МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ВЕБ-ПЛАТФОРМИ ОНЛАЙН КУРСІВ З ВИКОРИСТАННЯМ ANGULAR ТА EXPRESS.JS»**

Виконала: студентка 2 курсу, групи 8.1224-з-дн
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми комп'ютерні науки
(назва освітньої програми)
Т.Є. Дівчук
(ініціали та прізвище)

Керівник професор кафедри комп'ютерних наук, доцент, д.е.н. Полуектова Н.Р.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти магістр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук, д.т.н., професор

_____ Шило Г.М.
(підпис)

" ____ " _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Дівчук Тетяні Євгеніївні
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка веб-платформи онлайн курсів з використанням Angular та Express.js

керівник роботи Полуектова Наталія Робертівна, д.е.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 28 » лютого 2025 року № 327-с

2. Строк подання студентом роботи 29.01.2026 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі, аналіз предметної області.
2. Проєктування.
3. Реалізація та тестування.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 14.03.2025 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	28.03.2025	
2.	Збір вихідних даних.	18.04.2025	
3.	Обробка методичних та теоретичних	16.06.2025	
	джерел.		
4.	Розробка першого та другого розділу.	15.09.2025	
5.	Розробка третього розділу.	29.12.2025	
6.	Оформлення та нормоконтроль	26.01.2026	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	12.02.2026	

Студент _____ Т.Є. Дівчук
(підпис) (ініціали та прізвище)

Керівник роботи _____ Н.Р. Полуектова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г. Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка веб-платформи онлайн курсів з використанням Angular та Express.js»: 66 с., 49 рис., 1 табл., 19 джерел, 5 додатків.

ANGULAR, EXPRESS, NODE.JS, MODULE, MYSQL, REST API, RXJS, UML.

Об’єкт дослідження – веб-платформи онлайн курсів.

Мета роботи – розробити веб-платформу онлайн курсів.

Методи дослідження – аналітичний, моделювання, проєктування, програмний.

У роботі розглянуто проєктування та реалізацію веб-платформи онлайн курсів з урахуванням рольової моделі та низького порогу входження для користувача. Для проєктування використані діаграми UML. Клієнтська частина виконана засобами Angular, серверна – Node.js з Express, база даних на MySQL. Для взаємодії між клієнтом і сервером обрано «клієнт-серверну» модель та реалізовано REST API. Компоненти серверної та клієнтської частини мають модульну структуру.

Елементи інфтерфейсу мають просту та зрозумілу структуру з підказками для користувачів. Передбачено автоматичне оновлення контенту при додавані/оновленні параметрів, збереження результатів на сервері, захист від XSS-атак та SQL-ін’єкцій, а також валідацію даних. Таким чином, розроблена платформа забезпечує простий і зрозумілий інтерфейс, не вимагає від користувачів технічного досвіду, а також легко масштабується та підтримується.

SUMMARY

Master’s qualifying paper “Development of a Web Platform for Online Courses Using Angular and Express.js”: 66 pages, 49 figures, 1 table, 19 references, 5 supplements.

ANGULAR, EXPRESS, NODE.JS, MODULE, MYSQL, REST API, RXJS, UML.

The object of the study is web platforms for online courses.

The aim of the study is to develop a web platform for online courses.

The methods of research are analytical, modeling, design, programming.

The paper considers the design and implementation of a web platform for online courses, taking into account the role model and low entry threshold for the user. The client side is implemented using Angular, the server side – Node.js with Express, the database – MySQL. A client-server model was chosen for interaction between the client and server, and a REST API was implemented. The components of the server and client parts have a modular structure.

The following UML diagrams were constructed: use case, activity, sequence, and deployment. The interface elements have a simple and understandable structure with hints for users. Automatic content updates are provided when adding/updating parameters, saving results on the server, protection against XSS attacks and SQL injections, as well as data validation. Thus, the developed platform provides a simple and intuitive interface, does not require technical expertise from users, and is easily scalable and maintainable.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Вступ 8
1 Технічне завдання та функціональні вимоги системи 10
1.1 Перелік умовних позначень, скорочень, термінів 10
1.2 Функціональні вимоги 11
1.2.1 Призначення і цілі створення системи 11
1.2.2 Загальні функціональні можливості системи 11
1.3 Нефункціональні вимоги 12
1.4 Опис предметної області 12
1.5 Опис системи 13
1.6 Огляд та порівняння аналогів 13
2 Проєктування 16
2.1 Використання UML під час розробки системи 16
2.2 Діаграма варіантів використання 17
2.2.1 Опис варіантів використання 20
2.3 Діаграма діяльності 26
2.4 Діаграма послідовності 28
2.5 Діаграма розгортання 29
3 Програмна реалізація та тестування 30
3.1 Опис інструментів розробки 30
3.2 Основні класи системи 30
3.3 Встановлення та налаштування оточення 32
3.4 Опис API 37
3.5 Тестування проєкту 39
3.6 Керівництво користувача 40

3.6.1 Вхід та реєстрація 40
3.6.2 Взаємодія з кабінетом 44
3.6.3 Взаємодія з курсом 45
3.6.4 Взаємодія з елементами курсу 47
Висновки 51
Перелік посилань 52
Додаток А Angular CourseDetailsComponent 54
Додаток Б Angular CoursesService 57
Додаток В Express CourseController 59
Додаток Г Express CourseRepo 61
Додаток Д Unit-test Login Component 64

ВСТУП

З розвитком онлайн-освіти зростає і попит на платформи, які дозволяють всім бажаючим навчатися дистанційно, а викладачам – розробляти та надавати матеріал у своєму стилі. Веб-платформи для онлайн-курсів мають важливе значення для онлайн-освіти, адже вони забезпечують доступ до бази знань у будь-який час і з будь-якого місця.

Ринок вже пропонує гарні готові рішення у цій сфері або дотичній до неї. Однак більшість з таких рішень мають вузьку спеціалізацію на одному або декількох аспектах, складні у користувані та підтримці, вимагають від користувачів певного технічного досвіду або є комерційними.

Враховуючи описане вище і зростаючий кожен день попит на подібні системи, виникає необхідність у створенні легкої, адаптивної, зрозумілої для кінцевого користувача та масштабованої системи, що дозволяє легко та швидко як створювати курси, так і взаємодіяти з ним на різних рівнях при мінімальному порозі технічних навичок.

Актуальність роботи полягає у створенні легкої та масштабованої веб-платформи для створення онлайн-курсів без складного і довгого процесу вивчення функціоналу та налаштувань системи.

З огляду на це, можна виділити наступні цілі і задачі дослідження.

Мета: розробити веб-платформу онлайн курсів.

Досягнення поставленої мети передбачає реалізацію наступних завдань:

– сформулювати функціональні та нефункціональні вимоги до веб-платформи;
– проаналізувати, описати та порівняти аналогічні або дотичні рішення;
– спроєктувати та побудувати архітектуру веб-платформи;
– реалізувати веб-платформу онлайн курсів;
– протестувати веб-платформу та надати керівництво користувача.

Об’єкт дослідження: веб-платформи онлайн курсів.

Предмет дослідження: Angular, Express.js та взаємодія з СКБД MySQL.

Методи дослідження: аналітичний, моделювання, проєктування, програмний.

У першому розділі описані функціональні й нефункціональні вимоги, надано аналіз предметної області та опис системи, проаналізовані існуючі аналоги та надано таблицю порівняння.

У другому розділі аргументовано доречність використання UML, спроєктовано архітектуру веб-платформи, а також надані основні діаграми та описані варіанти використання.

У третьому розділі наведено інструменти розробки, процес встановлення та налаштування клієнтської та серверної частин платформи, описані основні API-endpoint, надано приклади unit-тестування, а також надано покрокове керівництво користувача.

1 ТЕХНІЧНЕ ЗАВДАННЯ ТА ФУНКЦІОНАЛЬНІ ВИМОГИ СИСТЕМИ

1.1 Перелік умовних позначень, скорочень, термінів

Система – веб-платформа онлайн курсів.

Angular – фреймворк для створення динамічних односторінкових веб-застосунків з використанням TypeScript [1].

Node.js – середовище виконання JavaScript на сервері для створення високопродуктивних веб-сервісів [11].

Express.js – серверний фреймворк для Node.js, що полегшує створення RESTful API [5].

MySQL – система керування реляційними базами даних для зберігання даних курсів, користувачів, оцінок тощо [10].

Користувач – зареєстрована особа (слухач, викладач, адміністратор), що взаємодіє із системою згідно з наданими правами.

Адміністратор – користувач з повними правами для управління користувачами та налаштуваннями системи.

Слухач – користувач, який може переглядати та проходити курси.

Викладач – користувач, що створює курси та додає матеріали.

ДВВ – діаграма варіантів використання.

ДД – діаграма діяльності.

ДП – діаграма послідовності.

ДР – діаграма розгортання.

СКБД – система керування базами даних.

БД – база дпних.

API – інтерфейс для взаємодії між клієнтом і сервером, який реалізує обмін даними в форматі JSON [15].

JWT – технологія для передачі токенів для аутентифікації та авторизації користувачів [13].

1.2 Функціональні вимоги

1.2.1 Призначення і цілі створення системи

Функціональне призначення системи – надати можливість викладачам швидко створювати онлайн-курси з мультимедійним контентом (відео, аудіо, зображення, файли, тощо) та керувати доступом.

Експлуатаційне призначення – система експлуатується слухачами для перегляду доступних курсів і проходження їх матеріалів, а також викладачами для створення та адміністрування власних курсів. Адміністратор забезпечує керування користувачами та системними налаштуваннями.

Мета створення системи – розробити сучасний вебзастосунок, який дозволить зручно створювати й поширювати навчальні курси, забезпечувати миттєву публікацію матеріалів і гнучке керування станом.

1.2.2 Загальні функціональні можливості системи

Система має надавати викладачам такі можливості:

– створювати/редагувати/видаляти курси;
– змінювати видимість/публічність курсу;
– переглядати власні курси;
– створювати/редагувати/видаляти елементи власних курсів;
– обирати тип контенту елементу при створенні/редагуванні.

Для слухачів та викладачів система має надавати такі можливості:

– переглядати список доступних курсів;
– реєструватися в системі;
– записуватися на курси.

Система має надавати адміністраторам такі додаткові можливості:

– створення/перегляд/редагування/видалення користувачів;
– оновлення ролі користувача.

Для всіх користувачів система має надавати спільні можливості:

– вхід/вихід до/з системи;
– перегляд інформації про профіль;
– перегляд курсів та їх елементів відповідно до ролі.

1.3 Нефункціональні вимоги

Інтерфейс користувача. Інтерфейс має бути інтуїтивно зрозумілим, адаптивним (desktop/tablet/mobile) та побудованим на Angular Material компонентній бібліотеці.

Підтримка браузерів. Система повинна коректно працювати в актуальних версіях Chrome, Firefox, Microsoft Edge, Opera, Safari.

Вимоги до продуктивності. Відкриття списку курсів не повинно перевищувати 2 секунди за наявності 100+ записів, а збереження/публікація елемента курсу – не довше 1 секунди.

Вимоги до безпеки. Пароль зберігається у вигляді хешу. Доступ до API обмежується JWT-токеном. Рольова модель доступу (слухач/викладач/адміністратор) контролює операції. Незареєстровані користувачі не повинні мати доступу до особистого кабінету та адмін-панелі.

1.4 Опис предметної області

Поява онлайн-навчання зумовила необхідність розробки інструментів, що сприяють швидкому створенню та поширенню навчальних матеріалів, що містять мультимедійні елементи та інструменти оцінювання.

Традиційні методи, за яких контент готується в офлайн-режимі і перед публікацією проходить тривалу процедуру затвердження, гальмують процес створення/оновлення курсів, що знижує зацікавленість слухачів.

Веб-платформа враховує сучасні вимоги, а саме: автоматизація публікації контенту, управління видимістю та спільний доступ. Ці пункти сприяють швидкому запуску курсів як для вузької, так і для широкої аудиторії.

Система повинна зберігати курси та їхні елементи в базі даних, підтримувати функції пошуку, фільтрування та сортування, а також забезпечувати захист від несанкціонованого доступу.

1.5 Опис системи

Система побудована за архітектурою «клієнт-сервер»:

– клієнт – використовується Angular 19 з Angular Material: відповідає за форми входу/реєстрації, особистий кабінет, перегляд доступних курсів, розділ «Мої курси» для викладача та інтерфейс створення/редагування курсів і елементів;
– сервер – реалізовано на Node.js з Express: REST API, маршрути для автентифікації та їх захист, роботи з курсами та елементами, валідація даних з клієнта;
– база даних – обрано MySQL, що забезпечує створення відповідних таблиць та зв’язків для курсів, елементів, користувачів, ролей, тощо.

Під час створення курсу викладач заповнює форму та додає елементи (відео/аудіо/медіа/файли/HTML). Для відео можуть використовуватися вбудовування з зовнішніх платформ (наприклад, YouTube).

1.6 Огляд та порівняння аналогів

Ринок онлайн-освіти представлений низкою глобальних платформ керування навчанням і сервісів для створення курсів. Багато з них дозволяють публікувати мультимедійні матеріали та організовувати оцінювання.

Розглянемо деякі з таких платформ.

LMS Moodle підтримує вбудовування аудіо/відео безпосередньо в курс і надає модуль Quiz для створення та керування широким спектром типів запитань. Це дозволяє організовувати як тести різного рівня складності, так і просто наповнювати сторінку курсу необхідним матеріалом. Moodle надає розширену систему налаштування, керування доступом і видимістю курсів та елементів, але для нового користувача може бути складним. Хоч Moodle і можна використовувати в різних областях, основне його застосування – великі освітні організації [8].

Udemy фокусується на відеокурсах і додатково пропонує інтерактивні активності: тести, завдання, практичні тести тощо. Udemy підтримує лише публічні курси, а доступ до них регулюється через налаштування курсу. Інструктори не мають можливості встановлювати різні рівні доступу для окремих користувачів або груп. Усі курси за замовчуванням доступні для всіх користувачів платформи. Створення курсу на Udemy є інтуїтивно-зрозумілим завдяки простому інтерфейсу. Користувачі можуть швидко додавати відео, текст, тести та завдання, і платформа автоматично формує курс у форматі, що відповідає стандартам Udemy [19].

Open edX має гнучку систему керування доступом. Курси можна зробити відкритими або закритими для певних груп. Крім того, доступність окремих частин курсу можна обмежувати в залежності від прогресу користувача. Створення курсів є досить складним, що нагадує Moodle, оскільки платформа пропонує широкий набір функцій для створення та адміністрування курсу. За аналогією з Moodle, платформа більше підходить для великих освітніх установ і потребує більше часу для навчання користувачів [14].

Google Classroom слугує централізованим хабом для класів: учні приєднуються за посиланням/кодом/запрошенням, викладачі розміщують матеріали та завдання [7].

У таблиці 1.1 наведено порівняння описаних вище систем з системою, що розробляється.

Таблиця 1.1 – Порівняння аналогів

Платформа	**Вбудовування медіа**	**Тести / оцінювання**	**Керування доступом**	**Простота створення та наповнення**	**Особливості**
Moodle	Так	Так	Гнучке керування видимістю та доступом	Складне, багато налаштувань	Гнучкість, для великих установ
Udemy	Так	Так	Лише публічні курси	Дуже просте	Простота для новачків
Open edX	Так	Так	Гнучке, для груп і частин курсу	Складне, для великих проєктів	Високий поріг входу
Наша система	Так	Можна розширити функціонал системи	Публічний / приватний	Дуже просте	Швидкий старт, простота використання

Аналіз систем демонструє, що існуючі рішення покривають більшість типових сценаріїв онлайн-навчання: мультимедіа, тести, керування доступом. Водночас вони або є важковаговими/всеосяжними (Moodle, Open edX), або орієнтованими на маркетплейс та масовий продаж курсів (Udemy). Наша система вирізняється фокусом на простоті використання та підтримки, мінімально необхідному функціоналі з можливістю масштабування і швидкій публікації контенту.

2 ПРОЄКТУВАННЯ

2.1 Використання UML під час розробки системи

UML – стандартизована нотація для візуалізації, специфікації, проєктування та документування компонентів програмних систем. Вона підтримується консорціумом Object Management Group (OMG) і слугує мовою проєктування, а не програмування. UML дає змогу зосередитися на логіці та структурі системи до початку реалізації, що особливо важливо для клієнт-серверних веб-застосунків із численними компонентами та ролями користувачів [6, 17].

Розглянемо переваги UML:

– стандартизованість – UML формалізує представлення вимог і рішень на основі офіційної специфікації OMG;
– візуалізація – діаграми сприяють спільному опрацюванню архітектури між замовником, аналітиком і командою;
– підтримка ООП – UML поєднується з інкапсуляцією, наслідуванням і поліморфізмом та дієвою декомпозицією;
– зниження витрат – формалізація вимог і сценаріїв на ранньому етапі зменшує кількість помилок під час імплементації [17].

У цій роботі застосовано наступні діаграми:

– діаграма варіантів використання – для відображення функціональних вимог та акторів;
– діаграма діяльності – для відображення послідовності дій та розгалудження бізнес-процесів;
– діаграма послідовності – для моделювання динамічних взаємодій між об’єктами в часовому проміжку;
– діаграма розгортання – для моделювання фізичного розміщення програмних компонентів системи на апаратних пристроях.

Наведені діаграми забезпечують узгодженість між логікою роботи, структурами даних і механізмами взаємодії інтерфейсів і сервісів.

2.2 Діаграма варіантів використання

Діаграма варіантів використання (Use Case Diagram) призначена для візуалізації функціональних вимог системи та взаємодії між її компонентами. Вона дає змогу відобразити варіанти використання, які можуть відбуватися в межах системи, а також актора, який виконує певні дії. Актори можуть бути як внутрішніми (користувачі системи), так і зовнішніми (інші системи чи пристрої) [18].

Розглянемо основні елементи ДВВ [18].

Актори – зовнішні сутності, що взаємодіють із системою, виконуючи певні дії. Актори можуть бути користувачами, іншими системами чи пристроями.

Варіанти використання – описують функціональні можливості системи з точки зору акторів.

Зв’язки – лінії, що вказують на взаємодію актора з певним варіантом використання. Вони можуть бути лініями або спеціальними типами зв’язків, такими як include або extend.

Діаграми варіантів використання пропонують зрозумілий спосіб документування вимог до системи, що робить їх доступними для всіх учасників проєкту, незалежно від їхнього технічного досвіду. Вони сприяють ефективному обміну інформацією між зацікавленими сторонами, дозволяючи бізнес-аналітикам, розробникам та замовникам спілкуватися на одній мові [18].

Ієрархічна декомпозиція функціоналу дозволяє структурувати складні системи від загальних сценаріїв до конкретних варіантів використання. Окрім того, діаграми варіантів використання легко трансформуються в інші типи діаграм, що спрощує перехід від аналізу вимог до етапів проєктування та реалізації системи [6, 18].

На рисунку 2.1 зображена ДВВ системи онлайн-курсів.

