Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська кваліфікаційна робота на тему:

РОЗРОБКА WEB-ПЛАТФОРМИ ДЛЯ ОРГАНІЗАЦІЇ ОСВІТНІХ КУРСІВ

Виконав: студент 4 курсу, групи 4КН-21Б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Ліщинський В. І.
(прізвище та ініціали)
Керівник: доктор філософії, асистент каф. КН
Перепелиця В. І.
(прізвище та ініціали)
" 04 " червня 2025 р.
Рецензент: к.т.н., доцент каф. САІТ
Варчук І. В.
(прізвище та ініціали)
" 05 " червня 2025 р.

Допущено до захисту
Зав. кафедри: проф., д.т.н Ярмолік А.А.
" 06 " червня 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань
– 12 "Інформаційні технології"
Спеціальність – 122 "Комп'ютерні науки"
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
"05" травня 2025 р

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Ліщинському Вадиму Ігоровичу

1. Тема роботи: Розробка WEB-платформи для організації освітніх курсів.
Керівник роботи: Перепелиця В'ячеслав Ігорович.
Затверджені наказом вищого навчального закладу "20" 03 2025 року № 97

2. Термін подання студентом роботи 30 травня 2025 р

3. Вихідні дані до роботи:
Мова програмування Python, JavaScript;
Використання об'єктно-орієнтованого програмування;
Середовище розробки JetBrains PyCharm.

4. Зміст текстової частини (перелік питань, які потрібно розробити):
вступ; обґрунтування доцільності WEB-платформи для організації освітніх курсів; реалізація модулі реєстрації та авторизації у застосунку, розробити функціонал створення оновлення та перегляду курсів; створення клієнтської частини застосунку висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень):
Послідовні діаграми схема алгоритму роботи авторизації; структури компонентів застосунків;

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Асистент кафедри КН Перепелиця В. І.		

7. Дата видачі завдання 05 травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Обгрунтування доцільності розробки додатку освітньої платформи	05.05.25	07.05.25.	
2	Проектування додатку освітньої платформи	08.05.25	11.05.25	
3	Реалізація модулі реєстрації та авторизації у застосунку, розробити функціонал створення оновлення та перегляду курсів	12.05.25	15.05.25.	
4	Створення клієнтської частини застосунку	16.05.25	26.05.25.	
5	Розробка інструкції користувача	27.05.25	29.05.25.	
6	Аналіз отриманих результатів та формування висновків	30.05.25.	01.06.25.	
7	Оформлення бакалаврської кваліфікаційної роботи	02.06.25	04.06.25.	

Студент _____ (підпис) **Ліщинський В. І.** (прізвище та ініціали)

Керівник роботи _____ (підпис) **Перепелиця В. І.** (прізвище та ініціали)

АНОТАЦІЯ

Ліщинський В. І. Розробка Web-платформи для створення освітніх курсів. Бакалаврська кваліфікаційна робота складається з 93 сторінок формату А4, містить 27 рисунків, список використаних джерел налічує 32 найменувань.

Мета роботи полягає в створенні ефективного інструменту для поширення накопичених знань шляхом організації курсів із забезпеченням зручного інтерфейсу для авторів і слухачів.

В результаті роботи створено адаптивний і розширюваний додаток, що забезпечує комфортний та продуктивний доступ користувачів до навчального контенту. Серверна частина системи реалізована за допомогою мови програмування Python, натомість клієнтський застосунок — із використанням JavaScript; вся розробка виконана в інтегрованому середовищі PyCharm.

Ключові слова: освітня платформа, онлайн-навчання, управління контентом, система адміністрування, Zustand, Onion архітектура.

ABSTRACT

Lishchynskyi V. I. Development of a Web Platform for Creating Educational Courses. The Bachelor's thesis comprises 93 A4-format pages, includes 27 figures, and the list of references contains 32 entries.

The aim of this work is to create an effective tool for disseminating accumulated knowledge through the organization of courses, providing a user-friendly interface for both instructors and learners.

As a result of this project, an adaptive and extensible application was developed, ensuring comfortable and productive access for users to educational content. The system's server side is implemented in the Python programming language, while the client-side application is built using JavaScript; all development was carried out in the PyCharm integrated development environment.

Keywords: educational platform, online learning, content management, administration system, Zustand, Onion architecture.

ЗМІСТ

ВСТУП

Актуальність досліджень. У сучасному світі, багато людей — як початківці, так і ті, хто вирішив змінити професію — прагнуть увійти в IT, проте часто не мають чіткої «дорожньої карти» для досягнення своїх цілей. Без належної підтримки та зрозумілої структури вони зазвичай покладаються на онлайн-курси, які або не забезпечують належного інтерактиву, або вимагають значних витрат.

На сьогодні переважна кількість онлайн-платформ пропонують або статичні відео роки без можливості безпосереднього зворотного зв’язку, або дорогі «живі» курси з викладачем у реальному часі. Як наслідок, студенти часто втрачають мотивацію через відсутність інтерактиву та можливості швидко отримати відповіді на свої запитання, або ж змушені платити велику ціну за повноцінну навчальну взаємодію [1].

Тому розробка рішення, яке поєднує переваги відеоматеріалів у вільному доступі й можливість оперативної консультації з викладачем у реальному часі, є вкрай актуальною. Такий підхід здатен не лише підвищити якість та доступність навчання, а й зменшити витрати на організацію курсів за рахунок оптимального використання часу викладача.

Отже, розробка освітньої веб платформи з гібридним форматом навчання є актуальною.

Метою дослідження є створення освітньої онлайн-платформи для надання можливості поширення накопичених знань методом створення курсів на платформі.

Об’єкт дослідження – процес розробки WEB-платформи для організації освітніх курсів.

Предмет дослідження – WEB-застосунок платформи для організації освітніх курсів.

Задачі дослідження:

1. проаналізувати предметну область та аргументувати доцільність розробки;

2. визначити вимоги та розробити архітектуру додатку;

3. проаналізувати та аргументувати вибір технологій для розробки;

4. розробити модулі авторизації та адміністрування курсів;

5. розробити клієнтську частину застосунку;

Апробація результатів роботи

Результати досліджень було апробовано на LIII науково-технічній конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ–2025) м. Вінниці у 2025 р. [1].

Публікації: за основними результатами досліджень опубліковано тези доповіді на науково-технічній конференції [1].

1 ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ ДОДАТКУ ОСВІТНЬОЇ ПЛАТФОРМИ

1.1 Аналіз предметної області

Навчальні платформи та курси стали невід’ємною частиною сучасного освітнього середовища, створюючи можливості для отримання нових знань та навичок незалежно від географічних чи часових обмежень. В Україні, як і у світі, стрімко зростає інтерес до онлайн-освіти, особливо у сфері інформаційних технологій, що зумовлено високим попитом на спеціалістів та можливостями кар'єрного зростання.

Однак, попри численні переваги дистанційного навчання, залишається ряд суттєвих проблем, які обмежують ефективність онлайн-курсів. Найбільш поширеними з них є недостатня інтерактивність навчального процесу та висока вартість повноцінних курсів із залученням викладачів у реальному часі.

З одного боку, значна кількість освітніх платформ пропонують користувачам статичні навчальні матеріали у форматі записаних відео-лекцій. Цей формат, хоча й зручний у використанні, часто не забезпечує належного рівня залучення та мотивації студентів, адже відсутня можливість отримати миттєвий зворотний зв'язок та роз'яснення від викладача. Як наслідок, студенти швидко втрачають мотивацію та інтерес до навчання через пасивний характер подачі матеріалу.

З іншого боку, платформи, які пропонують інтерактивні заняття з живим спілкуванням у реальному часі, часто ставлять перед користувачами фінансові бар'єри. Високі ціни на такі курси роблять їх недоступними для багатьох студентів, особливо тих, хто тільки розпочинає свою кар'єру в IT або бажає змінити професію. Таким чином, значна частина потенційних слухачів залишається без якісних освітніх послуг через фінансові обмеження.

У сучасному світі, багато людей — як початківці, так і ті, хто вирішив змінити професію — прагнуть увійти в IT, проте часто не мають чіткої

«дорожньої карти» для досягнення своїх цілей. Без належної підтримки та зрозумілої структури вони зазвичай покладаються на онлайн-курси, які або не забезпечують належного інтерактиву, або вимагають значних витрат.

Розробка гібридних освітніх платформ, які поєднують переваги записаних відео лекцій із можливістю оперативної консультації з викладачем у реальному часі, стає особливо актуальною. Такий формат дозволяє студентам отримувати доступ до якісного, структурованого контенту в зручному темпі, а при виникненні питань — оперативно отримувати допомогу від кваліфікованих менторів. Це дозволяє знизити вартість навчання завдяки оптимальному використанню часу викладачів та зменшити фінансове навантаження на слухачів.

Таким чином, актуальним завданням є створення освітньої платформи, яка ефективно вирішує проблему низького рівня інтерактивності та високих витрат, пропонуючи користувачам доступне та якісне навчання, адаптоване до сучасних потреб ринку праці.

1.2 Огляд аналогів освітніх платформ

На сьогодні у світі існує значна кількість онлайн-освітніх платформ, таких як Udemy, Coursera, Prometheus та багато інших. Кожна з них пропонує власний підхід до організації навчального процесу, має свої переваги, функціональні особливості та обмеження. Зокрема, деякі орієнтовані переважно на академічну освіту з акцентом на співпрацю з університетами (наприклад, Coursera), інші — на масове створення контенту незалежними викладачами (наприклад, Udemy), а ще інші — на локальні освітні потреби (як-от український Prometheus).

Проте жодна з наявних платформ не забезпечує повної відповідності сформульованим нами вимогам до навчального процесу, які включають більшу гнучкість у створенні авторських курсів, поєднання самостійного навчання з

інтерактивною підтримкою, глибоку індивідуалізацію програм та розширені можливості залучення експертів-менторів.

Особливо важливо підкреслити, що більшість існуючих рішень або зовсім не дозволяють авторам безпосередньо створювати власні інтерактивні курси з гнучкою структурою навчання та контролем прогресу слухачів, або ж передбачають суттєві технічні чи фінансові бар'єри для запуску таких курсів. Це суттєво обмежує потенціал для розвитку нових форматів освіти, орієнтованих на практичні навички, менторську підтримку та адаптацію до швидко змінних потреб ринку.

Як результат, жодна з цих платформ повністю не відповідає поставленим нами вимогам, що робить розробку нової платформи доцільною.

1.2.1 Udemy

Глобальний маркетплейс освітніх курсів, що налічує понад 200,000 курсів, які створюють приватні викладачі з усього світу [2]. Курси охоплюють широкий спектр тем від IT до особистого розвитку (рис 1.1).

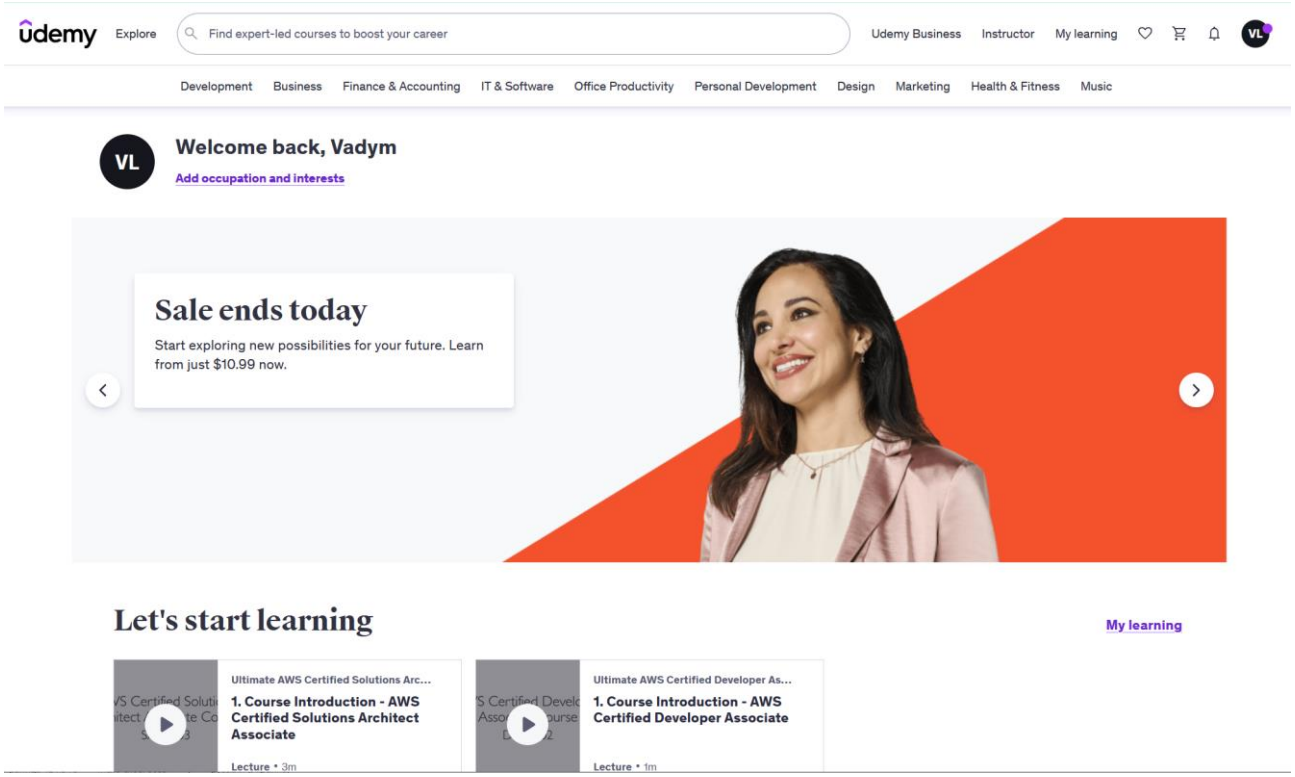


Рисунок 1.1 – Головна сторінка кабінету Udemy

Основний функціонал:

– Відео-лекції з можливістю самостійного проходження.

– Тести та практичні завдання у межах курсів.

– Довічний доступ до придбаних матеріалів.

– Система рейтингів і відгуків для оцінки якості курсів.

Недоліки:

– Відсутність безпосереднього інтерактиву та консультацій з викладачем.

– Сертифікати мають низьку академічну цінність.

– Мала кількість україномовних курсів.

1.2.2 Hillel IT School

Українська освітня платформа, яка спеціалізується на підготовці IT-фахівців з акцентом на практичне навчання та підтримку студентів менторською системою (рис. 1.2). Курси охоплюють різні напрямки: програмування, тестування, дизайну та менеджменту в IT-сфері. [3].

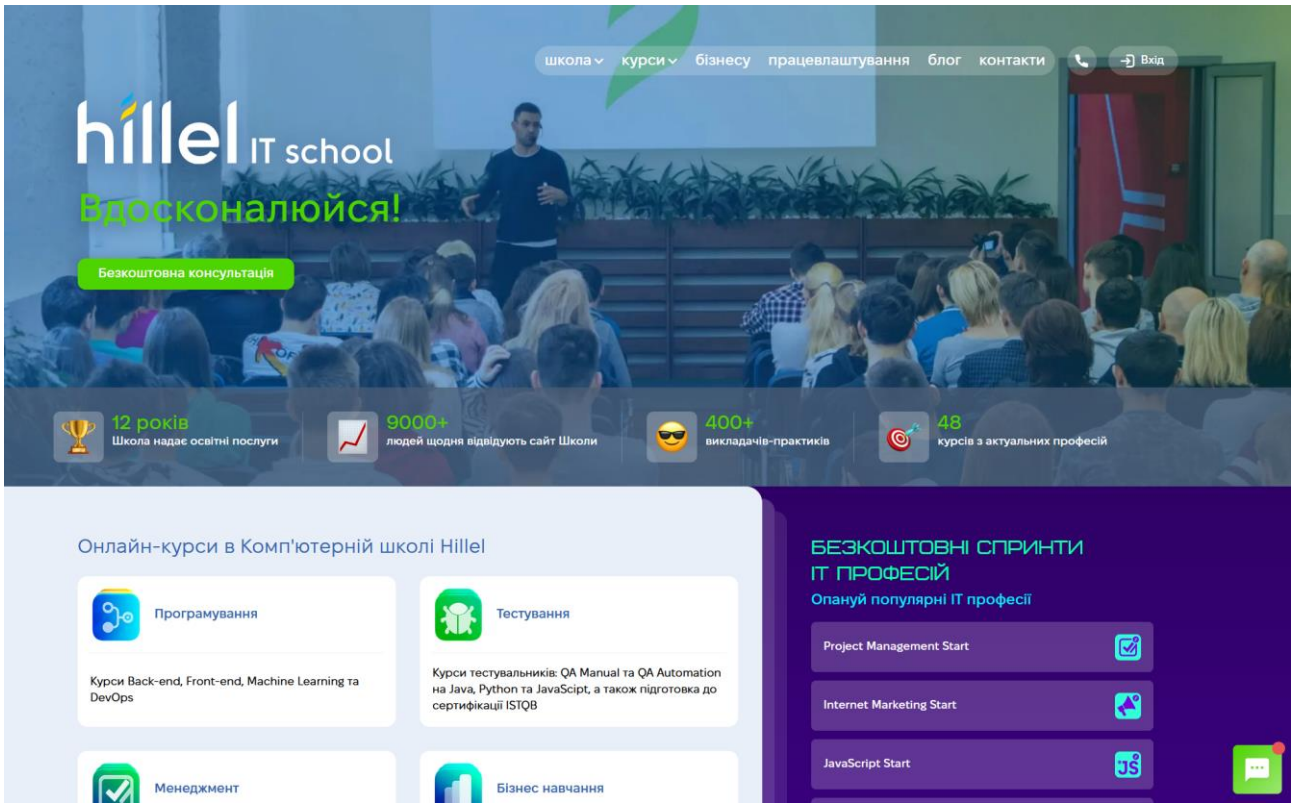


Рисунок 1.2 – Головна сторінка школи Hillel

Основний функціонал:

– Мала кількість україномовних курсів.

– Практичні курси із залученням професійних викладачів.

– Індивідуальний підхід до студентів, підтримка кураторів та менторів.

– Допомога у працевлаштуванні після завершення курсів.

Недоліки:

– Висока вартість навчання, що може бути недоступною для широкого кола слухачів.

– Інтенсивність курсів може бути занадто високою для осіб, які поєднують навчання з роботою.

– Деякі курси мають обмежену кількість місць, що ускладнює доступність.

1.2.3 Coursera

Глобальна освітня платформа (рис 1.3), що співпрацює з провідними університетами та компаніями для створення курсів, спеціалізацій та професійних сертифікатів [4]. Надає можливість отримати академічно визнані сертифікати та навіть повноцінні дипломи.

Основний функціонал:

– Курси від університетів і компаній (відео-лекції, інтерактивні завдання, тести).

– Можливість отримати сертифікати, дипломи та ступені.

– Деякі курси доступні безкоштовно (аудиторський режим).

– Гнучкі строки навчання.

– Мобільний додаток для навчання з телефону.

– Співпраця з компаніями для надання освітніх послуг

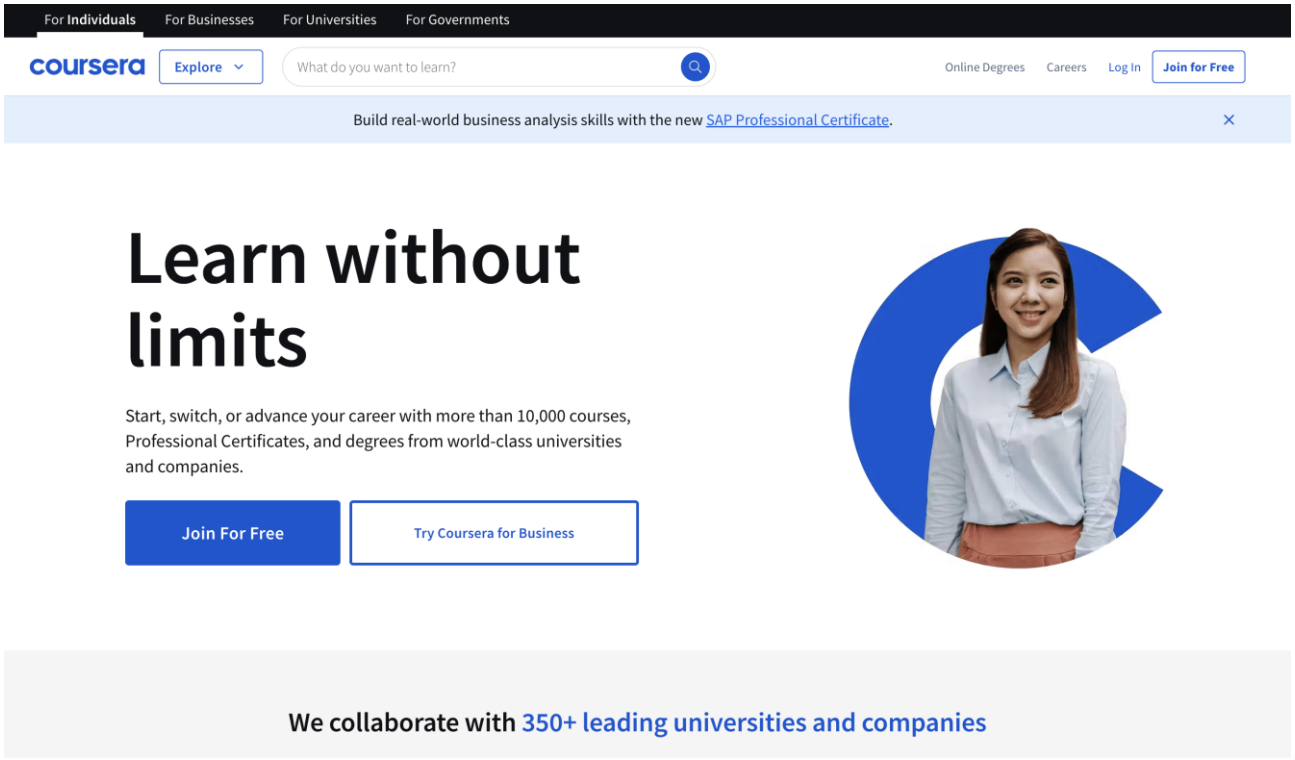


Рисунок 1.3 – Головна сторінка кабінету Coursera

Недоліки:

– Більшість сертифікатів потребують оплати.

– Обмежена можливість для прямої комунікації з викладачами.

– Деякі курси мають високий рівень складності без попередньої підготовки.

1.2.4 Prometheus

Prometheus — провідна українська платформа масових відкритих онлайн-курсів (МВОК), заснована у 2014 році (рис 1.4). Метою проєкту є забезпечення доступу до якісної освіти для широких верств населення в Україні. Платформа активно співпрацює з українськими університетами, громадськими організаціями, державними установами та бізнесом, пропонуючи актуальні курси у сфері державного управління, IT, підприємництва, права, освіти, психології тощо [5].

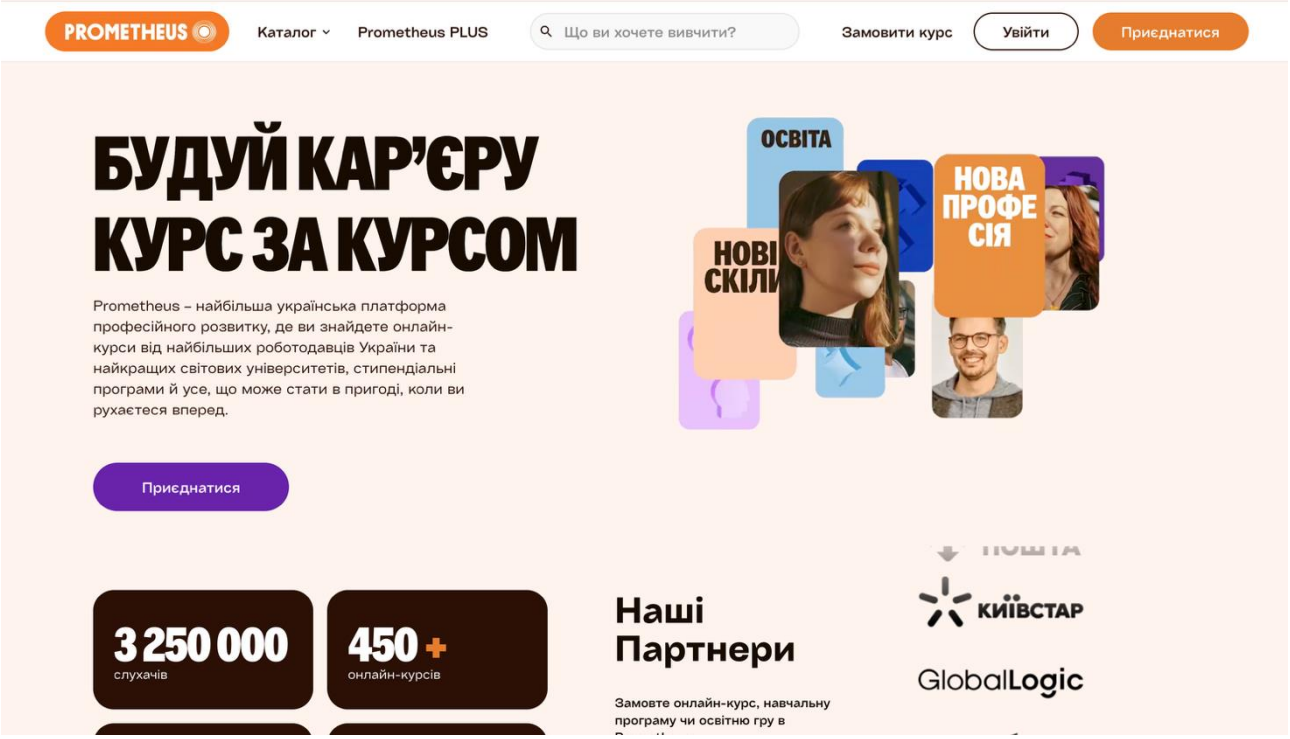


Рисунок 1.4 – Головна сторінка кабінету Prometheus

Основний функціонал:

– Безкоштовні онлайн-курси українською мовою.

– Співпраця з провідними українськими ВНЗ, урядовими структурами та громадськими організаціями.

– Відеолекції, інтерактивні завдання, тести, форуми.

– Сертифікати про завершення курсів, які інколи визнаються при офіційному навчанні.

