочному інтерфейсу, Postman дозволяє моделювати запити з різними HTTP-методами (GET, POST, PUT, DELETE), встановлювати необхідні заголовки, а також передавати дані у форматі JSON або форм. Усі запити структуруються в колекції, що дозволяє швидко орієнтуватися у великій кількості ендпоінтів. Крім того, Postman підтримує змінні середовища, які спрощують тестування у різних умовах (наприклад, розробка, staging, продакшн).

Однією з найцінніших функцій для адміністратора є можливість авторизації запитів через JWT токени або інші механізми автентифікації. Це дозволяє перевірити, як система реагує на дії автентифікованих та неавторизованих користувачів, включно з можливістю оновлення профілю, редагування контенту або підтвердження електронної пошти. Для прикладу, при тестуванні запиту PUT /api/questions/{questionId} у Postman можна додати заголовок Authorization: Bearer {{token}}, встановити тіло запиту й миттєво побачити результат.

Перевагою також є вбудовані можливості для написання автоматизованих тестів, які дозволяють здійснювати перевірку коректності відповіді на кожен запит. Наприклад, можна переконатися, що при правильному оновленні питання повертається статус-код 200 ОК, а тіло відповіді містить оновлені значення. Таким чином, Postman виступає не лише як інструмент для тестування, але й як платформа для документування та адміністрування API, що значно спрощує життєвий цикл розробки застосунку.

Для забезпечення зручного та ефективного тестування розробленого API у дипломному проєкті використовується Postman - універсальний інструмент для роботи з HTTP-запитами. Цей застосунок дозволяє тестувати REST API, налагоджувати роботу з параметрами запитів, перевіряти відповіді від сервера, автоматизувати перевірки та зберігати документацію у вигляді колекцій. Однією з ключових переваг Postman є можливість імпортувати готову документацію API з JSON-файлу, що містить набір запитів, опис параметрів, маршрутів, змінних середовища тощо. Такий файл зазвичай формується вручну або автоматично експортується з бекенд-середовища розробки, наприклад.

Щоб імпортувати документацію в Postman, спершу необхідно встановити і запустити сам додаток. Далі у верхньому лівому куті інтерфейсу потрібно натиснути кнопку “Import”. У вікні, що з’явиться, потрібно перейти на вкладку “File”, натиснути “Upload Files” і вибрати відповідний .json файл з

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		56

локального комп'ютера - наприклад, api-collection.json або інший файл, експортований з іншої системи.

Після завантаження файл буде автоматично проаналізовано, і Postman запропонує створити нову колекцію (Collection). Для підтвердження імпорту потрібно натиснути кнопку "Import". Після цього у лівій панелі з'явиться колекція з назвою документації. У ній міститимуться всі HTTP-запити, згруповані за категоріями, наприклад: /api/articles, /api/users, /api/questions. Кожен запит можна редагувати, запускати, переглядати та зберігати результати. У разі потреби можна експортувати колекцію у вигляді .json файлу для подальшого використання або спільного доступу з іншими користувачами.

Якщо в колекції використовуються змінні, наприклад {{baseUrl}} чи {{token}}, варто створити відповідне середовище (environment) у Postman. Це дозволить уникнути ручного редагування кожного запиту. Змінні задаються в розділі "Environments" → "Add", де вказуються їхні імена та значення.

Перед вами зображення документації для роутів API, створене в Postman. Це наочно демонструє два типи запитів: POST для входу користувача та GET для підтвердження електронної пошти. У кожному запиті надано необхідні параметри, такі як email, пароль та інші змінні для підтвердження пошти. Документація дає зрозуміле уявлення про те, як повинні виглядати запити та їх параметри для взаємодії з API. Зображення виглядає наступним чином (рис. 3.7).

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		57

POST Login

[Open request →](#)

https://localhost:5001/api/users/login

Add request description

Body raw (json)

json

```
{
  "email": "user@example.com",
  "password": "YourPassword123!"
}
```

GET Confirm Email

[Open request →](#)

https://localhost:5001/api/users/confirm-email?userId=your_user_id_here&token=your_confirm_token_here

Рис.3.7 - Документація роутів Postman

Практичне використання

У ході дипломної роботи за допомогою Postman здійснювалося тестування реєстрації та автентифікації користувачів через запити POST до /api/users/register та /api/users/login. Також проводилася перевірка CRUD-операцій над статтями, включаючи GET, POST, PUT та DELETE запити для /api/articles. Окрім цього, було перевірено отримання та оновлення питань за ідентифікатором через GET та PUT запити до /api/questions/{id}. Крім того, тестувалася функціональність підтвердження електронної пошти через GET запит до /api/users/confirm-email?userId=....