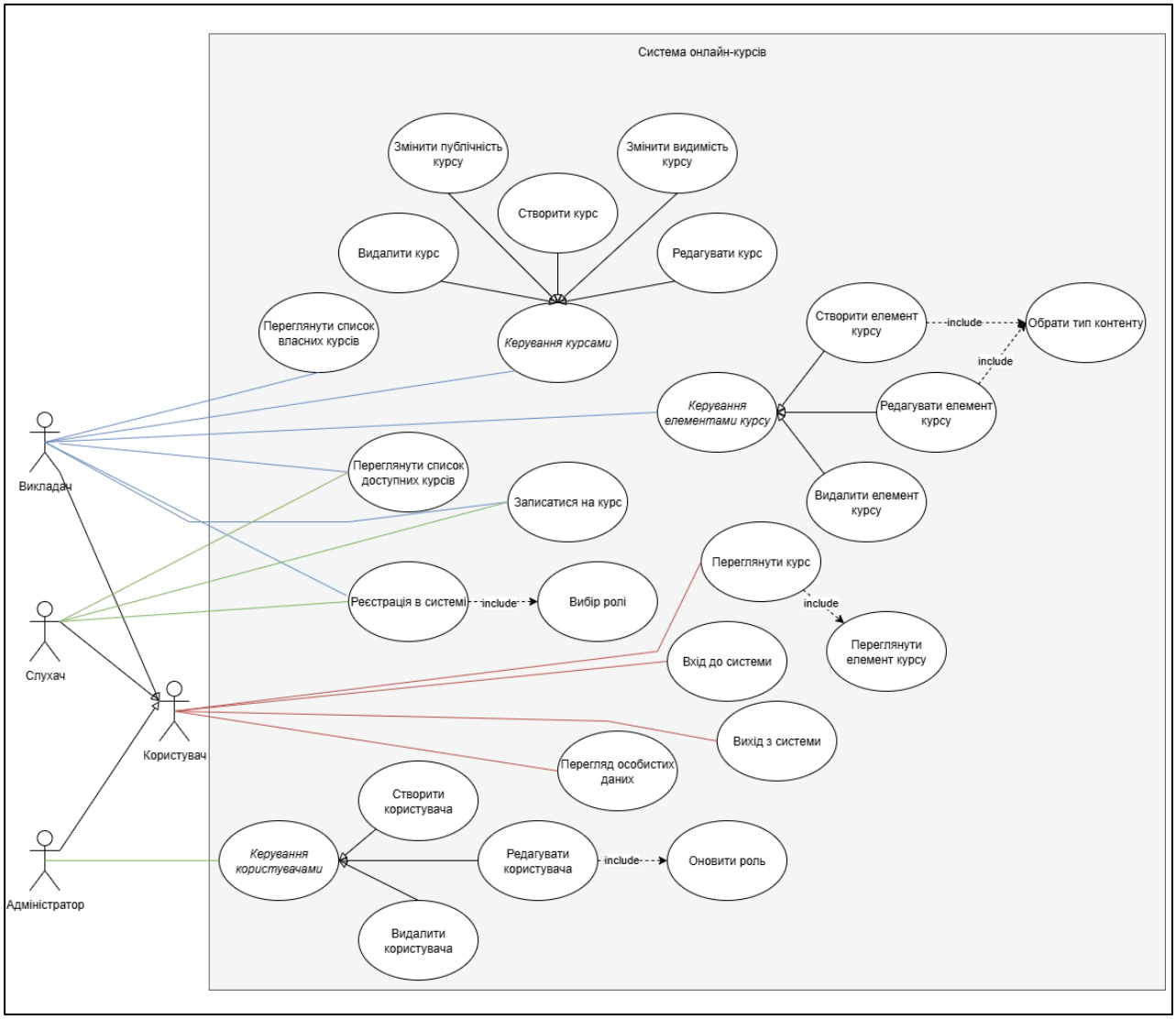


Рисунок 2.1 – Діаграма варіантів використання

Користувачі системи представлені наступними акторами: «Викладач» «Слухач» та «Адміністратор». Кожен з акторів взаємодіє з системою на різних рівнях, основним актором, який створює та взаємодіє з курсами на різних рівнях, є «Викладач» (див. рис. 2.2).

Адміністратор не має прав щодо курсів, але може керувати всіма користувачами. Слухач є базовим користувачем системи з правами на перегляд доступних курсів, запис на такі курси та взаємодію з їх вмістом. Всі актори об’єднані у одного спільного – «Користувач». Він відображає спільні функціональні можливості всіх акторів системи (див. рис. 2.3).

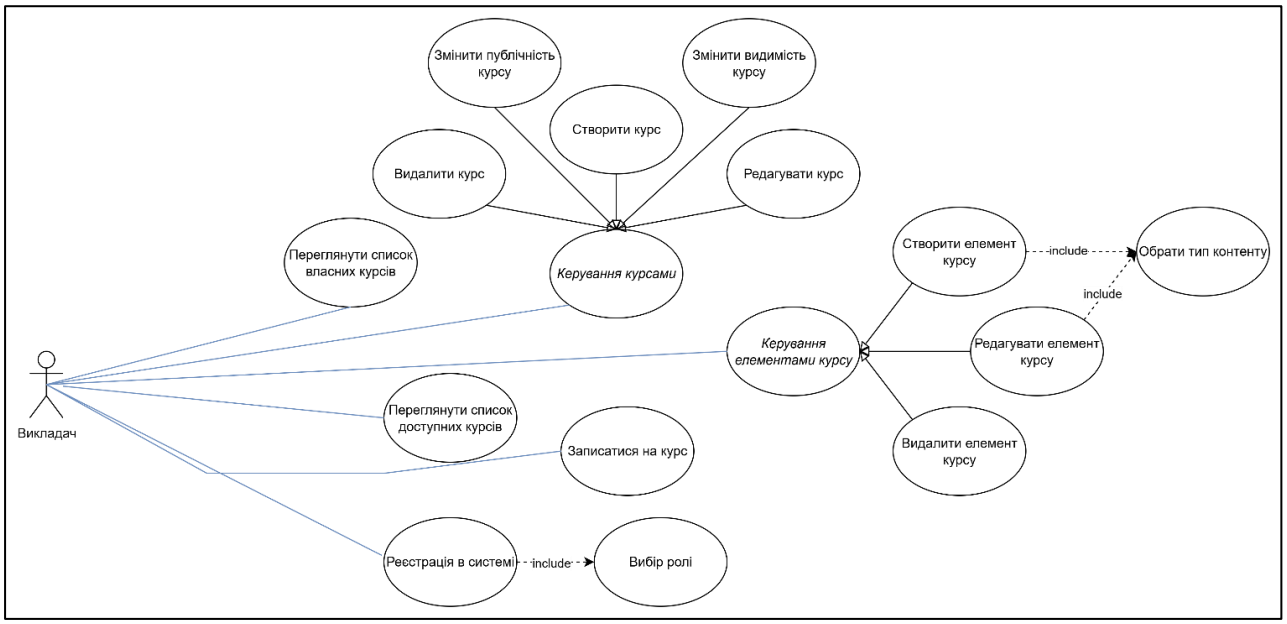


Рисунок 2.2 – ДВВ актора «Викладач»

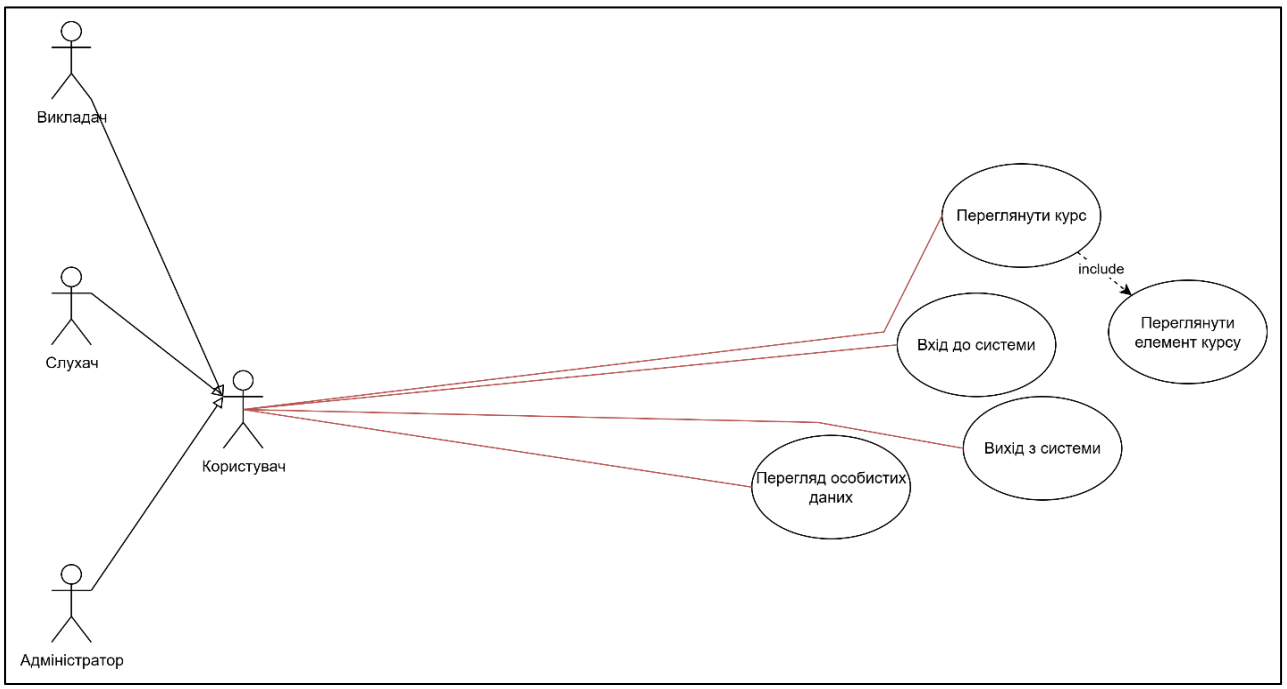


Рисунок 2.3 – ДВВ актора «Користувач»

Виділимо 1 з основних варіантів використання – «Перегляд курсу». Цей варіант використання хоч і не є повністю основним бізнес-процесом системи, але є важливою його складовою, бо доступний для всіх користувачів в залежності від рівня доступу.

Після того, як користувач натискає на пункт «Переглянути» спадного меню картки курсу або наводить на картку та натискає на кнопку «Переглянути»,

відображається сторінка курсу та її елементи. Якщо роль користувача «Викладач» та він є автором цього курсу, то система надає розширені можливості, як от: створення нового елементу (відео, аудіо, HTML, зображення тощо), редагування та видалення існуючих елементів. Якщо роль користувача «Слухач» або навіть «Адміністратор», то система надає можливість тільки взаємодіяти з існуючим контентом, те саме стосується викладачів, які не є авторами переглядаємих курсів.

2.2.1 Опис варіантів використання

Прецедент «Реєстрація в системі».

Призначення: даний варіант використання надає можливість створити обліковий запис із роллю «Слухач» або «Викладач».

Передумови: обліковий запис із вказаним email відсутній.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач відкриває сторінку реєстрації, вводить email і пароль, обирає роль та натискає на кнопку «Створити аккаунт». Система валідовує дані, створює обліковий запис і перенаправляє на сторнку входу до системи.

Виняткова ситуація 1: користувач з вказаним email уже існує – система виводить повідомлення про неможливість реєстрації з цим email.

Виняткова ситуація 2: невалідні дані – система виводить повідомлення про помилку під відповідним полем та блокує кнопку «Створити аккаунт».

Виняткова ситуація 3: помилка сервера – система виводить повідомлення про технічну проблему та рекомендує спробувати пізніше.

Прецедент «Вхід у систему».

Призначення: даний варіант використання надає можливість автентифікуватися для доступу до особистого кабінету.

Передумови: користувач зареєстрований.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач відкриває форму входу, вводить email і пароль та натискає на

кнопку «Вхід». Система перевіряє облікові дані та надає доступ до розділів відповідно до ролі.

Виняткова ситуація 1: неправильні облікові дані – система виводить повідомлення про помилку авторизації.

Виняткова ситуація 2: сервер недоступний – система виводить повідомлення про технічну помилку та радить спробувати пізніше.

Прецедент «Вихід із системи».

Призначення: даний варіант використання надає можливість завершити сесію користувача.

Передумови: користувач увійшов в систему.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає та піктограму профіля в правому верхньому кутку особистого кабінету та обирає пункт «Вихід». Система видаляє токен авторизації, очищує локальні дані сесії та перенаправляє на головну сторінку.

Прецедент «Перегляд особистих даних».

Призначення: даний варіант використання надає можливість переглянути інформацію про власний обліковий запис.

Передумови: користувач увійшов в систему.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає та піктограму профіля в правому верхньому кутку особистого кабінету та обирає пункт «Мій профіль». Система відображає сторінку з особистою інформацією користувача.

Виняткова ситуація 1: протермінований токен – система перенаправляє на сторінку входу иа виводить повідомлення про закінчення дії токена.

Виняткова ситуація 2: сервер недоступний – система виводить повідомлення про технічну помилку та радить спробувати пізніше.

Прецедент «Переглянути список доступних курсів».

Призначення: даний варіант використання надає можливість переглянути каталог публічних курсів та приватних, до яких надано доступ.

Передумови: користувач увійшов в систему.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач відкриває розділ «Доступні курси». Система завантажує перелік, дозволяє застосувати пошук і фільтри, перейти до картки курсу.

Виняткова ситуація 1: помилка отримання списку курсів – система виводить відповідне повідомлення та пропонує повторити спробу.

Виняткова ситуація 2: сервер недоступний – система виводить повідомлення про технічну помилку та радить спробувати пізніше.

Прецедент «Переглянути власні курси».

Призначення: даний варіант використання надає можливість викладачу переглянути каталог створених ним курсів.

Передумови: користувач увійшов в систему та має роль «Викладач».

Основний потік подій: даний варіант використання починає виконуватися, коли користувач відкриває розділ «Мої курси». Система завантажує перелік, дозволяє застосувати пошук і фільтри, дії щодо картки курсу.

Виняткова ситуація 1: помилка отримання списку курсів – система виводить відповідне повідомлення та пропонує повторити спробу.

Виняткова ситуація 2: сервер недоступний – система виводить повідомлення про технічну помилку та радить спробувати пізніше.

Прецедент «Переглянути курс».

Призначення: даний варіант використання надає можливість переглядати опубліковані елементи курсу.

Передумови: курс доступний та користувач увійшов в систему.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає на кнопку «Переглянути» відповідної картки курсу. Система завантажує структуру, користувач може взаємодяіти з елементами згідно рольової моделі.

Виняткова ситуація 1: медіаресурс недоступний – система виводить повідомлення про помилку завантаження.

Виняткова ситуація 2: спроба доступу до прихованого елемента – система відмовляє в доступі.

Виняткова ситуація 3: спроба доступу до прихованого/приватного курсу – система відмовляє в доступі.

Виняткова ситуація 4: сервер недоступний – система виводить повідомлення про технічну помилку та радить спробувати пізніше.

Прецедент «Створити курс».

Призначення: даний варіант використання надає можливість викладачу створити новий курс.

Передумови: користувач увійшов в систему, має роль «Викладач» та знаходиться на сторінці «Мої курси».

Основний потік подій: даний варіант використання починає виконуватися, коли викладач натискає «Створити новий курс» у розділі «Мої курси». Система відкриває форму, у якій викладач вводить назву/опис, додає зображення, налаштовує публічність/видимість і натискає на кнопку «Створити». Система створює курс та підтягує у розділ «Мої курси» відповідного викладача та у розділ «Доступні курси», якщо курс публічний та видимий.

Виняткова ситуація 1: невалідні дані – система виводить повідомлення про помилку під відповідним полем та блокує кнопку «Створити».

Виняткова ситуація 2: помилка збереження – система виводить повідомлення про помилку та пропонує повторити.

Прецедент «Редагувати курс».

Призначення: даний варіант використання надає можливість викладачу оновити властивості власного існуючого курсу.

Передумови: курс існує та користувач має роль «Викладач» із правами на курс.

Основний потік подій: даний варіант використання починає виконуватися, коли викладач натискає на піктограму «…» в правому нижньому кутку картки курсу та обирає пункт «Редагувати». Система відкриває форму редагування, викладач вносить зміни та натискає на кнопку «Зберегти». Система оновлює дані та відображає повідомлення про успіх та оновлену картку.

Виняткова ситуація 1: невалідні дані – система виводить повідомлення про помилку під відповідним полем та блокує кнопку «Зберегти».

Виняткова ситуація 2: помилка збереження – система виводить повідомлення про помилку та пропонує повторити.

Прецедент «Видалити курс».

Призначення: даний варіант використання надає можливість викладачу вилучити власний курс із системи.

Передумови: курс існує та користувач має роль «Викладач» із правами на курс.

Основний потік подій: даний варіант використання починає виконуватися, коли викладач натискає на піктограму «…» в правому нижньому кутку картки курсу, обирає пункт «Видалити» та підтверджує дію. Система видаляє курс та відображає повідомлення про успіх.

Виняткова ситуація 1: помилка видалення – система повідомляє про неможливість виконання дії.

Прецедент «Редагувати елемент курсу».

Призначення: даний варіант використання надає можливість викладачу змінювати налаштування існуючого елемента.

Передумови: елемент існує та користувач має роль «Викладач» із правами на курс.

Основний потік подій: даний варіант використання починає виконуватися, коли викладач натискає на піктограму «…» в лівому верхньому кутку картки елементу та обирає пункт «Редагувати». Система відкриває форму редагування, викладач вносить зміни та натискає на кнопку «Зберегти». Система оновлює дані та відображає повідомлення про успіх та оновлений елемент.

Виняткова ситуація 1: невалідні дані – система виводить повідомлення про помилку під відповідним полем та блокує кнопку «Зберегти».

Виняткова ситуація 2: помилка збереження – система виводить повідомлення про помилку та пропонує повторити.

Прецедент «Додати елемент курсу».

Призначення: даний варіант використання надає можливість викладачу додати відео, аудіо, зображення/медіа, файл або HTML.

Передумови: курс існує та користувач має роль «Викладач» із правами на курс.

Основний потік подій: даний варіант використання починає виконуватися, коли викладач відкриває сторінку курсу та натискає на кнопку «Додати елемент». Система відкриває форму, у якій викладач визначає тип, заповнює поля, вказує порядок, видимість та натискає на кнопку «Створити». Система створює елемент та додає у структуру курсу.

Виняткова ситуація 1: невалідні дані – система виводить повідомлення про помилку під відповідним полем та блокує кнопку «Створити».

Виняткова ситуація 2: помилка збереження – система виводить повідомлення про помилку та пропонує повторити.

Виняткова ситуація 3: недоступний зовнішній ресурс – система виводить повідомлення про.

Прецедент «Видалити елемент курсу».

Призначення: даний варіант використання надає можливість викладачу вилучити обраний елемент зі структури курсу.

Передумови: елемент існує та користувач має роль «Викладач» із правами на курс.

Основний потік подій: даний варіант використання починає виконуватися, коли викладач натискає на піктограму «…» в лівому верхньому кутку картки елементу, обирає пункт «Видалити» та підтверджує дію. Система видаляє елемент та відображає повідомлення про успіх.