– Курси на актуальні суспільно важливі теми (державне управління, медіаграмотність, IT, підприємництво, освіта дорослих тощо).

Недоліки:

– Обмежена кількість курсів у порівнянні з глобальними платформами.

– Менший вибір вузькоспеціалізованих міжнародних програм.

– Відсутність формальної академічної акредитації дипломних програм.

– Відсутність живого спілкування з лекторами.

1.3 Постановка задачі та формулювання вимог

Формулювання вимог до системи є ключовим етапом її створення, оскільки саме вони визначають очікувану функціональність та необхідні характеристики майбутнього програмного продукту. Чітко визначені вимоги дозволяють узгодити бачення замовника й розробників, мінімізувати ризики помилок під час розробки, а також забезпечити якісну реалізацію та подальшу підтримку системи. У зв’язку з цим далі наведено опис основних ролей користувачів системи та відповідних функціональних можливостей, які необхідно реалізувати під час розробки платформи.

Правильне визначення ключових ролей користувачів є важливою складовою під час проєктування будь-якої інформаційної системи. Саме ролі визначають, якими можливостями володіє кожен тип користувача, який обсяг даних доступний для перегляду та редагування, а також які функціональні завдання він може виконувати.

У системі передбачено три ключові ролі: слухач (студент), викладач і адміністратор. Така диференціація дає змогу чітко розподілити обов’язки, повноваження і доступ до функціоналу платформи. Слухач взаємодіє переважно з освітнім контентом і завданнями, викладач відповідає за формування курсу та супровід студентів, а адміністратор контролює технічне й організаційне налаштування всієї платформи.

Функціонал слухача.

Для слухача пропонується можливість реєстрації через email або соціальні мережі, що забезпечує швидку та зручну авторизацію. Особистий

кабінет студента містить перелік доступних і вже пройдених курсів, результати проходження курсів, а також індивідуальний план навчання. Система передбачає перегляд відео лекцій, завантаження супровідних матеріалів і отримання сертифікатів після успішного завершення програм.

Функціонал викладача.

Викладач отримує розширені права для створення та редагування курсів:

– завантаження відео контенту,
– побудову структури уроків,
– формулювання тестових завдань
– організацію живих консультацій

Кабінет викладача містить інструменти для відстеження прогресу студентів і надання своєчасного зворотного зв’язку. Для аналізу результативності курсу передбачено аналітичні модулі, що показують загальний рівень залученості слухачів і відсоток виконання завдань. Це дає змогу оперативно виявляти сильні та слабкі сторони навчальної програми.

Функціонал адміністратора.

Адміністратор забезпечує загальне керування платформою, здійснюючи перевірку та модерацію завантаженого контенту, контроль фінансових транзакцій і налаштування різних параметрів системи (надання промокодів, налаштування платіжних методів, управління обліковими записами). Завдяки доступу до розширеної аналітики він може оцінювати загальну результативність освітнього процесу й ухвалювати рішення щодо стратегії розвитку платформи.

1.4 Висновок до розділу 1

У даному розділі було проведено аналіз існуючих рішень у сфері освіти, виявлено їх основний функціонал та недоліки. Виявлено, що розвиток гібридного формату навчання зі сполученням відео контенту й живих

консультацій є вкрай актуальним на сучасному освітньому ринку, особливо з огляду на постійну потребу у персоналізованому підході та швидкій адаптації до змін.

Порівняно з традиційними онлайн-платформами, що працюють переважно в асинхронному режимі, запропонована модель додає інтерактивності та наближає студента до досвіду роботи у реальних професійних середовищах. Окрім того, впровадження навчальних планів, групових проєктів і механізмів зарахування попереднього досвіду суттєво підвищує ефективність та гнучкість освітнього процесу.

Усі основні вимоги і функціональний блок платформи визначено, що дає змогу спиратися на чітку структуру при подальшій реалізації проєкту.

2 ПРОЕКТУВАННЯ ДОДАТКУ ОСВІТНЬОЇ ПЛАТФОРМИ

Проектування архітектури — це не просто вибір стека технологій і малювання діаграм, а формування своєрідного «скелета» програми, на якому тримаються всі її частини: від користувацького інтерфейсу до зберігання даних. Чіткий поділ на шари показує, як запит користувача перетворюється у виконання бізнес-логіки та де ці дані зберігаються.

У світі стартапів успіх часто визначається швидкістю виходу MVP — тому не варто витрачати час на складні архітектурні патерни, якщо вони не потрібні для першого релізу [6]. Водночас уже на старті важливо закласти базові елементи: чітко описані API-контракти, інтерфейси для взаємодії з базою даних і модулі, що втілюють основну бізнес-логіку.

Такий підхід працює за двома напрямками. По-перше, він дає змогу команді оперативно створити робочий продукт і зібрати відгуки користувачів. По-друге, він відкриває шлях для подальшої еволюції системи: коли виникне потреба масштабуватися або перейти на мікросервіси, достатньо буде поетапно витягувати автономні частини без фундаментального рефакторингу всієї кодової бази. Таким чином, архітектура стає інструментом і для швидкого старту, і для стабільного зростання освітньої платформи.

Архітектурні рішення визначають розподіл відповідальностей між модулями системи, встановлюють межі компонентів та протоколи взаємодії. Правильно спроектована архітектура:

– Зменшує ймовірність помилок завдяки чіткому розмежуванню зон відповідальності: інтерфейс користувача, бізнес-логіка та доступ до даних не «переплітаються», що мінімізує ризик непередбаченої взаємодії й побічних ефектів.

– Підвищує стійкість до навантажень: за рахунок чіткої трирівневої структури (Presentation–Application–Data) можна масштабувати кожний рівень окремо, не втручаючись у внутрішню логіку інших компонентів.

Це вихідний рівень гарантії якості: навіть якщо команда розробників змінюється, добре побудована архітектура знижує поріг входу й допомагає швидше освоїтися в кодовій базі.

2.1 Вимоги до архітектури платформи

Проєктування архітектури освітньої платформи починається з формулювання чітких бізнес- та нефункціональних вимог, які враховують обмежені ресурси стартапу й одночасно закладають підґрунтя для подальшої еволюції системи.

Бізнес-орієнтовані вимоги

Першочерговою метою є мінімізація часу виходу на ринок (time-to-market). Це досягається шляхом обмеження обсягу початкового функціоналу лише ключовими сценаріями: реєстрація користувачів, управління курсами, оплата послуг та відтворення навчального контенту. Для прискорення розробки використовуються готові бібліотеки інтерфейсних компонентів та стандартні рішення для аутентифікації, що дозволяє зосередитися на бізнес-логіці замість реалізації другорядних сервісів. Крім того, передбачено застосування механізму feature-flags, який дає змогу поступово вводити нові можливості та за необхідності швидко здійснювати відкат змін.

Нефункціональні вимоги

– Модульність і розширюваність. Архітектура базується на Onion-підході з чітким розмежуванням доменної логіки, шарів сервісів та інфраструктурних адаптерів [7]. Така структура забезпечує незалежність кожного модуля та спрощує подальшу декомпозицію моноліту в набір мікросервісів.

– Конфігурована інфраструктура. Усі параметри оточення (строки підключення, ключі доступу, налаштування кешу) виносяться в змінні

середовища й файли конфігурації. Використання Infrastructure as Code (наприклад, Terraform або CloudFormation) гарантує відтворюваність середовищ і швидке масштабування нових інстансів [8].

– Масштабованість. Передбачено горизонтальне масштабування Presentation Tier (через CDN та автоматичне розгортання Next.js ISR) і Application Tier (через балансувальник навантаження та масштабування контейнерів FastAPI), а також поступове впровадження реплікації та шардінгу в Data Tier (MySQL) [9].

– Готовність до мікросервісної еволюції. Інтерфейси доступу до даних та зовнішніх сервісів реалізовано через абстракції, що дозволяє в майбутньому «витягти» окремі модулі в самостійні сервіси без змін у бізнес-логіці.

– Тестованість і якість. Кожен рівень архітектури супроводжується юніт- та інтеграційними тестами. CI/CD-конвеєр автоматизує прогін тестів, збірку та деплой, що підвищує стабільність релізів [10].

– Безпека та відмовостійкість. Забезпечується шифрування передавання даних (HTTPS), використання JWT для авторизації, обмеження швидкості запитів (rate limiting) і моніторинг стану системи (CloudWatch, Prometheus/Grafana) з налаштованими алертами для швидкого реагування на інциденти [11].

Усі вищезазначені вимоги поєднують прагнення до оперативного створення MVP із закладеною можливістю безшовного масштабування та еволюції архітектури відповідно до зростання бізнес-потреб.

2.2 Основні паттерни та підходи до створення онлайн-застосунку освітньої платформи

Сучасні освітні онлайн-платформи ставлять перед розробниками низку технологічних викликів, зокрема підтримку великої кількості користувачів, інтеграцію різноманітних типів контенту, високу гнучкість і масштабованість. У цьому контексті особливо важливими є вибір архітектурних рішень та правильна

організація взаємодії компонентів. Серед найбільш ефективних підходів до розв'язання цих завдань є застосування Onion Architecture («цибулинної архітектури») принципу Dependency Injection (ін'єкції залежностей) та використання Unit of Work та Repository патернів для створення надійної освітньої онлайн-платформи [12-14].

2.2.1 Переваги використання Onion Architecture

Концепція Onion Architecture (рис 2.1) передбачає поділ системи на кілька незалежних шарів, розміщених у вигляді концентричних кіл. Центральним є доменний шар (Domain Layer), що містить ключові моделі, наприклад, студенти, викладачі, курси, модулі, уроки тощо. Наступними йдуть шари прикладної логіки (Application Layer), інфраструктури (Infrastructure Layer) та презентації (Presentation Layer). Ключова особливість цієї архітектури полягає в тому, що всі залежності спрямовані до центру, а внутрішні шари не залежать від зовнішніх.

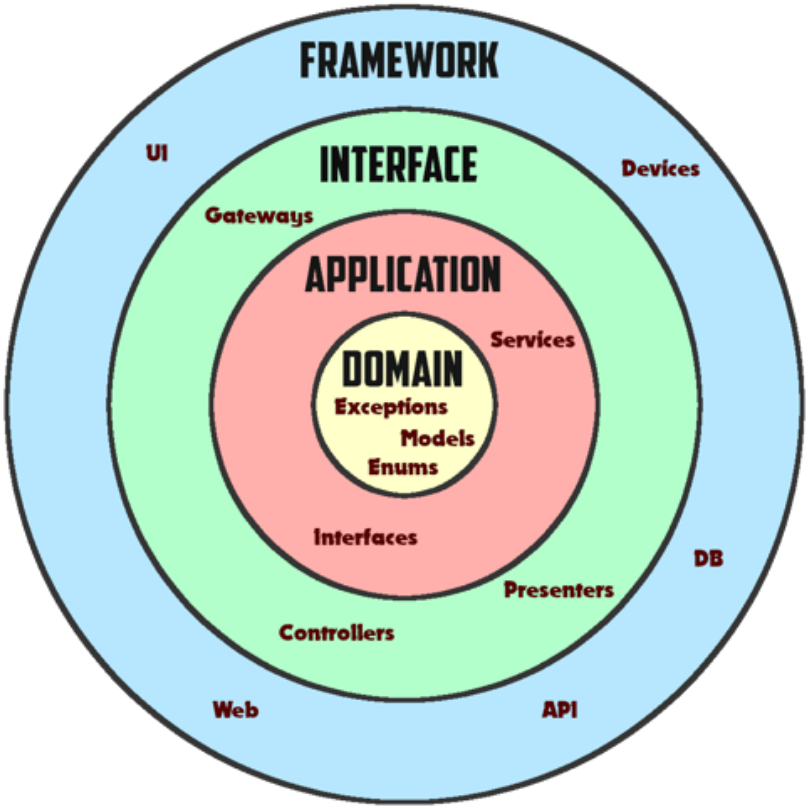


Рисунок 2.1 – Організація модулів в Onion архітектурі [15]

Такий підхід є особливо корисним для освітньої платформи з наступних причин:

Незалежність бізнес-логіки від технологій.

Наприклад, модель курсів чи користувачів не залежить від конкретних баз даних (MySQL, PostgreSQL) чи способів авторизації користувачів (OAuth, JWT). Це дозволяє платформі легко адаптуватися до нових технологій без зміни основної логіки.

Простота оновлення та розширення платформи.

Якщо в майбутньому буде потрібно інтегрувати новий тип контенту (наприклад, інтерактивні симуляції чи відеоуроки), Onion Architecture дозволить зробити це без глибоких змін у всій системі. Досить додати новий сервіс на рівні інфраструктури чи презентації.

Підвищена тестованість компонентів.

Кожен шар може бути протестований окремо, що дозволяє швидко виявляти помилки й мінімізувати їх вплив. Це важливо для освітньої платформи, де стабільність та якість контенту є критично важливими для користувачів.

Наприклад, якщо у платформі з'явиться необхідність змінити платіжну систему, то завдяки структурі Onion Architecture достатньо буде лише замінити сервіс на інфраструктурному шарі, без втручання у доменну логіку, що значно спрощує підтримку та модернізацію системи.

2.2.2 Важливість Dependency Injection у контексті освітньої платформи

Іншим важливим принципом, який доповнює Onion Architecture, є Dependency Injection (DI). DI передбачає, що залежності компонентів системи не створюються всередині самих компонентів, а передаються ззовні. Це дозволяє суттєво підвищити гнучкість і модульність застосунку.

Ключові переваги застосування Dependency Injection для освітньої платформи:

Слабка пов'язаність компонентів.

Завдяки використанню DI, компоненти освітньої платформи (наприклад, сервіси управління курсами чи системи оцінювання студентів) можуть бути легко замінені або оновлені без додаткових змін у інших частинах системи.

Покращення процесу тестування.

Наприклад, якщо платформа використовує сервіс надсилання повідомлень студентам (EmailSender), під час тестування його легко замінити тестовою реалізацією (mock-об'єктом), що дозволяє уникнути надсилання реальних повідомлень та забезпечує швидке та безпечне тестування [16].

Можливість легкого масштабування.

При зростанні кількості користувачів платформи чи потребі у розширенні функціоналу, Dependency Injection дозволяє оперативно інтегрувати додаткові компоненти (наприклад, додаткові сервіси авторизації, нові платіжні системи чи системи аналітики) без переписування існуючого коду.

Наприклад, якщо платформа на старті використовує базу даних MySQL [17], а в подальшому вирішує перейти на PostgreSQL, то завдяки Dependency Injection достатньо буде змінити конфігурацію компонента доступу до даних. Це дозволяє уникнути складних і ризикованих переробок у ядрі застосунку, що важливо для великих і тривалих проєктів.

Практичні аспекти застосування цих підходів.

У реальних умовах освітньої онлайн-платформи використання Onion Architecture та Dependency Injection допомагає суттєво скоротити витрати часу і ресурсів на підтримку системи. Наприклад:

Зручна інтеграція нових викладачів та курсів.

Платформа легко масштабується, швидко інтегруючи нових викладачів, модулі, курси без необхідності глобального втручання у структуру.

Швидка адаптація до змін потреб користувачів.

Якщо платформа потребує додавання нових функцій на запит користувачів (наприклад, чат з викладачем, оцінка домашніх завдань у реальному часі тощо), ці підходи забезпечують швидке і якісне впровадження.

Підвищена надійність системи.

Завдяки високій тестованості компонентів, платформа стабільно працює при великій кількості користувачів, що підвищує рівень довіри з боку студентів та викладачів.

2.2.3 Значення патерну Unit of Work для освітньої платформи

Unit of Work («одиниця роботи») – це шаблон проєктування, що забезпечує виконання операцій з даними у вигляді єдиної транзакції. Іншими словами, всі зміни, які вносяться під час створення нового курсу, модулів, уроків, повинні бути або виконані повністю, або не виконані взагалі. Це гарантує збереження узгодженості та цілісності домену, що критично важливо для платформи, яка обробляє велику кількість взаємопов’язаних сутностей. Принцип взаємодії компонентів в паттерні Unit of Work показано на рисунку 2.2.

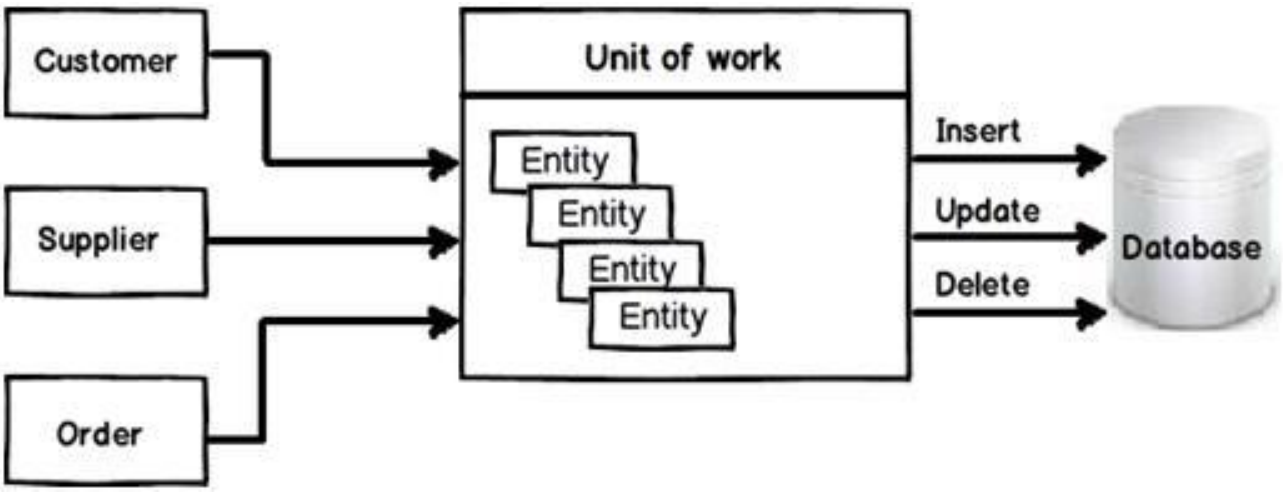


Рисунок 2.2 – Принцип взаємодії компонентів в паттерні Unit of Work [18]

Основні переваги застосування Unit of Work на освітній платформі:

Забезпечення атомарності транзакцій.

Створення нового курсу з усіма його складовими (модулями, уроками, завданнями) відбувається як єдина дія. Якщо хоча б одна частина цієї транзакції не виконається, всі попередні дії будуть автоматично скасовані, що захищає систему від некоректних чи часткових даних.

Контроль за змінами в адмініструванні.

Unit of Work дозволяє легко відстежувати всі зміни, зроблені адміністраторами платформи. Це важливо для аудиту та контролю процесів створення й редагування курсів, а також для виявлення потенційних помилок.

Оптимізація роботи з базою даних.

Патерн забезпечує оптимальну взаємодію з базою даних, мінімізуючи кількість запитів і підвищуючи продуктивність платформи, що особливо важливо при великій кількості активних користувачів.

Наприклад, при створенні нового курсу адміністратор може додати курс, модулі, уроки та завдання в одному операційному контексті. Unit of Work забезпечить, що всі ці операції або успішно збережуться в базі даних, або будуть повністю відкинуті у разі помилки.

Використання Repository-патерну в структурі освітньої платформи.

Repository («репозиторій») – це патерн, що забезпечує абстракцію над доступом до даних. Repository надає чітко визначений набір методів для роботи з конкретними сутностями, приховуючи деталі реалізації зберігання даних. Для освітньої платформи цей патерн є надзвичайно корисним, оскільки дозволяє працювати з даними більш прозоро, ізольовано від конкретної реалізації бази даних.

Основні переваги використання Repository на прикладі освітньої платформи:

Абстракція даних:

Репозиторії приховують деталі доступу до бази даних (MySQL, PostgreSQL тощо). Це дозволяє легко змінювати реалізацію баз даних без суттєвого впливу на код платформи.

Спрощення тестування:

Завдяки абстракції, що надає Repository, легко створити mock-реалізації для тестування, що суттєво полегшує написання unit-тестів для бізнес-логіки платформи.

Чіткість коду та легкість підтримки:

Всі операції з даними чітко структуровані у репозиторіях, що спрощує розуміння коду та його подальшу підтримку.

2.3 Структурне моделювання додатку освітньої платформи

Структурне моделювання є важливим етапом розробки будь-якого сучасного застосунку, особливо освітньої платформи, де важливою є логічна узгодженість сутностей та взаємодія між ними. Чітка структурна модель допомагає забезпечити зручність подальшого розширення функціоналу, стабільність роботи платформи та зрозумілість взаємозв'язків між компонентами системи.

Опис основних моделей застосунку.

Для моделювання предметної області було розроблено структуру бази даних, що представлена на рисунку 2.3.

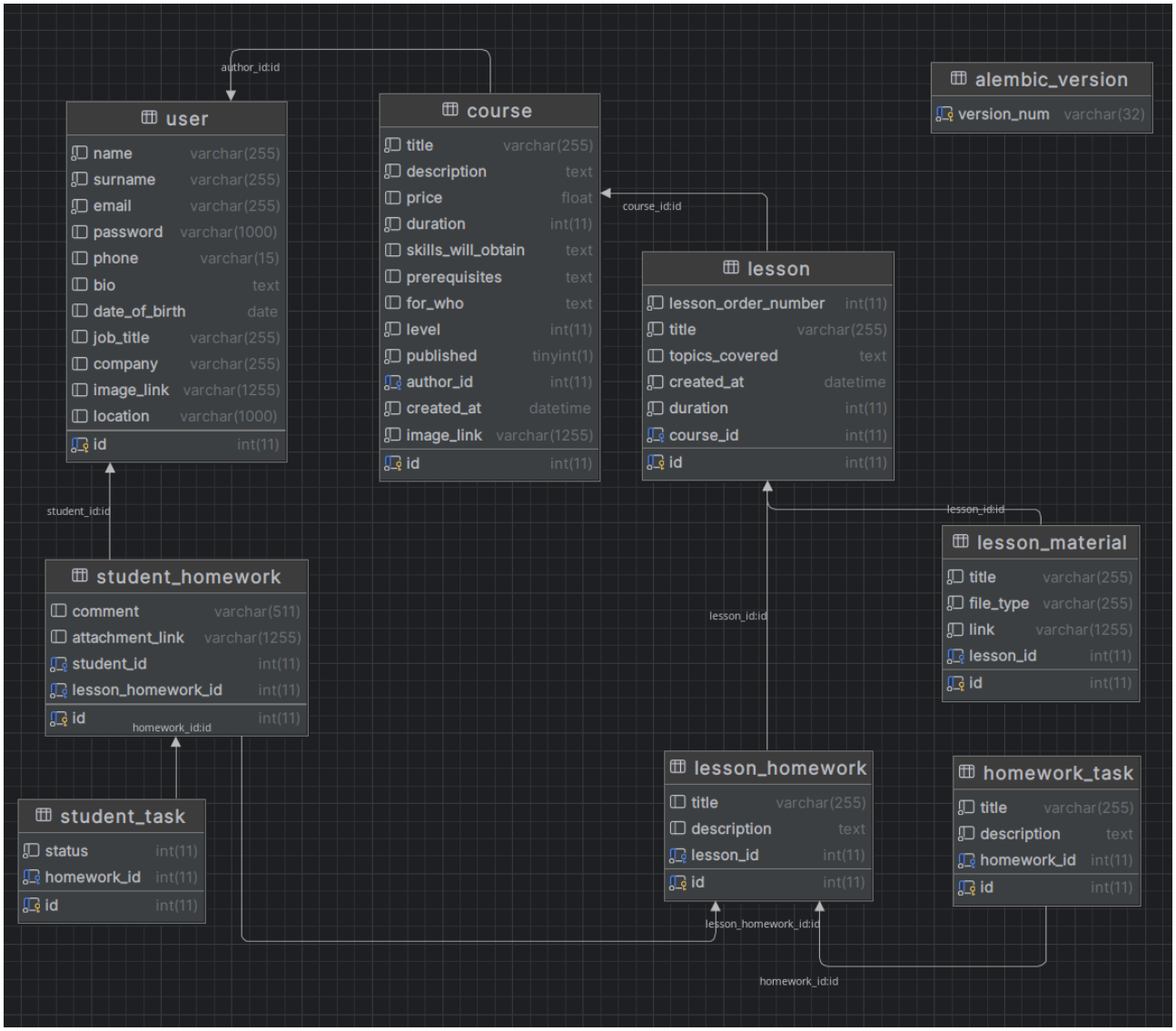


Рисунок 2.3 – Схема бази даних

Основними сутностями предметної області освітньої платформи є:

1. Користувач (user) Ця сутність містить всю необхідну інформацію для роботи з користувачами платформи: студентами, викладачами та адміністраторами.

Основні поля:

– id: унікальний ідентифікатор

– name, surname, email, password, phone: контактна інформація та аутентифікаційні дані

– bio, job_title, company, image_link, location: додаткова інформація профілю

2. Курс (course) Містить інформацію про курси, які створюються викладачами на платформі.

Основні поля:

– id: унікальний ідентифікатор

– title, description, price, duration: загальні відомості про курс

– skills_will_obtain, prerequisites, for_who: цільова аудиторія та передумови курсу

– published: стан публікації

– author_id: зв'язок з автором-викладачем курсу

– created_at: дата створення

3. Урок (lesson) Представляє модульну структуру курсів, які розбиті на окремі уроки.

Основні поля:

– id: унікальний ідентифікатор

– lesson_order_number: порядковий номер уроку в курсі

– title, topics_covered: опис і тематика уроку

– created_at, duration: додаткова інформація

– course_id: зв'язок із курсом

4. Матеріали уроків (lesson_material) Сутність для зберігання навчальних матеріалів, що додаються до уроків.

Основні поля:

– id: унікальний ідентифікатор

– title, file_type, link: інформація про навчальні ресурси

– lesson_id: зв'язок із уроком

5. Домашнє завдання (lesson_homework, homework_task, student_homework) Представляє систему завдань та їх перевірку.

– lesson_homework: Основна сутність домашнього завдання, що містить опис.

– homework_task: Окремі завдання в рамках домашнього завдання.

– student_homework: Здані завдання студентами.

6. Завдання студентів (student_task) Зберігає інформацію про виконання завдань студентами (статуси, оцінювання тощо).

