

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

ПОЯСНЮВАЛЬНА ЗАПИСКА

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

на тему:

**«Платформа для організації та адміністрування
мовних курсів»**

Виконав студент групи ІПЗ-21-4
ГОГОЛЮК Богдан Віталійович

Керівник роботи:
ЄФРЕМОВ Юрій Миколайович

Рецензент:
ЛОБАНЧИКОВА Надія Миколаївна

Житомир – 2025

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

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«26» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ГОГОЛЮК Богдан Віталійович**

Керівник роботи: **ЄФРЕМОВ Юрій Миколайович**

Тема роботи: **«Платформа для організації та адміністрування мовних курсів»**, затверджена наказом закладу вищої освіти №61/с від «14» лютого 2025 р.

Вихідні дані для роботи: **інформаційні технології з адміністрування мовних курсів. Використання веб-орієнтованих технологій для створення веб системи з адміністрування мовних курсів.**

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Юрій ЄФРЕМОВ	27.02.2025	27.02.2025
2	Юрій ЄФРЕМОВ	27.02.2025	27.02.2025
3	Юрій ЄФРЕМОВ	27.02.2025	27.02.2025

РЕФЕРАТ

Випускна кваліфікаційна робота складається з програмного продукту веб-платформи для організації та адміністрування мовних курсів та пояснювальної записки.

Пояснювальна записка складається з переліку умовних позначень, вступу, 3 розділів, висновків та джерел інформації.

Текстова частина викладена на 61 сторінці друкованого тексту.

Джерела інформації містять 30 найменувань і займають 3 сторінки. У роботі наведено 37 рисунків та 16 таблиць. Загальний обсяг пояснювальної записки – 69 сторінок.

Метою роботи є розробка веб-платформи для організації та адміністрування курсів з вивчення мов.

У роботі було визначено основні завдання для розробки програми та проведено аналіз аналогічних програмних продуктів. Обґрунтовано вибір архітектури програми й наведено сценарії її функціонування та тестування. Також узагальнено структуру веб-платформи, описано алгоритми взаємодії окремих модулів системи та об'єктну структуру програми.

КЛЮЧОВІ СЛОВА: ВЕБ-СИСТЕМА, МОВНІ КУРСИ, ТЕСТИ, СЛОВНИКИ, REACT, SPA.

					ІПЗ.КР.Б – 121 – 25 – ПЗ						
Змн.	Арк.	№ докум.	Підпис	Дата							
Розроб.		Б. В. Гоголюк			Платформа для організації та адміністрування мовних курсів Пояснювальна записка			Літ.	Арк.	Аркуші	
Керівник		Ю. М. Єфремов								3	61
								Житомирська політехніка, група ІПЗ-21-4			
Рецензент.		Н. М. Лобанчикова									
Зав. каф.		Т. А. Вакалюк									

ABSTRACT

The qualification work consists of a software product – a web platform for organizing and administering language courses, and an explanatory note.

The explanatory note includes a list of symbols and abbreviations, an introduction, 3 chapters, conclusions, and a list of references.

The text part is presented on 61 pages of printed text.

The list of references contains 30 sources and occupies 3 pages. The work includes 37 illustrations and 16 tables. The total volume of the explanatory note is 69 pages.

The aim of the work is the development of a web platform for organizing and administering language courses.

The work defines the main tasks for software development and analyzes similar software products. It justifies the choice of the program's architecture and provides scenarios of its operation. The structure of the web platform is also summarized, interaction algorithms of individual system modules are described, and the object structure of the program is presented.

KEYWORDS: WEB SYSTEM, LANGUAGE COURSES, TESTS, DICTIONARIES, REACT, SPA.

					ИПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						4

ЗМІСТ

ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ВЕБ-ПЛАТФОРМИ	9
1.1. Постановка задачі.....	9
1.2. Аналіз аналогів програмного продукту	10
1.3. Вибір архітектури веб-додатку	15
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	18
Висновок до першого розділу	22
РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБ-ПЛАТФОРМИ	23
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи	23
2.2. Розробка бази даних системи.....	27
2.3. Проектування та реалізація алгоритмів роботи системи	34
2.4. Реалізація веб-платформи	38
Висновки до другого розділу	41
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-ПЛАТФОРМОЮ.....	43
3.1. Порядок встановлення та налаштування системи	43
3.2. Структура інтерфейсу веб-платформи.....	44
3.3. Тестування платформи	53
Висновки до третього розділу	56
ВИСНОВКИ.....	57
ДЖЕРЕЛА ІНФОРМАЦІЇ	59
ДОДАТКИ.....	62

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення.