Виняткова ситуація 1: помилка видалення – система повідомляє про неможливість виконання дії.

Прецедент «Змінити публічність курсу».

Призначення: даний варіант використання надає можливість викладачу керувати доступністю курсу.

Передумови: курс існує та користувач має роль «Викладач» із правами на курс.

Основний потік подій: даний варіант використання починає виконуватися, коли викладач натискає на піктограму «…» в правому нижньому кутку картки

курсу та обирає пункт «Зробити публічним»/ «Зробити приватним». Система оновлює статус.

Виняткова ситуація 1: помилка оновлення статусу – система виводить повідомлення про помилку і повертає попереднє значення.

Прецедент «Змінити видимість курсу».

Призначення: даний варіант використання надає можливість викладачу змінити видимість курсу без зміни статусу доступу.

Передумови: курс існує та користувач має роль «Викладач» із правами на курс.

Основний потік подій: даний варіант використання починає виконуватися, коли викладач натискає на піктограму «…» в правому нижньому кутку картки курсу та обирає пункт «Показати»/ «Приховати». Система оновлює статус.

Виняткова ситуація 1: помилка оновлення статусу – система виводить повідомлення про помилку і повертає попереднє значення.

2.3 Діаграма діяльності

ДД моделює послідовності дій і розгалуження бізнес-процесу, відображаючи дії, переходи, умови, синхронізацію паралельних гілок тощо. Вона використовується для зображення алгоритмів, процедур та робочих процесів у межах системи. Умовні переходи, вузли розгалуження/об’єднання та бар’єри синхронізації дають можливість моделювати як послідовне, так і паралельне виконання [6].

На рисунку 2.4 надано ДД прецеденту «Переглянути курс».

ДД данного варіанту використання відображає одну із основних логік бізнес-процесу системи. Вона дозволяє зрозуміти порядок та взаємозв’язки між діями, а також продемонструвати моменти, в яких можуть виникнути розгалудження та виняткові ситуації [6, 9].

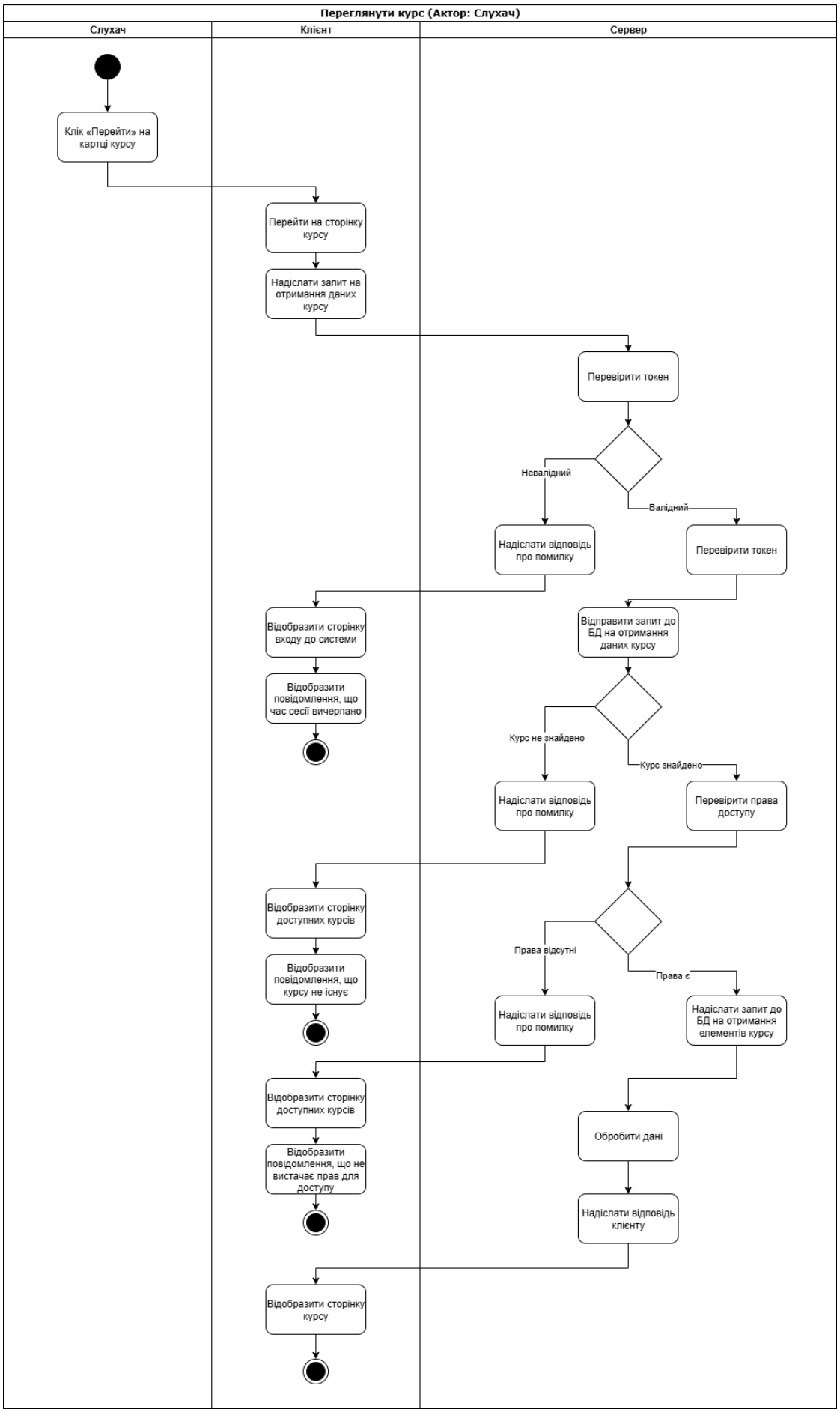


Рисунок 2.4 – ДД основного бізнес-процесу

2.4 Діаграма послідовності

ДП моделює динамічну взаємодію між об’єктами/компонентами в часі: життєві лінії, повідомлення/виклики методів, можливі комбіновані фрагменти (альтернативи, опції, цикли). У веб-застосунку це дає можливість відстежити шлях даних між клієнтом, сервером, репозиторієм і СКБД, що допомогає при проєктуванні REST-взаємодій і контролерів [9].

На рисунку 2.5 наведено ДП для сценарію «Переглянути курс».

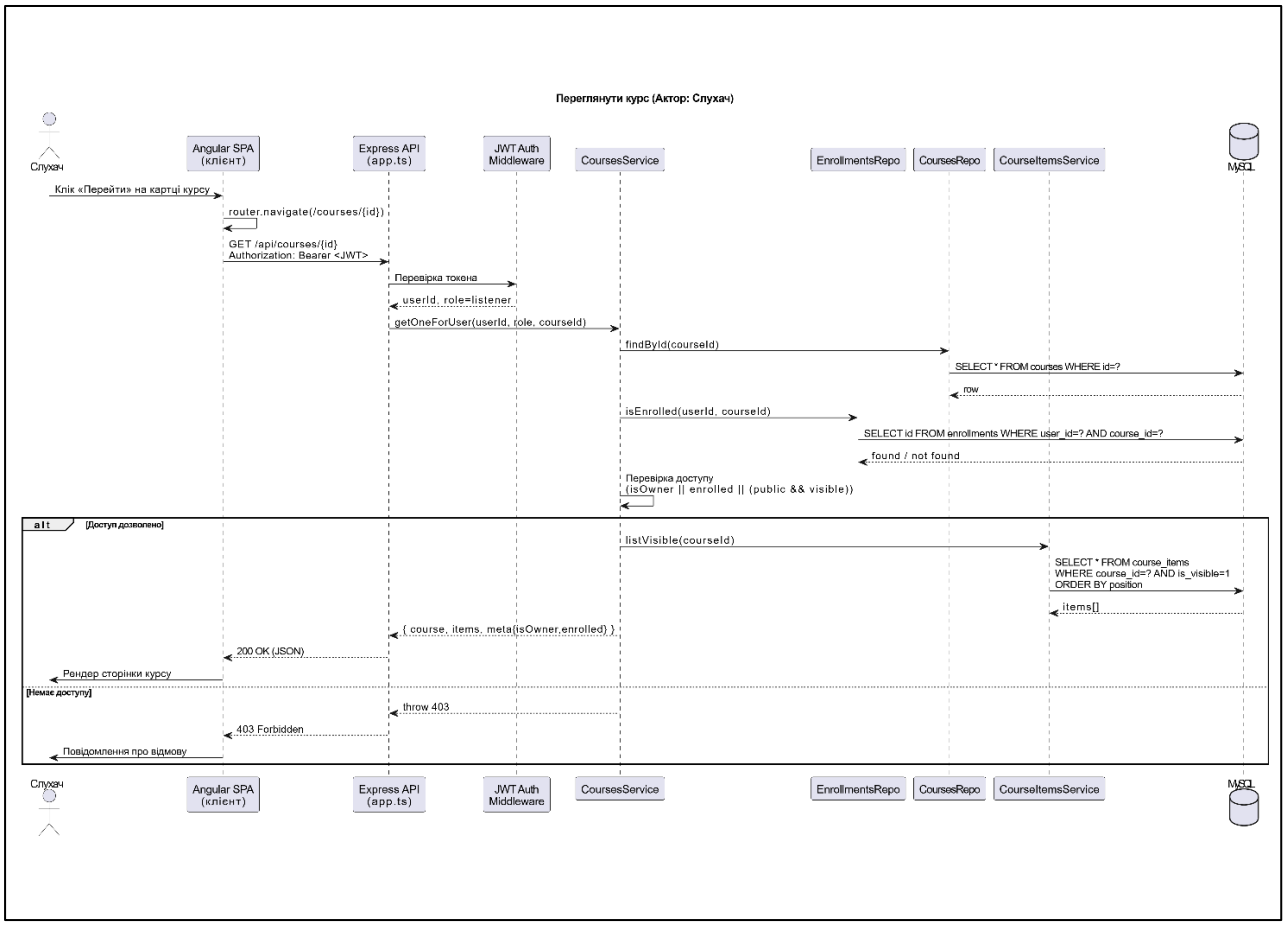


Рисунок 2.5 – ДП основного бізнес-процесу

Ця діаграма візуалізує логіку взаємодії з курсом, як лінійну взаємодію між компонентами. Її перевагами є те, що вона показує послідовність викликів дій та хто за них відповідає [9].

2.5 Діаграма розгортання

ДР відображає фізичне розміщення компонентів на вузлах, мережеві з’єднання та артефакти. Така візуалізація формалізує відповідальність вузлів і не накладає обмежень на конкретну інфраструктуру [6, 9].

На рисунку 2.6 показано розгортання системи онлайн-курсів.

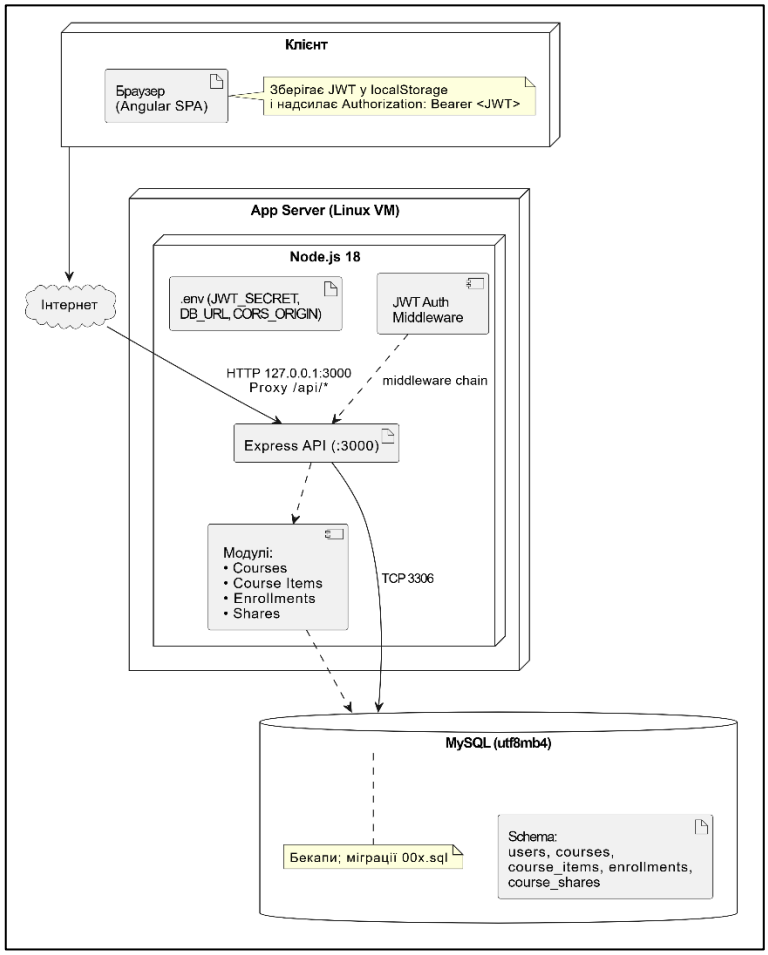


Рисунок 2.6 – Діаграма розгортання

У розділі обґрунтовано використання UML для проєктування системи онлайн-курсів і створено взаємопов’язані моделі: ДВВ для віображення акторів/функціонала, ДД для відображення логіка процесу взаємодії з курсом, ДП для взаємодії між компонентами в часі та ДР для відображення фізичного розміщення системи. Це забезпечує узгодженість вимог, архітектури й реалізації та слугує фундаментом для подальших етапів імплементації [9].

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Опис інструментів розробки

Система реалізована за допомогою сучасних технологій для створення подібних веб-платформ, що забезпечує масштабованість, гнучкість та високу продуктивність.

Angular – фреймворк для розробки SPA, використовує модульний підхід, декларативне програмування, реактивність та інкапсуляцію [1].

RxJS – бібліотека, що забезпечує реактивне програмування та асинхронну обробку даних через об’єкти Observable та спеціальні оператори [4, 16].

Node.js – серверна платформа, що використовується для створення масштабованих і продуктивних серверних додатків та побудови REST API [11].

Express.js – Node.js фреймворк, що забезпечує маршрутизацію запитів та їхню обробку через middleware [5].

MySQL – реляційна СКБД, яка зберігає інформацію у вигляді таблиць та зв’язків між ними [10].

Angular Material – бібліотека для створення сучасного UI/UX згідно з принципами Material Design [2].

3.2 Основні класи системи

Основним бізнес-процесом системи є взаємодія з курсами на різних рівнях, тому основними класами будуть ті, які забезпечують цей бізнес-процес.

Зі сторони клієнту можна виділити наступні класи:

– CourseDetailsComponent – основний Angular-компонент, який відповідає за відображення курсу, включаючи його елементи (відео, аудіо, медіа, HTML-блоки тощо);

– CoursesService – Angular-сервіс, що відповідає за обмін даними між клієнтом і сервером через REST API, зокрема, завантаження даних курсів, елементів курсу та управління підписками на курси;
– ContentEmbedComponent – Angular-компонент, який відповідає за вбудовування вмісту (відео, аудіо, медіа, HTML тощо) у картки елементів курсу.

Зі сторони серверу можна виділити наступні класи та каталоги:

– CoursesController.js – контролер Express.js, який приймає HTTP-запити на створення, оновлення, видалення курсів, а також повертає деталі курсів, включаючи його елементи;
– CourseItemsController.js – контролер Express.js, який приймає HTTP-запити на створення, оновлення, видалення елементів курсів;
– Models/ – каталог, що містить моделі БД: основною є модель Course, що описує поля курсу, включаючи назву, опис, зображення, доступність, видимість, власника тощо.

На рисунку 3.1 наведено приклад функції отримання курсів користувача на сервері.

```
export async function getOneForUser(userId:number, role:'listener'|'teacher', id:number) {  Show usages
    const row = await repo.findById(id);
    if (!row) throw { status:404, message:'Course not found' };
    const isOwner = row.owner_id === userId;
    const enrolled = await enrRepo.isEnrolled(userId, id);

    const isAccessible = isOwner || enrolled || (!!row.is_public && !!row.is_visible);
    if (!isAccessible) throw { status:403, message:'Access denied' };

    const course = mapRow(row);
    const itemsList = isOwner ? await items.listAll(id) : await items.listVisible(id);
    return { course, items: itemsList, meta: { isOwner, enrolled } };
}
```

Рисунок 3.1 – Функція отримання курсів користувача

Ознайомитися з компонентом курсу можна в додатку А, з сервісом – додаток Б, а з контролером і репозитарієм в додатках В та Г.

3.3 Встановлення та налаштування оточення

Перед початком розробки системи необхідно налаштувати оточення. І серверна, і клієнтська сторона працюють на JavaScript, тому необхідно встановити актуальну версію Node та npm.

Так як використоується ОС Linux Ubuntu, то у термінал необхідно ввести команди sudo apt install nodejs та sudo apt install npm [11, 12].

Після закінчення процесу встановлення, почнемо розгортати клієнтську частину, для створення базового Angular-проєкту необхідно встановити Angular CLI. Робимо це за допомогою команди npm install -g @angular/cli [1], після цього створюємо папку, яка буде містити код клієнтської частини та переходимо до неї (рис. 3.2). Далі виконуємо команду ng new name_project –key (рис. 3.3) [1, 3].

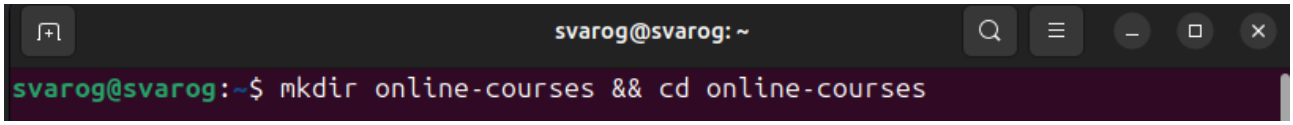


Рисунок 3.2 – Створення папки клієнту

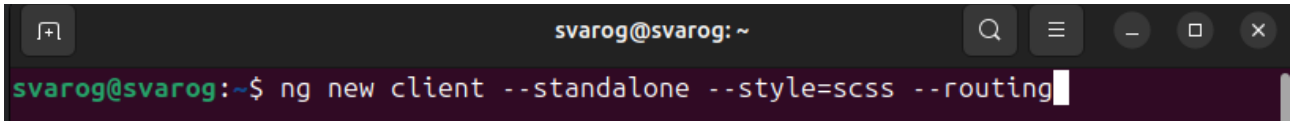


Рисунок 3.3 – Створення базового коду клієнтської частини

Переходимо до папки проєкту та встановлюєм Angular Material за допомогою команди ng add @angular/material (рис. 3.4) [2].

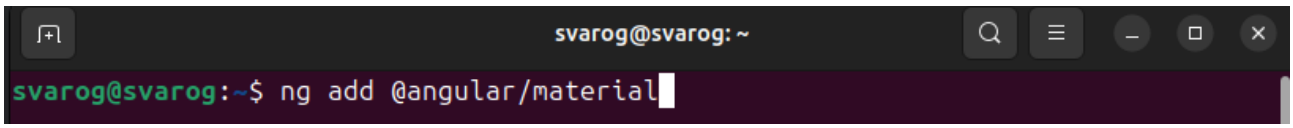


Рисунок 3.4 – Встановлення Angular Material

За аналогією створюємо у корені папку для серверу (рис. 3.5) та ініціалізуємо package.json за допомогою команди npm i -y (рис. 3.6) [11, 12].