Структурне моделювання дозволяє виявити й чітко визначити взаємозв'язки між різними сутностями, забезпечуючи логічну цілісність системи. Такий підхід дає змогу:

спростити розробку та підтримку платформи,

зменшити ймовірність помилок під час інтеграції нових функцій,

легко розширювати платформу новими модулями (наприклад, тестами, додатковими матеріалами, аналітикою навчання).

Завдяки цьому структурна схема бази даних платформи побудована таким чином, що її можна швидко й легко доповнювати новими функціями та реалізовувати за допомогою ORM системи (аналітика, інтеграція додаткових сервісів, система рекомендацій курсів тощо) [19].

Розподіл за репозиторіями.

Для ефективної роботи з базою даних запропоновано використання репозиторіїв, які забезпечують абстракцію над доступом до даних:

UserRepository. (Реєстрація, авторизація користувачів, Керування профілем та ролями)

CourseRepository. (Створення, редагування, публікація курсів, Управління переліком курсів та їх авторством)

LessonRepository. (Створення та редагування уроків, Управління порядком та змістом уроків)

MaterialRepository. (Завантаження, зберігання, керування навчальними матеріалами, Пошук матеріалів за уроками)

HomeworkRepository. (Створення домашніх завдань, редагування описів та завдань, Перевірка статусу виконання домашніх завдань студентами)

StudentHomeworkRepository. (Завантаження та оцінка виконаних студентами завдань, Відстеження прогресу студентів)

TaskRepository. (Відстеження виконання окремих завдань студентами, Виставлення оцінок та статусів)

Такий поділ забезпечує чистоту коду, високу швидкість розробки, зручність роботи та простоту підтримки у майбутньому.

2.4 Висновок до розділу 2

У цьому розділі було обґрунтовано вибір Onion Architecture як основи архітектури платформи — завдяки її модульності, гнучкості та зручності підтримки. Застосування принципів Dependency Injection спрощує тестування і модернізацію системи без порушення бізнес-логіки.

Також розглянуто патерни Unit of Work і Repository. Перший забезпечує цілісність і узгодженість змін при роботі з даними (створення, редагування, видалення курсів), другий — зручний доступ до БД і можливість розширення функціоналу (аналітика, рекомендації тощо).

Структура модулів побудована за відомими шаблонами проєктування, що дозволяє створювати масштабовані й добре організовані компоненти. Додатково проілюстровано взаємозв’язки між об'єктами через діаграми класів, що полегшує подальшу реалізацію, тестування та інтеграцію.

3 РОЗРОБКА ОСНОВНИХ МОДУЛІВ АДМІНІСТРУВАННЯ ОСВІТНЬОЇ ПЛАТФОРМИ

3.1 Аналіз і обґрунтування вибору засобів для реалізації

Під час створення освітньої онлайн-платформи однією з ключових задач є вибір оптимальних технологій, які забезпечать швидку, стабільну й ефективну роботу застосунку. Для реалізації поставлених завдань проєкту було обрано сучасні технології, які зарекомендували себе на ринку та широко використовуються для розробки надійних та масштабованих веб-застосунків.

Вибір технології для бекенду: FastAPI (Python) [20].

Для реалізації бекенд-частини платформи обрано мову програмування Python та фреймворк FastAPI [21]. Основні переваги цього вибору такі:

Висока продуктивність. FastAPI — один із найшвидших Python-фреймворків, побудований на основі бібліотеки Starlette та Uvicorn, що забезпечує високу швидкість обробки запитів і швидкий відгук сервера. Це особливо важливо для освітніх платформ, де навантаження на систему постійно зростає.

Чіткість та простота API. Завдяки вбудованій підтримці OpenAPI [22], FastAPI дозволяє автоматично створювати зрозумілу документацію API, що значно спрощує процес інтеграції бекенду з фронтендом та взаємодію з командою розробників.

Асинхронність і масштабованість. FastAPI ефективно підтримує асинхронні операції, що забезпечує стабільну роботу застосунку при великій кількості одночасних запитів користувачів.

Широка екосистема Python. Використання Python дає змогу швидко інтегрувати популярні бібліотеки (SQLAlchemy для роботи з базами даних, Pydantic для валідації даних, Pytest для тестування), що значно прискорює розробку та підтримку платформи [23,24].

3.2 Розробка серверної частини застосунку

Діаграма компонентів бекенд-частини, зображена на рисунку 3.1, освітньої платформи розроблена за принципами Onion-архітектури та покликана забезпечити чітке розділення відповідальностей між шарами додатка. Зовнішній Presentation Layer представлений трьома контролерами — AuthController, UserController і CourseController, — які реалізують прийом HTTP-запитів і первинну валідацію вхідних даних. Кожен контролер взаємодіє з відповідним сервісом із Service Layer, делегуючи туди виконання бізнес-логіки.

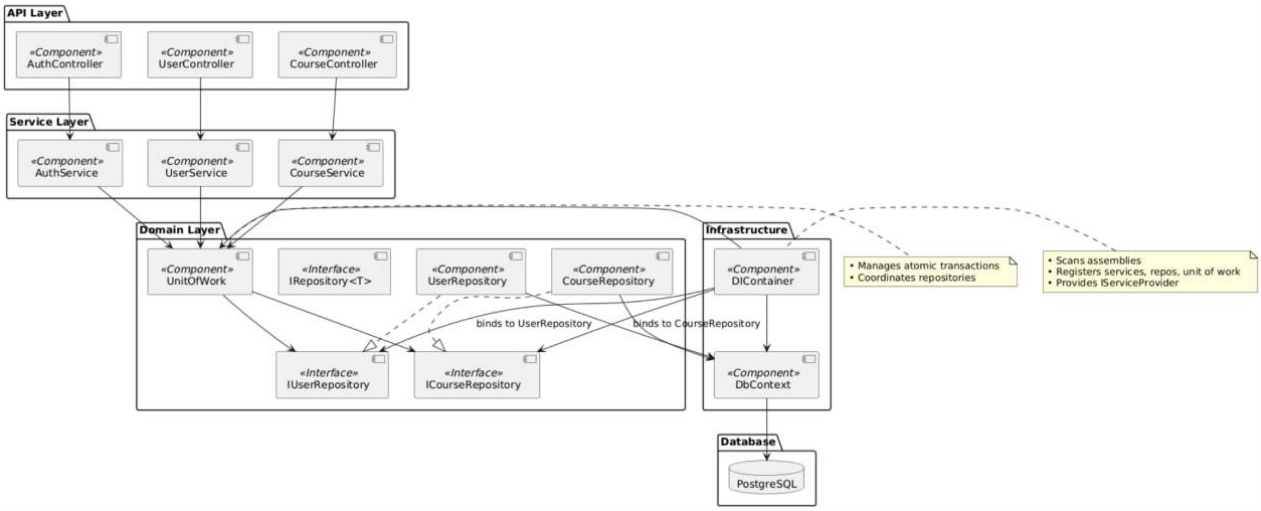


Рисунок 3.1. – Діаграма компонентів серверної частини застосунку

Сервісний шар включає класи AuthService, UserService і CourseService, що інкапсулюють ядро доменних операцій: аутентифікацію, керування користувачами та курсами. Для забезпечення цілісності й узгодженості змін використано паттерн UnitOfWork, який координує транзакції між репозиторіями. Безпосередню роботу з доменними сутностями виконує набір інтерфейсів IUserRepository та ICourseRepository, що визначені в Domain Layer і спрощують тестування завдяки явному контракту без прив’язки до технологічних деталей.

У Domain Layer також існують конкретні реалізації репозиторіїв — UserRepository та CourseRepository — які, обгорнуті в єдиний UnitOfWork, забезпечують доступ до даних і керування скопами транзакцій.

Інфраструктурний шар містить компонент DbContext, котрий інкапсулює специфіку роботи з СУБД (MySQL), та DIContainer — контейнер залежностей, що відповідає за реєстрацію сервісів, репозиторіїв і одиниць роботи відповідно до принципів інверсії керування.

Така організація коду в Python із використанням FastAPI гарантує модульність, легкість супроводу й масштабованість: кожен шар чітко ізольований, а зміни в реалізації сховища даних або бізнес-логіки не потребують переробки рівня контролерів чи сервісів. Подібна архітектурна побудова відповідає сучасним вимогам до розподілених систем і відкриває можливості для паралельної роботи над різними рівнями додатка.

3.2.1 Авторизація

У сучасних веб-додатках модуль аутентифікації та авторизації виконує ключову роль у забезпеченні безпечного доступу користувачів до ресурсів. У бекенд-платформі на базі FastAPI рекомендовано реалізувати набір REST-ендпоінтів, які дозволяють виконувати повний цикл роботи з обліковими записами: від реєстрації до завершення сессії. Нижче подано стислий опис основних маршрутів та їх функціональних можливостей.

1. POST /auth/register

Цей ендпоінт призначений для створення нового облікового запису користувача. Сервер приймає вхідні дані — електронну пошту та пароль, виконує валідацію формату, хешування пароля за допомогою стійкого алгоритму (наприклад, bcrypt) та зберігає результати в базі даних. У випадку успішного виконання клієнт отримує HTTP-відповідь із кодом 201 Created, яка підтверджує реєстрацію без повернення чутливої інформації.

2. POST /auth/login

Після реєстрації користувач може виконати автентифікацію, надіславши цей запит з тими ж параметрами (email і пароль). Сервер порівнює наданий пароль із хешем у сховищі, а в разі успіху генерує JWT-токен доступу (і, за потреби, refresh-токен). Токен укладає в собі ідентифікатор користувача та часові

обмеження життя сесії. Стандартна відповідь містить об’єкт із полями access_token, refresh_token (якщо застосовується) та token_type.

3. POST /auth/refresh

Щоб уникнути повторного введення пароля після закінчення терміну дії access-токена, передбачено механізм оновлення сесії. Клієнт надсилає дійсний refresh-токен, який перевіряється на предмет достовірності та строку дії. У разі позитивної верифікації сервер повертає новий JWT-токен доступу, що дозволяє продовжити роботу без повторного логіну.

4. POST /auth/logout

Для коректного завершення сесії та інвалідизації refresh-токена застосовується ендпоінт logout. Запит надходить із заголовком авторизації, де міститься саме refresh-токен. Сервер виконує його видалення або маркування як недійсного, після чого повертає відповідь 204 No Content, підтверджуючи успішний логаут.

5. GET /auth/me

Цей маршрут дозволяє отримати базову інформацію про поточного користувача на підставі дійсного access-токена. При успішній верифікації JWT запит повертає дані профілю (наприклад, email, ім’я, ролі), а при відсутності або простроченні токена — код 401 Unauthorized.

6. POST /auth/oauth/{provider}

Для інтеграції зі сторонніми провайдерами (наприклад, Google) реалізується універсальний ендпоінт, який приймає ID-токен від клієнта. Сервер верифікує цей сторонній токен, витягує інформацію про користувача та виконує операцію UPSERT у таблиці users, після чого видає внутрішній JWT-токен доступу. Це забезпечує безшовний досвід реєстрації й входу за допомогою сторонніх павідерів аутентифікації.

Таким чином, наведений набір ендпоінтів формує повноцінний API-інтерфейс для управління аутентифікацією та авторизацією, що відповідає принципам розподілених систем: кожен маршрут виконує свою чітко визначену функцію, а внутрішня логіка залишається ізольованою в межах відповідних

сервісів і репозиторіїв. Це сприяє зручному супроводу, розширюванню функціоналу та інтеграції додаткових механізмів безпеки.

3.2.2 Курси

У рамках модульної архітектури бекенд-платформи для управління курсами передбачено набір REST-ендпоінтів, що забезпечують повний цикл роботи з ресурсом «курс» та його підресурсом «заняття». Ці маршрути, реалізовані на базі FastAPI, формують чітку та послідовну структуру взаємодії клієнтської частини з сервером, гарантують гнучкість розширення та полегшують супровід.

Перш за все, для отримання переліку курсів служить маршрут **GET /courses**. Він підтримує параметри пагінації, сортування та фільтрації за категоріями чи рівнем складності, що дозволяє клієнту вибудовувати адаптивні інтерфейси з поступовим завантаженням даних. Детальну інформацію про конкретний курс можна витягти через **GET /courses/{course_id}**, який повертає не тільки метадані курсу (назву, опис, автора, дату створення), але й структуру занять і ключові налаштування, наприклад, кількість модулів чи обмеження доступу.

Для створення нових курсів призначений маршрут **POST /courses**. Він приймає у тілі запиту об’єкт із необхідними атрибутами – назвою, описом, категорією та рівнем складності – виконує їх валідацію та зберігає в базі даних у межах транзакції, забезпечуваної патерном UnitOfWork. Успішне створення курсу підтверджується кодом відповіді 201 Created та поверненням об’єкта згенерованого ресурсу, включно з його унікальним ідентифікатором.

Оновлення існуючого курсу здійснюється за допомогою **PUT /courses/{course_id}**. Цей маршрут дозволяє внести зміни до будь-яких атрибутів курсу – наприклад, оновити опис або змінити категорію – і повернути клієнту актуалізовану версію об’єкта. У разі спроби оновлення неіснуючого курсу сервер відповідає кодом 404 Not Found, а при помилках у структурі даних – 400 Bad Request.

Видалення курсу реалізовано через **DELETE /courses/{course_id}**. Залежно від бізнес-логіки можливе як «м’яке» інвалідизування (soft-delete), так і фізичне видалення запису. Успіх операції підтверджується кодом 204 No Content, що вказує на відсутність тіла відповіді.

Окремий набір маршрутів присвячений управлінню заняттями в рамках курсу. Для отримання списку занять певного курсу служить **GET /courses/{course_id}/lessons**, який повертає впорядкований масив об’єктів із полями id, title, order. Додавання нового заняття відбувається через **POST /courses/{course_id}/lessons**, куди в тілі запиту передаються назва, зміст і порядковий номер. Після успішного додавання клієнт отримує відповідь 201 Created із деталями створеного заняття.

Зміна параметрів заняття здійснюється маршрутом **PUT /courses/{course_id}/lessons/{lesson_id}**, який дозволяє оновлювати будь-які атрибути занять – від заголовка до змісту та порядкового індексу. Видалення конкретного уроку реалізовано через **DELETE /courses/{course_id}/ lessons/{lesson_id}**, що повертає 204 No Content при успішній операції або 404 Not Found, якщо ресурс відсутній.

Завдяки такій організації ендпоінтів забезпечується висока читабельність API, простота тестування та можливість поступового розширення функціоналу: до існуючих маршрутів легко додаються нові параметри фільтрації, рольова авторизація або динамічна генерація контенту без необхідності змінювати вже налагоджені шари контролерів і сервісів.

3.3 Висновок до розділу 3

У процесі реалізації серверної частини застосунку було поетапно сформовано набір базових модулів, які забезпечують як поточний функціонал, так і можливість подальшого масштабування системи. Насамперед, встановлено чітку обгортку для прийому зовнішніх HTTP-запитів із вбудованим механізмом маршрутизації та валідації вхідних даних. Це дозволяє на ранньому рівні

відокремити обробку клієнтських викликів від власне бізнес-логіки та гарантує стабільність роботи навіть при зростанні навантаження.

Далі було виділено доменні сервіси та шари, що відповідають за ключові бізнес-операції: роботу з користувацькими профілями, управління курсами, облік прав доступу та обробку подій навчального процесу. Кожен із цих сервісів інкапсулює суто свою сферу відповідальності, спираючись на інтерфейси репозиторіїв і єдину точку контролю транзакцій (UnitOfWork). Така організація не тільки підвищує читабельність коду, а й робить можливим одночасне його тестування та розгортання різних версій мікросервісів за потреби.

На рівні зберігання даних реалізовано універсальний репозиторій з підтримкою подальшої підміни сховища без зміни зовнішніх контрактів: досить додати новий адаптер для роботи з іншою СУБД або кешем. Додатково закладено каркас для асинхронної обробки повідомлень — це спрощує впровадження черг повідомлень і декомпозицію на окремі сервіси в майбутньому.

4 РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ ЗАСТОСУНКУ

Архітектура клієнтської частини освітньої платформи ґрунтується на сучасних підходах до веб-розробки, зокрема на компонентному підході, який забезпечує максимальну гнучкість і повторне використання коду. У ході проєктування особливу увагу було приділено тому, щоб кожен UI-елемент міг існувати автономно, мати чітко визначену відповідальність і легко тестуватися окремо від інших модулів. Завдяки цьому підходу команди розробників можуть паралельно працювати над різними частинами інтерфейсу без ризику конфліктів, а підтримка й розвиток проєкту залишаються керованими та структурованими.

4.1 Аналіз і обґрунтування вибору засобів для реалізації

Фронтенд частина освітньої платформи буде реалізована за допомогою комбінації технологій React, Next.js та Zustand. Вибір цих засобів є обґрунтованим завдяки наступним факторам:

React як основа UI: React — одна з найбільш популярних і надійних JavaScript-бібліотек для створення користувацьких інтерфейсів, яка дозволяє розробляти інтерактивні, динамічні та зручні в користуванні інтерфейси [25].

Next.js для продуктивності: Next.js є фреймворком на базі React, який забезпечує серверний рендеринг (SSR) та статичну генерацію сторінок (SSG) [26].Це забезпечує швидке завантаження сторінок платформи та кращу SEO-оптимізацію, що є критично важливим для залучення нових користувачів та їх комфортної роботи з контентом курсів [27].

Легке управління станом з Zustand: Zustand — це простий, ефективний та легкий у використанні інструмент для керування станом у React-застосунках. Завдяки йому спрощується зберігання і оновлення стану, такого як інформація про користувача, прогрес у курсах чи візуалізація завдань, без надмірного ускладнення коду та конфігурації [28].

Таким чином, вибір технологій Python (FastAPI) та React (Next.js, Zustand) є оптимальним рішенням для створення сучасної, швидкої та зручної освітньої

онлайн-платформи, яка ефективно вирішує поставлені завдання, забезпечує стабільну роботу, легкість супроводу та високий рівень комфорту для кінцевих користувачів.

4.1.1 Сервер-сайд рендеринг (SSR)

Використання сервер-сайд рендерингу стало ключовим рішенням для забезпечення швидкого відображення первинного контенту та оптимізації індексації в пошукових системах. Під час виконання SSR сервер формує готову до відображення HTML-сторінку з усіма необхідними даними, що дозволяє суттєво знизити час першого завантаження на клієнтських пристроях із обмеженими ресурсами та забезпечити користувачам більш плавний досвід переходу між сторінками. При цьому були враховані такі технічні аспекти, як передача початкового стану застосунку для подальшої гідратації на клієнті, а також запобігання розбіжностям між серверною та клієнтською візуалізацією.

У якості базового фреймворку обрано Next.js 14, який органічно поєднує в собі можливості як клієнтського, так і серверного рендерингу. Файлова маршрутизація App Router дозволяє компонувати вкладені макети та сторінки, що сприяє чіткому розмежуванню відповідальності між різними частинами інтерфейсу. Цей механізм у поєднанні з автоматичним розподілом коду гарантує, що кінцевий користувач завантажує лише необхідні для поточної взаємодії з платформою фрагменти, що істотно зменшує навантаження на мережу і прискорює відображення контенту.

Для побудови користувацького інтерфейсу застосовано React 18 у режимі функціональних компонентів із використанням хуків. Завдяки підтримці Concurrent Mode й механізмів Suspense та Error Boundaries інтерфейс здатен асинхронно завантажувати дані, коректно обробляти відмови окремих компонентів та відображати стан завантаження без заморожування основної взаємодії з користувачем. Писання всіх модулів на TypeScript дозволило значно знизити кількість помилок на етапі розробки: завдяки строгій статичній типізації

IDE забезпечує автодоповнення й зручний рефакторинг, а також документує API-інтерфейси, з якими працюють клієнтські модулі.

Систему стилізації організовано з використанням Tailwind CSS, що реалізує утилітарний підхід до CSS. Це дає змогу відмовитися від безлічі індивідуальних класів і швидко створювати адаптивні макети через набір вже готових правил для відступів, кольорів і розмірів шрифтів. Окремим достоинством є вбудована підтримка темної теми, яка реалізується через CSS-змінні, та можливість формувати єдину дизайн-систему без дублювання стилів.

Безпеку та управління доступом до ресурсів забезпечує шар аутентифікації на основі JWT. Після успішної авторизації на сервері формується токен, що містить інформацію про роль і права користувача, — студент, викладач або адміністратор. На клієнті токен зберігається у безпечному сховищі й додається до кожного запиту до API-роутів, реалізованих у Next.js. Механізми автоматичного оновлення токенів і обробки помилок авторизації гарантують захист від несанкціонованого доступу та підвищують стійкість системи до атак.

4.1.2 Реалізація адаптивного дизайну

Для того щоб інтерфейс коректно відображався на широкому спектрі пристроїв, від смартфонів до великих десктопних екранів, обрано підхід «mobile-first» з використанням утилітарних класів Tailwind CSS [29]. Така методика дає змогу задати базову функціональність і структуру макета для мобільних пристроїв, а потім поступово розширювати її для більших екранів, додаючи додаткові елементи й покращення. Філософія прогресивного поліпшення дозволяє забезпечувати доступність ключових можливостей навіть на старих браузерах або при обмежених мережевих ресурсах, одночасно надаючи більш складні візуальні та функціональні рішення для сучасних пристроїв та швидких з’єднань. Чітка організація стилів і політик відображення гарантує, що незалежно від розміру екрана користувач отримає інтуїтивно зрозумілий і приємний досвід взаємодії.

4.1.3 Інтерактивні елементи та анімації

Анімаційні переходи та інтерактивні ефекти реалізовані за допомогою бібліотеки Framer Motion, що дозволяє створювати плавні переходи між сторінками, керовані анімації входу та виходу компонентів, а також відгуки при наведенні курсора. Впроваджені спеціалізовані індикатори завантаження для асинхронних операцій, які інформують користувача про стан виконання запиту, а також системи сповіщень у вигляді тостів для підтвердження успішного завершення дій або повідомлення про помилки з можливістю швидкого відновлення. Таким чином, інтерактивність і візуальна зворотна реакція підвищують залученість користувачів і сприяють підвищенню зручності та прозорості роботи з платформою.

4.2 Архітектура клієнтського застосунку

Архітектура інтерфейсу, зображена на рисунку 4.1, побудована за принципом модульної організації, який забезпечує чіткий розподіл відповідальностей між різними елементами системи. Для досягнення максимальної гнучкості та повторного використання використовуються положення атомарного дизайну, згідно з яким окремі базові елементи, такі як кнопки та поля введення, об’єднуються в більш складні фрагменти, згодом формуючи структуровані розділи інтерфейсу та шаблони сторінок. Системна бібліотека компонентів shadcn/ui гарантує єдину візуальну мову в межах платформи, водночас забезпечуючи дотримання вимог доступності через коректні ARIA-атрибути та навігацію за допомогою клавіатури. Підтримка TypeScript і точне визначення властивостей (props) компонентів сприяють надійному типобезпечному розробленню й полегшують подальше розширення та масштабування інтерфейсу [30].

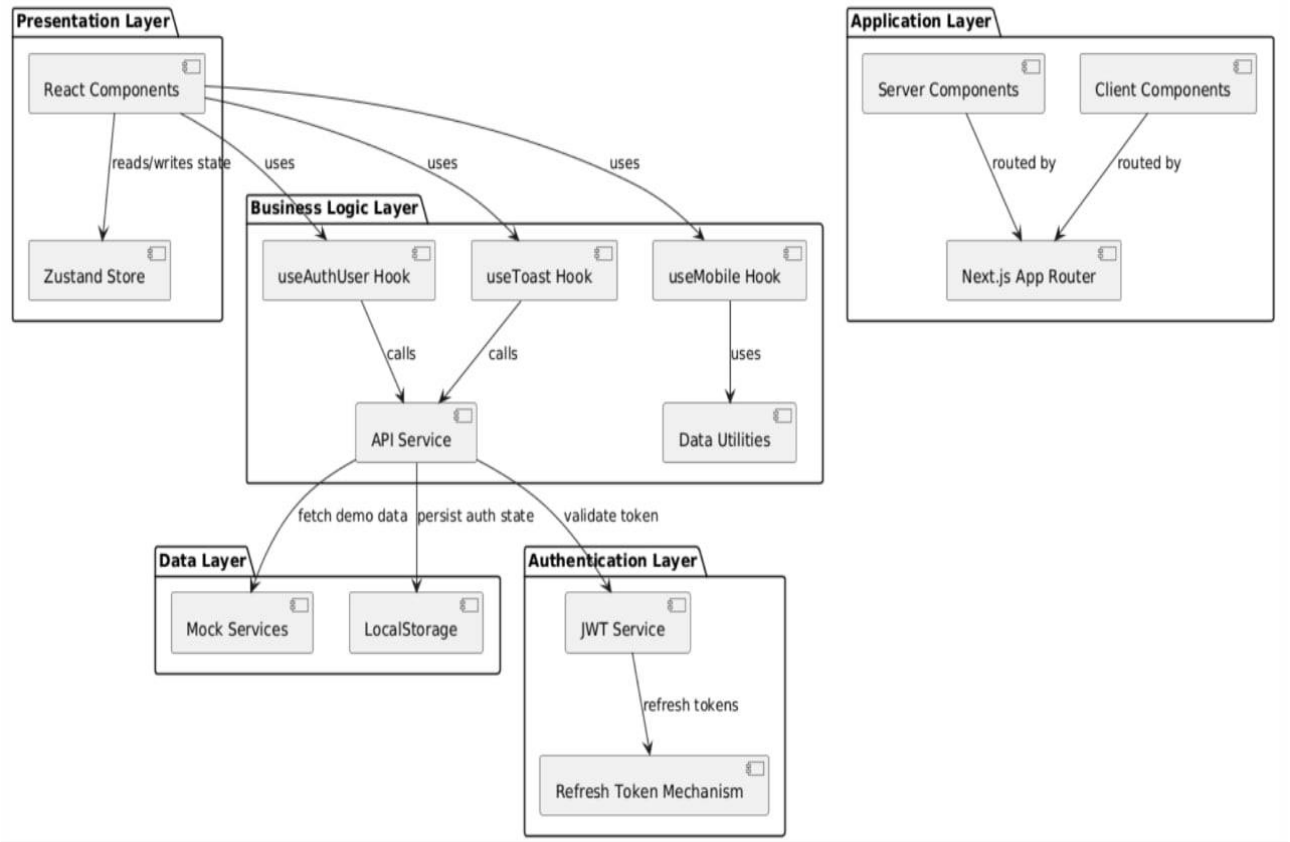


Рисунок 4.1 – Діаграма компонентів клієнтського застосунку

Presentation Layer: складається з компонентів React, які відповідають за відображення й взаємодію з користувачем. Кожен представницький компонент отримує дані через пропси або читає їх зі спільного стану (Zustand).