Це дозволило переконатися у правильності роботи кожного маршруту, перевірити обробку помилок, а також протестувати логіку авторизації через JWT токени.

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		58

Висновки до третього розділу

У розділі розглянуто архітектуру інтерфейсу та порядок взаємодії користувача з вебзастосунком, призначеним для організації онлайн курсів. Вебзастосунок реалізовано за моделлю клієнт-сервер, де фронтенд і бекенд мають чіткий розподіл обов'язків та взаємодіють між собою через REST API. Основну увагу приділено опису маршрутизації, побудованої за ресурсним принципом, з підтримкою типових HTTP-методів (GET, POST, PUT, DELETE), що забезпечує зрозумілу і масштабовану логіку обробки запитів як на стороні клієнта, так і на стороні сервера.

На фронтенді використано бібліотеку React [19], яка дозволяє реалізувати компонентну архітектуру інтерфейсу з логічним поділом на сторінки та підсторінки. Компоненти відповідають окремим функціональним частинам застосунку - реєстрації, перегляду курсів, управлінню контентом тощо. Вся навігація здійснюється за допомогою React Router, що синхронізується зі структурою бекенду, дозволяючи зберігати єдиний підхід до управління маршрутами та спрощувати підтримку коду при масштабуванні системи.

Окремо розглянуто механізм автентифікації користувачів за допомогою JWT (JSON Web Token). Цей підхід дозволяє здійснювати безстейтову перевірку користувача без необхідності зберігати сесії на сервері. Сервер верифікує токен і надає доступ до захищених ресурсів лише у разі його дійсності. Така схема забезпечує як безпеку, так і гнучкість, дозволяючи масштабувати систему без залежності від стану сесій.

Загалом, побудований інтерфейс вебзастосунку у поєднанні з правильно структурованою маршрутизацією та сучасним механізмом автентифікації формує стабільну, ефективну та безпечну платформу для взаємодії користувача з системою. Обрана архітектура [31] забезпечує зрозумілий порядок використання функціоналу, гнучкість розширення можливостей

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				59
Змн.	Арк.	№ докум.	Підпис	Дата		

системи та відповідає актуальним вимогам до вебзастосунків у сфері онлайн-освіти.

Крім того, важливою особливістю розробленого вебзастосунку є використання глобального стану за допомогою бібліотеки Redux, що забезпечує централізоване управління даними між компонентами. Це особливо корисно при реалізації функціоналу авторизації, управління профілем користувача та взаємодії з сервером. Інтерфейс користувача адаптований під різні типи пристроїв завдяки застосуванню адаптивної верстки, що покращує доступність платформи. Сторінки розроблені з урахуванням UX-принципів, зокрема передбачено валідацію форм, зворотній зв'язок при помилках та візуальні підказки. Таким чином, застосунок є не лише функціонально повним, а й зручним та зрозумілим у користуванні.

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				60
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У межах кваліфікаційної роботи бакалавра було реалізовано повнофункціональний вебдодаток для організації онлайн курсів на основі сучасного стеку технологій - ASP.NET Core для бекенду та React для фронтенду. Завдяки використанню REST API забезпечено чітке розмежування між клієнтською та серверною частинами, що підвищує масштабованість, продуктивність і зручність підтримки системи.

З боку серверної частини, за допомогою Entity Framework Core реалізовано ефективну взаємодію з реляційною базою даних. Це надало змогу швидко створювати, зберігати та оновлювати сутності, дотримуючись принципів чистої архітектури та забезпечуючи високу продуктивність. Завдяки розділенню логіки на контролери, сервіси та репозиторії код залишається підтримуваним і гнучким до змін.

Користувацький інтерфейс, реалізований на React, забезпечує швидко та динамічну взаємодію з додатком, зручну навігацію, а також дозволяє адаптуватися до сучасних потреб користувачів. Під час розробки були реалізовані ключові функціональні можливості - реєстрація, логін, підтвердження електронної пошти, перегляд та управління контентом (статтями, питаннями, профілями).