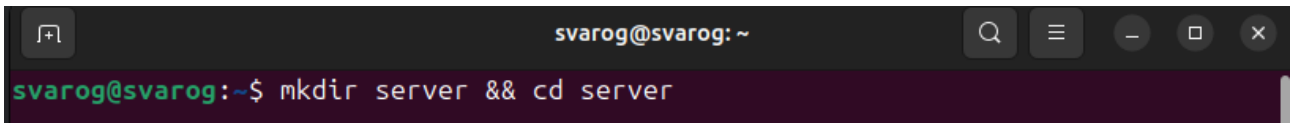


Рисунок 3.5 – Створення папки сервера

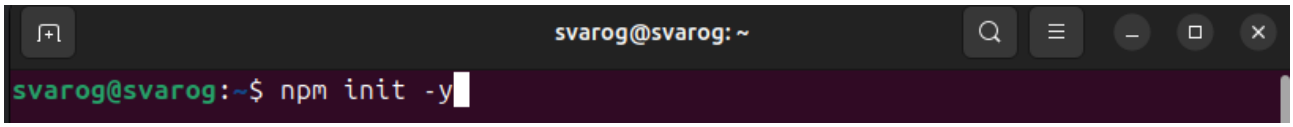


Рисунок 3.6 – Ініціалізація package.json

Далі необхідно встановити серверні залежності для роботи самого застосунку (рис. 3.7) та для процесу розробки із прапором -D (рис. 3.8). Серед залежностей, які необхідні для роботи застосунку (runtime-залежності) необхідно особливо встановити CORS, так як без нього запити зі сторони клієнта будуть блокуватися, інші залежності дозволяють використовувати фреймфорк express, підключення до mysql, авторизацію (jsonwebtoken, bycryptjs), валідацію (zod) та доступ до env-змінних. Серед залежностей для розробки встановлюємо typescript та пакети, які дозволяють йому працювати з іншими складовими проєкту.

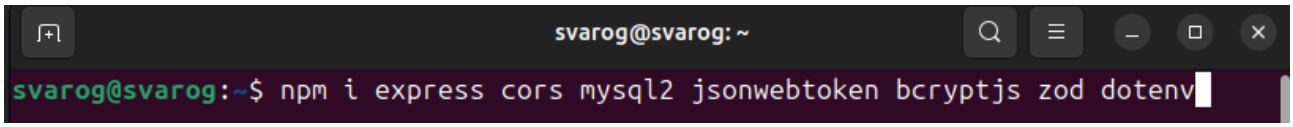


Рисунок 3.7 – Встановлення runtime-залежностей

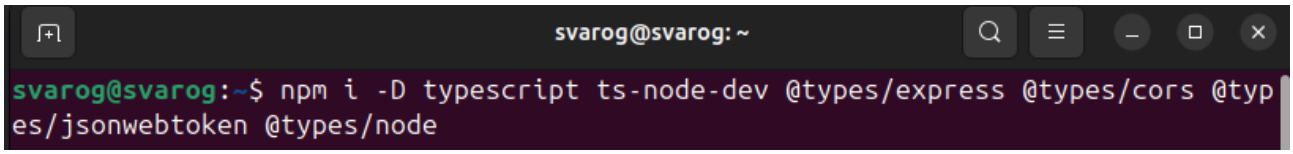
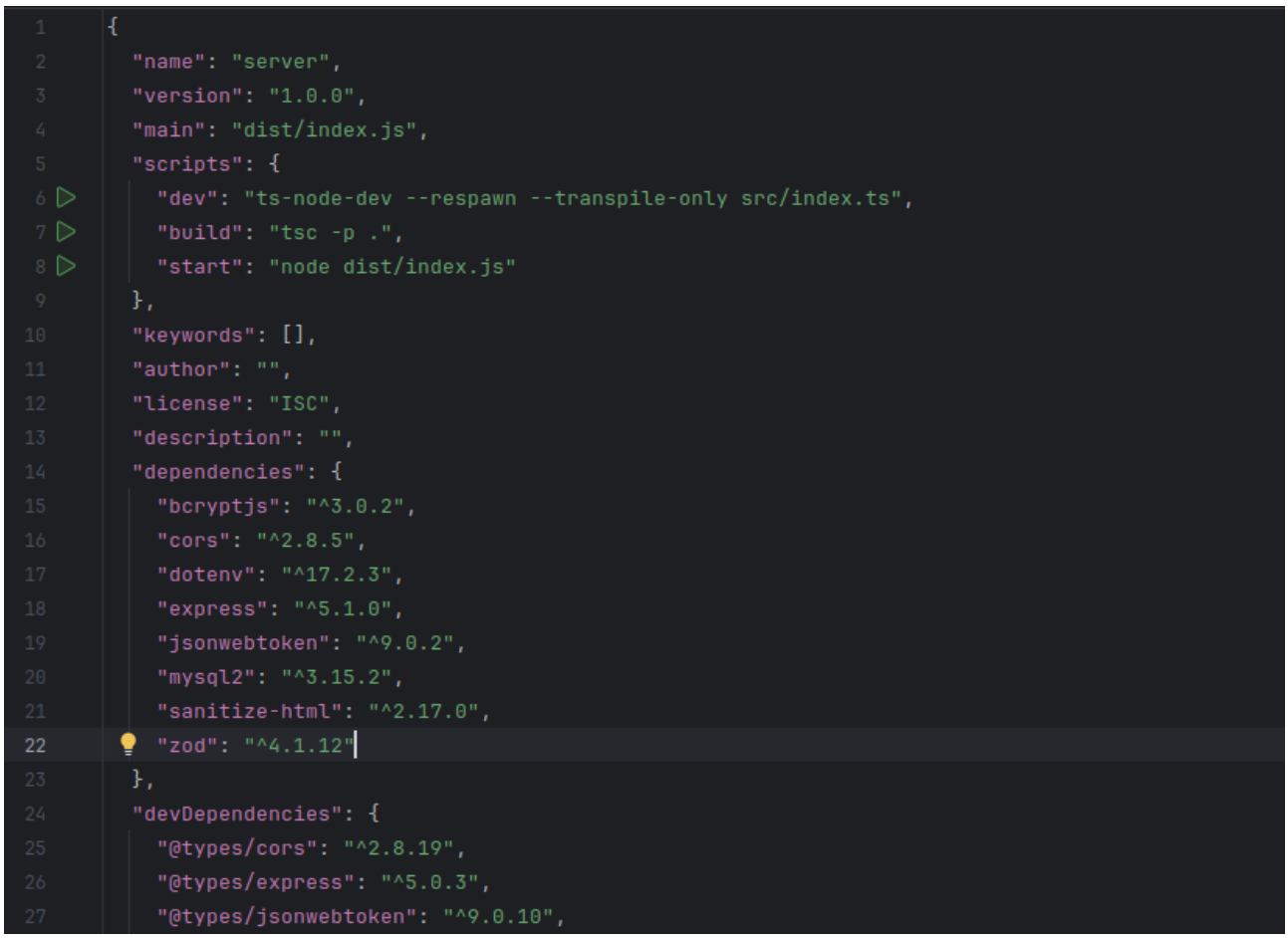


Рисунок 3.8 – Встановлення dev-залежностей

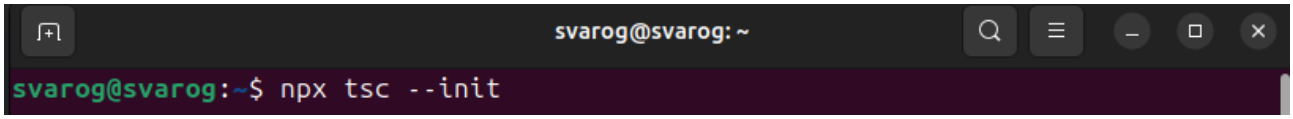
Після встановлення залежностей, формується папка node_modules, створюється файл package-lock.json, який фіксує дерево залежностей. Також версії залежностей додаються до властивостей dependencies та devDependencies файлу package.json (рис. 3.9).



```
{
  "name": "server",
  "version": "1.0.0",
  "main": "dist/index.js",
  "scripts": {
    "dev": "ts-node-dev --respawn --transpile-only src/index.ts",
    "build": "tsc -p .",
    "start": "node dist/index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "bcryptjs": "^3.0.2",
    "cors": "^2.8.5",
    "dotenv": "^17.2.3",
    "express": "^5.1.0",
    "jsonwebtoken": "^9.0.2",
    "mysql2": "^3.15.2",
    "sanitize-html": "^2.17.0",
    "zod": "^4.1.12"
  },
  "devDependencies": {
    "@types/cors": "^2.8.19",
    "@types/express": "^5.0.3",
    "@types/jsonwebtoken": "^9.0.10",
```

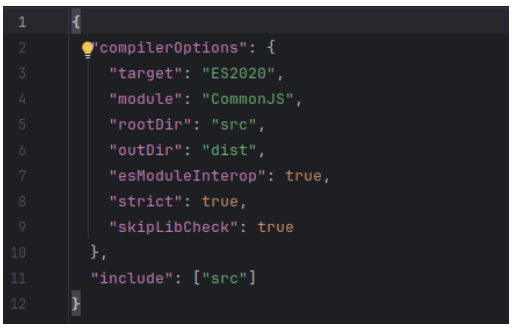
Рисунок 3.9 – Залежності

Далі створюємо файл tsconfig.json, який містить налаштування щодо typescript, за допомогою наступної команди npx tsc --init (рис. 3.10) [6, 10]. Змінюємо базовий код файлу під наш проєкт (рис. 3.11).



```
svarog@svarog: ~
svarog@svarog:~$ npx tsc --init
```

Рисунок 3.10 – Створення tsconfig.json



```
{
  "compilerOptions": {
    "target": "ES2020",
    "module": "CommonJS",
    "rootDir": "src",
    "outDir": "dist",
    "esModuleInterop": true,
    "strict": true,
    "skipLibCheck": true
  },
  "include": ["src"]
}
```

Рисунок 3.11 – Файл tsconfig.json

Далі створюємо файл оточення .env, який буде містити змінні для підключення до БД, порт на якому запускається сервер, а також JWT-токен та пов’язані з ним змінні (рис. 3.12).

```
APP_PORT=3000
CORS_ORIGIN=http://localhost:4200

MYSQL_HOST=127.0.0.1
MYSQL_PORT=3306
MYSQL_DB=online_courses
MYSQL_USER=root
MYSQL_PASSWORD=root

JWT_SECRET=52961142ae3449222dc4166c664c3b893f0cde9b5cca44597c2c714b84339b59
JWT_EXPIRES_IN=7d
```

Рисунок 3.12 – Файл .env

Для зручного доступу до змінних оточення створимо файл config/env.ts, саме його будемо імпортувати у ті файли, де потрібні змінні оточення (рис. 3.13).

```
import 'dotenv/config';

export const env = {  Show usages
    port: parseInt(process.env.APP_PORT || '3000', 10),
    corsOrigin: process.env.CORS_ORIGIN || 'http://localhost:4200',
    mysql: {
        host: process.env.MYSQL_HOST || '127.0.0.1',
        port: parseInt(process.env.MYSQL_PORT || '3306', 10),
        database: process.env.MYSQL_DB || 'online_courses',
        user: process.env.MYSQL_USER || 'root',
        password: process.env.MYSQL_PASSWORD || ''
    },
    jwt: {
        secret: process.env.JWT_SECRET || 'devsecret',
        expiresIn: process.env.JWT_EXPIRES_IN || '7d'
    }
};
```

Рисунок 3.13 – Файл config/env.ts

Далі створимо файл db/pool.ts, який буде відповідати за підключення до бази даних (рис. 3.14).

```
import mysql from 'mysql2/promise';
import { env } from '../config/env';

export const pool :Pool  = mysql.createPool({  Show usages
    host: env.mysql.host,
    port: env.mysql.port,
    user: env.mysql.user,
    password: env.mysql.password,
    database: env.mysql.database,
    waitForConnections: true,
    connectionLimit: 10,
    namedPlaceholders: true
});
```

Рисунок 3.14 – Файл db/pool.ts

Створимо файл app.ts, який буде відповідати за підключення маршрутів, middleware та express (рис. 3.15). Точкою доступу серверу є файл server.ts, також він підключає файл app.ts, що дозволяє не писати у ньому зайвий код (рис. 3.16) [6, 10].

```
import express from 'express';
import cors from 'cors';
import { env } from './config/env';
import { errorHandler } from './middlewares/error.middleware';
import authRoutes from './modules/auth/auth.routes';
import coursesRoutes from './modules/courses/courses.routes';
import enrollmentsRoutes from './modules/enrollments/enrollments.routes';
import courseItemsRoutes from './modules/course-items/course-items.routes';
import sharesRoutes from './modules/shares/shares.routes';

const app :Express  = express();
app.use(cors({ origin: env.corsOrigin, credentials: true }));
app.use(express.json());

app.get('/api/health', (_req :Request<{}, any, any, ParsedQs, Record<st... , res :Response<any, Record<string, any>, number... ) => res.json({ ok: true }));

app.use('/api/auth', authRoutes);
app.use('/api/courses', coursesRoutes);
app.use('/api/enrollments', enrollmentsRoutes);
app.use('/api/courses', courseItemsRoutes);
app.use('/api/shares', sharesRoutes);

app.use(errorHandler);
export default app;  Show usages
```

Рисунок 3.15 – Файл app.ts

```
import app from './app';
import { env } from './config/env';

app.listen(env.port, () :void  => {
    console.log(`API listening on http://localhost:${env.port}`);
});
```

Рисунок 3.16 – Файл server.ts

Так як сервер буде мати модульну структуру, то необхідно створити папку modules, а також папки для зберігання типів, utils та middleware (рис. 3.17).

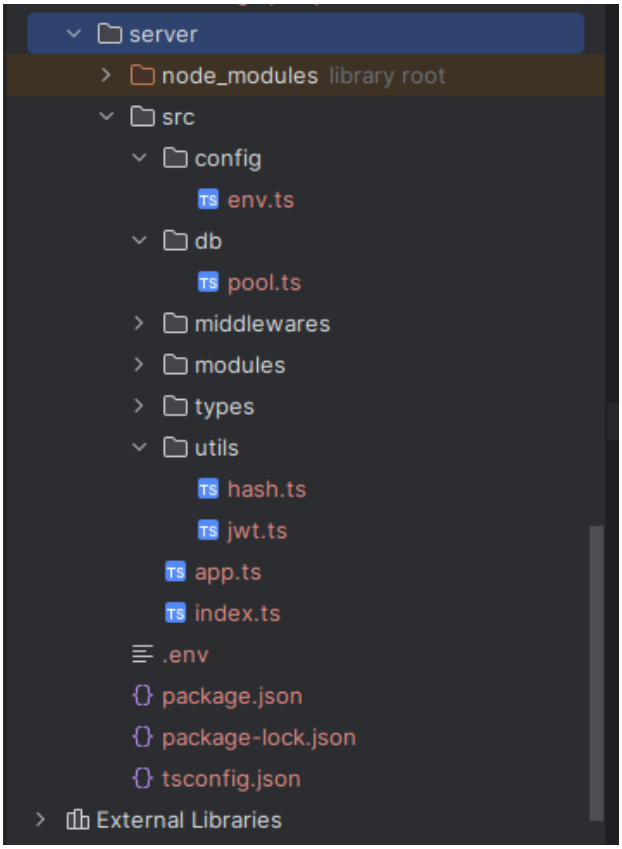


Рисунок 3.17 – Структура серверу

3.4 Опис API

У системі реалізовано REST API, яке забезпечує взаємодію між клієнтською та серверною частинами через HTTP-запити. API використовує методи HTTP POST, GET, PUT та DELETE для CRUD-операцій. Всі запити обробляються через фреймворк Express.js, а дані зберігаються в базі даних MySQL.

Розглянемо основні API-endpoints.

POST /api/courses відповідає за створення нового курсу. Приймає JSON-об’єкт з параметрами курсу (назва, опис, зображення, доступність тощо). Відповідь містить інформацію про створений курс.

GET /api/courses/:id відповідає за отримання деталей курсу за його ID. Приймає параметр id і повертає JSON-об’єкт з інформацією про курс і його елементи.

GET /api/courses/:id/items відповідає за отримання елементів курсу за його ID. Відповідь містить список елементів (відео, аудіо, медіа, HTML).

POST /api/courses/:id/items відповідає за створення елемента курсу (відео, аудіо, медіа, HTML тощо).

PUT /api/courses/:id відповідає за оновлення даних курсу.

DELETE /api/courses/:id відповідає за видалення курсу.

Структура API описана за допомогою Swagger UI, що надає можливість переглядати маршрути, тестувати запити та перевіряти валідацію вхідних даних.

Приклад опису API у Swagger наведено на рисунку 3.18.

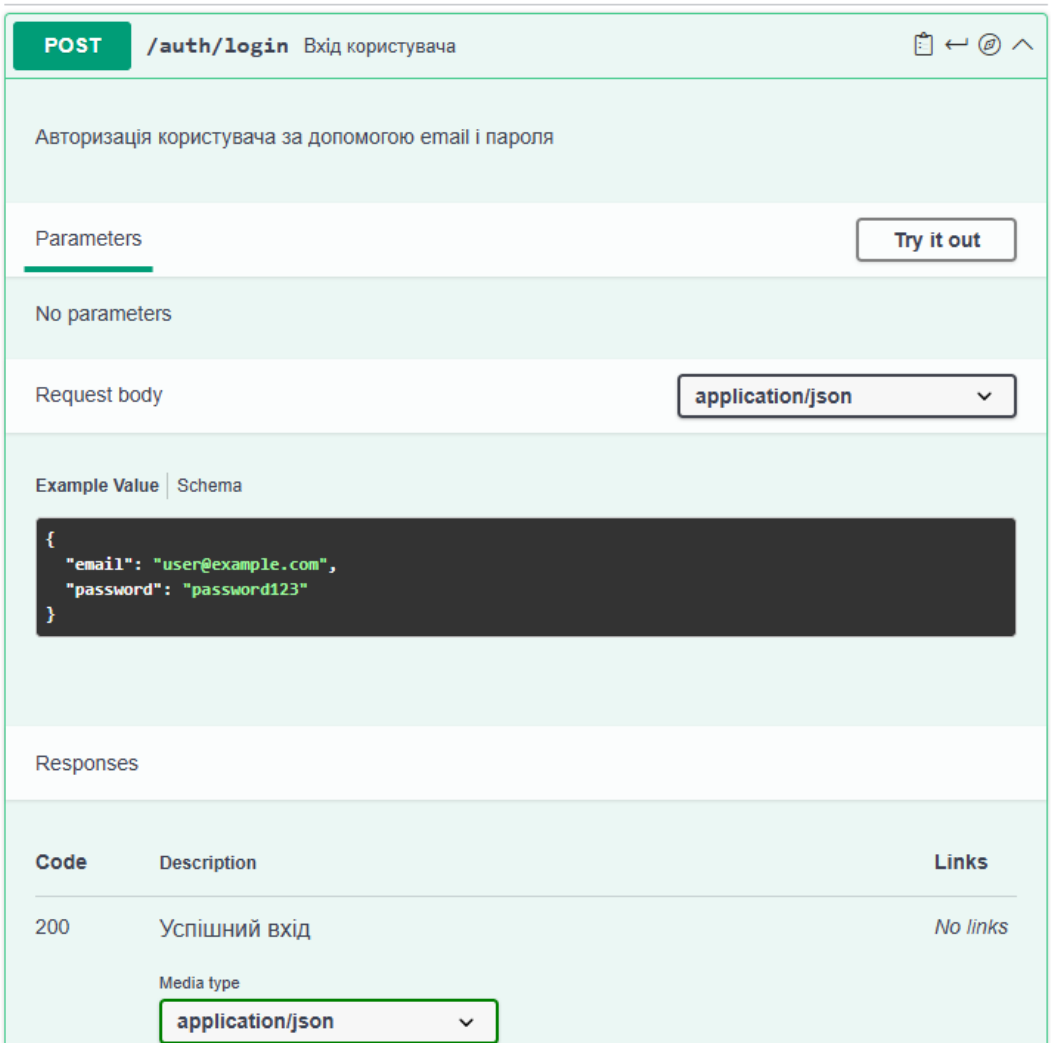


Рисунок 3.18 – Опис API у Swagger

Детально з описом API можна ознайомитися за посиланням https://app.swaggerhub.com/apis/znu-3ce/online-service/1.0.0.