Application Layer: реалізує маршрутизацію та вхідні точки для серверного й клієнтського рендерингу. Використовуються серверні компоненти Next.js для критичних частин інтерфейсу та клієнтські компоненти для динамічних елементів, які потребують взаємодії з користувачем у реальному часі.

Business Logic Layer: містить кастомні React-хуки (useAuthUser, useToast, useMobile), сервіси для роботи з API, а також утиліти для обробки й трансформації даних.

Data Layer: для демонстраційних цілей застосовано mock-сервіси та LocalStorage. В реальному проєкті цей шар замінюється на звернення до бекенду через API-роути з подальшою обробкою відповідей і помилок.

Authentication Layer: забезпечує генерацію, збереження та перевірку JWT. Впроваджено механізми автоматичного оновлення токенів (refresh tokens) та обмеження часу дії (token expiry), що підвищує загальний рівень безпеки системи.

4.3 Авторизація на платформі

Процес реєстрації через традиційну форму Email/Password, зображений на рисунку 4.2, розпочинається з ініціативи клієнтської частини системи, реалізованої на базі React. Після введення користувачем адреси електронної пошти та пароля формується HTTP-запит типу POST до ендпоінту /auth/register на сервері FastAPI. На сервері отримані дані спершу проходять валідацію відповідно до бізнес-правил (перевірка коректності формату емейлу та достатньої складності пароля). У разі успішного проходження валідації пароль передається у функцію hash_password(), яка застосовує сучасний криптографічний алгоритм хешування (наприклад, bcrypt або Argon2) з метою забезпечення конфіденційності та стійкості до атак типу «груба сила». Після формування хешу пароля сервер виконує операцію збереження нових облікових даних у реляційній базі даних.

У разі успішного виконання транзакції та відсутності конфліктів система створює токен доступу у форматі JWT, застосовуючи функцію create_access_token(user_id), де user_id — унікальний ідентифікатор щойно зареєстрованого користувача. Згенерований токен із позначкою «bearer» повертається клієнту в тілі відповіді з кодом 200 OK. Клієнт, отримавши JWT, зберігає його локально (наприклад, у localStorage) та додає до заголовків кожного наступного запиту до захищених ресурсів у вигляді Authorization: Bearer <JWT>.

Для доступу до захищених ресурсів сервер реалізує механізм проміжного програмного забезпечення (middleware), який виконує функцію decode_and_verify_JWT(token). Під час обробки запиту система перевіряє

цифровий підпис JWT і строк його дії. У разі задовільної валідації запит передається далі до обробника бізнес-логіки, який повертає відповідь з кодом 200 OK та необхідними даними. Якщо ж токен виявився недійсним або простроченим, сервер одразу повертає код 401 Unauthorized із повідомленням "Could not validate credentials", що забезпечує відмову в обслуговуванні неавторизованих запитів.

Окремо розглядається механізм реєстрації та аутентифікації через Google OAuth. У цьому випадку клієнтська частина ініціює авторизаційний запит до Google OAuth 2.0 із передачею параметрів client_id та scope, після чого користувач перенаправляється на сторінку входу Google. Після успішного входу Google генерує ID-токен, який клієнт надсилає на сервер FastAPI за допомогою POST-запиту до ендпоінту /auth/google. На стороні сервера проводиться верифікація отриманого ID-токена через офіційну бібліотеку Google API, що підтверджує достовірність підпису та перевіряє відповідність поля audience.

Отримавши валідний payload із даними користувача (Google ID, емейл, ім’я тощо), сервер виконує UPSERT-операцію над таблицею користувачів, створюючи нового запис або оновлюючи існуючий за ключем google_id. Після завершення транзакції також створюється JWT за допомогою create_access_token(user_id), який повертається клієнту у відповіді. Далі клієнт працює з платформою точно так само, як і при звичайній реєстрації: додає отриманий JWT у заголовок кожного запиту й отримує доступ до захищених ресурсів лише після успішної валідації токена. Таким чином, реалізований підхід поєднує високу зручність для користувача з належним рівнем безпеки та контролем доступу.

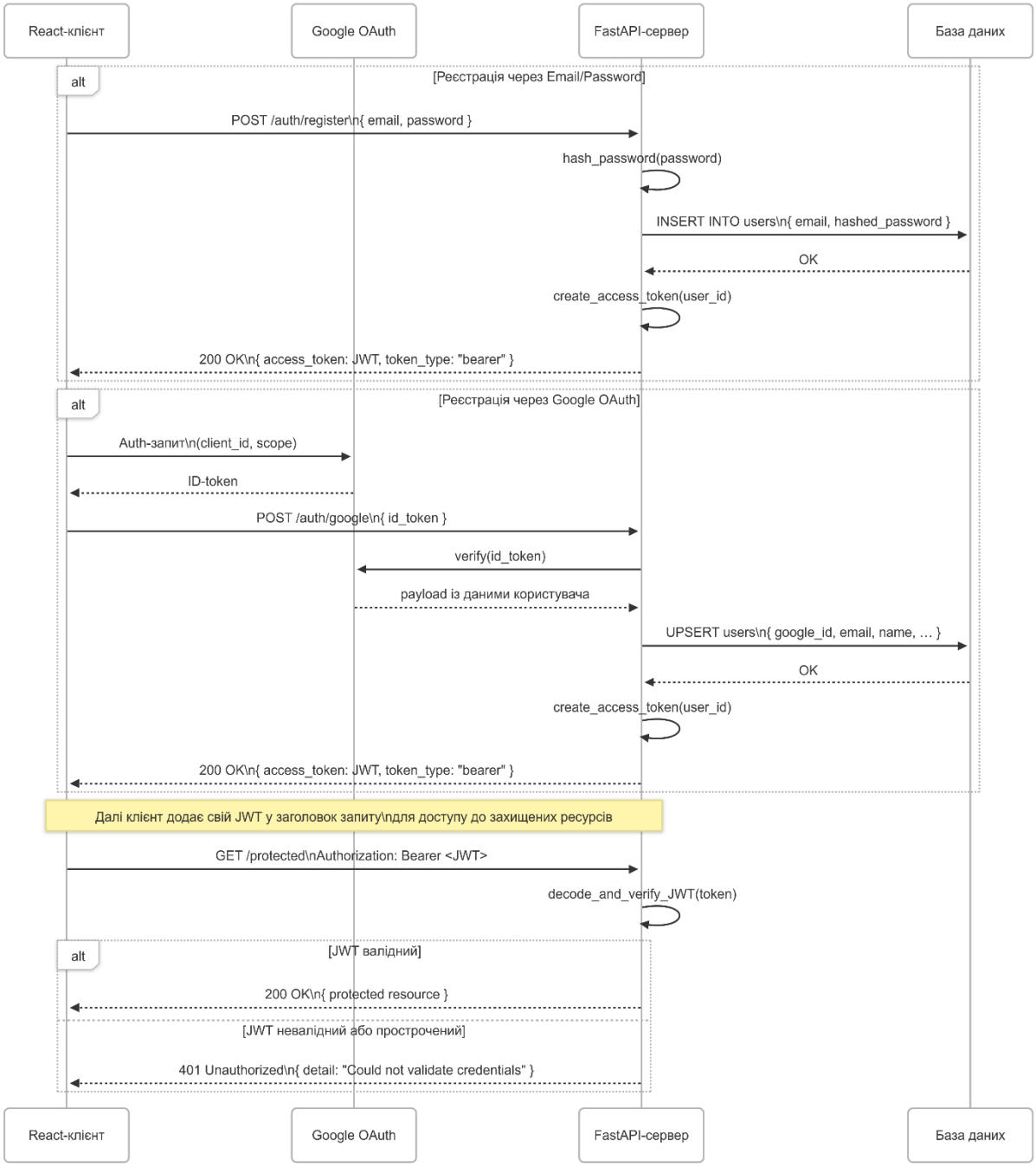


Рисунок 4.2 – Процес реєстрації на платформі

Процес входу розпочинається на клієнтській стороні (React-клієнт) з ініціації OAuth-запиту до сервісу Google. Клієнт надсилає запит типу Auth-запит(client_id, scope), де client_id визначає ідентифікатор застосунку в консолі розробника Google, а scope — набір дозволів (наприклад, доступ до адреси електронної пошти та базових профільних даних). Після успішної аутентифікації

користувача на стороні Google клієнт отримує ID-токен, який містить цифрово підписаний набір даних про користувача [31].

Отримавши ID-токен, React-клієнт формує HTTP-запит типу POST до ендпоінту /auth/google на сервері FastAPI, передаючи його в тілі запиту. Серверна частина приймає токен та виконує його верифікацію за допомогою бібліотеки Google API: перевіряє цифровий підпис, строк дії та відповідність поля audience (ідентифікатора клієнта). У разі позитивного результату верифікації ([Токен коректний]) сервер витягує з payload необхідні дані користувача (Google ID, ім’я, електронну пошту тощо).

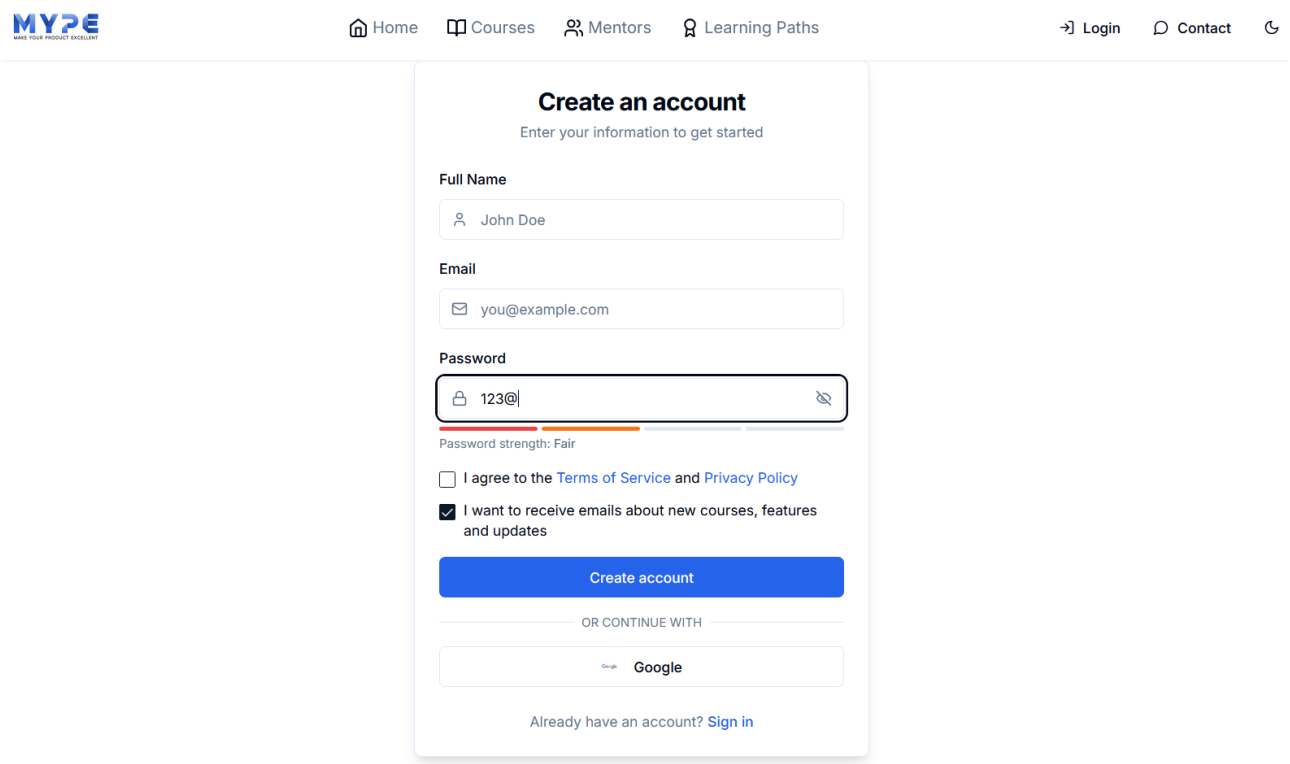


Рисунок 4.3 –Форма реєстрації на платформі

Під час реєстрації користувача система виконує двоступеневу перевірку складності пароля з метою забезпечення належного рівня безпеки облікового запису. На клієнтській стороні, ще до відправлення форми, зображеної на рисунку 4.3, валідація здійснюється за допомогою спеціалізованої бібліотеки, яка аналізує довжину пароля, різноманітність символів (включно з великими й

малими літерами, цифрами та спеціальними знаками) та оцінює його ентропію, відображаючи результат у вигляді індикатора сили (Weak–Fair–Good–Strong). У разі виявлення недостатньої стійкості система рекомендує користувачу посилити пароль і не дозволяє завершити реєстрацію, доки мінімальні порогові значення не будуть виконані.

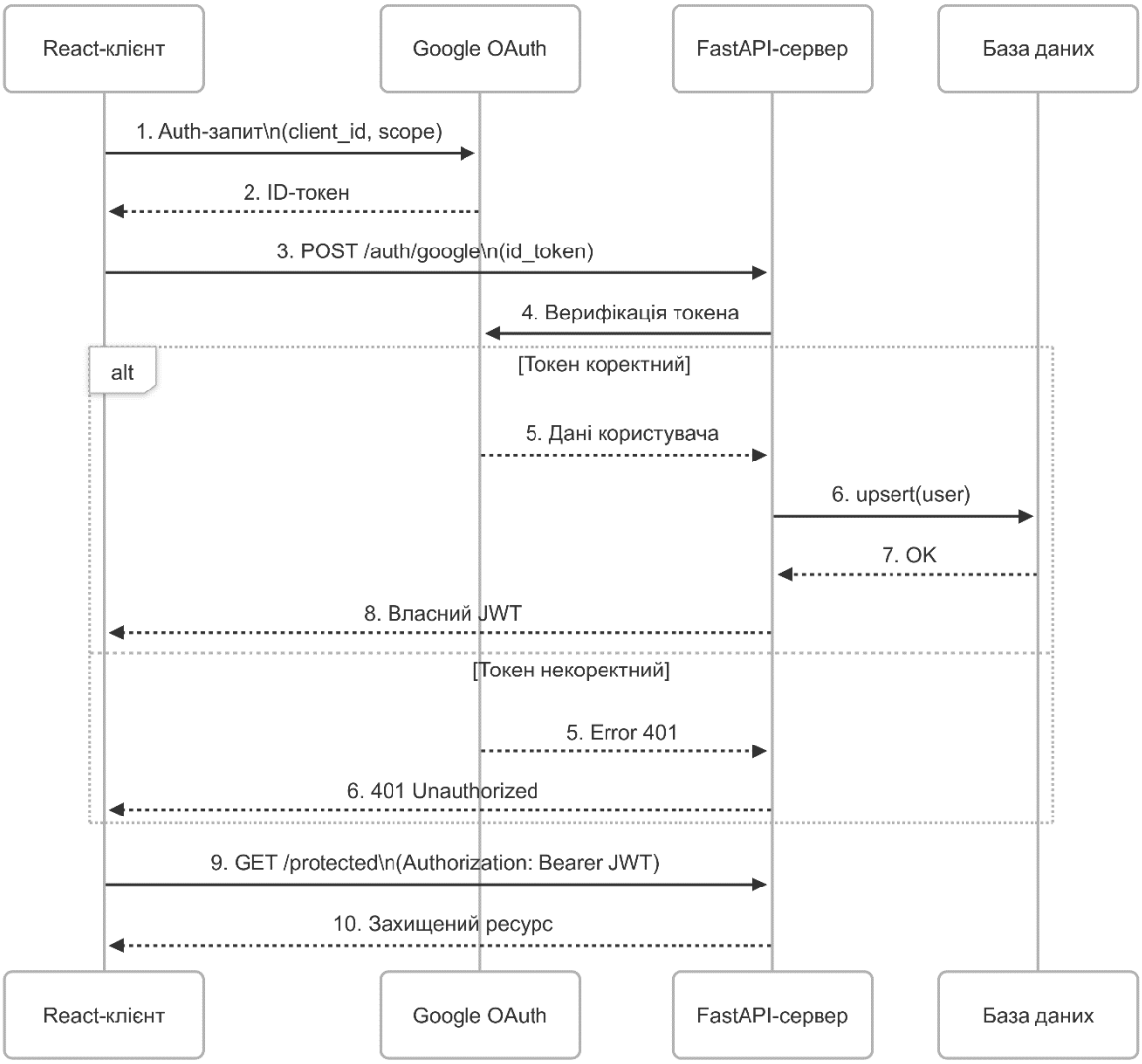


Рисунок 4.4– Процес входження в систему за допомогою Google OAuth

Після підтвердження достовірності інформації сервер здійснює операцію UPSERT над таблицею users у базі даних, рисунок 4.4. Це означає, що за наявності запису з відповідним google_id відбувається його оновлення, а в іншому випадку — створення нового рядка. Після успішного виконання

SQL-запиту (upsert(user)) система генерує власний JWT-токен за допомогою функції create_access_token(user_id) і повертає його клієнту у вигляді відповіді з кодом 200 OK. Цей токен містить інформацію про унікальний ідентифікатор користувача та строки дії, що дозволяє здійснювати подальшу авторизацію на сервері. Процес логіну в систему токеном описаний на рисунку 4.5, форма на стороні клієнтського застосунку наведена на рисунку 4.6.

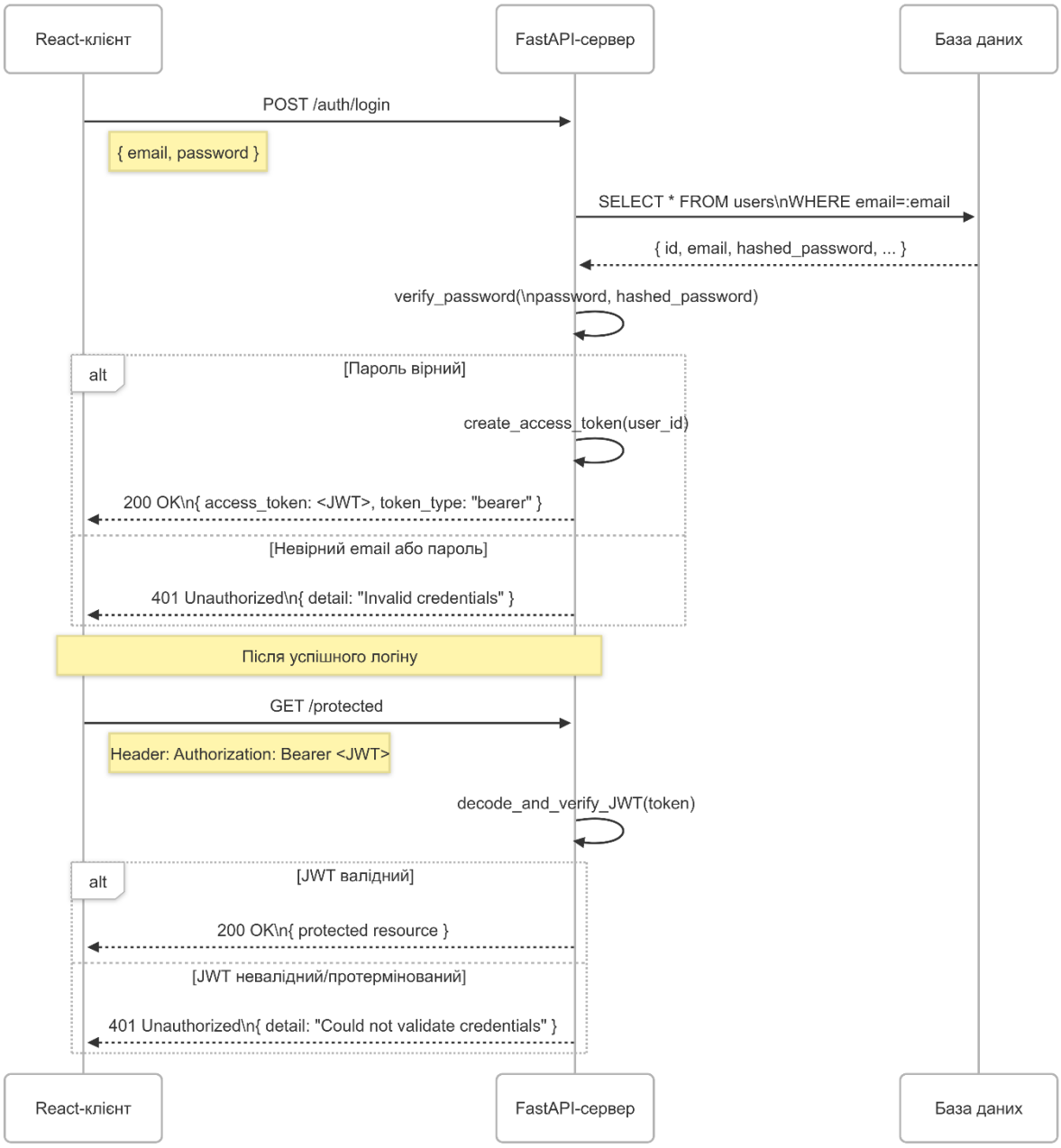


Рисунок 4.5 – Процес входження в систему за допомогою пошти та паролю

Якщо під час верифікації Google-токена було встановлено його недійсність (наприклад, термін дії минув або підпис не співпадає), сервер повертає клієнту HTTP-статус 401 Unauthorized з відповідним повідомленням про помилку. У разі успішного входу клієнт надалі додає отриманий JWT до заголовків кожного захищеного запиту у форматі Authorization: Bearer <JWT>. Серверна частина перед обробкою запиту викликає процедуру декодування та перевірки токена (decode_and_verify_JWT(token)), що гарантує відмову у доступі для неавторизованих запитів і видачу захищеного ресурсу лише для користувачів із дійсними повноваженнями.

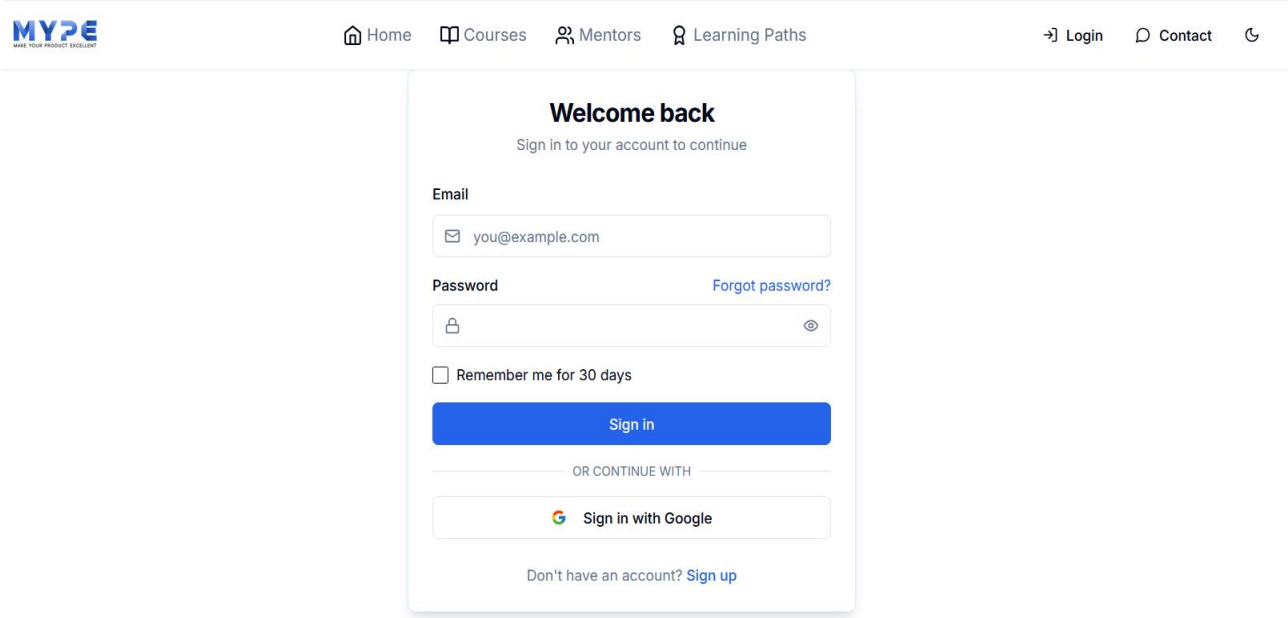


Рисунок 4.6 – Форма входу у персональний кабінет

4.4 Функціональні можливості клієнтського застосунку

4.4.1 Сторінки у вільному доступі

Головна сторінка платформи MYPE функціонує як центральний хаб, що надає користувачам комплексний огляд доступних освітніх можливостей (рис. 4.7).