Використання Postman для документування та тестування API надало змогу проводити ефективне тестування запитів, перевірку прав доступу та взаємодії між фронтендом і бекендом. Це значно спростило процес розробки, налагодження та адміністрування системи. У підсумку, результати дипломної роботи підтверджують доцільність обраного підходу та дозволяють розглядати розроблений додаток як основу для впровадження у реальних умовах або подальшого розвитку.

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ПЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		61

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Pro ASP.NET Core 6: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages / Adam Freeman, 2022 - Apress;
2. Entity Framework Core in Action / Jon P. Smith, 2018 - Manning Publications;
3. C# 10 and .NET 6 - Modern Cross-Platform Development / Mark J. Price, 2021 - Packt Publishing;
4. ASP.NET Core in Action / Andrew Lock, 2021 - Manning Publications;
5. Learning React: Modern Patterns for Developing React Apps / Alex Banks, Eve Porcello, 2020 - O'Reilly Media;
6. TypeScript Quickly / Yakov Fain, Anton Moiseev, 2020 - Manning Publications;
7. Fullstack React with TypeScript [Електронний ресурс]. - Режим доступу до ресурсу: <https://fullstackopen.com/en/> ;
8. React TypeScript Cheatsheets [Електронний ресурс]. - Режим доступу до ресурсу: <https://react-typescript-cheatsheet.netlify.app/> ;
9. Docker Deep Dive / Nigel Poulton, 2021 - Leanpub;
10. Docker for Developers / Richard Bullington-McGuire, 2020 - Packt Publishing;
11. PostgreSQL: Up and Running / Regina O. Obe, Leo S. Hsu, 2021 - O'Reilly Media;
12. The Internals of PostgreSQL / Hironobu Suzuki, 2021 - [Електронний ресурс]. - Режим доступу до ресурсу: <https://www.interdb.jp/pg/> ;
13. Model-View-Controller (MVC) Design Pattern [Електронний ресурс]. - Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Glossary/MVC> ;
14. ASP.NET Core Documentation [Електронний ресурс]. - Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/aspnet/core> ;

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		62

15. PostgreSQL Official Documentation [Електронний ресурс]. - Режим доступу до ресурсу: <https://www.postgresql.org/docs/> ;
16. Microsoft Entity Framework Core Documentation [Електронний ресурс]. - Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/ef/core/> ;
17. TypeScript Handbook [Електронний ресурс]. - Режим доступу до ресурсу: <https://www.typescriptlang.org/docs/> ;
18. Clean Architecture with .NET Core / Jason Taylor, 2022 - GitHub Repository. - Режим доступу: <https://github.com/jasontaylordev/CleanArchitecture> ;
19. React - A JavaScript Library for Building User Interfaces [Електронний ресурс]. - Режим доступу до ресурсу: <https://reactjs.org/> ;
20. Visual Studio Code Documentation [Електронний ресурс]. - Режим доступу до ресурсу: <https://code.visualstudio.com/docs> ;
21. Docker Documentation [Електронний ресурс]. - Режим доступу до ресурсу: <https://docs.docker.com/> ;
22. PostgreSQL Performance Tuning / Ibrar Ahmed, 2021 - Packt Publishing. .NET Microservices - Architecture for Containerized ;
23. .NET Applications / Cesar de la Torre, 2022 - Microsoft Docs;
24. RESTful Web API Patterns and Practices in ASP.NET Core / Fanie Reynders, 2022 - Apress;
25. Mastering TypeScript / Nathan Rozentals, 2021 - Packt Publishing;
26. Blazor in Action / Chris Sainty, 2023 - Manning Publications;
27. Pro Docker / Deepak Vohra, 2015 - Apress;
28. Hands-On RESTful Web Services with TypeScript 3 / Biharek Muniz Araújo, 2019 - Packt Publishing;
29. Professional C# and .NET 6 / Christian Nagel, 2022 - Wiley;