3.5 Тестування проєкту

У ході створення системи було проведено unit-тестування, з метою перевірки цілісності основної бізнес-логіки, надійність взаємодії з API та зручності користувацького інтерфейсу.

Для тестування використовувалися інтегровані в Angular CLI фреймворки тестування Jasmine та Karma [1, 3]. Тестування мало дві мети: по-перше, перевірити валідність форм, а по-друге перевірити взаємодію з CoursesService.

На рисунку 3.19 наведено приклад тестування поля «Email».

```
it('should enforce required and email format for email field', () => {
  const email = component.form.controls['email'];

  email.setValue('');
  expect(email.errors?.['required']).toBeTrue();

  email.setValue('not-an-email');
  expect(email.errors?.['email']).toBeTrue();

  email.setValue('test@example.com');
  expect(email.errors).toBeNull();
});
```

Рисунок 3.19 – Unit-тест LoginComponent

Тест запускаються у браузері за шлюзом 9876, для цього використовується консольна Angular CLI команда ng test. Результати тестування відкриваються в окремому вікні та мають позначки про успіх або невдачу (рис. 3.20) [1, 3].



Рисунок 3.20 – Результат тестування

Тестування дозволяє виявити помилки в бізнес-логіці, а також забезпечити стабільну взаємодію з сервером і коректну відображувану інформацію [3].

Детально ознайомитися з тестами LoginComponent можна в додатку Д.

3.6 Керівництво користувача

3.6.1 Вхід та реєстрація

Для реєстрації на веб-платформі користувачу необхідно перейти за її адресою в браузері. Після завантаження сторінки автоматично відкриється форма входу до системи (рис. 3.21).

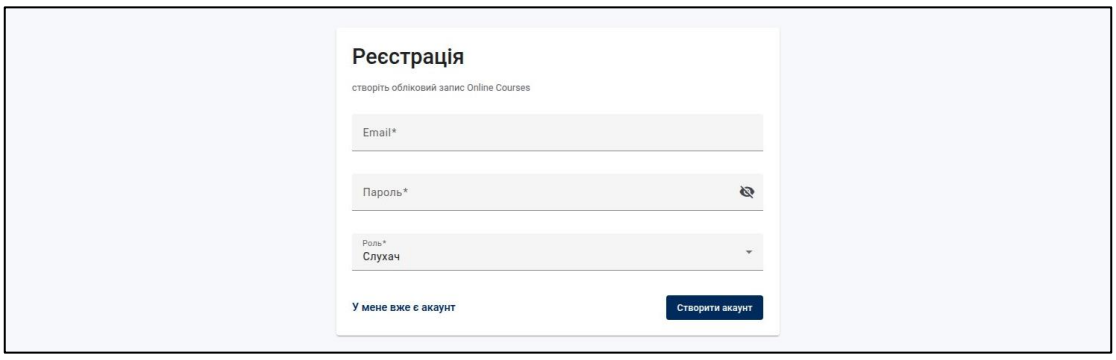


Рисунок 3.21 – Форма входу до системи

Якщо користувача не зареєстровано, то необхідно натиснути на кнопку «Створити аккаунт» у лівому нижньому кутку форми входу. Система відобразить сторінку реєстрації (рис. 3.22).

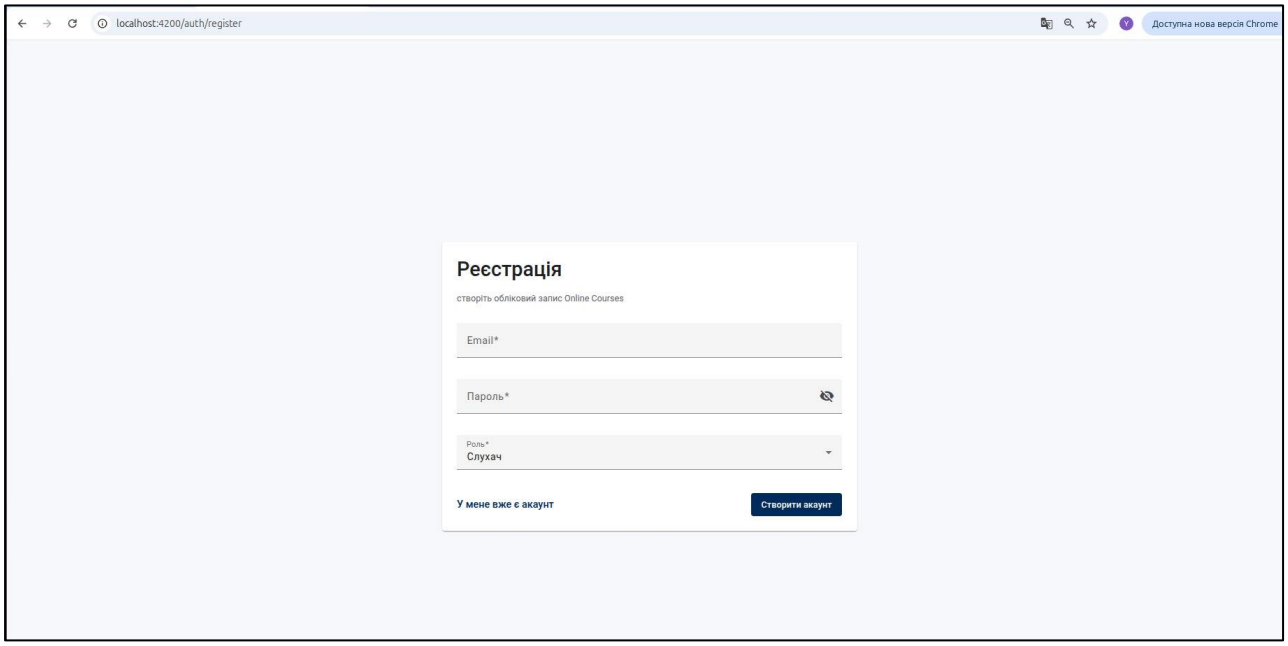


Рисунок 3.22 – Форма реєстрації

Користувач заповнює поля форми та натискає на кноку «Створити акаунт» (рис. 3.23).

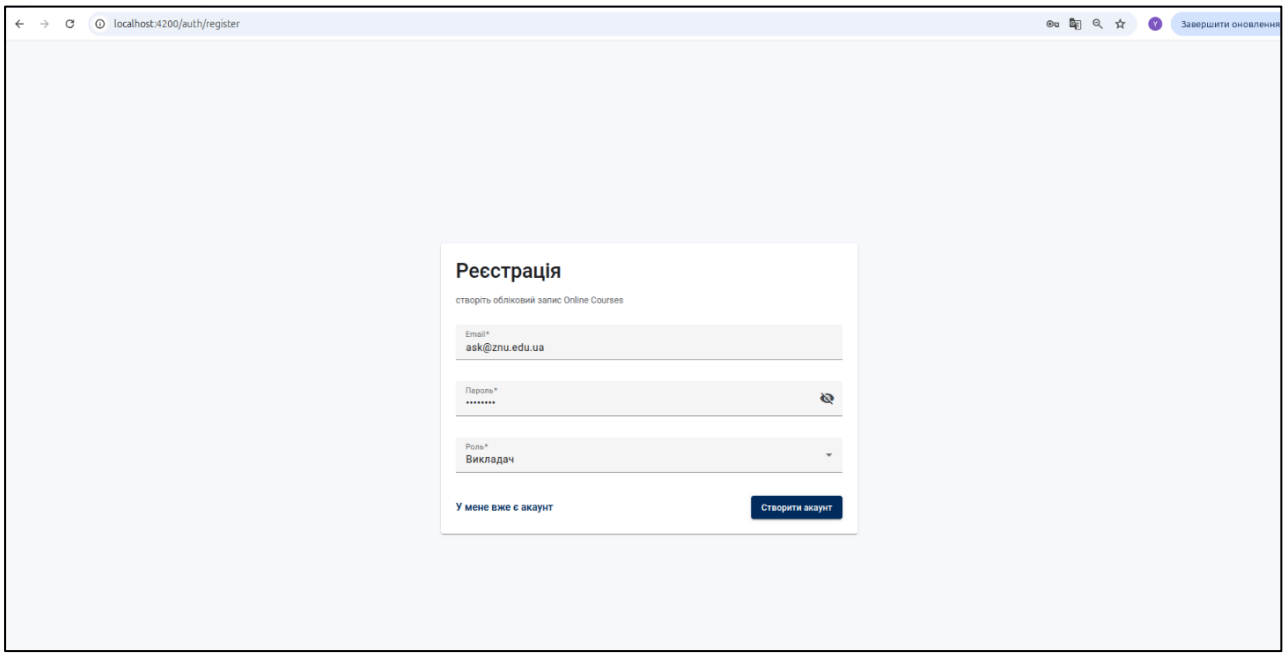


Рисунок 3.23 – Заповнена форма реєстрації

Якщо для вказаного користувачем email вже існує обліковий запис, то система повідомляє про це відповідним повідомленням (рис. 3.24).

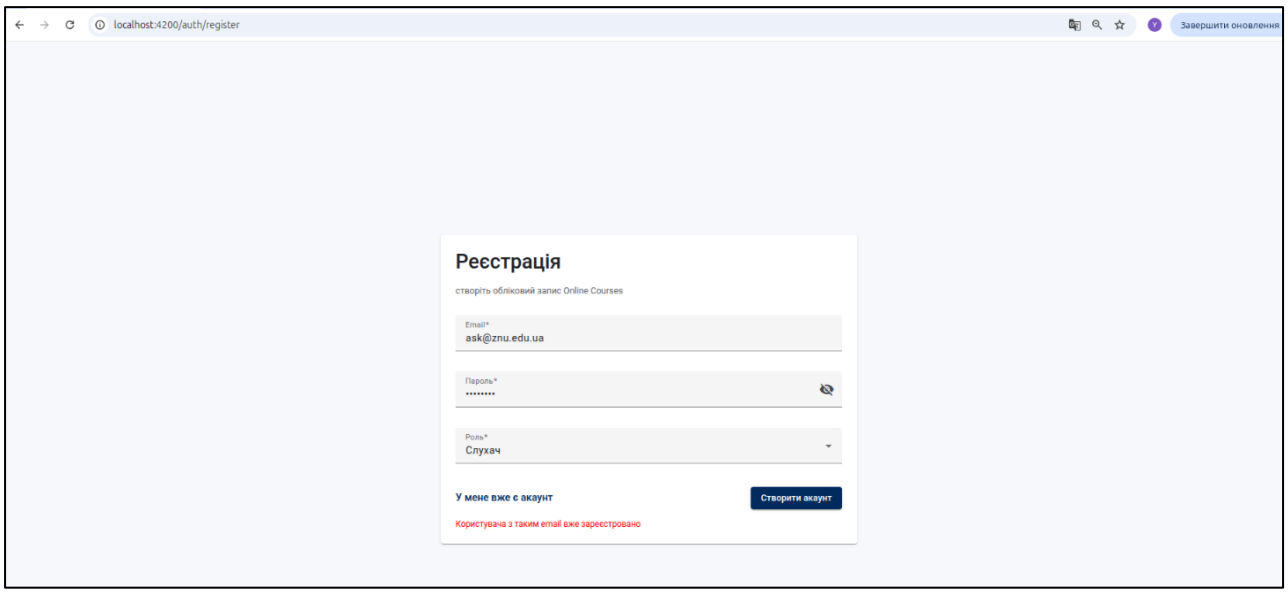


Рисунок 3.24 – Аккаунт вже існує

Якщо користувача з вказаним email не існує і поля форми заповнені правильно, то система створює новий профіль та перенаправляє на сторінку входу.

Для входу до системи користувачу необхідно ввести вказаний при реєстрації email та пароль, і натиснути на кнопку «Вхід». Якщо користувача з вказаним email не існує або пароль введено неправильно, то система видасть відповідне помилці повідомлення (див. рис. 3.25 – 3.26).

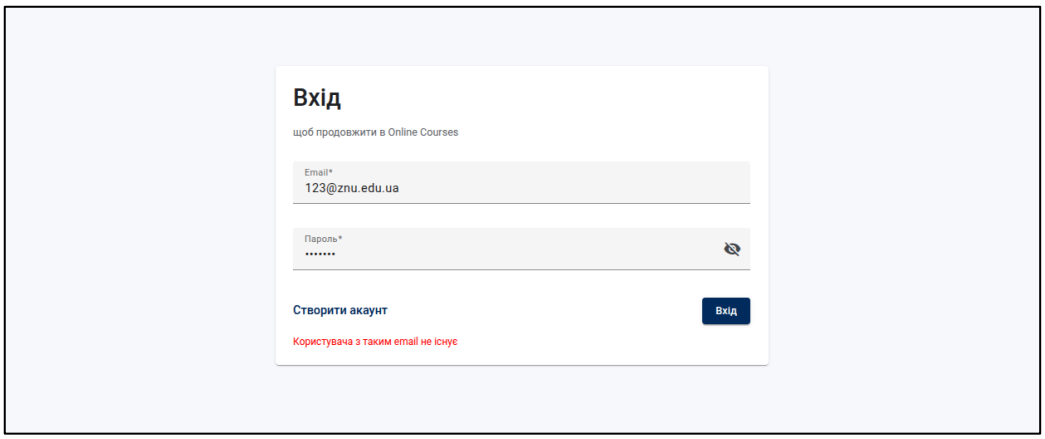


Рисунок 3.25 – Користувача з вказаним email не існує

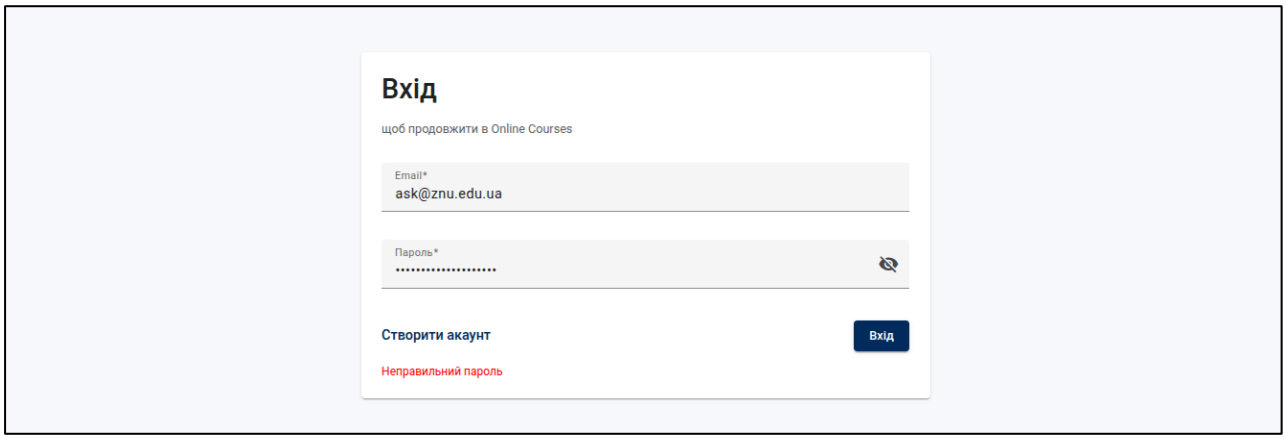


Рисунок 3.26 – Неправильний пароль

Також, при заповнені користувачем полів форми, система автоматично перевіряє валідність введених значень та повідомляє користувача у разі помилки (див. рис. 3.27 – 3.28).

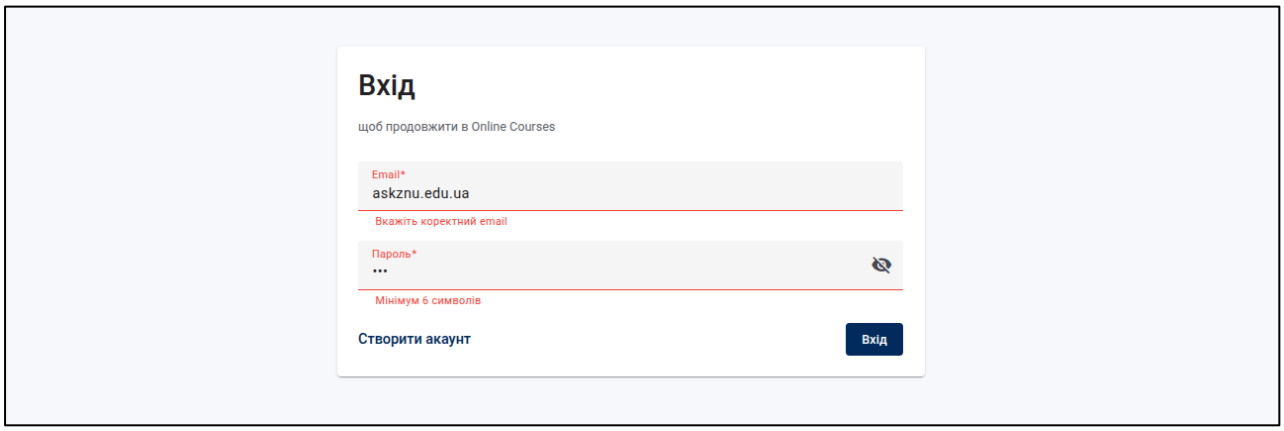


Рисунок 3.27 – Валідація форми входу

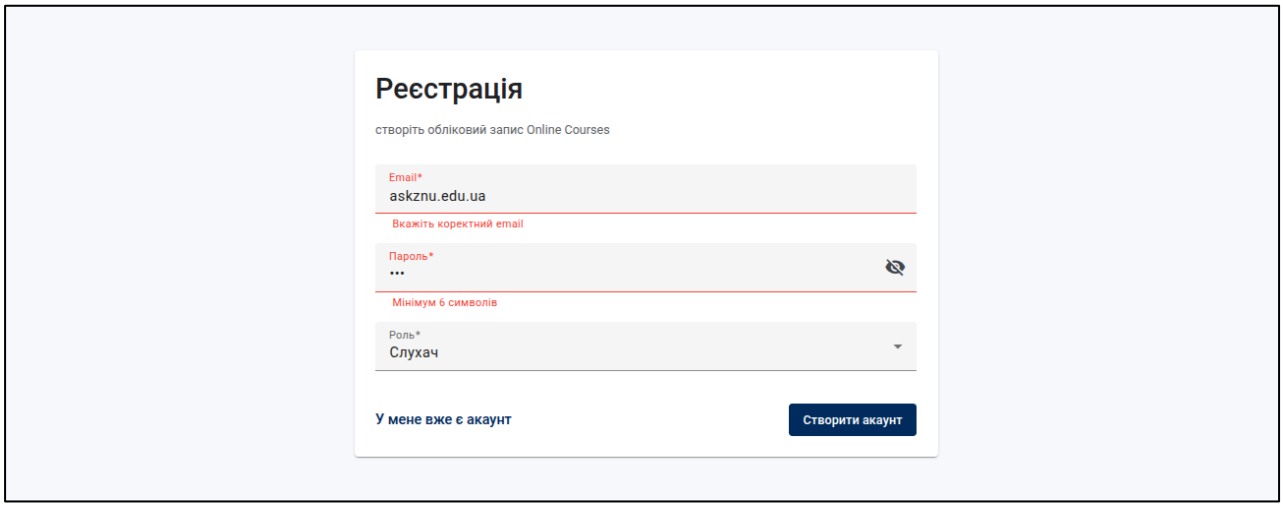


Рисунок 3.28 – Валідація форми реєстрації

Якщо дані, введені у поля форми валідні, система підсвічує поле зеленою рискою знизу (рис. 3.29).