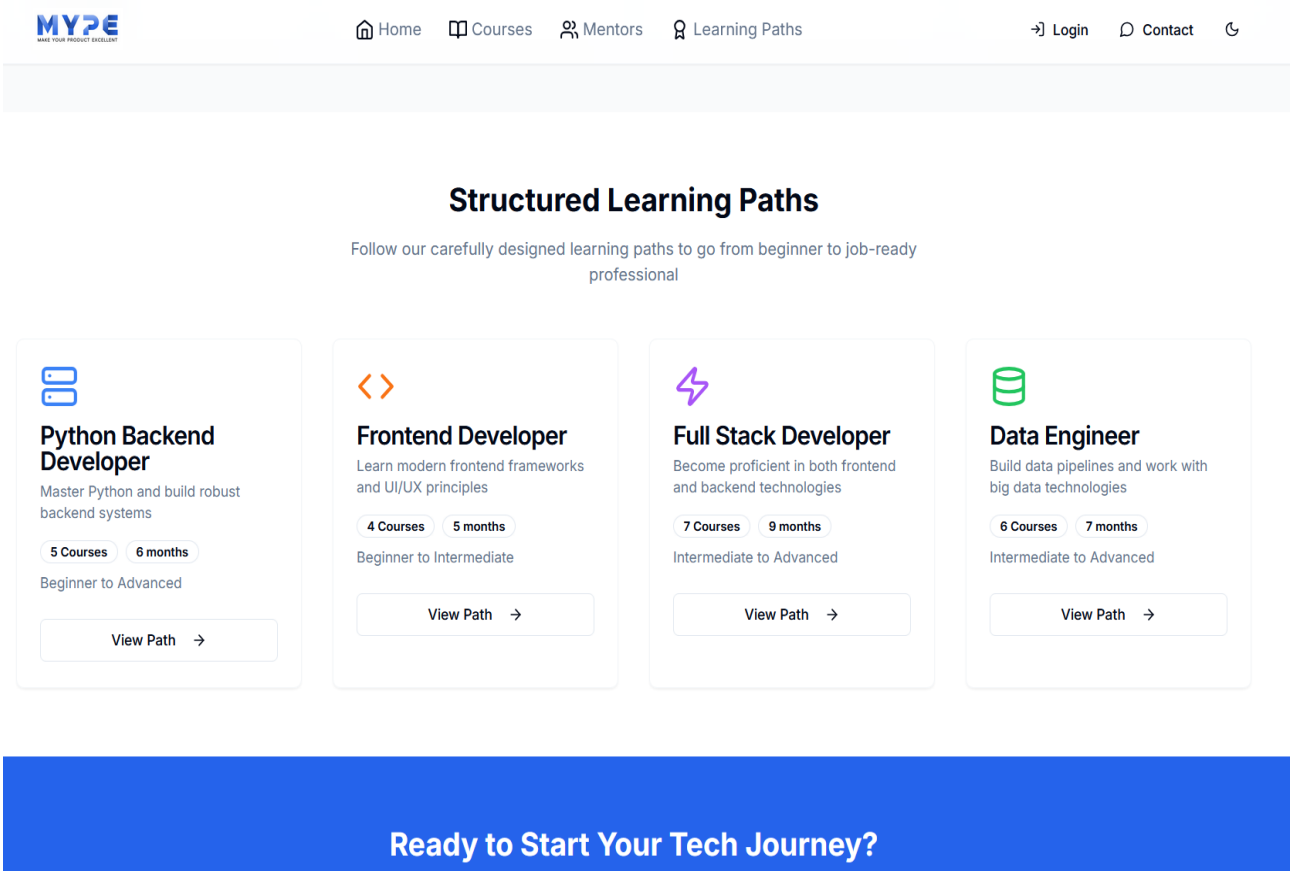


Рисунок 4.7 – Головна сторінка застосунку користувача

Розділ курсів представляє собою структурований каталог освітніх програм, організований за принципами зручності навігації та ефективності пошуку (рис. 4.8).

Кожен курс у каталозі супроводжується детальною інформацією про навчальні цілі, структуру модулів, кваліфікацію інструктора та очікувані результати навчання.

На сторінці (рис 4.9) користувач має змогу ознайомитися з усіма деталями обраного курсу: прочитати вступний опис, переглянути ключові характеристики (тривалість, складність, рейтинг) та вивчити перелік очікуваних навичок.

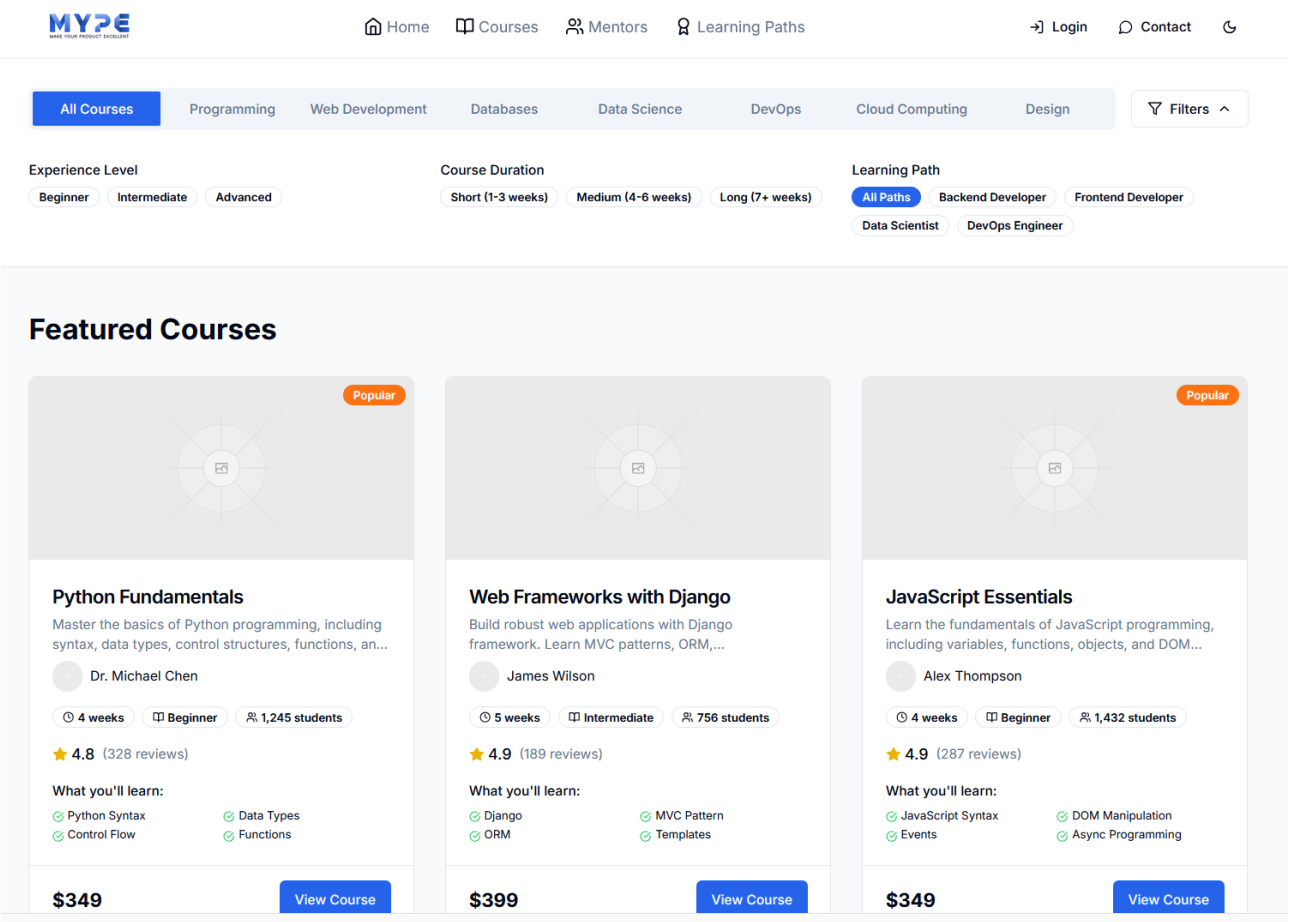


Рисунок 4.8 – Сторінка переліку освітніх курсів

Якщо ви ще не зареєстровані, платформа запропонує вам увійти або створити обліковий запис — після підтвердження доступу відкриються онлайн-лекції, відеоматеріали та додаткові ресурси.

На цій сторінці користувач має змогу ознайомитися з усіма деталями обраного курсу: прочитати вступний опис, переглянути ключові характеристики (тривалість, складність, рейтинг) та вивчити перелік очікуваних навичок. Якщо ви ще не зареєстровані, платформа запропонує вам увійти або створити обліковий запис — після підтвердження доступу відкриються онлайн-лекції, відеоматеріали та додаткові ресурси.

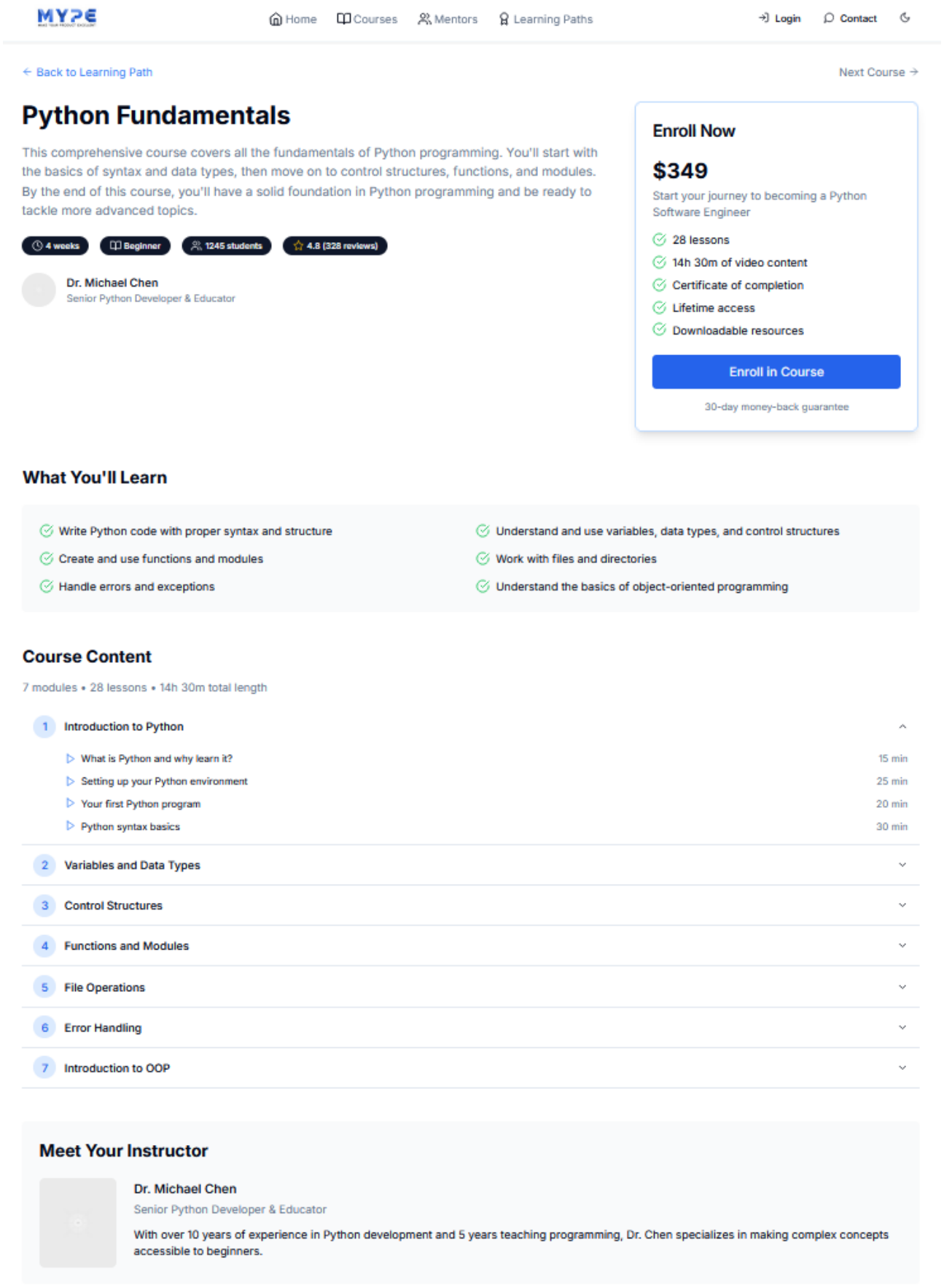


Рисунок 4.9 – Сторінка детальної інформації про освітній курс

4.4.2 Особистий кабінет та персоналізація

Особистий кабінет користувача функціонує як центр управління навчальною діяльністю, надаючи комплексний огляд академічного прогресу та доступних ресурсів (рис. 4.10). Зареєстровані кортувачі можуть відстежувати свій прогрес у активних курсах, переглядати статистику навчальної активності та планувати майбутні освітні цілі.

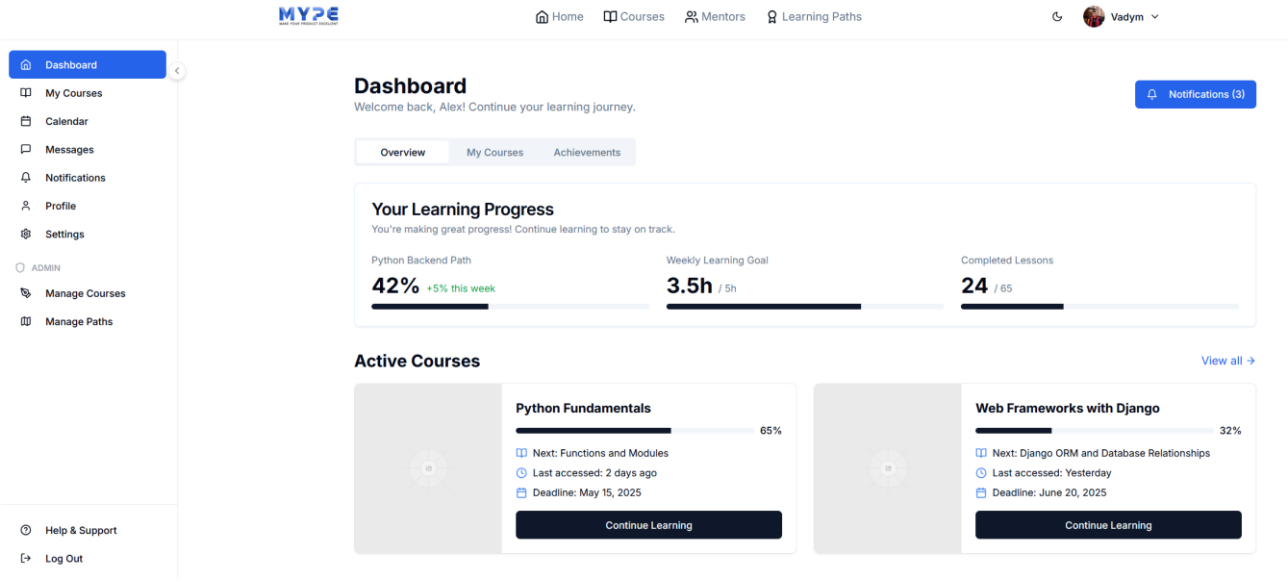


Рисунок 4.10 – Головна сторінка зареєстрованого

Сторінки окремих курсів представляють собою повнофункціональне навчальне середовище, оптимізоване для ефективного засвоєння знань (рис. 4. 11). Зареєстровані користувачі отримують доступ до мультимедійного контенту, включаючи відеолекції високої якості, інтерактивні завдання та додаткові навчальні матеріали. Система прогресивного розкриття контенту забезпечує логічну послідовність вивчення матеріалу та запобігає когнітивному перевантаженню.

Сторінка проходження курсу організована як єдиний робочий простір для студентів і складається з кількох взаємодоповнюючих блоків. У верхній частині розміщено навігаційну стрічку (Dashboard / My Courses / Python Fundamentals), заголовок курсу й стислий опис із датами початку й завершення модуля та

інформацією про наступний дедлайн. Безпосередньо під описом розташовано індикатор прогресу — лінійний бар із відсотком виконання та кількістю пройдених уроків із загального обсягу.

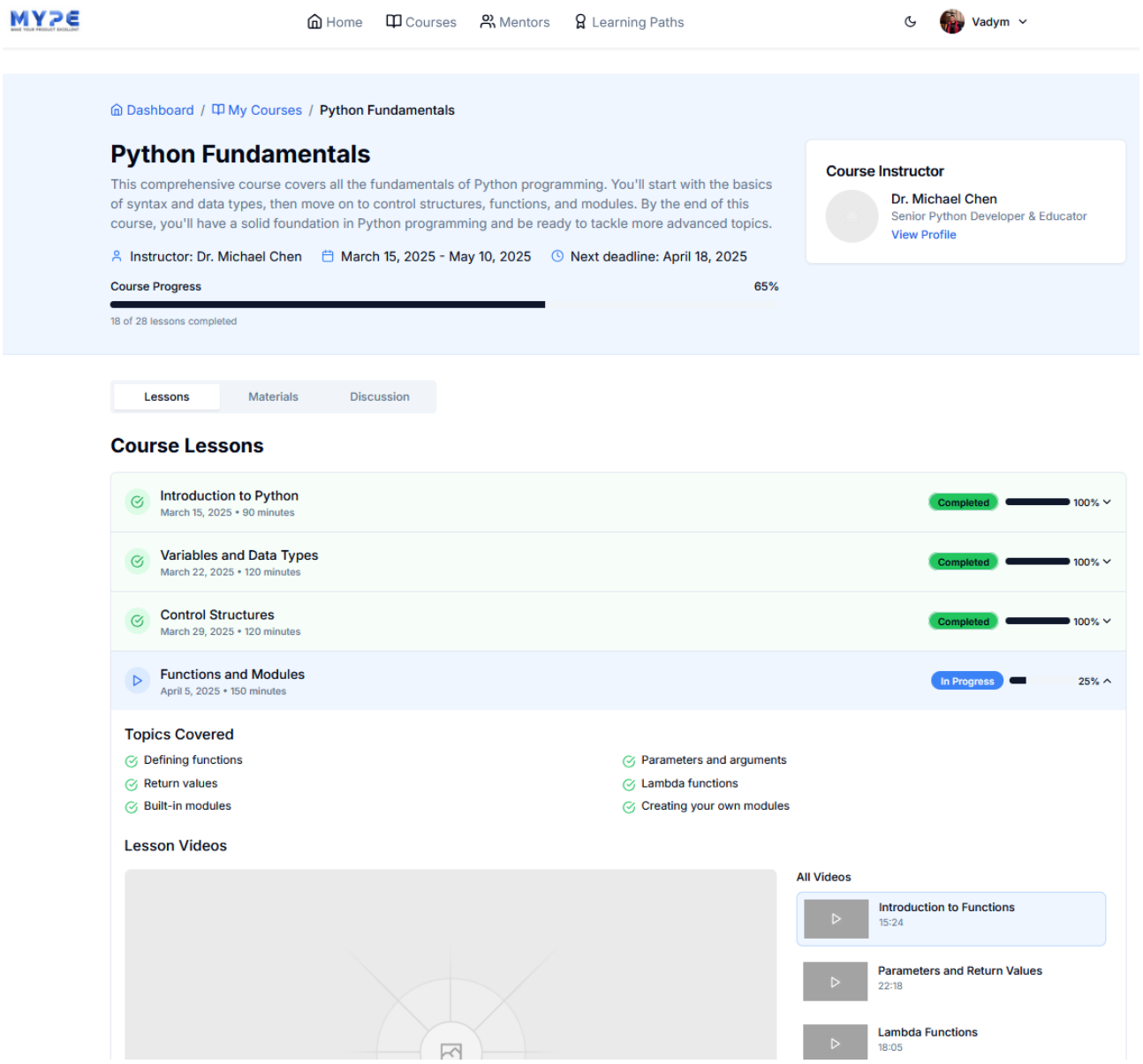


Рисунок 4.11 – Сторінка проходження курсу ч.1

Нижче основної інформації передбачено три вкладки: Lessons, Materials та Discussion. За замовчуванням відкривається вкладка Lessons, де представлено перелік усіх занять із датами та рекомендованою тривалістю. Кожний урок позначається статусом: «Completed» із повною шкалою виконання, «In

Progress» — із поточним відсотком, або «Locked», якщо час відкриття ще не настав. Активний урок розгортається, відкриваючи деталі модуля (Topics Covered), список конкретних тем та віджет із плеєром для перегляду відео.

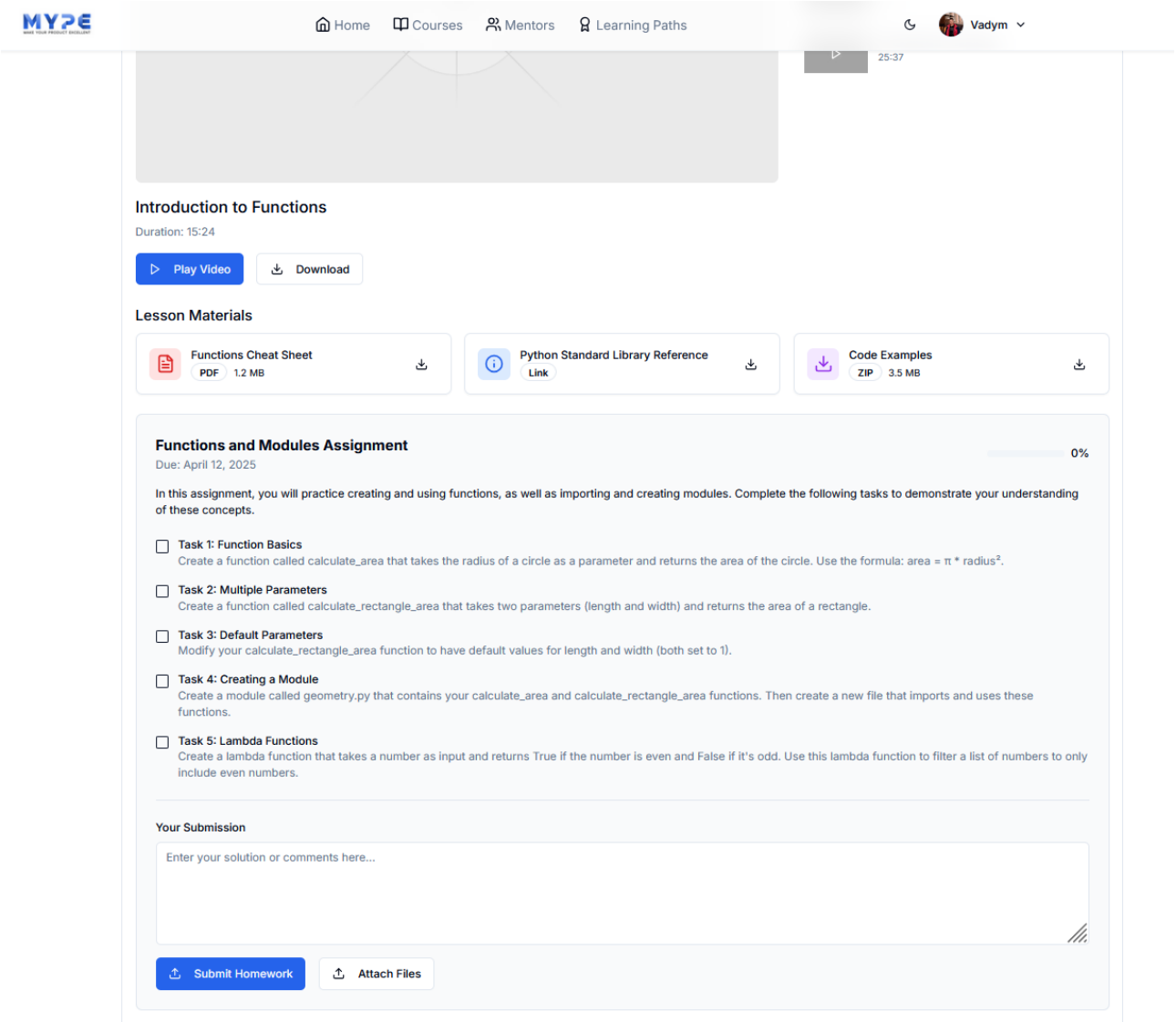


Рисунок 4.12 – Сторінка проходження курсу ч.2 з завданням

У бічній колонці Lessons наведено панель All Videos, яка дозволяє швидко переключатися між програванням різних відео в межах одного уроку.

Перейшовши на вкладку Materials, студент може завантажити всі супровідні ресурси: PDF-конспекти, зовнішні посилання на документацію, архіви з прикладами коду тощо (рис. 4.12).

Під відеоплеєром розміщено секцію Lesson Materials, де кожний файл або посилання супроводжується іконкою типу ресурсу та кнопкою завантаження.

Найактивніший блок сторінки — Functions and Modules Assignment: тут викладач формулює кілька завдань із контрольними точками (Task 1, Task 2 тощо), кожне з яких можна позначити у вигляді чекбоксу після виконання. Дедлайн відображається над списком завдань, а відсоток загальної завершеності підказує рівень прогресу. Після виконання студент залишає коментар у полі Your Submission або прикріплює файли через кнопку Attach Files, а потім відправляє роботу кнопкою Submit Homework.

4.4.3 Адміністрування курсів

Сторінка «Manage Courses» призначена для централізованого перегляду та керування всіма курсами викладача або адміністратора (Рис. 4.13). У лівому бічному меню доступні основні розділи облікового запису, а в секції ADMIN підсвічено пункт Manage Courses.

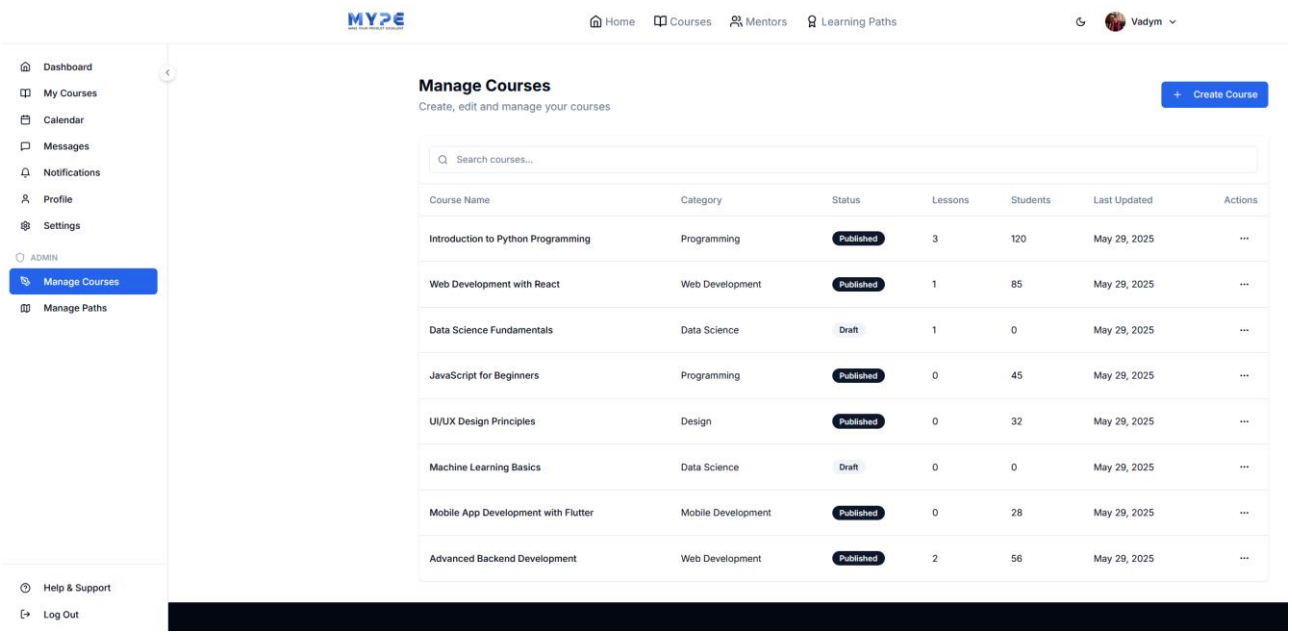


Рисунок 4.13 – Панель адміністрування курсів

Центральна область займає табличний перелік курсів із такими стовпцями:

Course Name – назва курсу;

Category – тематика або напрямок (програмування, дизайн, тощо);

Status – поточний стан курсу (Published, Draft);

Lessons – кількість уроків, що входять до програми;

Students – число зареєстрованих слухачів;

Last Updated – дата останньої модифікації;

Actions – набір кнопок для швидкого редагування, попереднього перегляду (Preview) або видалення курсу.