СУБД – система управління базами даних.

БД – База даних.

API (application programming interface) – прикладний програмний інтерфейс.

CRUD – основні операції, що використовуються для управління даними (Create , Read , Update , Delete).

ORM – (Object-relational mapping) – об’єктно-реляційна проєкція, технологія яка зв’язує бази даних з концепціями об’єктно-орієнтованих мов програмування.

ВСТУП

Актуальність розробки веб-платформи для організації та адміністрування курсів з вивчення мов обумовлена підвищенням інтересу до вивчення іноземних мов та необхідністю зручного інструменту для ефективної організації і структурування занять у різноманітних гуртках з вивчення мов, курсах, тренінгових програмах, школах, навчальних закладах, тощо.

Мета даної кваліфікаційної роботи є розробка веб-платформи для організації та адміністрування курсів з вивчення мов, яка надає ефективний та зручний інтерфейс для організації подібних курсів.

Встановлена мета обумовлює наступні завдання:

- проведення аналізу вимог до веб-платформи;
- визначення архітектури та основних компонентів платформи;
- розробка алгоритмів та моделей взаємодії з платформою;
- обґрунтування та вибір технологій для реалізації платформи;
- тестування та оптимізація роботи системи.

Розробка веб-платформи дозволяє централізовано планувати та контролювати навчання, студентів, викладачів, розклад занять, відстеження успішності студентів. Це дозволяє швидко вирішувати організаційні питання.

За допомогою інструментів для управління студентами, викладачами та курсами можна мінімізувати паперову документацію та автоматизувати більшість рутинних дій. Це дає змогу адміністраторам більше уваги приділяти розвитку курсів, роботі зі студентами та покращенню якості послуг.

Веб-платформа забезпечує доступ до актуальних даних про студентів, викладачів, навчальні програми та процес навчання в режимі реального часу.

Об'єктом дослідження є інформаційні технології з адміністрування курсів з вивчення мов.

Предметом дослідження є використання веб-орієнтованих технологій для розробки системи з адміністрування курсів з вивчення мов.

У процесі роботи використовувалися методи аналізу вимог користувачів.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						7

Інформаційна база дослідження включала ознайомлення з робочими процесами адміністраторів, викладачів та студентів, а також вивчення аналогів.

Результати даної роботи можуть бути використані для розробки та впровадження на реальну освітню установу веб-платформи для організації та адміністрування курсів з вивчення мов.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ВЕБ-ПЛАТФОРМИ

1.1. Постановка задачі

Розробка веб-платформи потребує ретельного аналізу потреб користувачів. Визначення ключових вимог дозволить створити зручний та ефективний продукт, який відповідатиме очікуванням користувачів.

Застосунок має бути зручним інструментом для ефективної організації і структурування занять у різноманітних гуртках, курсах, тренінгових програмах, школах з вивчення мов, навчальних закладах, тощо.

Для проектування архітектури програмного комплексу потрібно буде визначитися зі структурою веб-платформи, яка забезпечить ефективну взаємодію між користувачами, базою даних та серверною частиною. Проектування архітектури включатиме кілька ключових аспектів:

Структура бази даних – розробка схеми для зберігання інформації про користувачів, курси, тести, словники та інші дані [1]. Оптимізація структури для швидкого доступу та ефективного оброблення запитів.

Серверна частина – вибір технологій для реалізації серверної логіки, розробка API для обміну даними між клієнтом і сервером, обробка запитів користувачів та збереження даних [2].

Клієнтська частина – проектування веб-інтерфейсу, який буде інтуїтивно зрозумілим та зручним у використанні. Розробка логіки взаємодії користувачів із платформою [3].

Правильне проектування архітектури платформи забезпечить її масштабованість, продуктивність і зручність використання, а це є важливими критеріями для успішного функціонування веб-платформи.