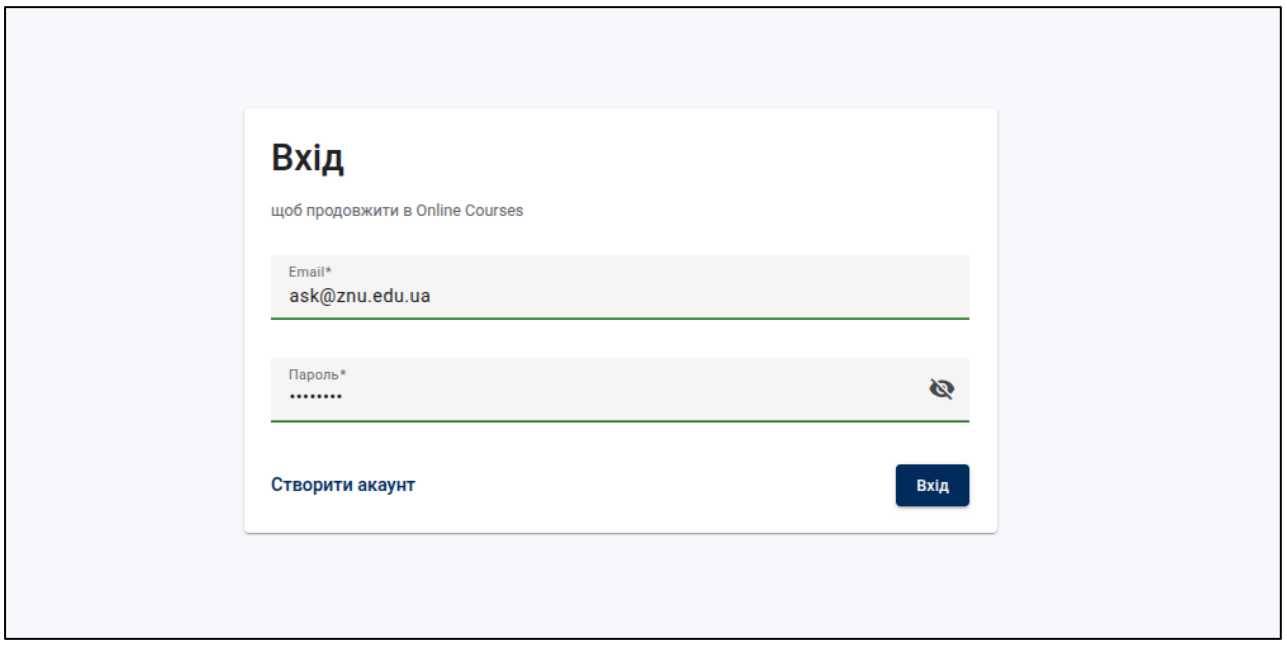


Рисунок 3.29 – Валідна форма входу

Якщо всі поля валідні та користувача зареєстровано у системі, то відкривається сторінка «Мої курси» (рис. 3.30).



Рисунок 3.30 – Вкладка «Мої курси»

3.6.2 Взаємодія з кабінетом

Після входу до особистого кабінету, користувач може взаємодіяти з вкладками «Мої курси» (відображення залежить від ролі), «Доступні курси» та з меню профіля у правому верхньому кутку (рис. 3.31).



Рисунок 3.31 – Особистий кабінет

У вкладках «Мої курси» та «Доступні курси» для всіх користувачів доступна фільтрація та пошук. Для викладачів доступна кнопка «Створити новий курс».

3.6.3 Взаємодія з курсом

Для сворення нового курсу, користувачу з правами «Викладач» необхідно натиснути на кнопку «Створити новий курс» та заповнити форму, що відобразилась (рис. 3.32).

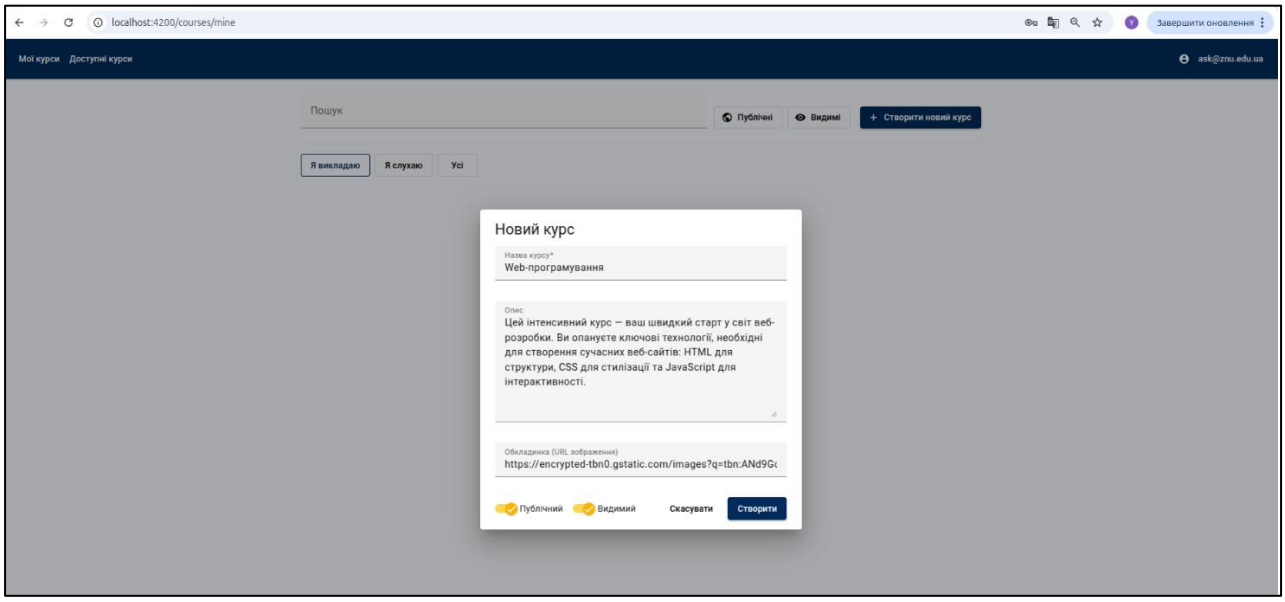


Рисунок 3.32 – Форма створення нового курсу

Форма має простий та інтуїтивно-зрозумілий інтерфейс. Користувачу необхідно тільки заповнити поля назви, опису та обрати параметри

відображення/доступу. Зображення картки курсу є опціональним параметром, і якщо користувач його не вказує, то підтягується системне значення.

Після створення курсу, його картка відображається у вкладці «Мої курси» (рис. 3.33), а також за умови його публічності – «Доступні курси».

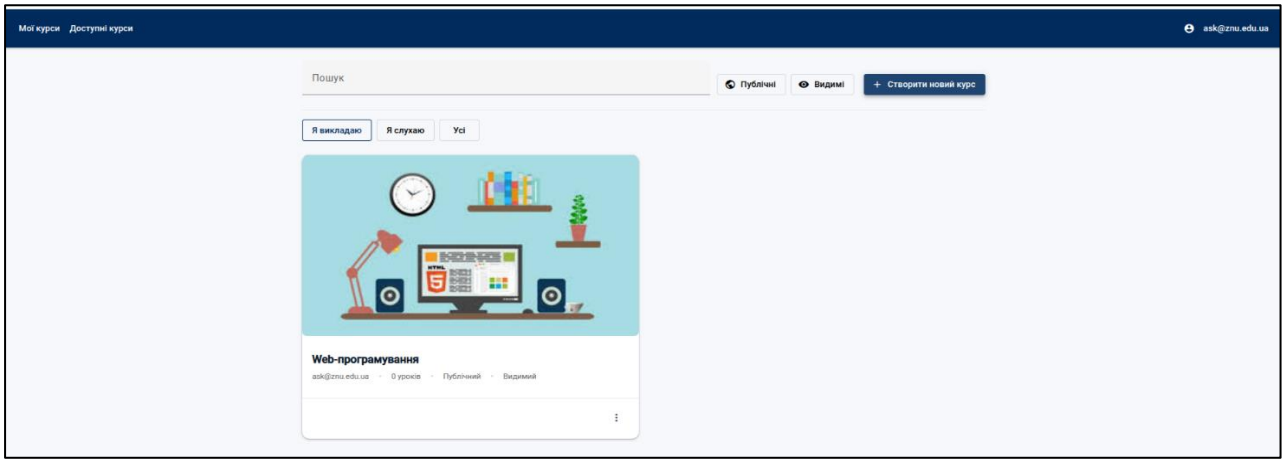


Рисунок 3.33 – Картка курсу

Кожна карткаа курсу має зображення, назву курсу, автора, кулькість активностей, публічність та доступ. Взаємодіяти з карткою курсу можна через меню, яке викликається по натисканню піктограми «…» у нижньому правому кутку (рис. 3.34).

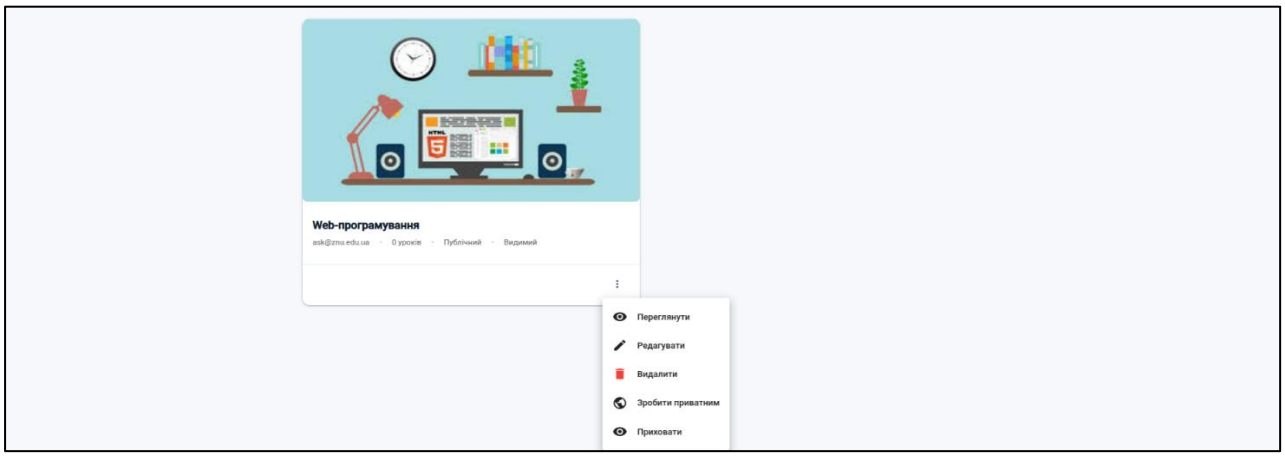


Рисунок 3.34 – Меню картки

Також доступна функція «Переглянути» при наведені на зображення картки (рис. 3.35).

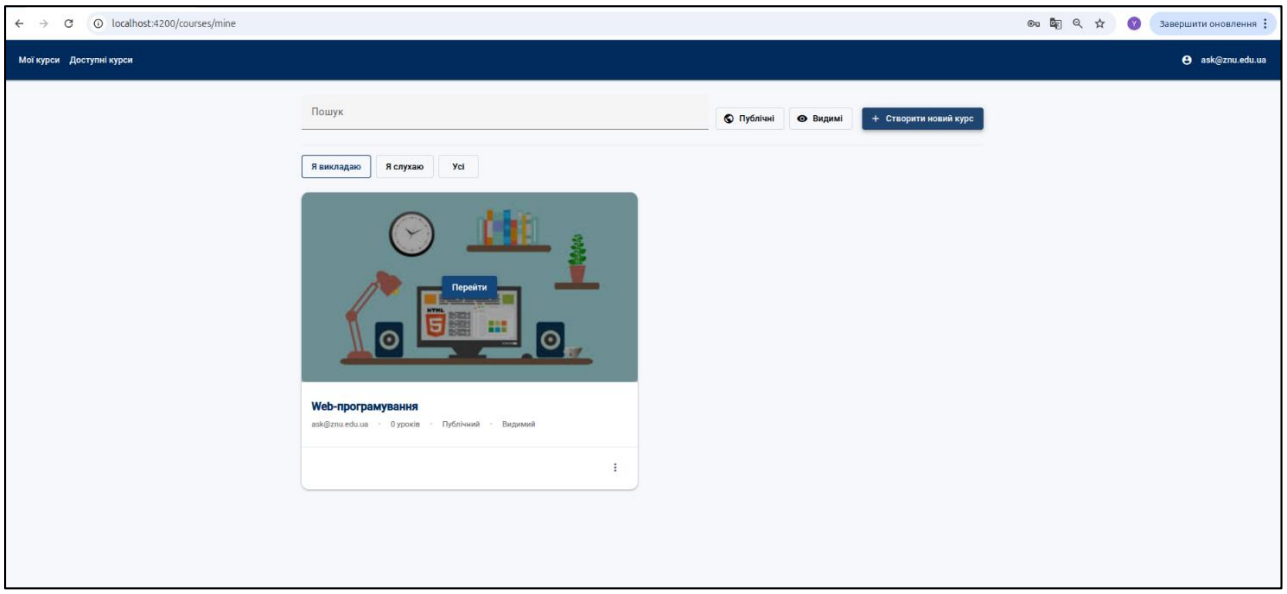


Рисунок 3.35 – Наведення на зображення картки

Для переходу на сторінку курсу, користувач може скористатися одним із наведених вище способів обравши пункт «Переглянути». Система відкриває сторінку курсу, яка за замовчуванням має назву та опис, а також кнопки дій за умови, що поточний користувач – автор курсу (рис. 3.36).

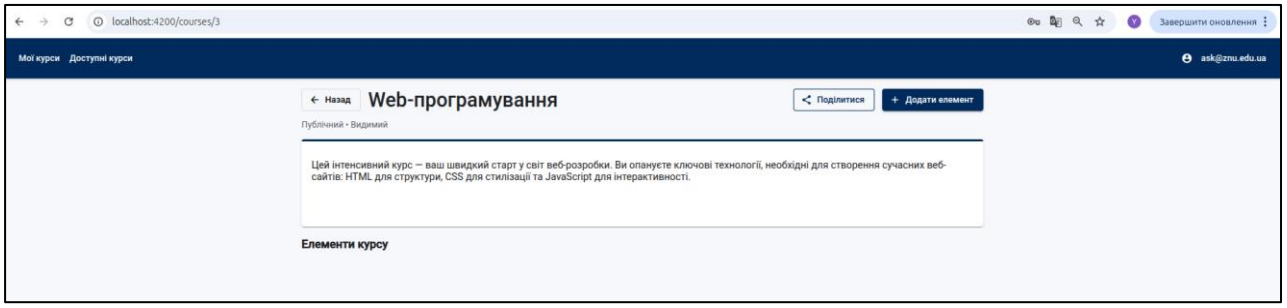


Рисунок 3.36 – Сторінка нового курсу

3.6.4 Взаємодія з елементами курсу

Для створення нового елементу курсу, користувачу з правами «Викладач», необхідно натиснути на кнопку «Додати елемент». Відкриється форма із можливісттю вибору типу контенту, назви та відповідного до типу вмісту. Також можна керувати порядком появи на сторінці та відображенням (рис. 3.37).

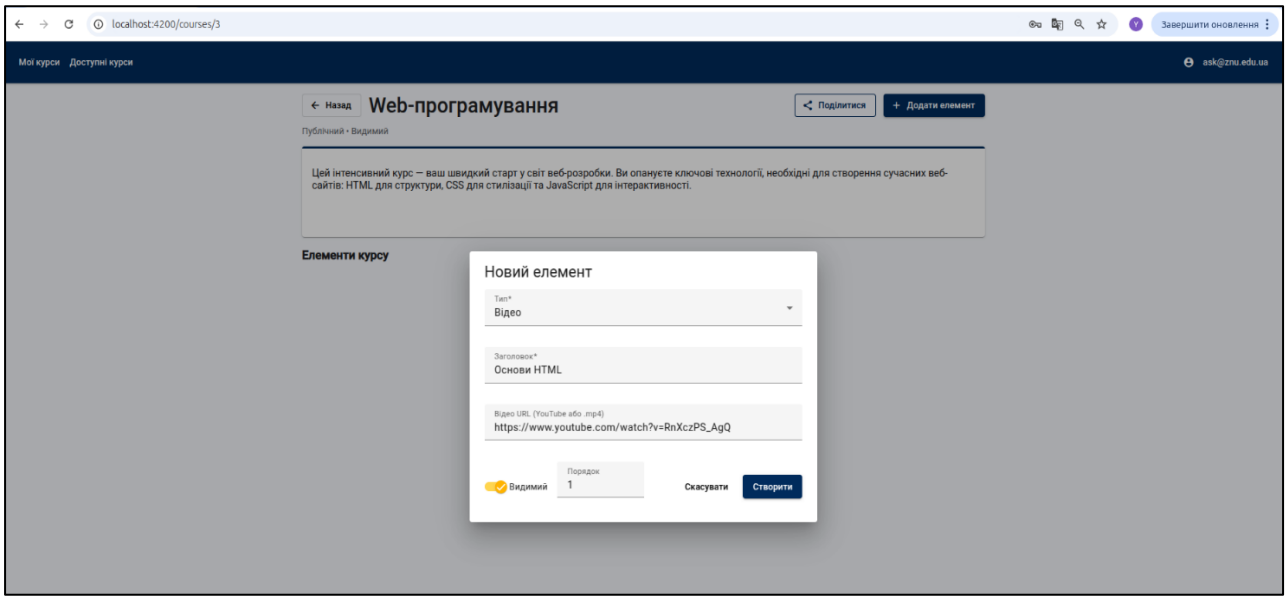


Рисунок 3.37 – Форма створення нового елементу курсу

Після створення відповідний елемент відображається на сторінці курсу (рис. 3.38).