За допомогою пошукового поля над таблицею можна відфільтрувати курси за назвою. Кнопка Create Course у верхньому правому куті відкриває форму створення нового курсу, а опція Preview у списку дій дозволяє побачити, як курс виглядатиме для слухачів, ще до публікації.

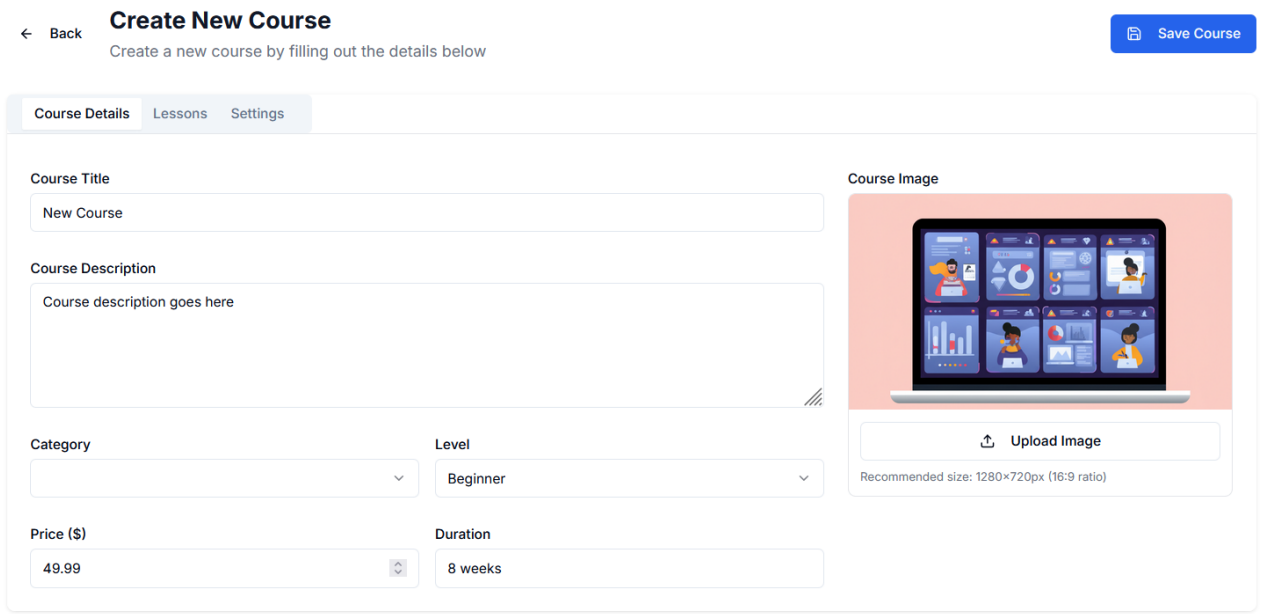


Рисунок 4.14 – Форма створення курсів

Сторінка «Create New Course» реалізована у вигляді багатокрокової форми з вкладками (рис. 4.14). У вкладці Course Details викладач задає загальні метадані:

Course Title – назва курсу;

Course Description – докладний опис змісту та цілей;

Category і Level – тематика та рівень складності;

Price та Duration – вартість і очікувана тривалість курсу;

Course Image – ілюстрація з можливістю завантаження власного зображення.

Перехід на вкладку Lessons дозволяє створити структуру курсу: додати модулі, назви уроків, завантажити відео чи матеріали, а також вказати тип заняття (лекція, тест, практична робота).

У вкладці Settings налаштовуються додаткові параметри: формат доступу (безстроковий, із дедлайнами), можливість коментування, активації сертифікатів та інші опції, що впливають на поведінку курсу в продакшені.

Після заповнення всіх розділів кнопка Save Course зберігає чернетку або, за бажанням, публікує курс одразу. Якщо курс створено в статусі «Draft», його можна попередньо переглянути через функцію Preview, щоб переконатися в коректності оформлення та структури, перш ніж зробити курс доступним для слухачів.

4.5 Висновок до розділу 4

У результаті розробки клієнтської частини застосунку на базі React, TypeScript, Zustand і Next.js було створено структуровану та розширювану фронтенд-архітектуру, що складається з незалежних компонентів із чітко визначеними обов’язками. Компоненти авторизації, реалізовані через Google OAuth та традиційний логін/пароль, інкапсулюють логіку взаємодії з API та обробку помилок, що дозволяє легко підключати нові методи аутентифікації без зміни решти інтерфейсу.

Для централізованого зберігання й керування станом застосовано Zustand, що забезпечує мінімальний об’єм шаблонного коду та високу продуктивність оновлення UI. Використання Next.js дало змогу поєднати серверний рендеринг (SSR) для швидкого початкового завантаження сторінок із клієнтською

навігацією (SPA), а також реалізувати статичне генерування (SSG) окремих маршрутів для покращення SEO та зниження навантаження на сервер.

Адміністрування курсів реалізовано як набір ізольованих сторінок і форм, що взаємодіють із відповідними API-ендоінтами, — створення, редагування, публікація та архівація курсів. Завдяки модульній організації коду та чіткому розподілу відповідальностей між сервісними файлами, хука-ми та UI-компонентами, додавання нових властивостей курсу або інтеграція сторонніх віджетів (наприклад, аналізу прогресу студентів) здійснюється мінімальними зусиллями.

Отже, обрана технологічна стек-комбінація та архітектурні підходи забезпечують не лише зручність розробки й високу швидкість прототипування, а й створюють міцний фундамент для масштабування інтерфейсу та впровадження нових функцій у майбутньому без кардинальних змін у вже існуючій системі.

ВИСНОВКИ

У ході виконання роботи було здійснено комплексний аналіз предметної області, що дозволив виокремити ключові потреби як викладачів, так і студентів при організації освітнього процесу в онлайні; зокрема, з’ясовано вимоги до зручного інтерфейсу створення та проходження курсів, механізмів зворотного зв’язку й гнучкого керування контентом. На основі отриманих даних спроектовано багаторівневу архітектуру системи, що поєднала принципи Onion-модульності та патерн UnitOfWork / Repository у бекенд-шарі й компонентний підхід із централізованим станом у фронтенді.

Розробка серверної частини застосунку з використанням Python і шаблонів Repository забезпечила чітке розмежування бізнес-логіки та інфраструктурних деталей, уніфікований доступ до даних і готовність до впровадження нових сервісів. Паралельно клієнтський застосунок на базі React, Next.js, TypeScript і Zustand забезпечив високу продуктивність, адаптивність та зрозумілу структуру компонентів, включно з підтримкою авторизації через Google та традиційний логін/пароль, а також адміністрування курсів.

В результаті створено повноцінний прототип гібридної освітньої платформи, що відповідає визначеній меті — надати ефективний інструмент для поширення накопичених знань і навчального контенту в зручному форматі. Закладена модульна архітектура та вбудовані механізми масштабування відкривають широкі можливості для подальшого розширення функціоналу, інтеграції нових сервісів і підтримки зростаючого навантаження без кардинальних перебудов системи.

Під час виконання бакалаврської кваліфікаційної роботи всі поставлені задачі були успішно реалізовані, а саме:

– проаналізовано предметну область та обґрунтовано доцільність розробки;

– визначено функціональні й нефункціональні вимоги та спроектовано архітектуру додатку;

– проаналізовано доступні технології та аргументовано вибір стеку для розробки;

– розроблено модулі авторизації (Google OAuth та логін/пароль) і адміністрування курсів;

– розроблено клієнтську частину застосунку з адаптивним інтерфейсом і централізованим керуванням станом.

Таким чином, усі поставлені завдання виконано повною мірою, а пояснювальна записка оформлена відповідно до методичних рекомендацій бакалаврської кваліфікаційної роботи [32].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вадим Ліщинський, В’ячеслав Перепелиця. “Аналіз потреб у безперервному навчанні: функціональні вимоги для освітньої онлайн-платформи” в Матеріалах конференції “LIII Науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2025)”, Вінниця, 2025. URL: https://conferences.vntu.edu.ua/index.php/mn/-mn2025/paper/view/25387 (дата звернення: 05.06.2025).
2. Освітня платформа Udemy. URL: https://www.udemy.com/ (дата звернення: 08.05.2025).
3. Освітня платформа Hillel IT School. URL: https://ithillel.ua/ (дата звернення: 08.05.2025).
4. Освітня платформа Coursera. URL: https://www.coursera.org/ (дата звернення: 08.05.2025).
5. Освітня платформа Prometheus. URL: https://prometheus.org.ua/ (дата звернення: 08.05.2025).
6. Minimum Viable Product (MVP). URL: https://en.wikipedia.org/wiki /Minimum_viable_product (дата звернення: 12.05.2025)
7. Onion архітектура. URL: https://medium.com/expedia-group-tech/onion-architecture-deed8a554423 (дата звернення: 13.05.2025).
8. Infrastructure as Code (Terraform) Documentation. URL: https://www.terraform.io/docs (дата звернення: 16.05.2025)
9. Content Delivery Network (CDN). URL: https://en.wikipedia.org-/wiki/Content_delivery_network (дата зверення: 16.05.2025)
10. Continuous Integration and Continuous Delivery (CI/CD). URL: https://en.wikipedia.org/wiki/CI/CD (дата зверення: 16.05.2025)
11. JSON Web Tokens (JWT) Documentation. URL: https://jwt.io/introduction (дата зверення: 16.05.2025)
12. Dependency Injection URL: https://python-dependency-injector.ets-labs.org (дата звернення: 17.05.2025).

13. Unit Of Work Pattern URL: https://www.cosmicpython.com/book/chapter_06_uow.html (дата звернення: 18.05.2025)
14. Implementing the Repository and Unit of Work Patterns. URL: https://learn.microsoft.com/en-us/archive/msdn-magazine/2010/february/data-points-implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application (дата зверення: 18.05.2025)
15. Onion архітектура. URL: https://softwareengineering.stackexchange.com/questions/450423/how-to-structure-libraries-solution-for-onion-architecture-with-model-view-prese (дата звернення: 18.05.2025).
16. Mock Objects (unittest.mock) Documentation. URL: https://docs.python.org/3/library/unittest.mock.html (дата зверення: 19.05.2025).
17. MySQL Reference Manual. URL: https://dev.mysql.com/doc (дата зверення: 19.05.2025)
18. Unit Of Work Pattern URL: https://www.codeproject.com/Articles/581487/Unit-of-Work-Design-Pattern (дата звернення: 20.05.2025).
19. Object–Relational Mapping (ORM). URL: https://en.wikipedia.org/wiki/Object-relational_mapping (дата звернення: 20.05.2025)
20. Python language documentation. URL: https://docs.python.org/3/ (дата звернення: 20.05.2025).
21. FastAPI Documentation. URL: https://fastapi.tiangolo.com/ (дата звернення: 22.05.2025)
22. OpenAPI Specification. URL: https://swagger.io/specification (дата звернення: 23.05.2025)
23. Pydantic Documentation. URL: https://docs.pydantic.dev (дата зверення: 23.05.2025)
24. Pytest Documentation. URL: https://docs.pytest.org/en/stable/ (дата зверення: 24.05.2025
25. React Documentation. URL: https://reactjs.org/docs/getting-started.html (дата звернення: 24.05.2025)

26. Server-Side Rendering (Next.js) Documentation. URL: https://nextjs.org/docs/-basic-features/pages#server-side-rendering (дата звернення: 26.05.2025)
27. Next.js Documentation. URL: https://nextjs.org/docs (дата звернення: 28.05.2025)
28. Zustand Documentation. URL: https://github.com/pmndrs/zustand (дата звернення: 30.05.2025)
29. Tailwind CSS Documentation. URL: https://tailwindcss.com/docs (дата зверення: 30.05.2025)
30. TypeScript Language Documentation. URL: https://www.typescriptlang.org-/docs/ (дата звернення: 30.05.2025)
31. Google OAuth 2.0 Documentation. URL: https://developers.google.com/identity/protocols/oauth2 (дата зверення: 30.05.2025)
32. Методичні вказівки до виконання бакалаврської кваліфікаційної роботи для студентів денної та заочної форм навчання спеціальності 122 - «Комп`ютерні науки». URL: https://iq.vntu.edu.ua/method/ (дата звернення: 30.05.2025)

ДОДАТКИ

ДОДАТОК А (обов'язковий)
ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка WEB-платформи для організації освітніх курсів

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 0,27 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Перепелиця В.І. асис. каф.КН
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Ліщинський В.І.
(підпис) (прізвище, ініціали)

ДОДАТОК Б (обов’язковий)
ФРАГМЕНТ ЛІСТИНГУ ПРОГРАМНОГО КОДУ

БЕКЕНД ЧАСТИНИ

```
from datetime import datetime, timedelta, timezone
from typing import Annotated

import jwt
from dependency_injector.wiring import Provide, inject
from fastapi import APIRouter, Depends, HTTPException
from fastapi.security import OAuth2PasswordBearer
from passlib.context import CryptContext
from pydantic import BaseModel

from mype_backend.models.config.loader import settings
from mype_backend.models.containers import ApplicationContainer
from mype_backend.models.user_service import UserService

# to get a string like this run:
# openssl rand -hex 32
ALGORITHM = 'HS256'
ACCESS_TOKEN_EXPIRE_MINUTES = 30

oauth2_scheme = OAuth2PasswordBearer(tokenUrl='token')
pwd_context = CryptContext(schemes=['bcrypt'], deprecated='auto')

def get_current_user(token: str = Depends(oauth2_scheme)):
    try:
        payload = jwt.decode(token, settings.SECRET_KEY, algorithms=[ALGORITHM])
        user_email: str = payload.get('sub')
        if user_email is None:
            raise HTTPException(status_code=401, detail='Invalid authentication credentials')
        return payload
    except jwt.PyJWTError:
```

```
            raise HTTPException(status_code=401, detail='Invalid authentication credentials')


        def create_access_token(data: dict, expires_delta: timedelta | None = None):
            to_encode = data.copy()
            if expires_delta:
                expire = datetime.now(timezone.utc) + expires_delta
            else:
                expire = datetime.now(timezone.utc) + timedelta(days=7)
            to_encode.update({'exp': expire})
            encoded_jwt = jwt.encode(to_encode, settings.SECRET_KEY,
algorithm=ALGORITHM)
            return encoded_jwt

        router = APIRouter(
            tags=['auth'],
            prefix='/auth',
        )

        class Token(BaseModel):
            token: str
            token_type: str

        # Define token schema
        class TokenRequest(BaseModel):
            access_token: str

        class TokenData(BaseModel):
            username: str | None = None

        # Define the endpoint for Google token verification
        @router.post('/google/token')
        @inject
        async def google_auth(
            access_token: TokenRequest,
```

```
    user_service: Annotated[UserService,
Depends(Provide[ApplicationContainer.services.user_service])],
) -> Token:
    user = await user_service.return_user_on_login_sign_up(access_token)

    access_token_expires = timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    user_role = 'Student'  # UPDATE - create role logic
    user_data = {
       'name': user.name,
       'surname': user.surname,
       'email': user.email,
       'image_link': user.image_link,
       'role': user_role,
    }
    access_token = create_access_token(data={'sub': user_data},
expires_delta=access_token_expires)
    return Token(token=access_token, token_type='bearer')
```

КЛІЄНТСЬКА ЧАСТИНА – ФОРМА ВХОДУ В СИСТЕМУ

```
"use client"

import type React from "react"

import { useState, useEffect } from "react"
import Link from "next/link"
import { useRouter } from "next/navigation"
import { motion } from "framer-motion"
import { Button } from "@/components/ui/button"
import { Input } from "@/components/ui/input"
import { Label } from "@/components/ui/label"
import { Checkbox } from "@/components/ui/checkbox"
import { Card, CardContent, CardDescription, CardFooter, CardHeader, CardTitle } from
"@/components/ui/card"
import { Header } from "@/components/header"
```

```
import { Footer } from "@/components/footer"
import { Eye, EyeOff, Mail, Lock } from "lucide-react"
import { useAuthStore } from "@/store/auth-store"
import { GoogleOAuthProvider, useGoogleLogin } from "@react-oauth/google"
import { initializeAuthFromToken } from "@/lib/token-service"

// Custom Google Login Button Component
function GoogleLoginButton() {
  const router = useRouter()
  const { loginWithGoogle } = useAuthStore()
  const [isLoading, setIsLoading] = useState(false)
  const [error, setError] = useState("")

  // Use the Google login hook from @react-oauth/google
  const googleLogin = useGoogleLogin({
    onSuccess: async (response) => {
      setIsLoading(true)
      try {
        // Call your backend to create/update user and session
        await loginWithGoogle(response.access_token)

        // Redirect to dashboard
        router.push("/dashboard")
      } catch (err) {
        console.error("Google login error:", err)
        setError("Failed to complete Google authentication")
      } finally {
        setIsLoading(false)
      }
    },
    onError: (errorResponse) => {
      console.error("Google login error:", errorResponse)
      setError("Google login failed")
    },
  })
```

```
        return (
          <>
            {error && (
              <div className="p-3 rounded-md bg-red-50 dark:bg-red-900/20 text-red-500 dark:text-red-400 text-sm mb-4">
                {error}
              </div>
            )}
            <Button
              type="button"
              variant="outline"
              className="w-full flex items-center justify-center"
              onClick={() => googleLogin()}
              disabled={isLoading}
            >
              {isLoading ? (
                <div className="flex items-center justify-center">
                  <div className="h-4 w-4 border-2 border-primary border-t-transparent rounded-full animate-spin mr-2"></div>
                  Signing in with Google...
                </div>
              ) : (
                <>
                  <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 48 48" className="w-5 h-5 mr-2">
                    <path
                      fill="#FFC107"
                      d="M43.611,20.083H42V20H24v8h11.303c-1.649,4.657-6.08,8-11.303,8c-6.627,0-12-5.373-12-12c0-6.627,5.373-12,12-12c3.059,0,5.842,1.154,7.961,3.039l5.657-5.657C34.046,6.053,29.268,4,24,4C12.955,4,4,12.955,4,24c0,11.045,8.955,20,20,20c11.045,0,20-8.955,20-20C44,22.659,43.862,21.35,43.611,20.083z"
                    />
                    <path
                      fill="#FF3D00"
```

```
d="M6.306,14.691l6.571,4.819C14.655,15.108,18.961,12,24,12c3.059,0,5.842,1.154,7.961,3.039l5
.657-5.657C34.046,6.053,29.268,4,24,4C16.318,4,9.656,8.337,6.306,14.691z"
                    />
                    <path
                      fill="#4CAF50"
                      d="M24,44c5.166,0,9.86-1.977,13.409-5.192l-6.19-
5.238C29.211,35.091,26.715,36,24,36c-5.202,0-9.619-3.317-11.283-7.946l-
6.522,5.025C9.505,39.556,16.227,44,24,44z"
                    />
                    <path
                      fill="#1976D2"
                      d="M43.611,20.083H42V20H24v8h11.303c-0.792,2.237-2.231,4.166-
4.087,5.571c0.001-0.001,0.002-0.001,0.003-
0.002l6.19,5.238C36.971,39.205,44,34,44,24C44,22.659,43.862,21.35,43.611,20.083z"
                    />
                  </svg>
                  Sign in with Google
                </>
              )}
            </Button>
          </>
        )
      }

      export default function LoginPage() {
        const router = useRouter()
        const { login, isAuthenticated, isLoading, error, clearError } = useAuthStore()

        const [email, setEmail] = useState("")
        const [password, setPassword] = useState("")
        const [showPassword, setShowPassword] = useState(false)
        const [rememberMe, setRememberMe] = useState(false)
        const [localError, setLocalError] = useState("")
```

```
// Google Client ID - Replace with your actual Google Client ID
const googleClientId = process.env.NEXT_PUBLIC_GOOGLE_CLIENT_ID || "YOUR_GOOGLE_CLIENT_ID"

// Initialize auth from token on page load
useEffect(() => {
  initializeAuthFromToken()
}, [])

// Redirect if already authenticated
useEffect(() => {
  if (isAuthenticated) {
    router.push("/dashboard")
  }
}, [isAuthenticated, router])

// Clear errors when unmounting
useEffect(() => {
  return () => {
    clearError()
  }
}, [clearError])

// Update local error when store error changes
useEffect(() => {
  if (error) {
    setLocalError(error)
  }
}, [error])

const handleSubmit = async (e: React.FormEvent) => {
  e.preventDefault()
  setLocalError("")

  if (!email || !password) {
```

```
      setLocalError("Please fill in all fields")
      return
    }

    try {
      await login(email, password)
      console.log('Login finished')
      // The router.push is handled by the useEffect that watches isAuthenticated
    } catch (err) {
      // Error is handled in the store
    }
  }

  return (
    <div className="min-h-screen bg-background flex flex-col">
      <Header />

      <main className="flex-grow flex items-center justify-center px-4 py-16">
        <div className="w-full max-w-md">
          <motion.div initial={{ opacity: 0, y: 20 }} animate={{ opacity: 1, y: 0 }} transition={{ duration: 0.5 }}>
            <Card className="border-blue-100 dark:border-blue-900 shadow-lg">
              <CardHeader className="space-y-1">
                <CardTitle className="text-2xl font-bold text-center">Welcome back</CardTitle>
                <CardDescription className="text-center">Sign in to your account to continue</CardDescription>
              </CardHeader>

              <CardContent className="space-y-4">
                {(localError || error) && (
                  <div className="p-3 rounded-md bg-red-50 dark:bg-red-900/20 text-red-500 dark:text-red-400 text-sm">
                    {localError || error}
                  </div>
```

```
)}

<form onSubmit={handleSubmit} className="space-y-4">
 <div className="space-y-2">
  <Label htmlFor="email">Email</Label>
  <div className="relative">
   <Mail className="absolute left-3 top-3 h-4 w-4 text-muted-foreground" />
   <Input
    id="email"
    type="email"
    placeholder="you@example.com"
    className="pl-10"
    value={email}
    onChange={(e) => setEmail(e.target.value)}
    disabled={isLoading}
    required
   />
  </div>
 </div>

 <div className="space-y-2">
  <div className="flex items-center justify-between">
   <Label htmlFor="password">Password</Label>
   <Link
    href="/auth/reset-password"
    className="text-sm  text-blue-600  hover:text-blue-700  dark:text-blue-400
dark:hover:text-blue-300"
   >
    Forgot password?
   </Link>
  </div>
  <div className="relative">
   <Lock className="absolute left-3 top-3 h-4 w-4 text-muted-foreground" />
   <Input
    id="password"
```

```
                    type={showPassword ? "text" : "password"}
                    className="pl-10"
                    value={password}
                    onChange={(e) => setPassword(e.target.value)}
                    disabled={isLoading}
                    required
                  />
                  <button
                    type="button"
                    className="absolute right-3 top-3 text-muted-foreground hover:text-foreground"
                    onClick={() => setShowPassword(!showPassword)}
                  >
                    {showPassword ? <EyeOff className="h-4 w-4" /> : <Eye className="h-4 w-4" />}
                    <span className="sr-only">{showPassword ? "Hide password" : "Show password"}</span>
                  </button>
                </div>
              </div>

              <div className="flex items-center space-x-2">
                <Checkbox
                  id="remember"
                  checked={rememberMe}
                  onCheckedChange={(checked) => setRememberMe(checked as boolean)}
                />
                <Label htmlFor="remember" className="text-sm font-normal">
                  Remember me for 30 days
                </Label>
              </div>

              <Button type="submit" className="w-full bg-blue-600 hover:bg-blue-700" disabled={isLoading}>
                {isLoading ? (
```

```
                    <div className="flex items-center justify-center">
                     <div className="h-4 w-4 border-2 border-white border-t-transparent rounded-full animate-spin mr-2"></div>
                      Signing in...
                    </div>
                   ) : (
                    "Sign in"
                   )}
                  </Button>
                 </form>

                 <div className="relative">
                  <div className="absolute inset-0 flex items-center">
                   <div className="w-full border-t border-gray-200 dark:border-gray-700"></div>
                  </div>
                  <div className="relative flex justify-center text-xs uppercase">
                   <span className="bg-background px-2 text-muted-foreground">Or continue with</span>
                  </div>
                 </div>

                 <GoogleOAuthProvider clientId={googleClientId}>
                  <GoogleLoginButton />
                 </GoogleOAuthProvider>
                </CardContent>

                <CardFooter className="flex justify-center">
                 <p className="text-sm text-muted-foreground">
                  Don&apos;t have an account?{" "}
                  <Link
                   href="/auth/signup"
                   className="text-blue-600 hover:text-blue-700 dark:text-blue-400 dark:hover:text-blue-300 font-medium"
                  >
```

```
                    Sign up
                  </Link>
                </p>
              </CardFooter>
            </Card>
          </motion.div>
        </div>
      </main>

      <Footer />
    </div>
  )
}
```

КЛІЄНТСЬКА ЧАСТИНА – ЗБЕРЕЖЕННЯ СТАНУ КОРИСТУВАЧА

```
import { create } from "zustand"
import { persist, devtools} from "zustand/middleware"
import { setToken, removeToken, getUserInfoFromToken } from "@/lib/token-service"
import { useUserInfoStore } from "@/store/user-info-store"

const APP_URL = process.env.APP_URL || "http://localhost:8000"

interface AuthState {
  isAuthenticated: boolean
  isLoading: boolean
  error: string | null

  // Actions
  login: (email: string, password: string) => Promise<void>
  loginWithGoogle: (accessToken: string) => Promise<void>
  signup: (name: string, email: string, password: string, receiveUpdates: boolean) =>
Promise<void>
  logout: () => void
```

```
  clearError: () => void
  setAuthenticated: (isAuthenticated: boolean) => void
}

export const useAuthStore = create<AuthState>()(
  devtools(persist(
    (set, get) => ({
      isAuthenticated: false,
      isLoading: false,
      error: null,

      login: async (email: string, password: string) => {
        set({ isLoading: true, error: null })
        try {
          // Call your authentication API
          const response = await fetch("/api/auth/login/token ", {
            method: "POST",
            headers: {
              "Content-Type": "application/json",
            },
            body: JSON.stringify({ email, password }),
          })

          const data = await response.json()

          if (!response.ok) {
            throw new Error(data.message || "Login failed")
          }

          // Store the JWT token
          setToken(data.token)

          // Extract user info from token and store it
          const userInfo = getUserInfoFromToken(data.token)
          if (userInfo) {
```

```
      useUserInfoStore.getState().setUserInfo(userInfo)
    }

    set({
      isAuthenticated: true,
      isLoading: false,
    })
  } catch (error: any) {
    set({
      error: error.message || "Login failed",
      isLoading: false,
    })
    throw error
  }
},

loginWithGoogle: async (accessToken: string) => {
  set({ isLoading: true, error: null })
  try {
    // Send the access token to your backend for validation
    const response = await fetch(APP_URL + "/api/auth/google/token", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({ access_token: accessToken }),
    })

    const data = await response.json()
    if (!response.ok) {
      throw new Error(data.message || "Google login failed")
    }

    // Store the JWT token
    setToken(data.token)
```

```
        // Extract user info from token and store it
        const userInfo = getUserInfoFromToken(data.token)
        if (userInfo) {
          useUserInfoStore.getState().setUserInfo(userInfo)
        }

        set({
          isAuthenticated: true,
          isLoading: false,
        })
      } catch (error: any) {
        set({
          error: error.message || "Google login failed",
          isLoading: false,
        })
        throw error
      }
    },

    signup: async (name: string, email: string, password: string, receiveUpdates: boolean)
=> {
      set({ isLoading: true, error: null })
      try {
        const response = await fetch("/api/auth/signup", {
          method: "POST",
          headers: {
            "Content-Type": "application/json",
          },
          body: JSON.stringify({
            name,
            email,
            password,
            receiveUpdates,
          }),
```

```
    })

    const data = await response.json()

    if (!response.ok) {
      throw new Error(data.message || "Signup failed")
    }

    set({ isLoading: false })
    return data
  } catch (error: any) {
    set({
      error: error.message || "Signup failed",
      isLoading: false,
    })
    throw error
  }
},

logout: () => {
  // Remove the token
  removeToken()

  // Clear user info
  useUserInfoStore.getState().clearUserInfo()

  // Update auth state
  set({ isAuthenticated: false })
},

clearError: () => {
  set({ error: null })
},

setAuthenticated: (isAuthenticated: boolean) => {
```

```
      set({ isAuthenticated })
    },
  }),
  {
    name: "auth-storage",
    partialize: (state) => ({ isAuthenticated: state.isAuthenticated }),
  },
 )),
)
```

ДОДАТОК В (обов'язковий)

ГРАФІЧНА ЧАСТИНА

РОЗРОБКА WEB-ПЛАТФОРМИ ДЛЯ ОРГАНІЗАЦІЇ ОСВІТНІХ КУРСІВ

Виконав: студент 4 курсу, групи 4КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Ліщинський В. І.
(прізвище та ініціали)
Керівник: доктор філософії, асистент каф. КН
Перепелиця В. І.
(прізвище та ініціали)
«03» червня 2025 р.