В результаті повинно бути створено веб-платформу для зручного адміністрування курсів для вивчення мов. Основними етапами розробки платформи є:

1. Розробка дизайну веб-сайту:
 - створення загальної концепції дизайну додатку;
 - розробка дизайну сторінок.
2. Побудова архітектури клієнтської частини додатку;
3. Побудова архітектури серверної частини додатку;
4. Розробка системи авторизації та ролей користувачів;
5. Розробка основного функціоналу:
 - створення та налаштування курсів;
 - редагування списку учасників загалом на платформі та на курсі;
 - публікація новин курсу;
 - створення та перегляд електронних словників;
 - можливість створення та проходження тестів;
 - відстеження оплати курсу та результатів тестувань;
 - створення віддаленої бази даних для зберігання інформації;
 - організація спілкування серверної частини з клієнтською;

Далі буде проведено тестування веб-платформи для перевірки її функціональності. Виконуються тести для виявлення можливих помилок і збоїв у роботі системи, після чого здійснюється їх виправлення. Це дозволяє забезпечити стабільну та ефективну роботу веб-платформи перед її впровадженням у використання.

1.2. Аналіз аналогів програмного продукту

При розробці веб-платформи для організації та адміністрування курсів з вивчення мов важливо враховувати існуючі рішення, щоб отримати повну картину та розуміння потреб користувачів. Аналіз аналогічних платформ дозволить визначити їхні сильні та слабкі сторони й виділити ключові функції. Для проведення аналізу було обрано такі сервіси: Moodle [4]; Teachable [5]; Preply [6].

Moodle – це система управління навчанням з відкритим кодом, яка дозволяє створювати, адмініструвати та проводити онлайн курси. Вона підтримує різні формати контенту, тести, форуми та оцінювання (рис.1.1).

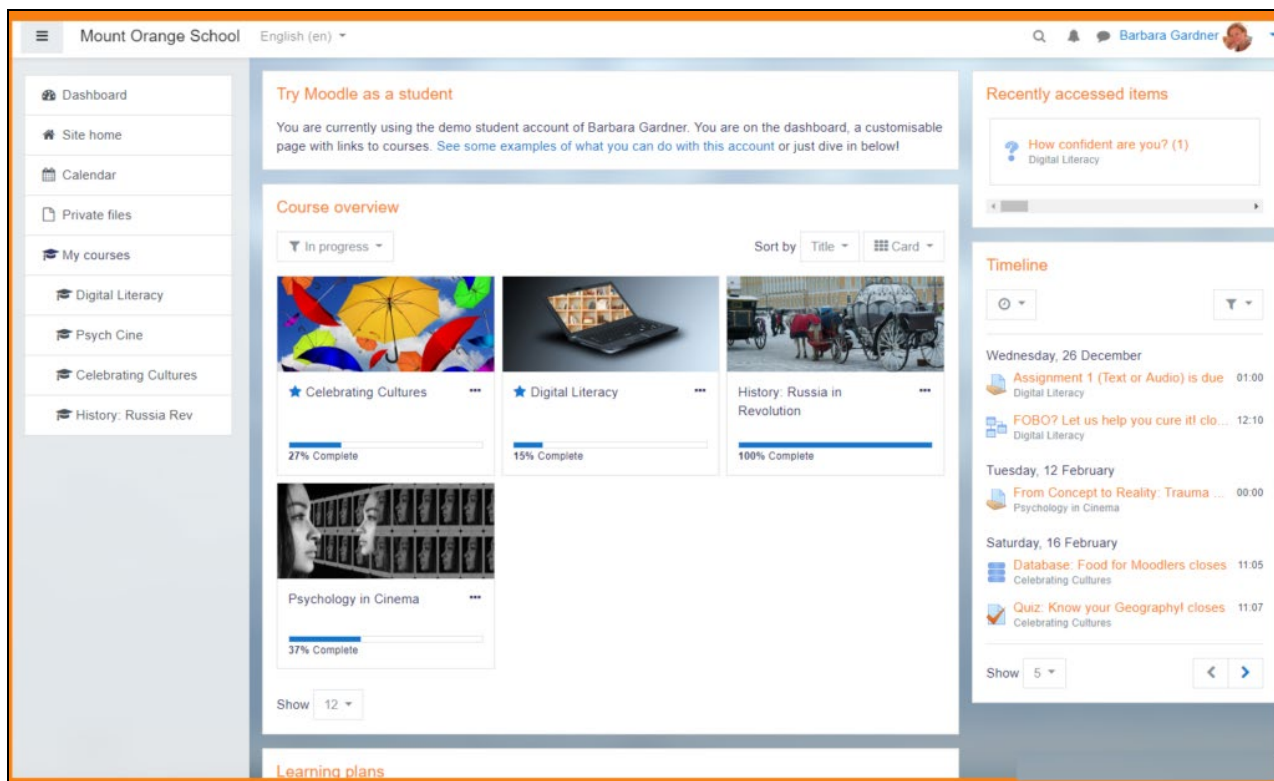


Рис. 1.1. Платформа Moodle

До переваг даного додатку можна віднести те, що гнучкість та налаштування Moodle дозволяє створювати курси та платформу відповідно до потреб користувача, а багатофункціональність платформи підтримує різноманітні формати контенту, в тому числі тести та їх різноманітні методи оцінювання.