30. Beginning Database Programming Using ASP.NET Core / Bipin Joshi, 2021 - Apress;
31. Software Architecture with C# 10 and .NET 6 / Gabriel Baptista, Francesco Abbruzzese, 2022 - Packt Publishing;
32. Effective TypeScript: 62 Specific Ways to Improve Your TypeScript / Dan Vanderkam, 2019 - O'Reilly Media;
33. PostgreSQL: The Comprehensive Guide to Building, Programming, and Administering PostgreSQL Databases / Korrry Douglas, Susan Douglas, 2003 - Sams;
34. TypeScript for Professionals / Bjorn Lu, 2020 - Leanpub;
35. React and TypeScript: The Complete Developer's Guide / Stephen Grider, 2021 - UdeMy Course;
36. The Complete Guide to Docker for Developers / Bret Fisher, 2022 - UdeMy Course;

		Д.О. Герасимчук			ІІЗ – 121 – 25 – ІІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		64

ДОДАТКИ

		Д.О. Герасимчук			ІПЗ – 121 – 25 – ІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		65

Додаток А

Лістинг AttemptQuestionsApi.cs

```

using System.Net.Mime;
using Courses.API.Extensions;
using Courses.API.Services;
using Courses.Application.Common.Models;
using Courses.Application.AttemptQuestions.Commands.CreateAttemptQuestion;
using Courses.Application.AttemptQuestions.Commands.UpdateAttemptQuestion;
using Courses.Application.AttemptQuestions.Dto;
using Microsoft.AspNetCore.Http.HttpResults;
using Microsoft.AspNetCore.Mvc;
using Courses.Application.AttemptQuestions.Queries.GetAttemptQuestionsByTestAttempt;
using Courses.Application.AttemptQuestions.Queries.GetAttemptQuestion;

namespace Courses.API.Apis;

public static class AttemptQuestionsApi
{
    public static RouteGroupBuilder MapAttemptQuestionsApi(this IEndpointRouteBuilder app)
    {
        var api = app.MapGroup("api/attemptquestions").WithTags("AttemptQuestions").DisableAntiforgery();

        api.MapGet("/testattempts/{testAttemptId:guid}",
            GetAttemptQuestionsByTestAttemptAsync).RequireAuthorization();
        api.MapGet("/{attemptQuestionId:guid}", GetAttemptQuestionAsync).RequireAuthorization();
        api.MapPost("/", CreateAttemptQuestionAsync).RequireAuthorization();
        api.MapPut("/{attemptQuestionId:guid}", UpdateAttemptQuestionAsync).RequireAuthorization();

        return api;
    }

    [ProducesResponseType<Ok<PagedList<AttemptQuestionResponse>>>(StatusCodes.Status200OK,
        MediaTypeNames.Application.Json)]
    public static async Task<Results<Ok<PagedList<AttemptQuestionResponse>>, ProblemHttpResult>>
        GetAttemptQuestionsByTestAttemptAsync(
        [AsParameters] TestServices services,
        Guid testAttemptId,
        [FromQuery] int pageIndex = 0,
        [FromQuery] int pageSize = 10)
    {
        var result = await services.Sender.Send(new
            GetAttemptQuestionsByTestAttemptQuery(testAttemptId, pageIndex, pageSize));

        return result.IsSuccess ? TypedResults.Ok(result.Value) : result.ToProblemHttpResult();
    }

    [ProducesResponseType<Ok<AttemptQuestionResponse>>(StatusCodes.Status200OK,
        MediaTypeNames.Application.Json)]
    [ProducesResponseType<ProblemHttpResult>(StatusCodes.Status404NotFound,
        MediaTypeNames.Application.ProblemJson)]
    public static async Task<Results<Ok<AttemptQuestionResponse>, ProblemHttpResult>>
        GetAttemptQuestionAsync(
        [AsParameters] TestServices services,