Вінниця ВНТУ – 2025 рік

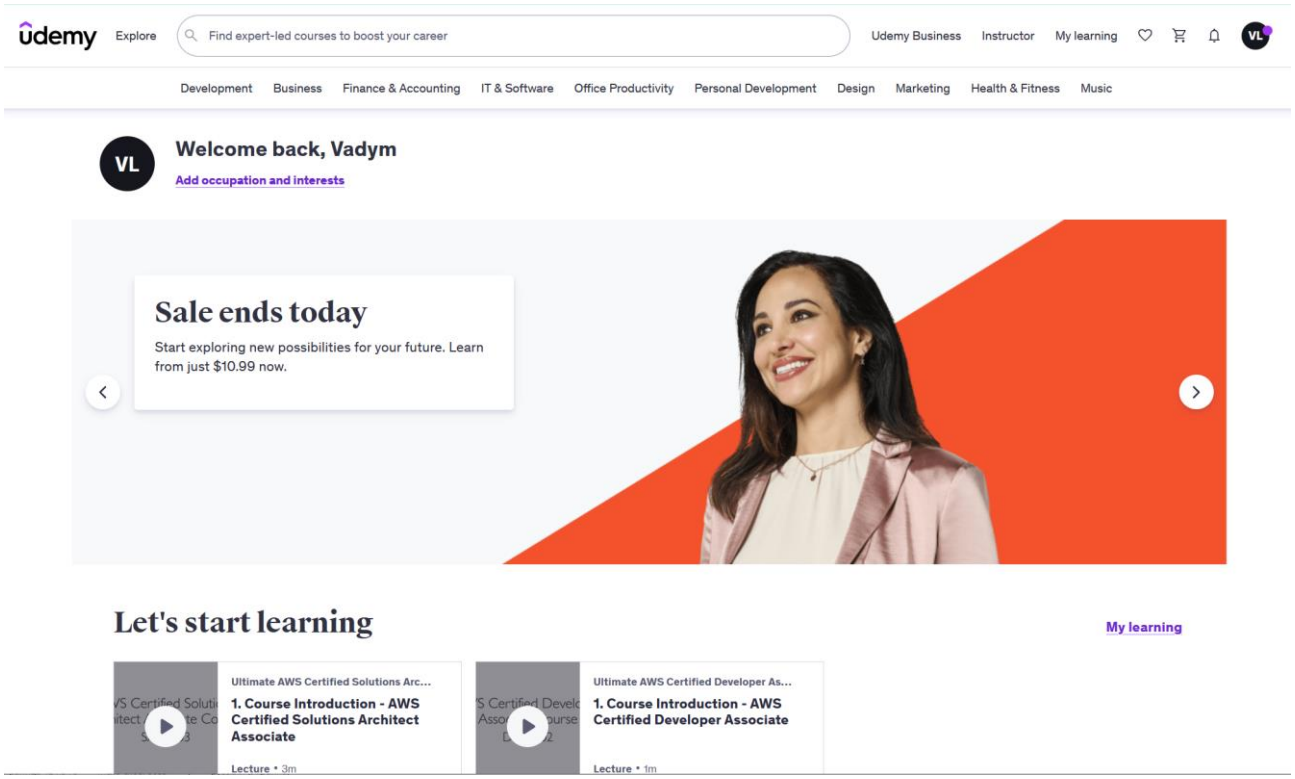


Рисунок В.1 – Головна сторінка кабінету Udemy

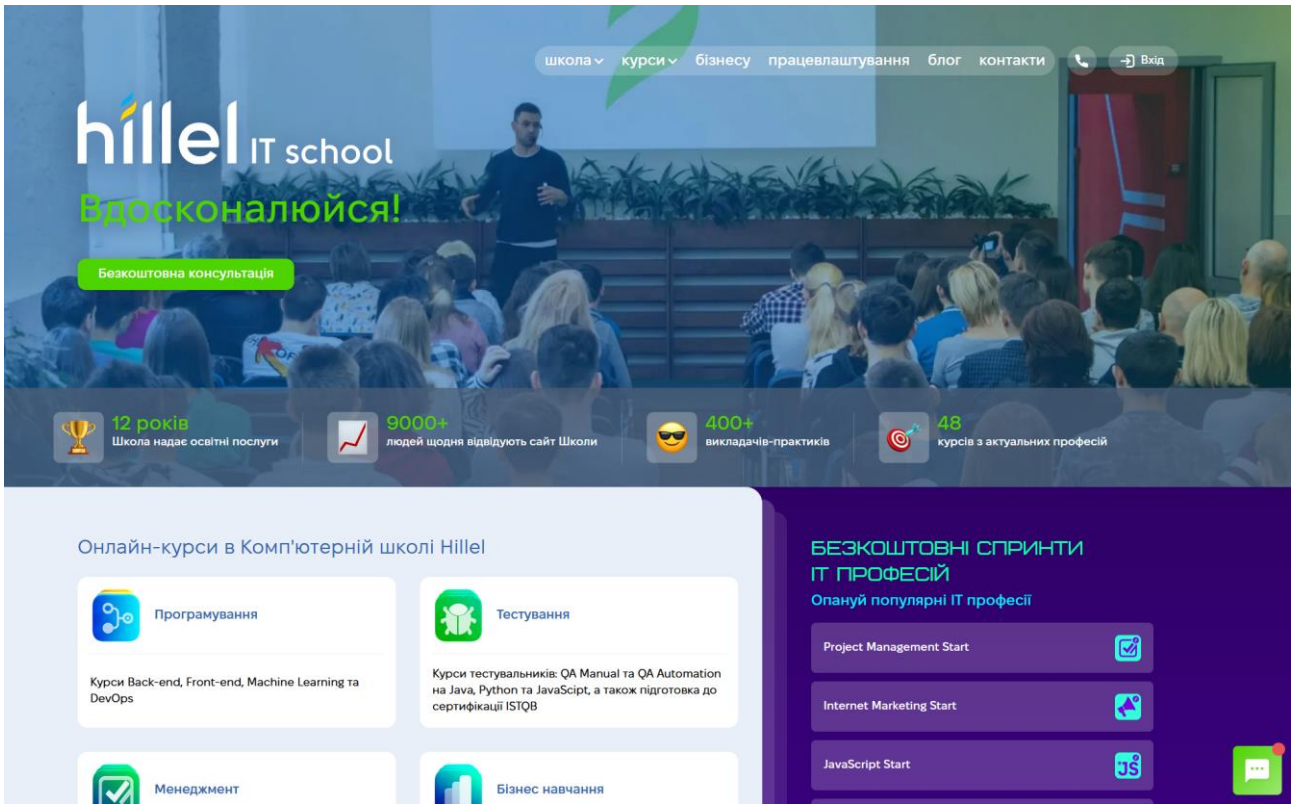


Рисунок В.2 – Головна сторінка школи Hillel

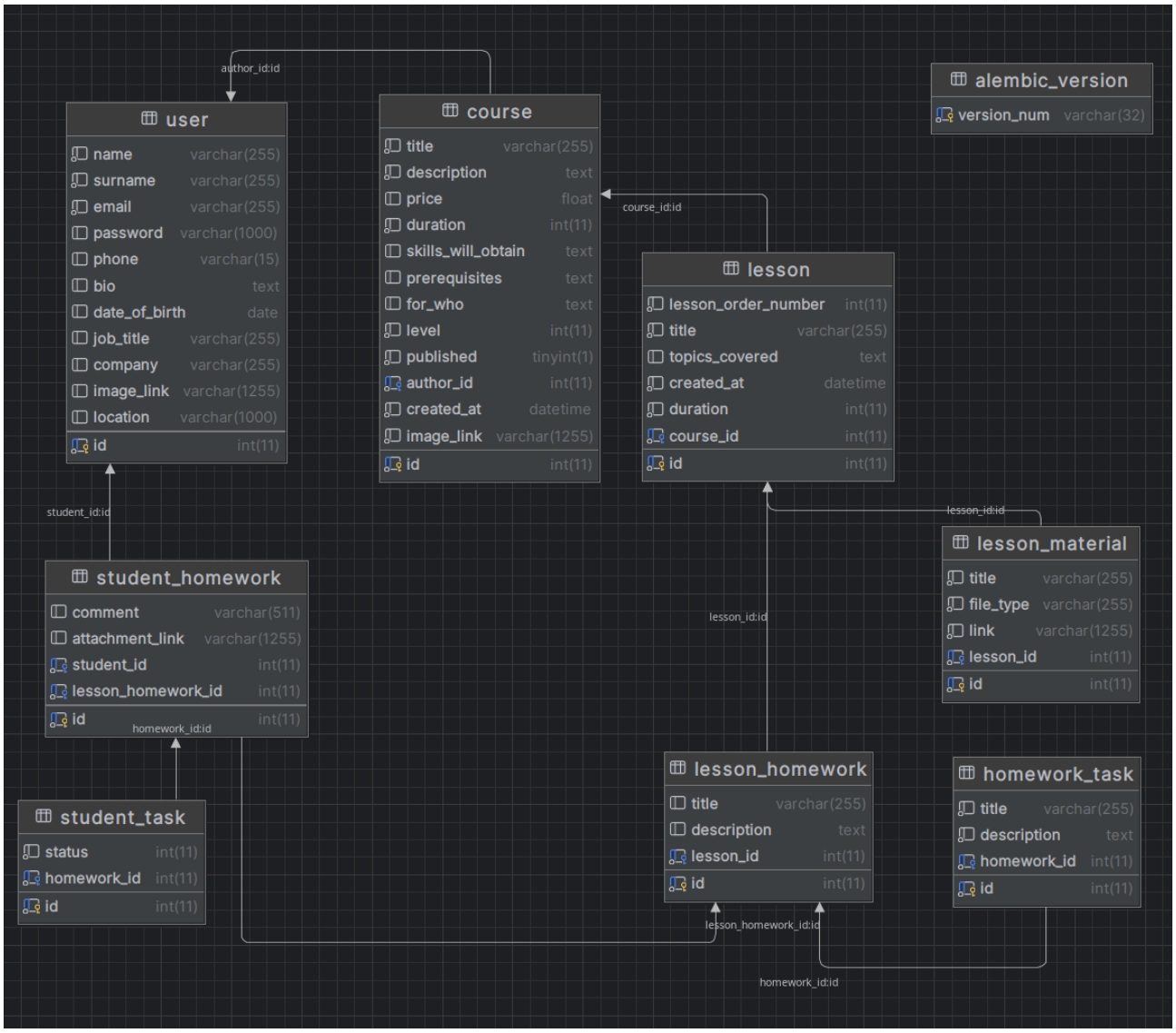


Рисунок В.3 – Схема бази даних

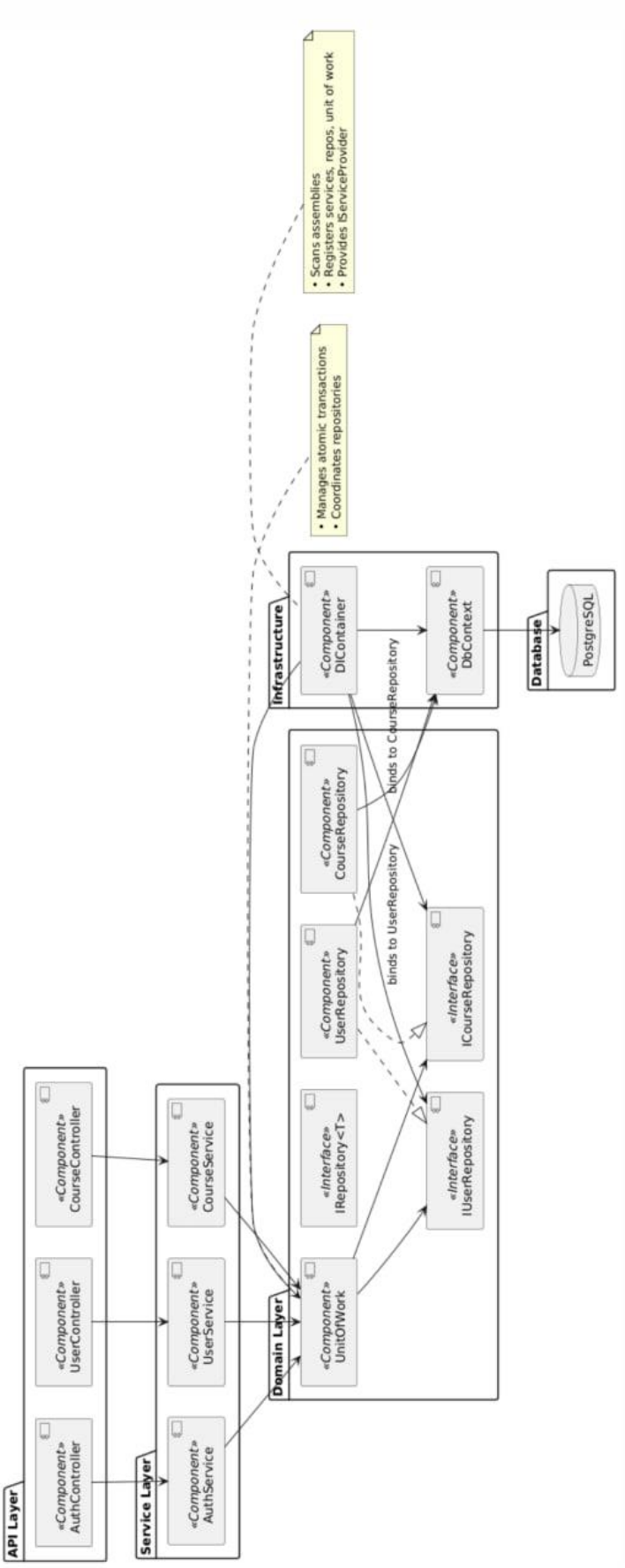


Рисунок В.4 – Діаграма компонентів серверної частини застосунку

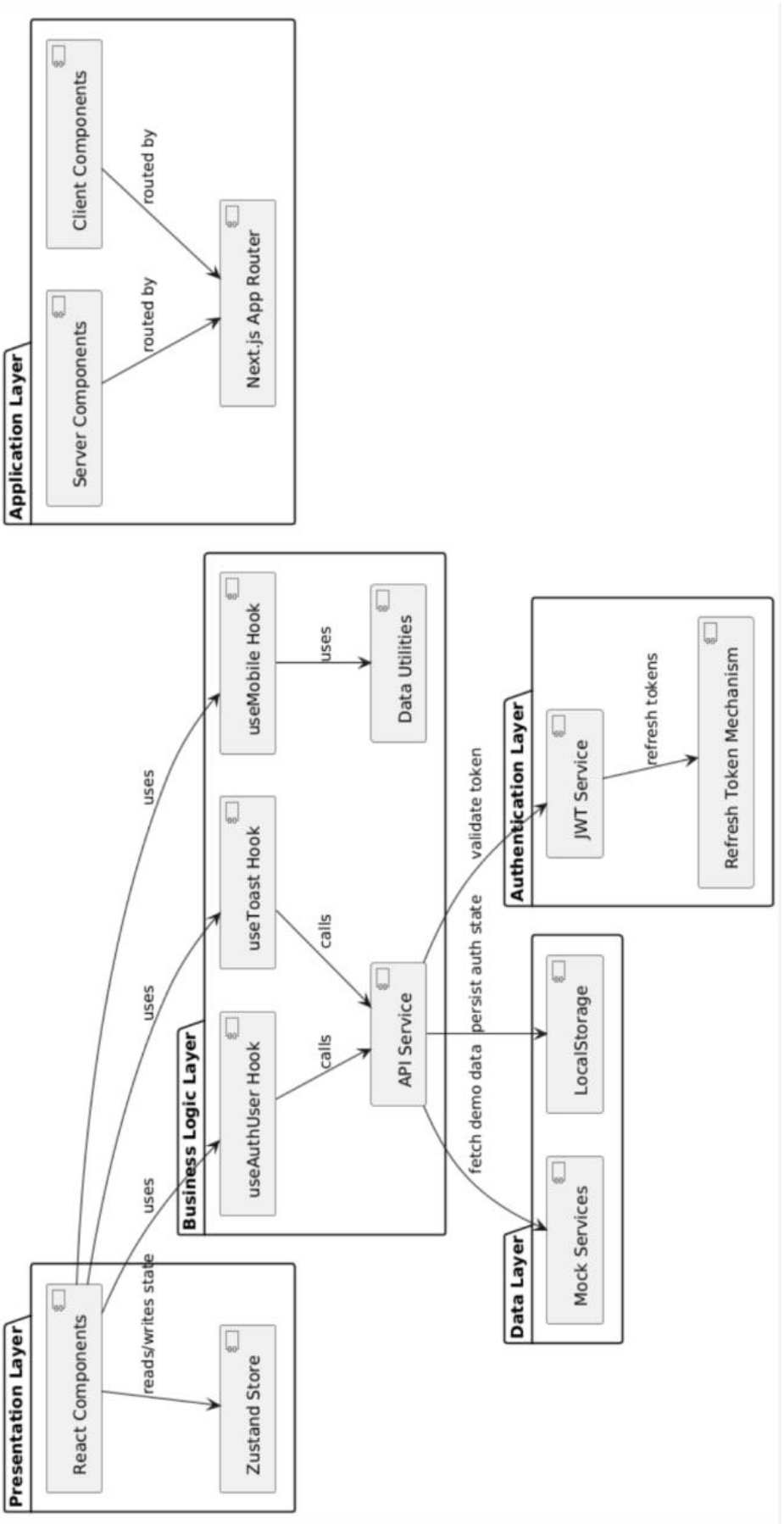


Рисунок В.5 – Діаграма компонентів клієнтського застосунку

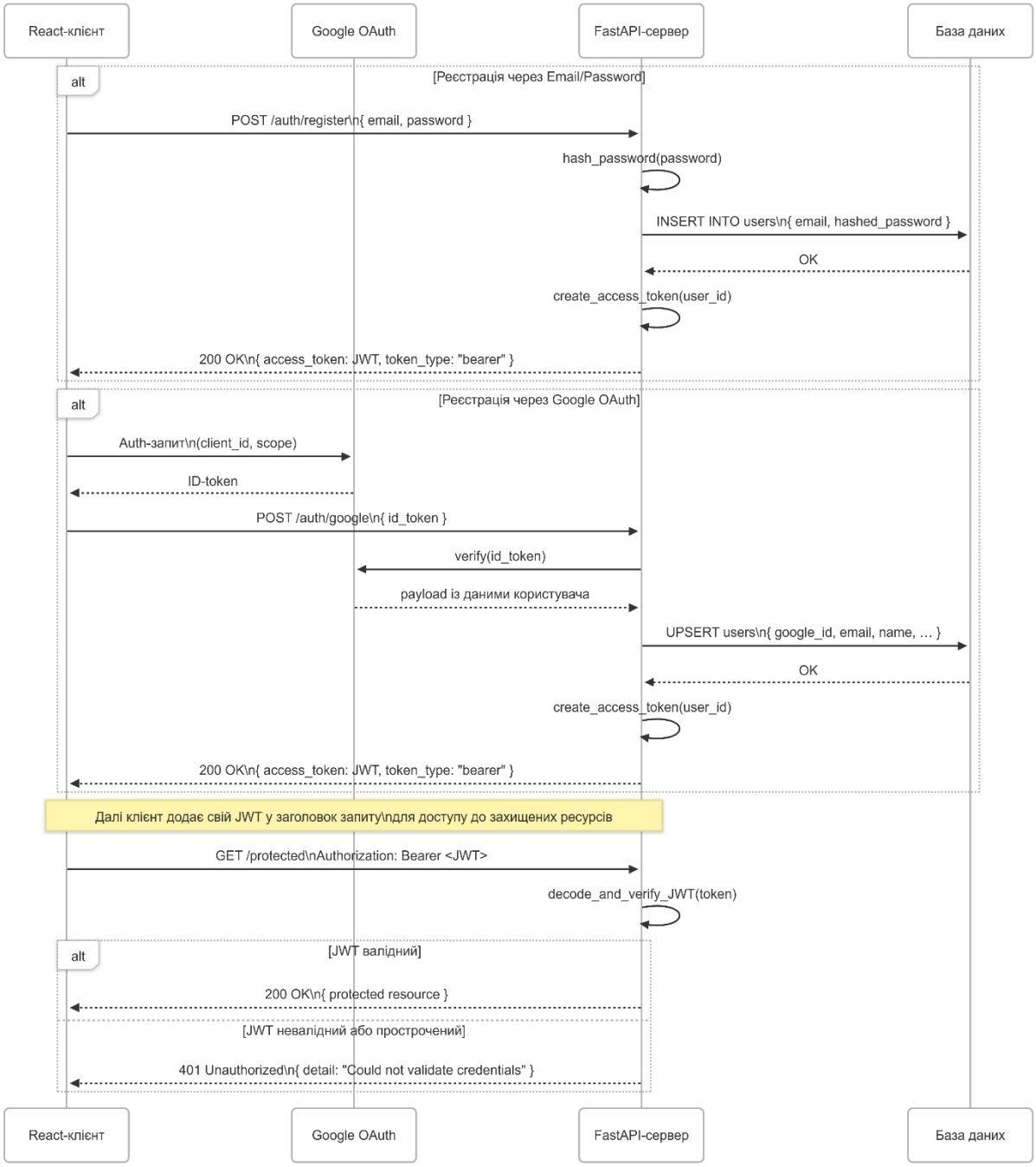


Рисунок В.6 – Процес реєстрації на платформі

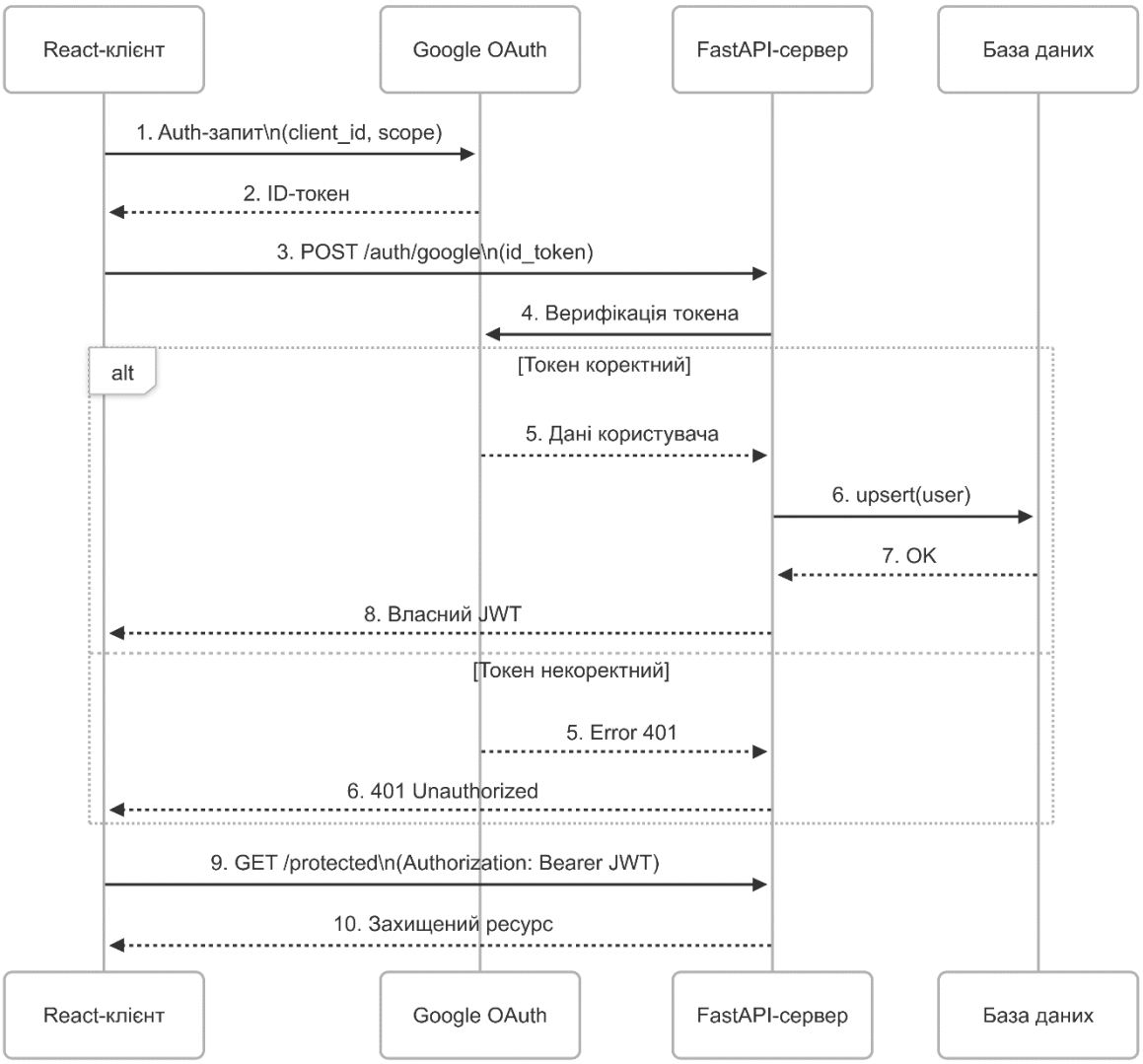


Рисунок В.7. – Процес входження в систему за допомогою Google OAuth

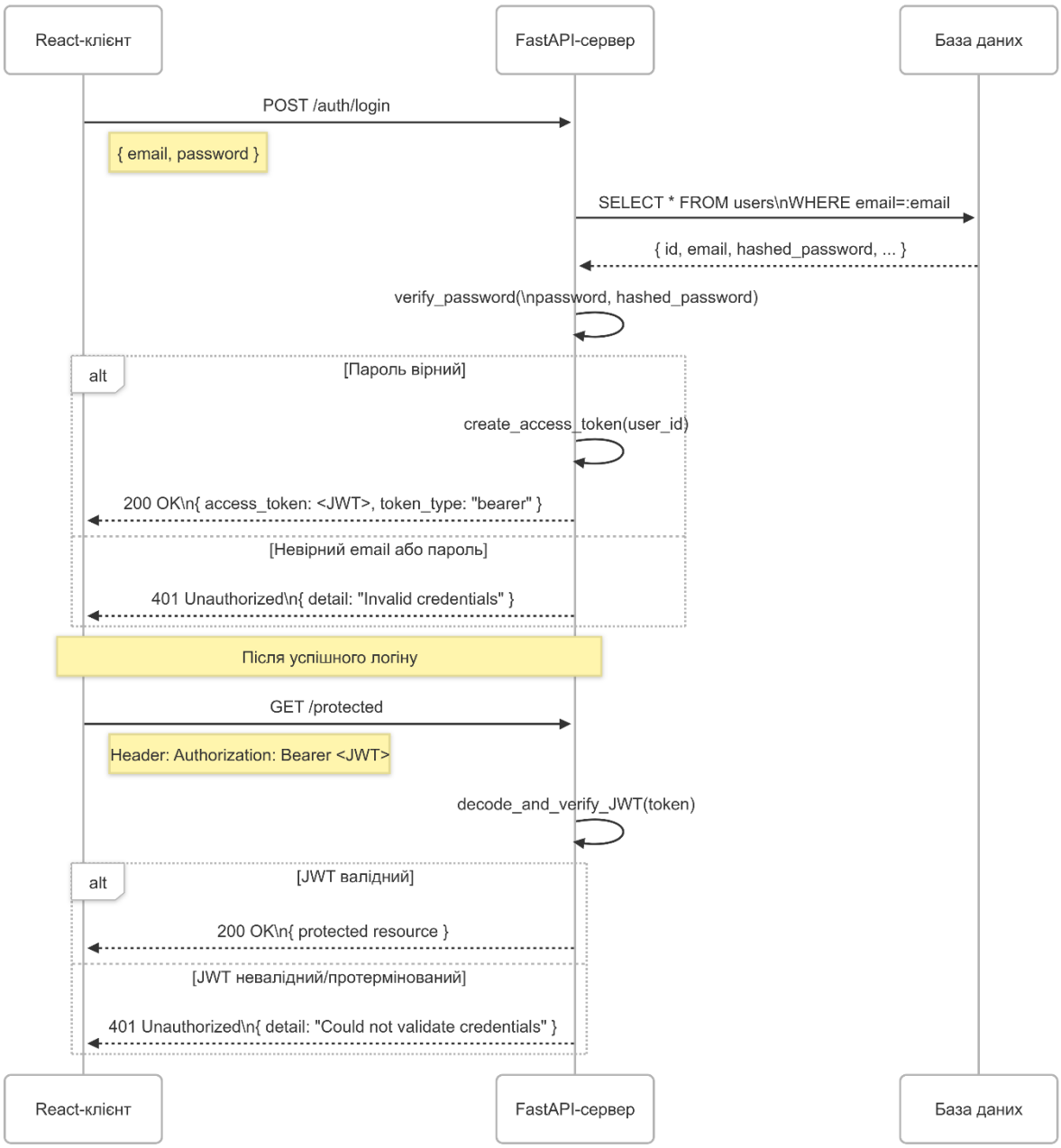


Рисунок В.8 – Процес входження в систему за допомогою пошти та паролю

ДОДАТОК Г (довідниковий)
ІНСТРУКЦІЯ КОРИСТУВАЧА

Користувач починає знайомство з платформою на сторінці Courses, де представлений перелік доступних програм у вигляді карток із назвою, коротким описом, інформацією про викладача та рейтингом (рис. Г.1). За допомогою панелі пошуку та фільтрів можна швидко відсортувати курси за категоріями чи рівнем складності і перейти до детального опису за кліком на обрану програму.

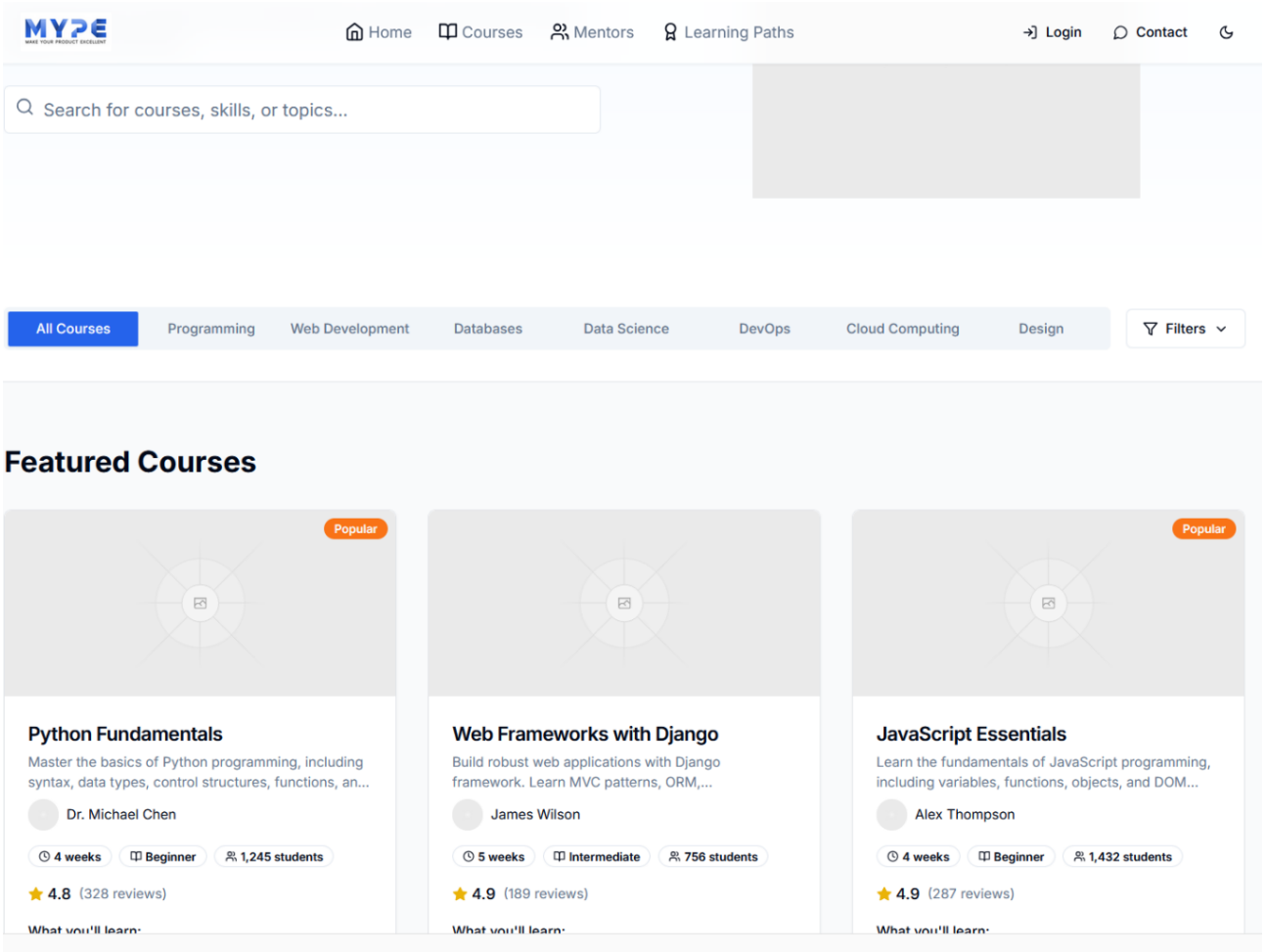


Рисунок Г.1 – Сторінка переліку курсів

На сторінці конкретного курсу відображаються всі ключові відомості: стислий вступний опис, характеристики (тривалість, кількість уроків, відгуки), перелік отримуваних навичок та картка запису з опціями Self-Paced або Guided Group Learning (рис. Г.2). Щоб розпочати навчання, користувач натискає кнопку

Enroll Now. Якщо акаунту ще немає, система запропонує зареєструватися або авторизуватися через Google OAuth чи за допомогою класичного логіна й пароля. Після підтвердження облікового запису в електронній пошті доступ до матеріалів курсу буде відкрито автоматично.

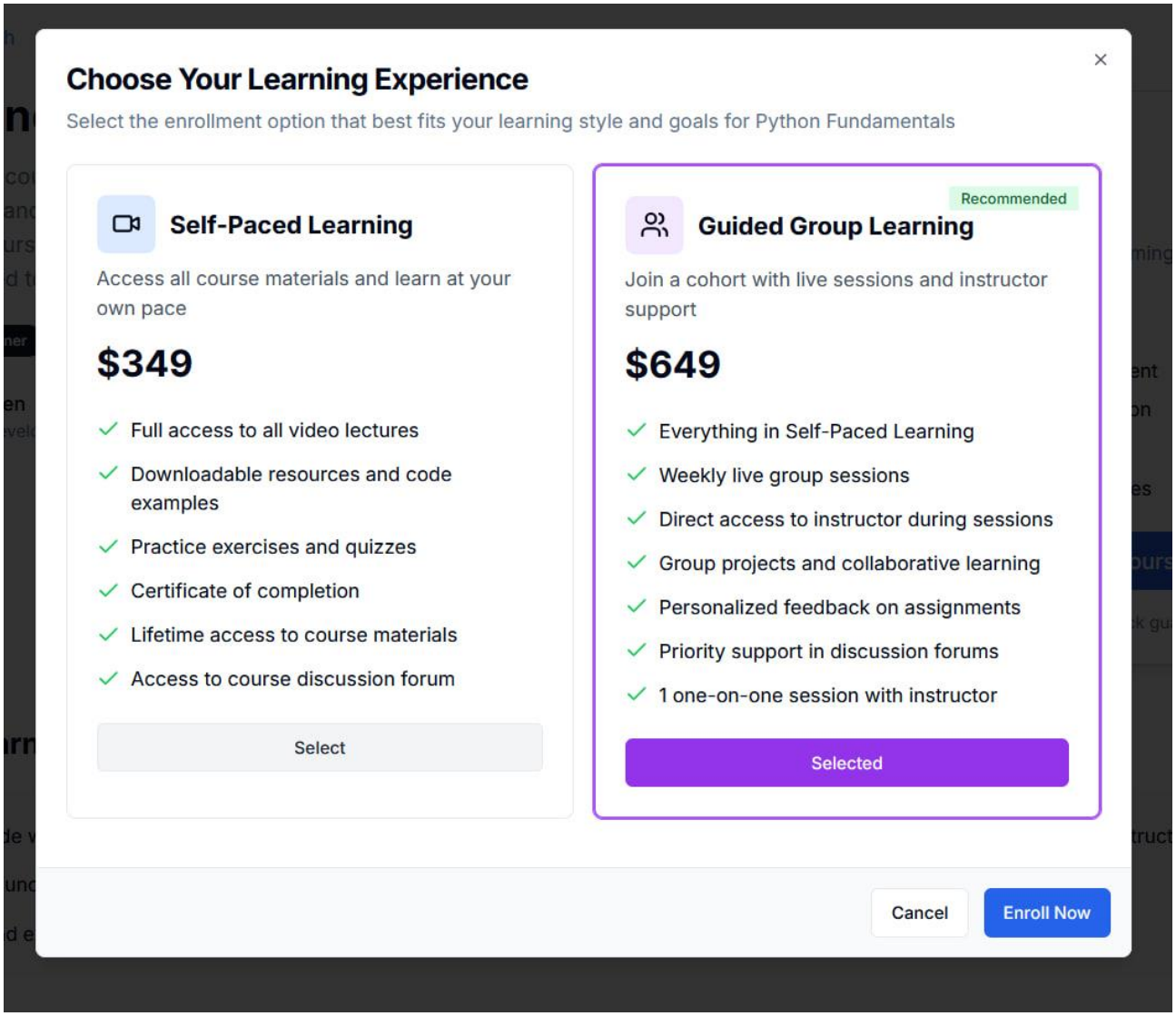


Рисунок Г.2 – Сторінка переліку курсів