До недоліків же можна віднести те, що для деяких користувачів налаштування може здатися складним і вимагати значного додаткового часу на освоєння. Також Moodle має досить неінтуїтивний інтерфейс, він не завжди є найбільш зручним для новачків, і можуть виникати складнощі з навігацією та його освоєнням.

Teachable – це платформа для створення онлайн курсів. Вона дозволяє легко створювати курси, додавати матеріали та управляти учнями. В порівнянні з Moodle має більш зрозумілий інтерфейс (рис.1.2).

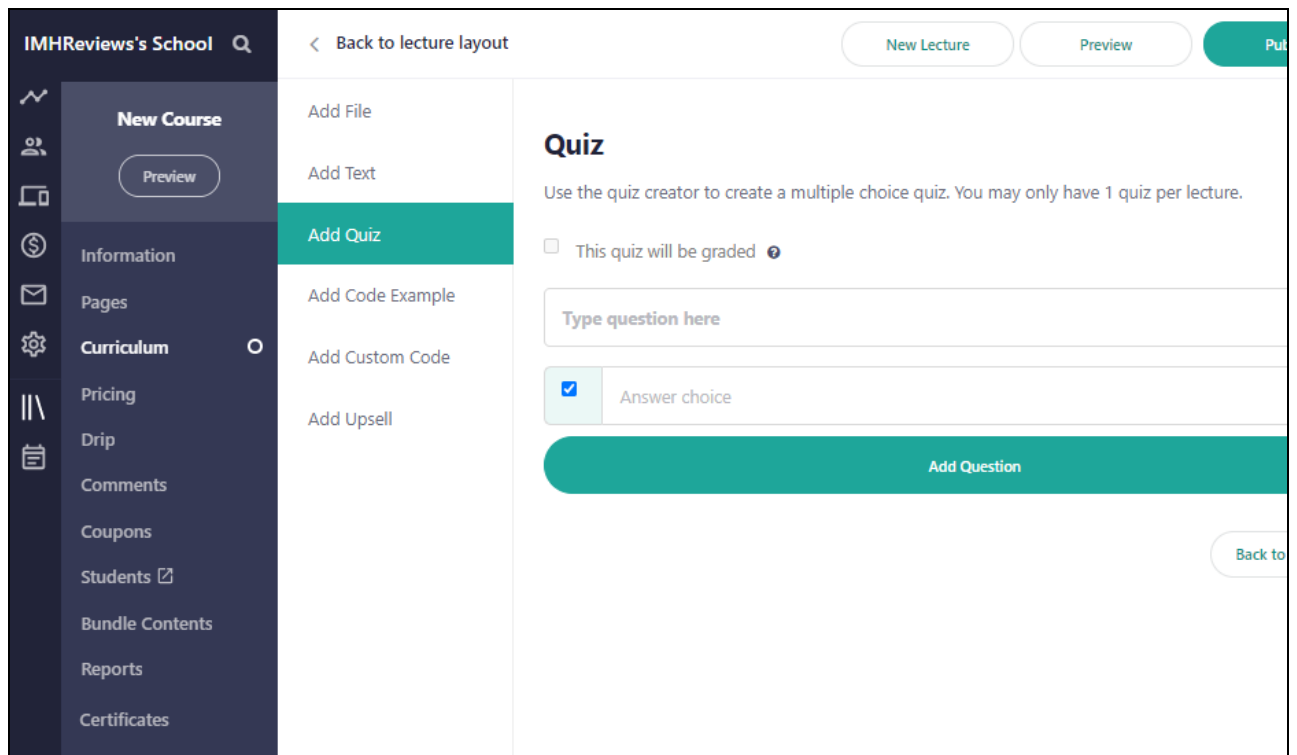


Рис. 1.2. Платформа Teachable

До переваг даної платформи можна віднести те, що вона пропонує простий та інтуїтивно зрозумілий інтерфейс для створення та управління курсами. Платформа має вбудовані інструменти для відстеження прогресу учнів, наприклад їх результатів тестування.