```

		Л.О. Герасимчук			ІІЗ – 121 – 25 – ІІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		66

```

    Guid attemptQuestionId)
    {
        var result = await services.Sender.Send(new GetAttemptQuestionQuery(attemptQuestionId));

        return result.IsSuccess ? TypedResults.Ok(result.Value) : result.ToProblemHttpResult();
    }

    [ProducesResponseType<Ok<AttemptQuestionResponse>>(StatusCodes.Status200OK,
    MediaTypeNames.Application.Json)]
    [ProducesResponseType<ProblemHttpResult>(StatusCodes.Status400BadRequest,
    MediaTypeNames.Application.ProblemJson)]
    public static async Task<Results<Ok<AttemptQuestionResponse>, ProblemHttpResult>>
    CreateAttemptQuestionAsync(
        [AsParameters] TestServices services,
        CreateAttemptQuestionCommand request)
    {
        var result = await services.Sender.Send(request);

        return result.IsSuccess ? TypedResults.Ok(result.Value) : result.ToProblemHttpResult();
    }

    [ProducesResponseType<Ok<AttemptQuestionResponse>>(StatusCodes.Status200OK,
    MediaTypeNames.Application.Json)]
    [ProducesResponseType<ProblemHttpResult>(StatusCodes.Status400BadRequest,
    MediaTypeNames.Application.ProblemJson)]
    [ProducesResponseType<ProblemHttpResult>(StatusCodes.Status404NotFound,
    MediaTypeNames.Application.ProblemJson)]
    public static async Task<Results<Ok<AttemptQuestionResponse>, ProblemHttpResult>>
    UpdateAttemptQuestionAsync(
        [AsParameters] TestServices services,
        Guid attemptQuestionId,
        UpdateAttemptQuestionCommand request)
    {
        if (attemptQuestionId != request.Id)
        {
            return TypedResults.Problem(new ProblemDetails
            {
                Status = StatusCodes.Status400BadRequest,
                Title = "Bad Request",
                Detail = "The ID in the route does not match the ID in the request body."
            });
        }

        var result = await services.Sender.Send(request);

        return result.IsSuccess ? TypedResults.Ok(result.Value) : result.ToProblemHttpResult();
    }
}

```

Лістинг UsersApi.cs

```

using System.Net.Mime;
using Courses.API.Extensions;
using Courses.API.Services;
using Courses.Application.Common.Models;
using Courses.Application.Users.Commands.ConfirmEmail;

```

		Д.О. Герасимчук			ІІЗ – 121 – 25 – ІІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		67

```

using Courses.Application.Users.Commands.DeleteUser;
using Courses.Application.Users.Commands.LoginUser;
using Courses.Application.Users.Commands.RegisterUser;
using Courses.Application.Users.Commands.UpdateUser;
using Courses.Application.Users.Dto;
using Courses.Application.Users.Queries.GetArticle;
using Courses.Application.Users.Queries.GetArticles;
using Courses.Application.Users.Queries.GetCurrentUser;
using Microsoft.AspNetCore.Http.HttpResults;
using Microsoft.AspNetCore.Mvc;

namespace Courses.API.Apis;

public static class UsersApi
{
    public static RouteGroupBuilder MapUsersApi(this IEndpointRouteBuilder app)
    {
        var api = app.MapGroup("api/users").WithTags("users");

        api.MapGet("/", GetUsersAsync);
        api.MapGet("{userId:guid}", GetUserAsync);
        api.MapGet("/me", GetCurrentUserAsync).RequireAuthorization();

        api.MapGet("/confirm-email", ConfirmEmailAsync);
        api.MapPost("/register", RegisterUserAsync);
        api.MapPost("/login", LoginUserAsync);

        api.MapPut("/", UpdateUserAsync).RequireAuthorization();
        api.MapDelete("/{userId:guid}", DeleteUserAsync).RequireAuthorization();

        return api;
    }

    [ProducesResponseType<Ok<PagedList<UserResponse>>>(StatusCodes.Status200OK,
    MediaTypeNames.Application.Json)]
    public static async Task<Results<Ok<PagedList<UserResponse>>, ProblemHttpResult>>
    GetUsersAsync(
        [AsParameters] UserServices services,
        [AsParameters] GetUsersQuery request)
    {
        var result = await services.Sender.Send(request);

        return result.IsSuccess ? TypedResults.Ok(result.Value) : result.ToProblemHttpResult();
    }

    [ProducesResponseType<Ok<UserResponse>>(StatusCodes.Status200OK,
    MediaTypeNames.Application.Json)]
    [ProducesResponseType<ProblemHttpResult>(StatusCodes.Status404NotFound,
    MediaTypeNames.Application.ProblemJson)]
    public static async Task<Results<Ok<UserResponse>, ProblemHttpResult>> GetUserAsync(
        [AsParameters] UserServices services,
        Guid userId)
    {
        var result = await services.Sender.Send(new GetUserQuery(userId));
    }
}