Далі всі активні курси з індикаторами прогресу, датою останнього доступу та наступним дедлайном можна переглянути у розділі My Courses (рис. Г.3). Кожна картка містить кнопку Continue Learning, яка веде до інтерфейсу проходження: список уроків з відмітками «Completed» чи «In Progress», відеоплеєр, супровідні матеріали та інтерактивні завдання з можливістю завантажити файли й відмітити виконання кожного пункту.

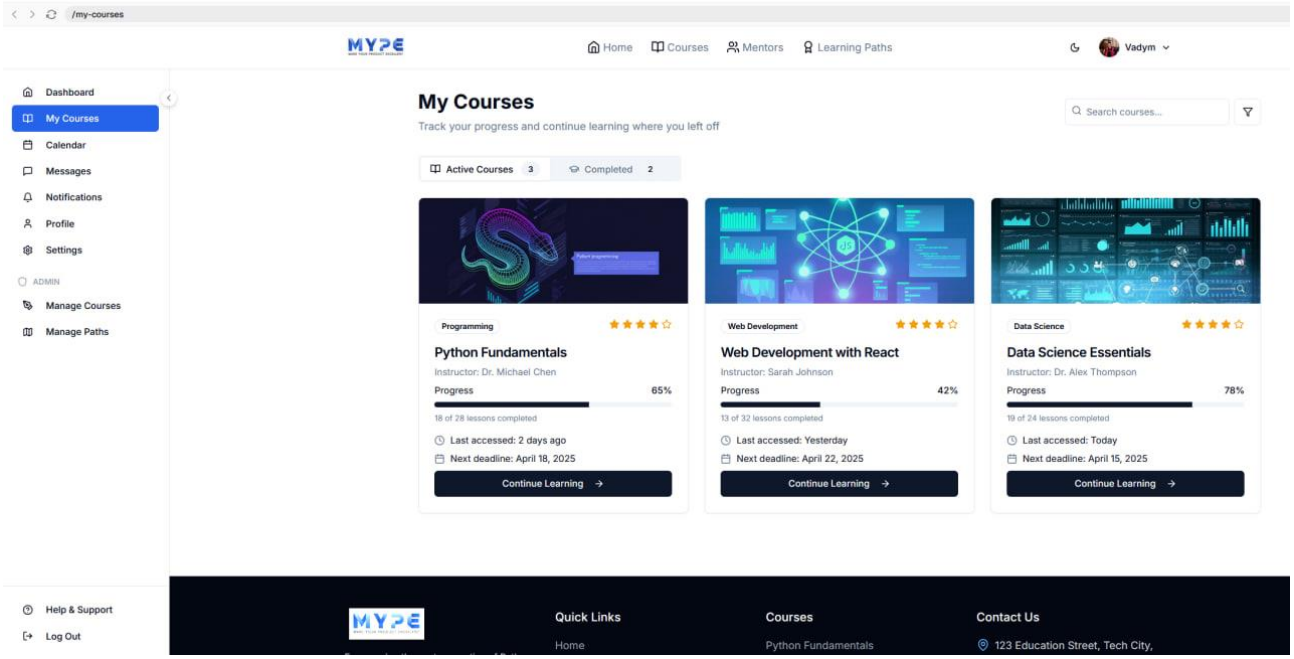


Рисунок Г.3 – Сторінка переліку курсів

Для викладачів і адміністраторів передбачено спеціальну панель управління — Manage Courses. Тут відображається таблиця всіх власних курсів із полями «Status», «Lessons», «Students» і «Last Updated». За допомогою кнопки Create Course (рис. Г.4) можна перейти до форми створення нового курсу, де послідовно заповнюються вкладки Course Details, Lessons та Settings. Після збереження чернетки курс можна попередньо переглянути через функцію Preview, щоб оцінити його вигляд перед публікацією.

Таким чином, платформа забезпечує природний потік взаємодії: від пошуку та ознайомлення з курсом до реєстрації, запису, власне навчання та — за необхідності — адміністрування власних програм у зручному інтерфейсі. Такий підхід гарантує швидкий доступ до знань для слухачів і простоту управління контентом для викладачів без зайвих технічних складнощів.

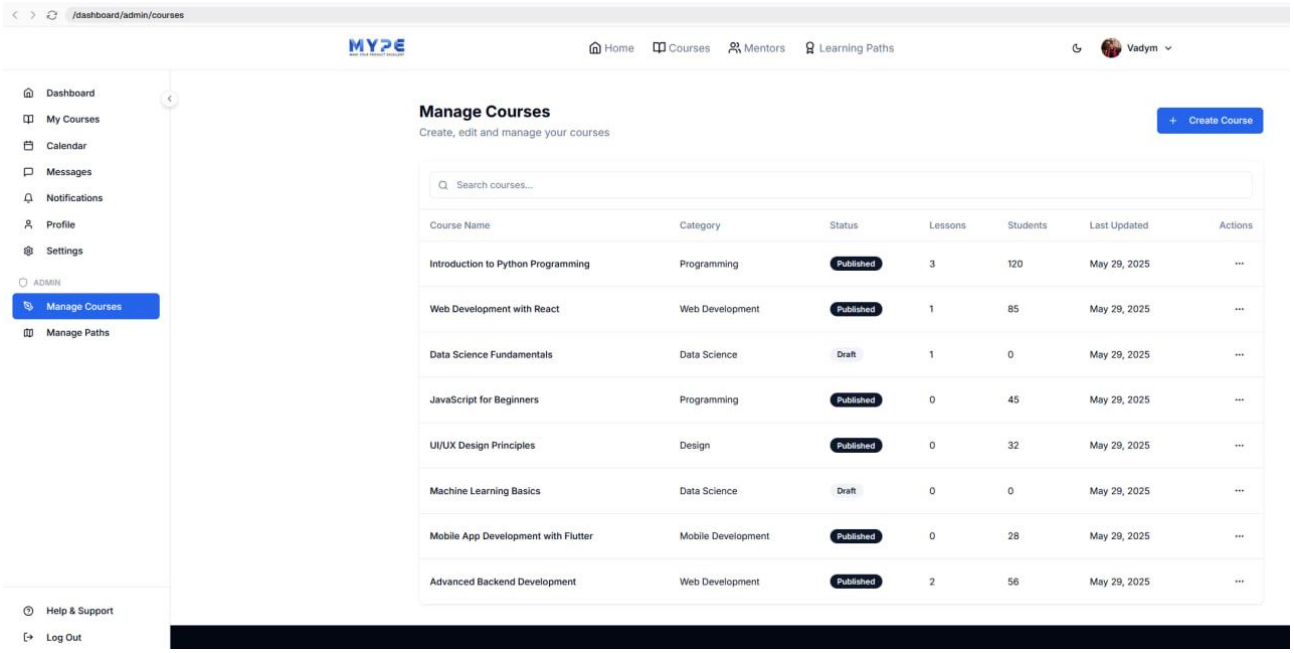
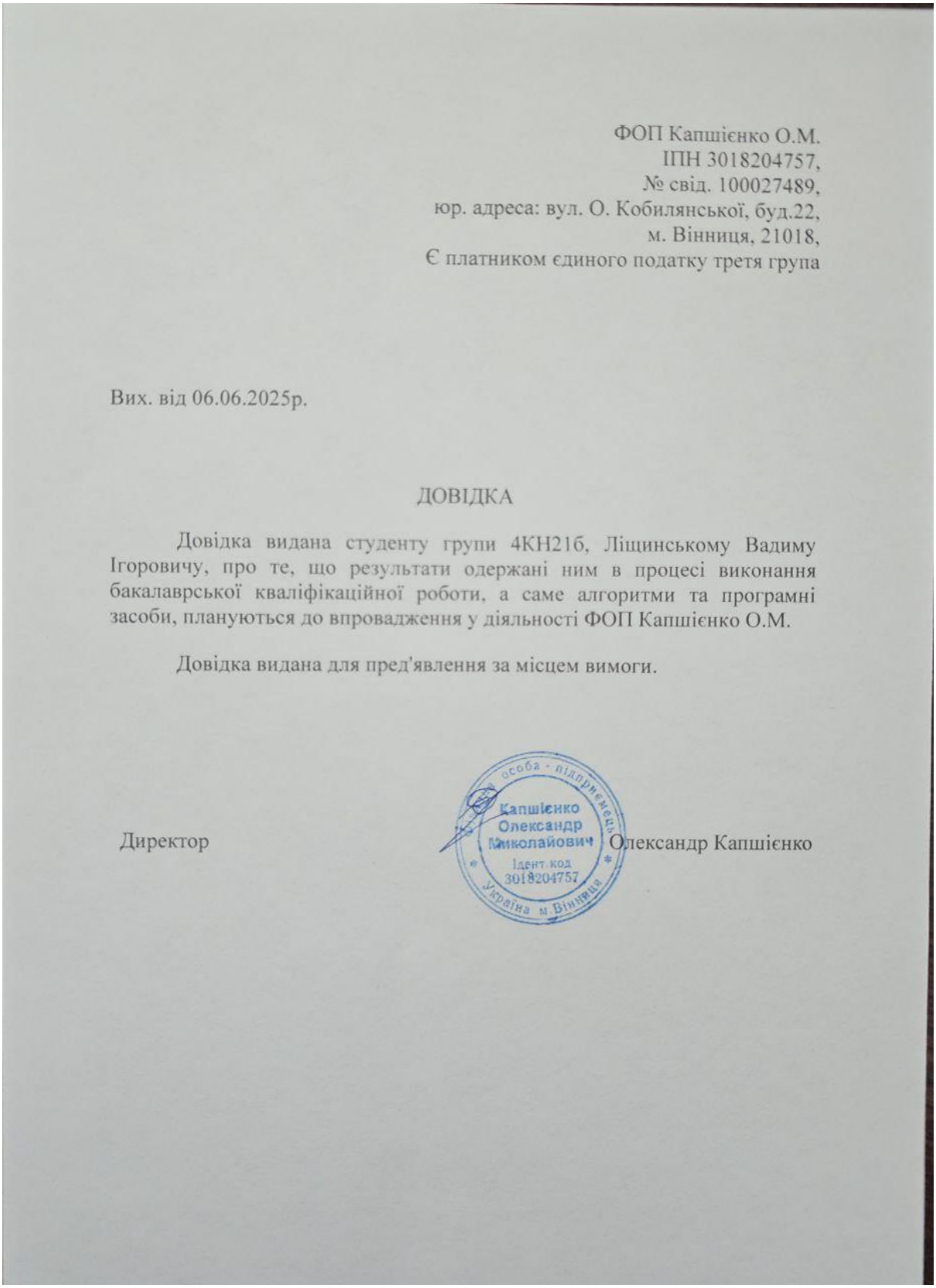


Рисунок Г.4 – Сторінка переліку курсів

ДОДАТОК Д (обов’язковий)
ДОВІДКА ПРО ВПРОВАДЖЕННЯ



ФОП Капшієнко О.М.
ІПН 3018204757,
№ свід. 100027489,
юр. адреса: вул. О. Кобилянської, буд.22,
м. Вінниця, 21018,
Є платником єдиного податку третя група

Вих. від 06.06.2025р.

ДОВІДКА

Довідка видана студенту групи 4КН21б, Ліщинському Вадиму Ігоровичу, про те, що результати одержані ним в процесі виконання бакалаврської кваліфікаційної роботи, а саме алгоритми та програмні засоби, плануються до впровадження у діяльності ФОП Капшієнко О.М.

Довідка видана для пред'явлення за місцем вимоги.

Директор Капшієнко Олександр Миколайович Ідент.код 3018204757 Україна м.Вінниця Олександр Капшієнко

Рисунок Д.1 – Довідка про впровадження