До недоліків можна віднести те, що платформа не підтримує групові курси з великою кількістю учасників. Teachable більше орієнтована на індивідуальне навчання, і її функціональність не так розвинена для проведення великих групових курсів.

Preply – це досить популярна онлайн-платформа для вивчення іноземних мов, вона надає користувачам можливість знаходити викладачів за їх вподобаннями для індивідуальних занять у онлайн, а іноді навіть оффлайн форматах. Ця платформа має базу викладачів з усього світу, що дає змогу обрати репетитора відповідно до рівня підготовки, бюджету, розкладу та навіть рідної мови викладача. Інтерфейс платформи є інтуїтивно зрозумілим і досить адаптований до зручного використання (рис.1.3). Але Preply фокусується на персоналізованому підході до навчання, можливість керувати груповими заняттями тут також відсутня.

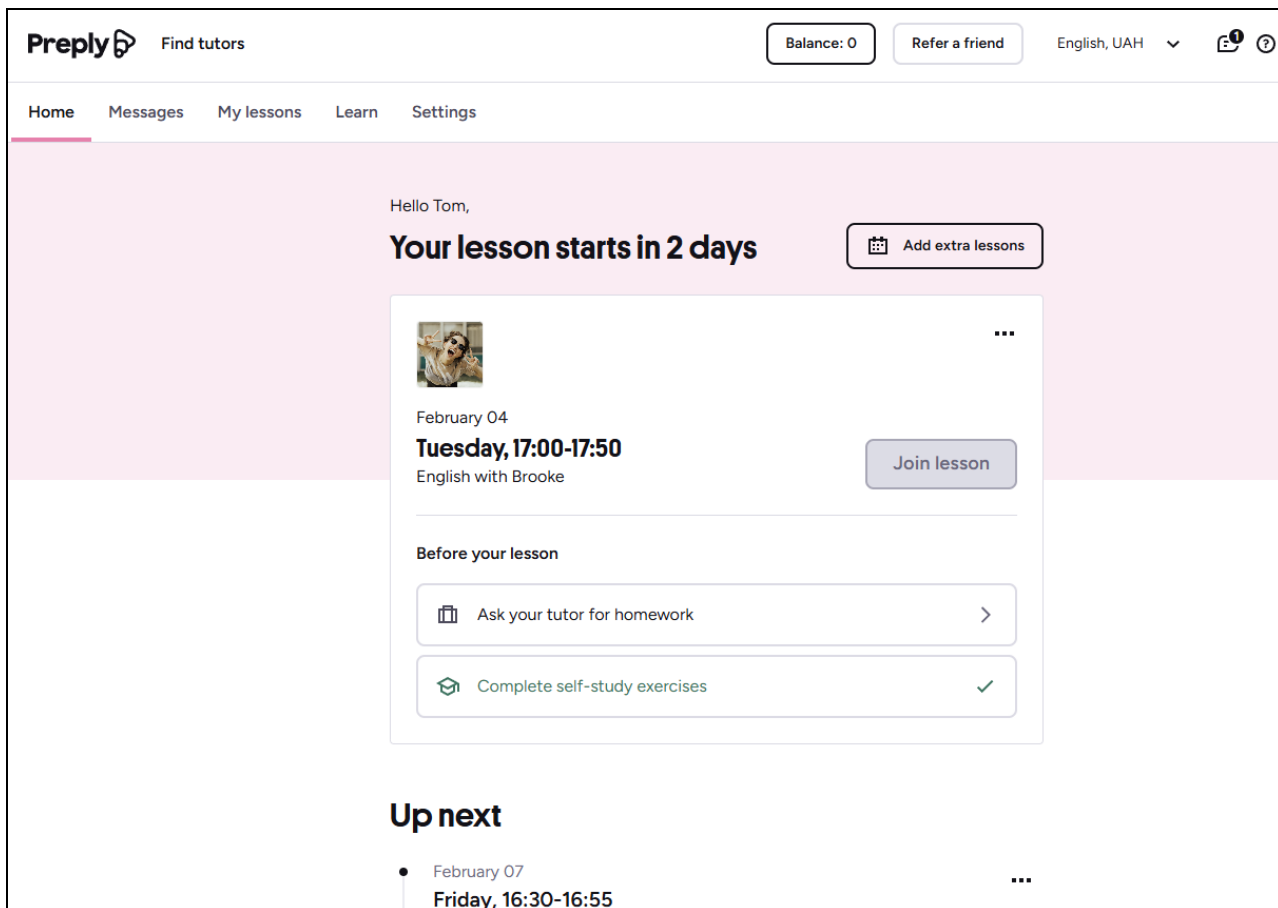


Рис. 1.3. Платформа Preply

До переваг даної платформи можна віднести те, що вона має досить інтерактивні та зрозумілі інструменти. Платформа надає доступ до інтерактивних навчальних матеріалів, має зручні інструменти для вивчення слів (рис. 1.4 – рис. 1.5), проведення тестувань, перегляду матеріалу, тощо.

Preply має простий і зручний інтерфейс, він дозволяє взаємодіяти з платформою без зайвих труднощів, що сильно полегшує звикання до платформи.

До недоліків можна віднести те, що платформа не має підтримки групових занять. Preply орієнтована виключно на індивідуальне навчання, тому можливість організації групових занять наразі відсутня.

Виходячи з наявних переваг та недоліків, складемо порівняльну таблицю вищезгаданих програмних аналогів (табл. 1.1).