```

```

        return result.IsSuccess ? TypedResults.Ok(result.Value) : result.ToProblemHttpResult();
    }

    [ProducesResponseType<Ok<UserResponse>>(StatusCodes.Status200OK,
    MediaTypeNames.Application.Json)]
    [ProducesResponseType<ProblemHttpResult>(StatusCodes.Status404NotFound,
    MediaTypeNames.Application.ProblemJson)]
    public static async Task<Results<Ok<UserResponse>, ProblemHttpResult>> GetCurrentUserAsync(
        [AsParameters] UserServices services)
    {
        var result = await services.Sender.Send(new GetCurrentUserQuery());

        return result.IsSuccess ? TypedResults.Ok(result.Value) : result.ToProblemHttpResult();
    }

    [ProducesResponseType<Ok<UserResponse>>(StatusCodes.Status200OK,
    MediaTypeNames.Application.Json)]
    [ProducesResponseType<ProblemHttpResult>(StatusCodes.Status400BadRequest,
    MediaTypeNames.Application.ProblemJson)]
    public static async Task<Results<Ok<UserResponse>, ProblemHttpResult>> RegisterUserAsync(
        [AsParameters] UserServices services,
        RegisterUserCommand request)
    {
        var result = await services.Sender.Send(request);

        return result.IsSuccess ? TypedResults.Ok(result.Value) : result.ToProblemHttpResult();
    }

    [ProducesResponseType<Ok<TokenResponse>>(StatusCodes.Status200OK,
    MediaTypeNames.Application.Json)]
    [ProducesResponseType<ProblemHttpResult>(StatusCodes.Status400BadRequest,
    MediaTypeNames.Application.ProblemJson)]
    public static async Task<Results<Ok<TokenResponse>, ProblemHttpResult>> LoginUserAsync(
        [AsParameters] UserServices services,
        LoginUserCommand request)
    {
        var result = await services.Sender.Send(request);

        return result.IsSuccess ? TypedResults.Ok(result.Value) : result.ToProblemHttpResult();
    }

    [ProducesResponseType<Ok>(StatusCodes.Status200OK, MediaTypeNames.Application.Json)]
    [ProducesResponseType<ProblemHttpResult>(StatusCodes.Status400BadRequest,
    MediaTypeNames.Application.ProblemJson)]
    public static async Task<Results<Ok, ProblemHttpResult>> ConfirmEmailAsync(
        [AsParameters] UserServices services,
        [FromQuery] Guid userId,
        [FromQuery] string token)
    {
        var result = await services.Sender.Send(new ConfirmEmailCommand(userId, token));

        return result.IsSuccess ? TypedResults.Ok() : result.ToProblemHttpResult();
    }
}

```

```

[ProducesResponseType<Ok<UserResponse>>(StatusCodes.Status200OK,
MediaTypeNames.Application.Json)]
[ProducesResponseType<ProblemHttpResult>(StatusCodes.Status400BadRequest,
MediaTypeNames.Application.ProblemJson)]
[ProducesResponseType<ProblemHttpResult>(StatusCodes.Status404NotFound,
MediaTypeNames.Application.ProblemJson)]
public static async Task<Results<Ok<UserResponse>, ProblemHttpResult>> UpdateUserAsync(
    [AsParameters] UserServices services,
    UpdateUserCommand request)
{
    var result = await services.Sender.Send(request);

    return result.IsSuccess ? TypedResults.Ok(result.Value) : result.ToProblemHttpResult();
}

```

```

[ProducesResponseType<Ok>(StatusCodes.Status200OK, MediaTypeNames.Application.Json)]
[ProducesResponseType<ProblemHttpResult>(StatusCodes.Status400BadRequest,
MediaTypeNames.Application.ProblemJson)]
[ProducesResponseType<ProblemHttpResult>(StatusCodes.Status404NotFound,
MediaTypeNames.Application.ProblemJson)]
public static async Task<Results<Ok, ProblemHttpResult>> DeleteUserAsync(
    [AsParameters] UserServices services)
{
    var result = await services.Sender.Send(new DeleteUserCommand());

    return result.IsSuccess ? TypedResults.Ok() : result.ToProblemHttpResult();
}
}