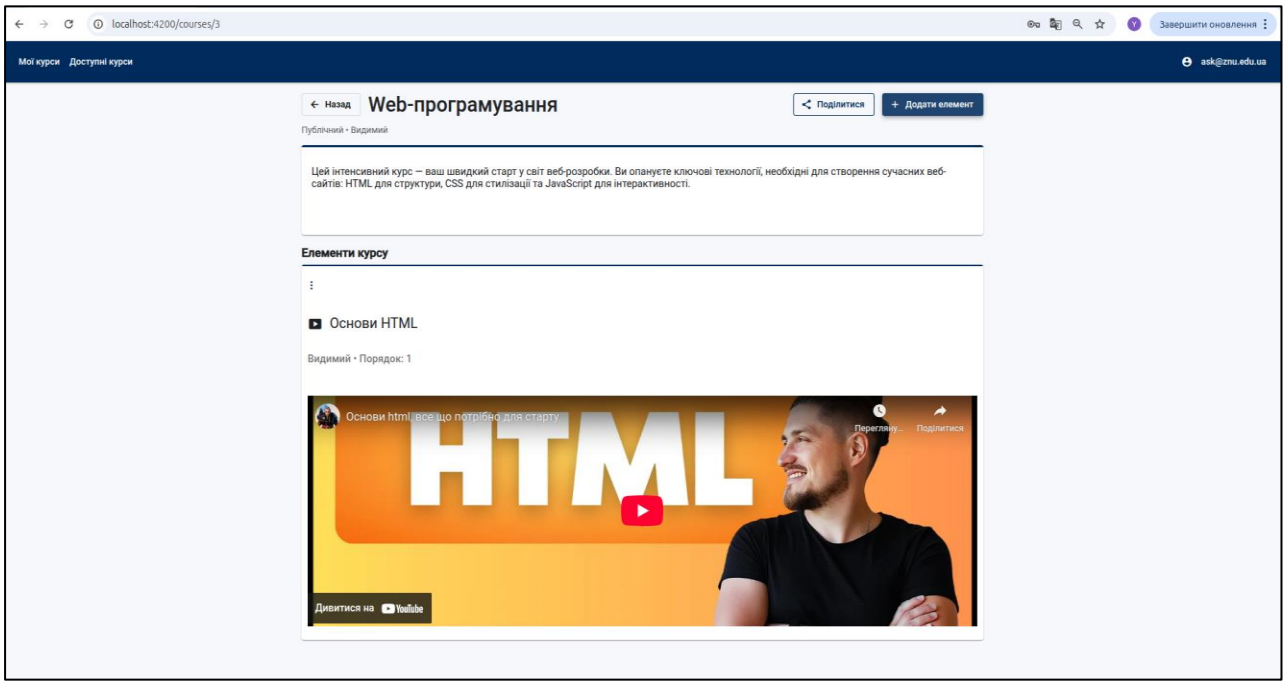


Рисунок 3.38 – Елемент курсу «Відео»

Якщо поточний користувач – автор курсу, то система надає можливість редагувати дані елементу або видалити його. Для цього необхідно натиснути на піктограму «…» у лівому верхньому кутку елементу (рис. 3.39).

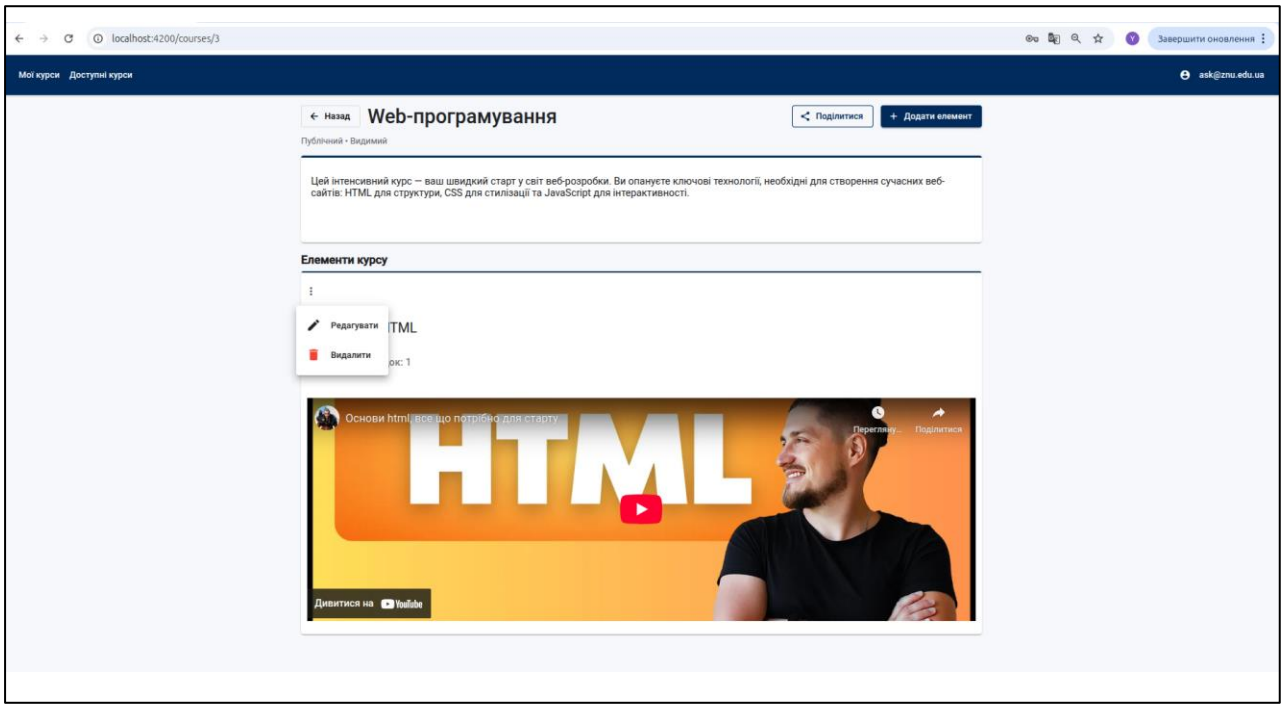


Рисунок 3.39 – Взаємодія з елементом курсу

Також на сторінку курсу можна вбудовувати інші інтерактивні елементи, наприклад Google Docs або презентацію Prezi. Для цього користувачу необхідно обрати тип «HTML-блок» та вставити у нього відповідний iframe (див. рис. 3.40 – 3.41).

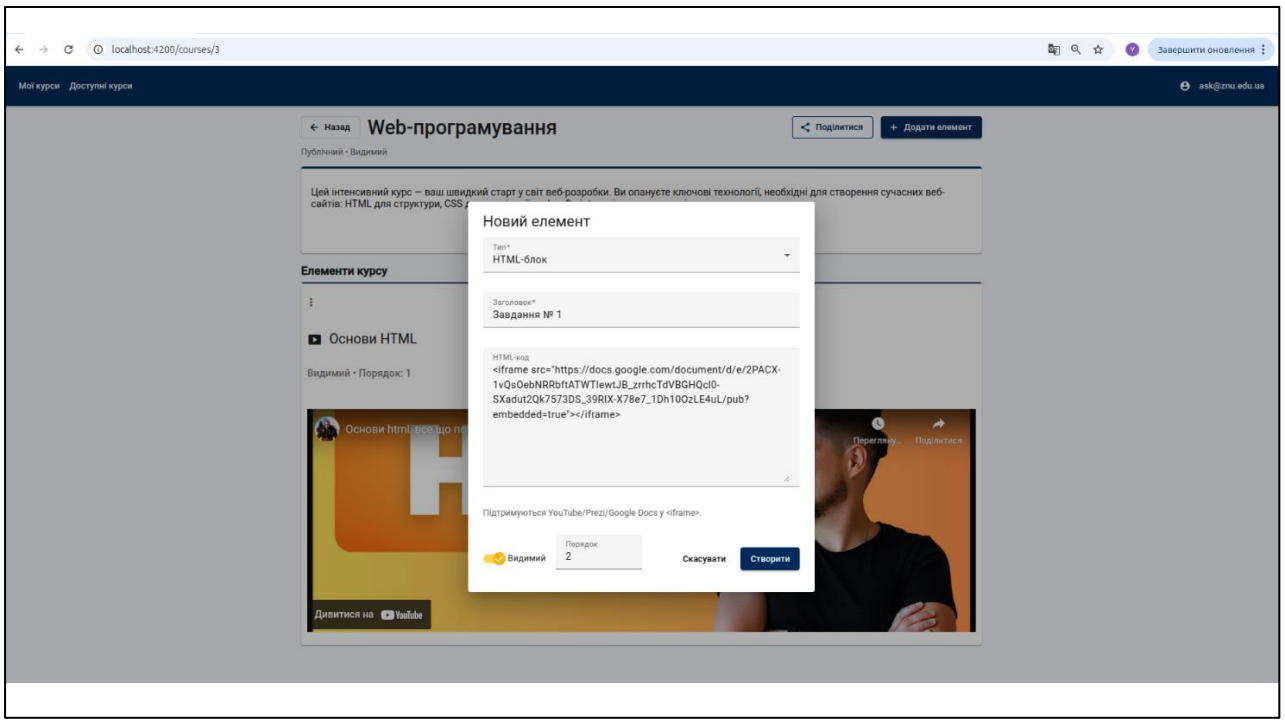


Рисунок 3.40 – Створення «HTML-блок» для Google Docs

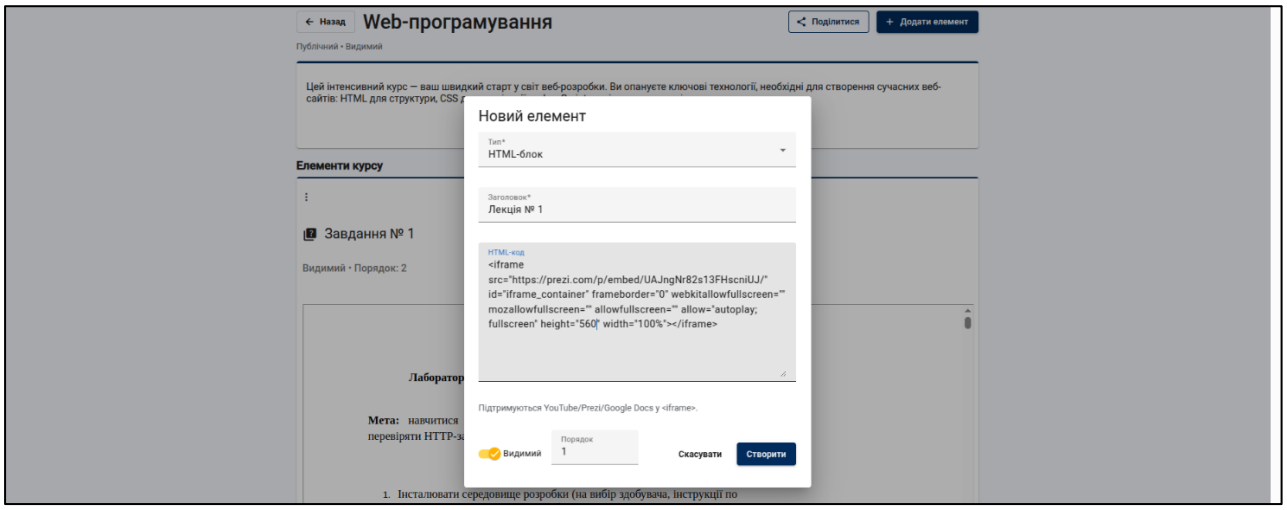


Рисунок 3.41 – Створення «HTML-блок» для Prezi

Після створення відповідних елементів, користувач може взаємодіяти з ними так, як би він відкрив їх через відповідний сервіс (див. рис. 3.42 – 3.43).

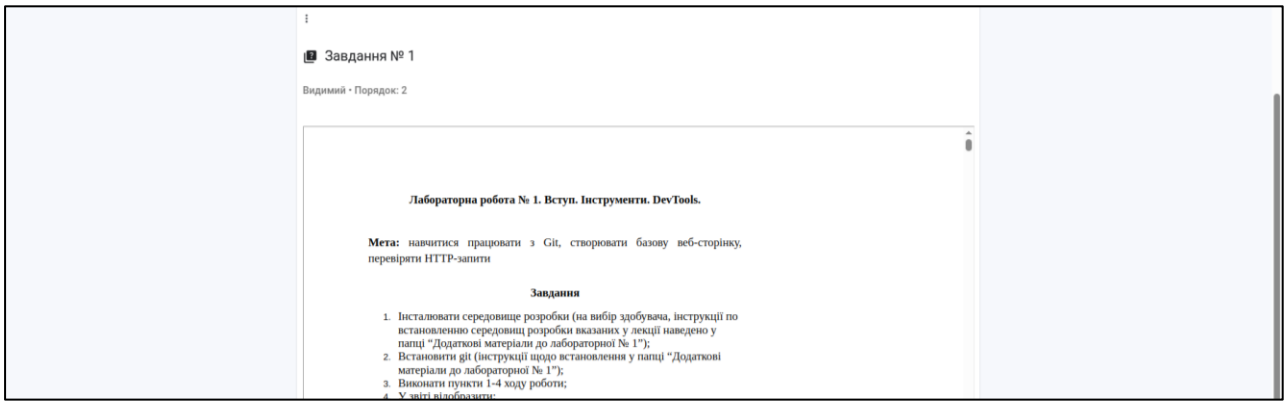


Рисунок 3.42 – Вбудований Google Docs

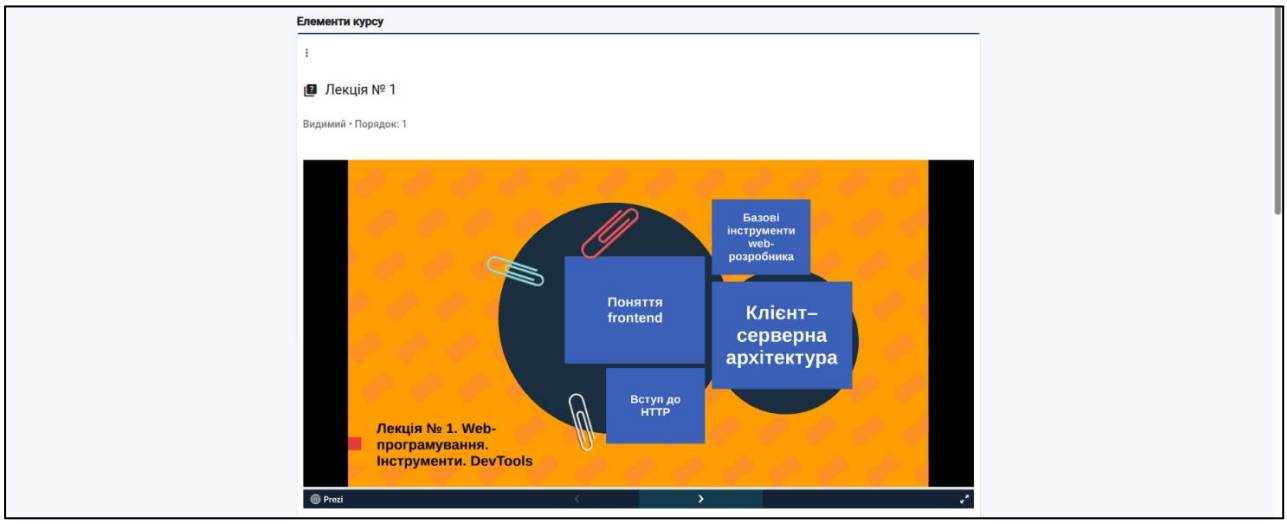


Рисунок 3.43 – Вбудована презентація Prezi

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено веб-платформу онлайн курсів. Система надає можливість легко та швидко створити курс та наповнити його різним інтерактивним, текстовим та мультемедійним контентом. Клієнт-серверна архітектура та модульність, дозволили побудувати легкопідтримувану, масштабовану та гнучку платформу.

У відповідності до мети кваліфікаційної роботи було розроблено веб-платформу онлайн курсів з використанням наступних технологій:

– Angular з боку клієнтської частини;
– Node.js + Express з боку серверної частини та реалізації REST API;
– СКБД MySQL для зберігання даних.

У відповідності до поставлених задач були виконані наступні етапи створення платформи:

– сформовані вимоги до платформи (функціональні/нефункціональні), а також наведено аналіз предметної області та системи;
– проаналізовані, описані та порівняні аналоги, надано аналіз та висновки;
– спроєктувана та побудована архітектура платформи (створено наступні UML-діаграми: варіантів використання, діяльності, послідовності, розгортання; наданий розгорнутий опис прецедентів);
– реалізовано веб-платформу онлайн курсів (надано детальне керівництво користувача, опис основних API-endpoints, інструкція по врозгортанню оточнення, описані інструменти розробки та основні класи);
– протестовано роботу платформи.

ПЕРЕЛІК ПОСИЛАНЬ

1. Angular Developer Documentation. URL: https://angular.io/docs/ (дата звернення: 21.05.2025).

2. Angular Material Guide. URL: https://material.angular.dev/guides (дата звернення: 02.07.2025).

3. Bampakos A., Deeleman P. Learning Angular: A no-nonsense guide to building web applications with Angular 15. Birmingham : Packt Publishing, 2023. 446 p.

4. Chebbi L. Reactive Patterns with RxJS for Angular: A practical guide to managing your Angular application’s data reactively and efficiently using RxJS 7. Birmingham : Packt Publishing, 2022. 224 p.

5. Express.js Guide. URL: https://expressjs.com/ (дата звернення: 14.08.2025).

6. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. Addison-Wesley Professional, 2003. 208 p.

7. Google Classroom. URL: https://classroom.google.com/ (дата звернення: 12.05.2025).

8. LMS Moodle Documentaion. URL: https://docs.moodle.org/500/en/Main_page (дата звернення: 12.05.2025).

9. Mrzyglocki F. UML Summarized: Key Concepts and Diagrams for Software Engineers, Architects, and Designers. Phillip Mrzyglocki, 2023. 59 p.

10. MySQL Documentation. URL: https://dev.mysql.com/doc/ (дата звернення: 15.07.2025).

11. Node.js Documentation. URL: https://nodejs.org/docs/latest/api/ (дата звернення: 14.08.2025).

12. Npm Docs. URL: https://docs.npmjs.com/ (дата звернення: 14.08.2025).

13. Npm jsonwebtoken. URL: https://www.npmjs.com/package/jsonwebtoken (дата звернення: 14.08.2025).

14. Open edX Documentaion. URL: https://openedx.org/community/documentation/ (дата звернення: 12.05.2025).

15. Postman. API documentation. URL: https://www.postman.com/api-platform/api-documentation/ (дата звернення: 14.08.2025).