```

Лістинг MiddlewareExtensions.cs

```

using Courses.API.Middleware;

namespace Courses.API.Extensions;

public static class MiddlewareExtensions
{
    public static IApplicationBuilder UseExceptionHandlerMiddleware(this IApplicationBuilder builder)
    {
        return builder.UseMiddleware<ExceptionHandlerMiddleware>();
    }
}

```

Лістинг launchSettings.json

```

{
  "profiles": {
    "http": {
      "commandName": "Project",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      },
      "dotnetRunMessages": true,
      "applicationUrl": "http://localhost:5216"
    },
  }
}

```

		Л.О. Герасимчук			ІПЗ – 121 – 25 – ІІЗ	Арк.
		Ю.М. Єфремов				70
Змн.	Арк.	№ докум.	Підпис	Дата		

```

"https": {
  "commandName": "Project",
  "environmentVariables": {
    "ASPNETCORE_ENVIRONMENT": "Development"
  },
  "dotnetRunMessages": true,
  "applicationUrl": "https://localhost:7035;http://localhost:5216"
},
"Container (Dockerfile)": {
  "commandName": "Docker",
  "launchUrl": "{Scheme}://{ServiceHost}:{ServicePort}",
  "environmentVariables": {
    "ASPNETCORE_HTTPS_PORTS": "5001",
    "ASPNETCORE_HTTP_PORTS": "5000"
  },
  "publishAllPorts": true,
  "useSSL": true
}
},
"$schema": "https://json.schemastore.org/launchsettings.json"
}

```

Лістинг docker-compose.yml

```

services:
  courses.api:
    image: ${DOCKER_REGISTRY-}coursesapi
    container_name: courses.api
    build:
      context: ..
      dockerfile: Courses/src/Courses.API/Dockerfile
    depends_on:
      - courses.db
    networks:
      - courses-net

  courses.db:
    image: postgres:latest
    container_name: courses.db
    restart: always
    environment:
      - POSTGRES_DB=courses
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
    ports:
      - 5432:5432
    networks:
      - courses-net

networks:
  courses-net:
    driver: bridge

```

Лістинг App.tsx

```
import { Provider } from 'react-redux';
```

		Л.О. Герасимчук			ІПЗ – 121 – 25 – ІІЗ	Арк.
		Ю.М. Єфремов				71
Змн.	Арк.	№ докум.	Підпис	Дата		

```

import { Route, BrowserRouter as Router, Routes } from 'react-router-dom';
import './App.css';
import MainLayout from './components/layout/MainLayout';
import ProtectedRoute from './components/ProtectedRoute';
import CourseDetails from './pages/CourseDetails';
import Courses from './pages/Courses';
import Home from './pages/Home';
import Login from './pages/Login';
import Profile from './pages/Profile';
import Register from './pages/Register';
import TestDetails from './pages/TestDetails';
import { TestAttempt } from './pages/TestAttempt';
import { TestAttemptReview } from './pages/TestAttemptReview';
import Topics from './pages/Topics';
import { store } from './store';

```

```

function App() {
  return (
    <Provider store={store}>
      <Router>
        <Routes>
          <Route element={<MainLayout />}>
            <Route path="/" element={<Home />} />
            <Route path="/courses" element={<Courses />} />
            <Route path="/courses/:id" element={<CourseDetails />} />
            <Route path="/courses/:id/topics" element={<Topics />} />
            <Route path="/tests/:testId" element={<TestDetails />} />
            <Route path="/test-attempts/:attemptId" element={
              <ProtectedRoute>
                <TestAttempt />
              </ProtectedRoute>
            } />
            <Route path="/test-attempts/:attemptId/review" element={
              <ProtectedRoute>
                <TestAttemptReview />
              </ProtectedRoute>
            } />
            <Route path="/login" element={<Login />} />
            <Route path="/register" element={<Register />} />
            <Route path="/profile" element={
              <ProtectedRoute>
                <Profile />
              </ProtectedRoute>
            } />
          </Route>
        </Routes>
      </Router>
    </Provider>
  );
}

```

export default App;

Лістинг Home.tsx

```

import { Typography, Row, Col, Card } from 'antd';

```

		Л.О. Герасимчук			ІПЗ – 121 – 25 – ІІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		72

```

import { Link } from 'react-router-dom';
import { useSelector } from 'react-redux';
import type { RootState } from '../store';

const { Title, Paragraph } = Typography;

const Home = () => {
  const { user } = useSelector((state: RootState) => state.auth);

  return (
    <div>
      <Title level={2}>Welcome to Courses Platform</Title>
      <Paragraph>
        {user ? (
          `Welcome back, ${user.username}!`
        ) : (
          'Please login to access your courses.'
        )}
      </Paragraph>

      <Row gutter={[16, 16]} style={{ marginTop: 24 }}>
        <Col xs={24} sm={12} md={8}>
          <Card title="Browse Courses" hoverable>
            <Paragraph>
              Explore our wide range of courses designed to help you learn and grow.
            </Paragraph>
            <Link to="/courses">View Courses</Link>
          </Card>
        </Col>
        <Col xs={24} sm={12} md={8}>
          <Card title="Track Progress" hoverable>
            <Paragraph>
              Monitor your learning progress and complete assessments to earn certificates.
            </Paragraph>
          </Card>
        </Col>
        <Col xs={24} sm={12} md={8}>
          <Card title="Get Support" hoverable>
            <Paragraph>
              Need help? Our support team is here to assist you with any questions.
            </Paragraph>
          </Card>
        </Col>
      </Row>
    </div>
  );
};

export default Home;

```

Лістинг Login.tsx

```

import { LockOutlined, UserOutlined } from '@ant-design/icons';
import { Alert, Button, Card, Form, Input, Typography, Space } from 'antd';
import { useDispatch, useSelector } from 'react-redux';
import { useNavigate, Link, useLocation } from 'react-router-dom';

```

		Л.О. Герасимчук			ІПЗ – 121 – 25 – ІІЗ	Арк.
		Ю.М. Єфремов				
Змн.	Арк.	№ докум.	Підпис	Дата		73

```

import type { AppDispatch, RootState } from '../store';
import { login } from '../store/slices/authSlice';

const { Title, Text } = Typography;

const Login = () => {
  const [form] = Form.useForm();
  const dispatch = useDispatch<AppDispatch>();
  const navigate = useNavigate();
  const location = useLocation();
  const { status, registerError: error } = useSelector((state: RootState) => state.auth);

  const from = location.state?.from?.pathname || '/';

  const onFinish = async (values: { email: string; password: string }) => {
    dispatch(login(values))
      .unwrap()
      .then(() => navigate(from, { replace: true }));
  };

  return (
    <div style={{ maxWidth: 400, margin: '100px auto', padding: '0 24px' }}>
      <Card>
        <Space direction="vertical" size="large" style={{ width: '100%' }}>
          <Title level={2} style={{ textAlign: 'center', marginBottom: 0 }}>
            Welcome Back
          </Title>
          <Text type="secondary" style={{ textAlign: 'center', display: 'block' }}>
            Please sign in to continue
          </Text>

          {error && (
            <Alert
              type="error"
              message={error}
              style={{ marginBottom: 0 }}
              showIcon
              closable
            />
          )}

          <Form
            form={form}
            name="login"
            onFinish={onFinish}
            layout="vertical"
            requiredMark={false}
          >
            <Form.Item
              name="email"
              rules={[
                { required: true, message: 'Please enter your email' },
                { type: 'email' }
              ]}
            >

```

```

    <Input
      prefix={<UserOutlined />}
      placeholder="Email"
      size="large"
      disabled={status === 'loading'}
    />
  </Form.Item>

  <Form.Item
    name="password"
    rules={[
      { required: true, message: 'Please enter your password' },
    ]}
  >
    <Input.Password
      prefix={<LockOutlined />}
      placeholder="Password"
      size="large"
      disabled={status === 'loading'}
    />
  </Form.Item>

  <Form.Item>
    <Button
      type="primary"
      htmlType="submit"
      size="large"
      block
      loading={status === 'loading'}
    >
      {status === 'loading' ? 'Signing in...' : 'Sign in'}
    </Button>
  </Form.Item>
</Form>

<div style={{ textAlign: 'center' }}>
  <Text type="secondary">
    Don't have an account? <Link to="/register">Sign up</Link>
  </Text>
</div>
</Space>
</Card>
</div>
);
};

```