16. RxJS Documentation. URL: https://rxjs.dev/api (дата звернення: 04.09.2025).

17. Visual Paradigm. What is Unified Modeling Language (UML)? URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/ (дата звернення: 07.09.2025).

18. Visual Paradigm. Use Case Diagram Guide. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/ (дата звернення: 07.09.2025).

19. Udemy. URL: https://www.udemy.com/ (дата звернення: 12.05.2025).

ДОДАТОК А

Angular CourseDetailsComponent

```
import { Component, OnInit, computed, inject, signal } from '@angular/core';
import { CommonModule } from '@angular/common';
import { ActivatedRoute, Router, RouterLink } from '@angular/router';
import { CoursesService, CourseDetails, CourseItem, ItemType } from '../data/courses.service';
import { SessionService } from '../../core/services/session.service';
import { MatButtonModule } from '@angular/material/button';
import { MatIconModule } from '@angular/material/icon';
import { MatCardModule } from '@angular/material/card';
import { MatListModule } from '@angular/material/list';
import { MatDialog, MatDialogModule } from '@angular/material/dialog';
import { ItemFormDialogComponent } from '../ui/item-form-dialog.component';
import { ShareDialogComponent } from '../ui/share-dialog.component';
import {ContentEmbedComponent} from '../ui/content-embed.component';
import {MatMenuModule} from '@angular/material/menu';


@Component({
 standalone: true,
 selector: 'app-course-details',
 imports: [CommonModule, RouterLink, MatMenuModule, MatIconModule, MatButtonModule, MatIconModule, MatCardModule, MatListModule, MatDialogModule, ContentEmbedComponent],
 template: `
 <div class="page">
   <div class="container" *ngIf="vm() as v">
     <div style="display:flex;align-items:center;gap:12px;margin-bottom:12px;">
       <button mat-stroked-button routerLink="/courses/mine"><mat-icon>arrow_back</mat-icon> Назад</button>
       <h1 class="auth-title" style="margin:0;">{{ v.course.title }}</h1>
       <span class="spacer"></span>
       <button *ngIf="v.meta?.isOwner" mat-stroked-button color="primary" (click)="openShare()">
         <mat-icon>share</mat-icon> Поділитися
       </button>
       <button *ngIf="v.meta?.isOwner" mat-raised-button color="primary" class="primary-btn" (click)="openCreate()">
         <mat-icon>add</mat-icon> Додати елемент
       </button>
     </div>
     <p class="auth-subtitle" style="margin:0 0 16px;">
       {{ v.course.isPublic ? 'Публічний' : 'Приватний' }} • {{ v.course.isVisible ? 'Видимий' : 'Прихований' }}
     </p>
     <mat-card class="course" style="padding:16px;">
       <p>{{ v.course.description || 'Без опису' }}</p>
```

```
    </mat-card>
    <h3 style="margin:16px 0 8px;">Елементи курсу</h3>
    <div class="grid">
      <mat-card class="course" *ngFor="let it of v.items" style="position:relative;">
        <!-- Меню "..." -->
        <button *ngIf="v.meta?.isOwner" mat-icon-button class="item-more" [matMenuTriggerFor]="itm"
(click)="$event.stopPropagation()">
          <mat-icon>more_vert</mat-icon>
        </button>
        <mat-menu #itm="matMenu">
          <button mat-menu-item (click)="openEdit(it)"><mat-icon>edit</mat-
icon><span>Редагувати</span></button>
          <button mat-menu-item (click)="remove(it.id)"><mat-icon color="warn">delete</mat-
icon><span>Видалити</span></button>
        </mat-menu>
        <mat-card-title>
          <mat-icon style="vertical-align:middle;margin-right:6px;">{{ icon(it.type) }}</mat-icon>
          {{ it.title }}
        </mat-card-title>
        <mat-card-subtitle>
          {{ it.isVisible ? 'Видимий' : 'Прихований' }} • Порядок: {{ it.position }}
        </mat-card-subtitle>
        <mat-card-content>
          <app-content-embed [item]="it"></app-content-embed>
        </mat-card-content>
      </mat-card>
    </div>
  </div>
 </div>
 `
})
export class CourseDetailsComponent implements OnInit {
 private route = inject(ActivatedRoute);
 private api = inject(CoursesService);
 private dialog = inject(MatDialog);
 private router = inject(Router);
 vm = signal<CourseDetails | null>(null);
 ngOnInit() { this.load(); }
 load() {
   const id = Number(this.route.snapshot.paramMap.get('id'));
   this.api.getOne(id).subscribe(v => this.vm.set(v));
 }
 icon(t: ItemType){ return t==='video'?'smart_display': t==='audio'?'music_note': t==='media'?'image': 'quiz';
}
 openCreate() {
   const courseId = this.vm()!.course.id;
   const ref = this.dialog.open(ItemFormDialogComponent, { data: { mode: 'create' }, width: '560px' });
   ref.afterClosed().subscribe(v => { if (v) this.api.createItem(courseId, v).subscribe(()=>this.load()); });
 }
 openEdit(item: CourseItem) {
```

```
  const ref = this.dialog.open(ItemFormDialogComponent, { data: { mode: 'edit', item }, width: '560px' });
  ref.afterClosed().subscribe(v => { if (v) this.api.updateItem(item.id, v).subscribe(()=>this.load()); });
 }
 remove(id:number){ if(!confirm('Видалити елемент?')) return;
this.api.deleteItem(id).subscribe(()=>this.load()); }
 openShare(){
  const courseId = this.vm()!.course.id;
  this.dialog.open(ShareDialogComponent, { data: { courseId }, width: '560px' });
 }
}
```

ДОДАТОК Б

Angular CoursesService

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpParams } from '@angular/common/http';
import { environment } from '../../../environments/environment';
import { Course, CreateCourseDto, UpdateCourseDto } from './course.model';

export interface CourseDetails { course: Course; items: CourseItem[]; meta?: { isOwner:boolean;
enrolled:boolean }; }
export type ItemType = 'video'|'audio'|'media'|'quiz';
export interface CourseItem {
 id:number; courseId:number; type:ItemType; title:string;
 contentUrl?:string|null; contentJson?:any|null;
 position:number; isVisible:boolean; createdAt:string;
}

@Injectable({ providedIn: 'root' })
export class CoursesService {
 private api = `${environment.apiUrl}/courses`;
 private enr = `${environment.apiUrl}/enrollments`;
 private share = `${environment.apiUrl}/shares`;
 constructor(private http: HttpClient) {}

 getAvailable(q?: string) {
   let params = new HttpParams();
   if (q) params = params.set('q', q);
   return this.http.get<Course[]>(this.api, { params });
 }
 getMine() {
   return this.http.get<Course[]>(`${this.api}/mine`);
 }
 getEnrolled() { return this.http.get<Course[]>(`${this.enr}/mine`); }

 getOne(id:number){ return this.http.get<CourseDetails>(`${this.api}/${id}`); }

 enroll(courseId: number) { return this.http.post<{ok:true}>(`${this.api}/${courseId}/enroll`, {}); }
 unenroll(courseId: number) { return this.http.delete<{ok:true}>(`${this.api}/${courseId}/enroll`); }

 createItem(courseId:number, dto: Partial<CourseItem> & {type:ItemType; title:string}){
   return this.http.post<CourseItem>(`${this.api}/${courseId}/items`, {
     type: dto.type, title: dto.title, contentUrl: dto.contentUrl ?? null, contentJson: dto.contentJson ?? null,
     position: dto.position, isVisible: dto.isVisible
   });
 }
 updateItem(id:number, patch: Partial<CourseItem>){
```

```
  return this.http.patch<CourseItem>(`${this.api}/items/${id}`, {
    title: patch.title, type: patch.type, contentUrl: patch.contentUrl ?? null,
    contentJson: patch.contentJson ?? null, position: patch.position, isVisible: patch.isVisible
  });
 }
 deleteItem(id:number){ return this.http.delete<{ok:true}>(`${this.api}/items/${id}`); }

getShare(courseId:number){ return this.http.get<{token:string; expiresAt:string|null} |
null>(`${this.share}/course/${courseId}`); }
 createShare(courseId:number, ttlDays?:number){ return this.http.post<{token:string;
expiresAt:string|null}>(`${this.share}/course/${courseId}`, { ttlDays }); }
 revokeShare(courseId:number){ return this.http.delete<{ok:true}>(`${this.share}/course/${courseId}`); }
 getShared(token:string){ return this.http.get<CourseDetails>(`${this.share}/${token}` as any); }

 create(dto: CreateCourseDto) {
  return this.http.post<Course>(this.api, dto);
 }

 update(id: number, dto: UpdateCourseDto) {
  return this.http.patch<Course>(`${this.api}/${id}`, dto);
 }

 delete(id: number) {
  return this.http.delete<{ok:true}>(`${this.api}/${id}`);
 }
}
```

ДОДАТОК В

Express CourseController

```
import * as repo from './courses.repo';
import * as items from '../course-items/course-items.service';
import * as enrRepo from '../enrollments/enrollments.repo';


const mapRow = (r: repo.CourseRow) => ({
  id: r.id,
  title: r.title,
  description: r.description,
  image_url: r.image_url || null,
  isPublic: !!r.is_public,
  isVisible: !!r.is_visible,
  ownerId: r.owner_id,
  ownerEmail: r.owner_email || undefined,
  itemsCount: r.items_count ?? undefined,
  createdAt: r.created_at
});


export async function getPublic(q?: string) {
  const rows = await repo.listPublic(q);
  return rows.map(mapRow);
}


export async function getMine(ownerId: number) {
  const rows = await repo.listByOwner(ownerId);
  return rows.map(mapRow);
}


export async function create(ownerId: number, dto: { title:string; description?:string; image_url?:string|null;
isPublic?:boolean; isVisible?:boolean; }) {
  const id = await repo.createCourse(ownerId, {
     title: dto.title, description: dto.description, image_url: dto.image_url ?? null,
     is_public: dto.isPublic ?? false, is_visible: dto.isVisible ?? true
  } as any);
  const row = await repo.findById(id);
  return mapRow(row!);
}


export async function update(ownerId: number, id: number, patch: Partial<{title:string;
```

```
description:string|null; image_url?:string|null; isPublic:boolean; isVisible:boolean;}>) {
  console.log(patch);
  const affected = await repo.updateCourse(id, ownerId, {
     title: patch.title,
     description: patch.description ?? null,
     image_url: patch.image_url ?? null,
     is_public: patch.isPublic,
     is_visible: patch.isVisible
  } as any);
  if (!affected) throw { status: 404, message: 'Course not found or not owned by you' };
  const row = await repo.findById(id);
  return mapRow(row!);
}


export async function remove(ownerId: number, id: number) {
  const affected = await repo.deleteCourse(id, ownerId);
  if (!affected) throw { status: 404, message: 'Course not found or not owned by you' };
  return { ok: true };
}


export async function getEnrolled(userId: number) {
  const rows = await repo.listEnrolled(userId);
  return rows.map(mapRow);
}


export async function getOneForUser(userId:number, role:'listener'|'teacher', id:number) {
  const row = await repo.findById(id);
  if (!row) throw { status:404, message:'Course not found' };
  const isOwner = row.owner_id === userId;
  const enrolled = await enrRepo.isEnrolled(userId, id);


  const isAccessible = isOwner || enrolled || (!!row.is_public && !!row.is_visible);
  if (!isAccessible) throw { status:403, message:'Access denied' };


  const course = mapRow(row);
  const itemsList = isOwner ? await items.listAll(id) : await items.listVisible(id);
  return { course, items: itemsList, meta: { isOwner, enrolled } };
}
```

ДОДАТОК Г

Express CourseRepo

```
import { pool } from '../../db/pool';
import { ResultSetHeader, RowDataPacket } from 'mysql2';


export interface CourseRow {
  id:number; title:string; description:string|null; image_url:string|null;
  is_public:0|1; is_visible:0|1; owner_id:number; created_at:string;
  owner_email?:string; items_count?:number;
}


export async function listPublic(q?: string) {
  const where = ['c.is_public = 1', 'c.is_visible = 1'];
  const params: any = {};
  if (q) { where.push('(c.title LIKE :q OR c.description LIKE :q)'); params.q = `%${q}%`; }


  const [rows] = await pool.query<RowDataPacket[]>(
    `SELECT c.id, c.title, c.description, c.image_url, c.is_public, c.is_visible, c.owner_id, c.created_at,
          u.email AS owner_email,
          (SELECT COUNT(*) FROM course_items i WHERE i.course_id=c.id AND i.is_visible=1) AS
items_count
     FROM courses c
          JOIN users u ON u.id = c.owner_id
     WHERE ${where.join(' AND ')}
     ORDER BY c.created_at DESC`,
    params
  );
  return rows as CourseRow[];
}
export async function listByOwner(ownerId: number) {
  const [rows] = await pool.query<RowDataPacket[]>(
    `SELECT c.id, c.title, c.description, c.image_url, c.is_public, c.is_visible, c.owner_id, c.created_at,
          u.email AS owner_email,
          (SELECT COUNT(*) FROM course_items i WHERE i.course_id=c.id) AS items_count
     FROM courses c
          JOIN users u ON u.id = c.owner_id
     WHERE c.owner_id = :ownerId
     ORDER BY c.created_at DESC`,
    { ownerId }
  );
  return rows as CourseRow[];
}
```

```
export async function listEnrolled(userId: number) {
  const [rows] = await pool.query<RowDataPacket[]>(
    `SELECT c.id, c.title, c.description, c.is_public, c.is_visible, c.owner_id, c.created_at
   FROM enrollments e
   JOIN courses c ON c.id = e.course_id
   WHERE e.user_id = :userId
   ORDER BY c.created_at DESC`,
    { userId }
  );
  return rows as CourseRow[];
}


export async function findById(id: number) {
  const [rows] = await pool.query<RowDataPacket[]>(
    `SELECT id, title, description, is_public, is_visible, owner_id, created_at
   FROM courses WHERE id = :id LIMIT 1`, { id }
  );
  return (rows[0] as CourseRow) || null;
}


export async function createCourse(ownerId: number, data: { title:string; description?:string;
image_url?:string|null; is_public?:boolean; is_visible?:boolean; }) {
  const [res] = await pool.query<ResultSetHeader>(
    `INSERT INTO courses (title, description, image_url, is_public, is_visible, owner_id)
     VALUES (:title, :description, :image_url, :is_public, :is_visible, :owner_id)`,
    {
      title: data.title,
      description: data.description ?? null,
      image_url: data.image_url ?? null,
      is_public: data.is_public ? 1 : 0,
      is_visible: data.is_visible ?? true ? 1 : 0,
      owner_id: ownerId
    }
  );
  return res.insertId;
}


export async function updateCourse(id:number, ownerId:number, patch: Partial<{ title:string;
description:string|null; image_url:string|null; is_public:boolean; is_visible:boolean; }>) {
  const sets: string[] = []; const params: any = { id, ownerId };
  if (patch.title !== undefined) { sets.push('title = :title'); params.title = patch.title; }
  if (patch.description !== undefined) { sets.push('description = :description'); params.description =
patch.description; }
  if (patch.image_url !== undefined) { sets.push('image_url = :image_url'); params.image_url =
patch.image_url; }
  if (patch.is_public !== undefined) { sets.push('is_public = :is_public'); params.is_public = patch.is_public ?
1 : 0; }
```

```
  if (patch.is_visible !== undefined) { sets.push('is_visible = :is_visible'); params.is_visible =
patch.is_visible ? 1 : 0; }
  if (!sets.length) return 0;
  console.log(patch);
  const [res] = await pool.query<ResultSetHeader>(`UPDATE courses SET ${sets.join(', ')} WHERE id=:id
AND owner_id=:ownerId`, params);
  return res.affectedRows;
}


export async function deleteCourse(id: number, ownerId: number) {
  const [res] = await pool.query<ResultSetHeader>(
    `DELETE FROM courses WHERE id = :id AND owner_id = :ownerId`, { id, ownerId }
  );
  return res.affectedRows;
}
```

ДОДАТОК Д

Unit-test Login Component

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { LoginComponent } from './login.component';
import { FormBuilder } from '@angular/forms';
import { Router } from '@angular/router';
import { of } from 'rxjs';
import { RouterTestingModule } from '@angular/router/testing';
import { AuthService } from '../data/auth.service';


class MockAuthService {
 login = (credentials: any) => of({ token: 'mock-token', user: { id: 1, email: credentials.email } });
}

class MockSessionService {
 setAuth = jasmine.createSpy('setAuth');
}

describe('LoginComponent', () => {
 let component: LoginComponent;
 let fixture: ComponentFixture<LoginComponent>;
 let authService: AuthService;
 let router: Router;

 beforeEach(async () => {
   await TestBed.configureTestingModule({
     imports: [LoginComponent, RouterTestingModule.withRoutes([])],
     providers: [
       FormBuilder,
       { provide: AuthService, useClass: MockAuthService },
       { provide: SessionService, useClass: MockSessionService },
     ],
   }).compileComponents();
 });

 beforeEach(() => {
   fixture = TestBed.createComponent(LoginComponent);
   component = fixture.componentInstance;
   authService = TestBed.inject(AuthService);
   sessionService = TestBed.inject(SessionService);
   router = TestBed.inject(Router);
   spyOn(router, 'navigateByUrl');
   component.ngOnInit();
   fixture.detectChanges();
```

```
});

it('should create and initialize the form structure', () => {
  expect(component).toBeTruthy();
  expect(component.form).toBeDefined();
  expect(component.form.contains('email')).toBeTrue();
  expect(component.form.contains('password')).toBeTrue();
  expect(component.hide).toBeTrue();
});

it('should make the form invalid when fields are empty', () => {
  expect(component.form.valid).toBeFalse();
});

it('should enforce required and email format for email field', () => {
  const email = component.form.controls['email'];

  email.setValue('');
  expect(email.errors?.['required']).toBeTrue();


  email.setValue('not-an-email');
  expect(email.errors?.['email']).toBeTrue();
  email.setValue('test@example.com');
  expect(email.errors).toBeNull();
});

it('should enforce minimum length (6) for password field', () => {
  const password = component.form.controls['password'];
  password.setValue('');
  expect(password.errors?.['required']).toBeTrue()
  password.setValue('12345');
  expect(password.errors?.['minlength']).toBeTruthy();
  password.setValue('123456');
  expect(password.errors).toBeNull();
});

it('should NOT call the login API if the form is invalid on submit', () => {
  spyOn(authService, 'login').and.callThrough();
  component.submit();
  expect(authService.login).not.toHaveBeenCalled();
  expect(router.navigateByUrl).not.toHaveBeenCalled();
});

it('should mark all fields as touched if the form is invalid on submit', () => {
  const emailControl = component.form.controls['email'];
  const passwordControl = component.form.controls['password'];
  expect(emailControl.touched).toBeFalse();
  expect(passwordControl.touched).toBeFalse();
  component.submit();
```

```
    expect(emailControl.touched).toBeTrue();
    expect(passwordControl.touched).toBeTrue();
  });
});
```