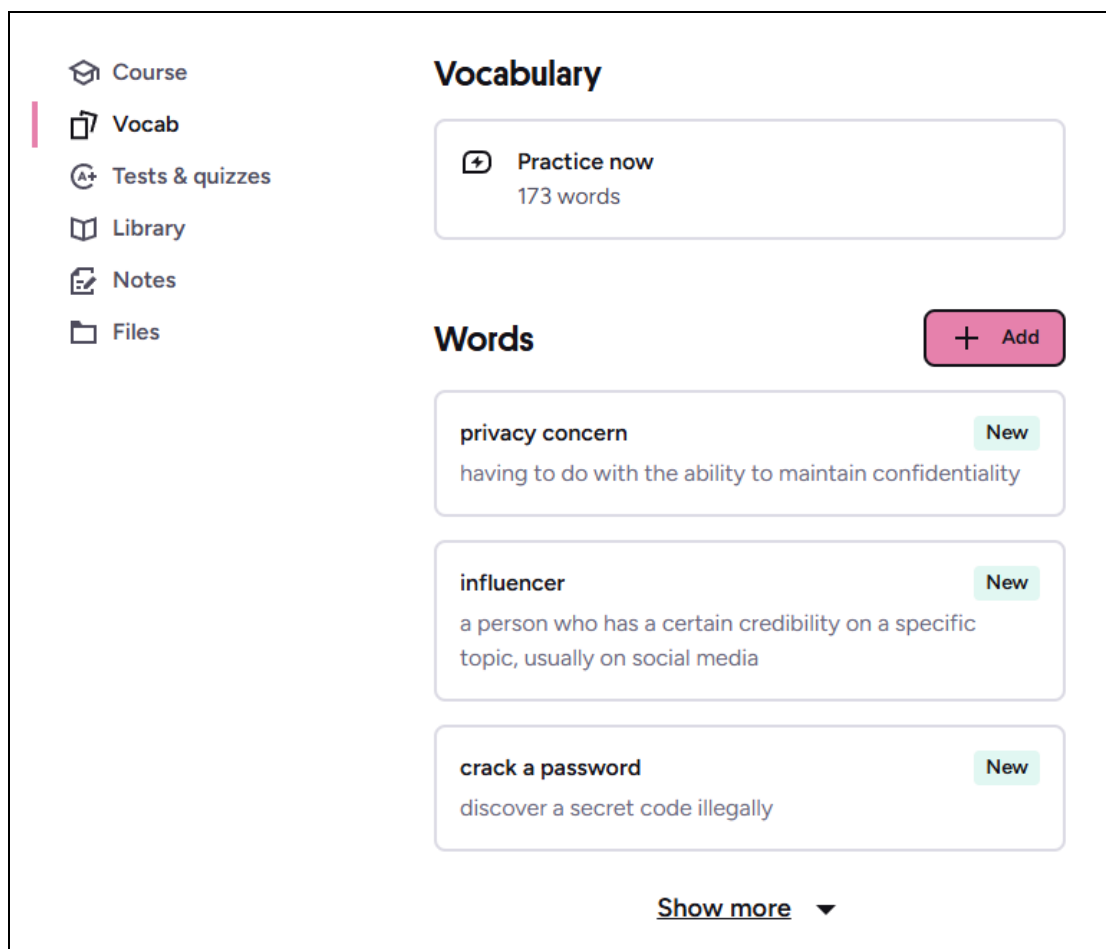


Рис. 1.4. Preply словник

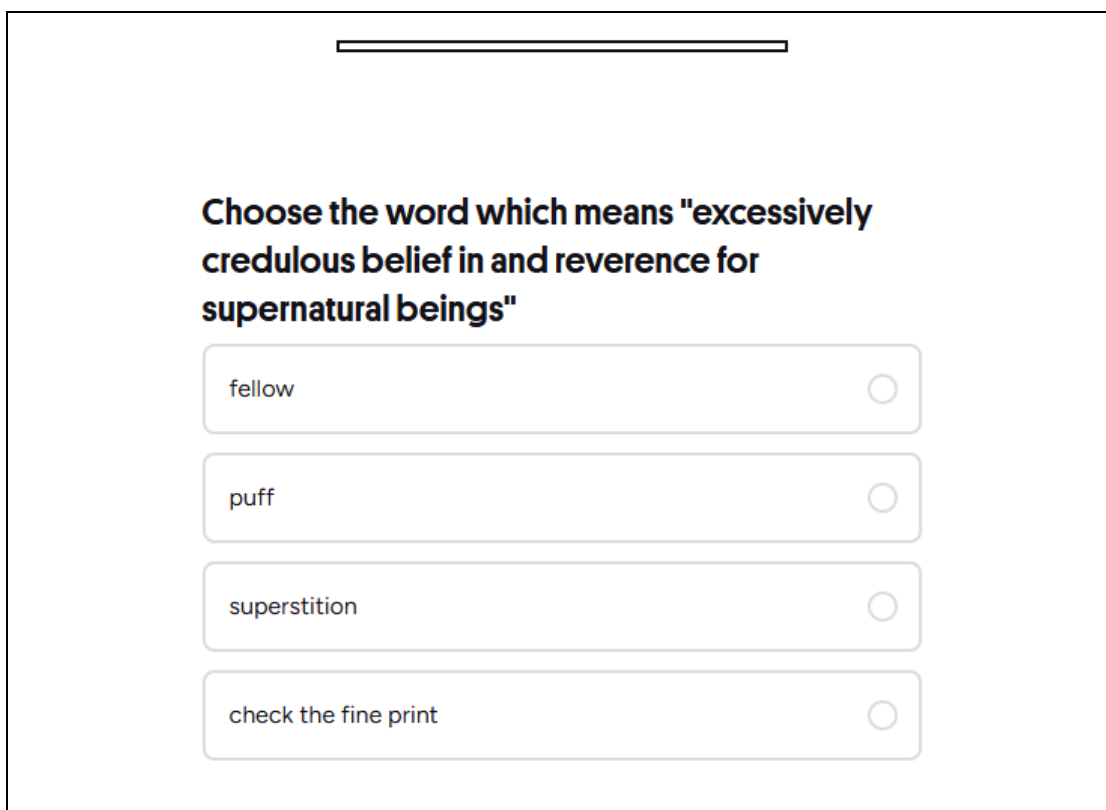


Рис. 1.5. Preply картки зі словами

Аналіз аналогів програмного продукту

Характеристика	Moodle	Teachable	Preply
Інтуїтивність та легкість у використанні	–	+ –	+
Індивідуальні курси	+	+	+
Підтримка створення групових курсів/занять	+	–	–
Зручне створення тестів та оцінювання	+	+	+ –
Зручне створення електронних словників	+	–	+

На основі отриманих результатів, визначаємо наступні критерії для розроблюваного додатку: інтуїтивний, не перевантажений та зрозумілий інтерфейс; підтримка створення групових курсів; зручне створення тестів, можливість їх проходження та перегляд результатів; створення електронних словників, з можливістю переглядати їх в режимі вивчення; публікація новин курсу; зручне налаштування списку користувачів як на платформі загалом, та і на конкретному курсі.

1.3. Вибір архітектури веб-додатку

У проекті буде застосовуватися кілька ключових патернів та архітектурних підходів. Архітектура веб-додатку клієнт-серверна [7], вона відповідає за розділення відповідальностей між клієнтом і сервером. Клієнт – надсилає запити і отримує відповіді. Сервер – обробляє запити, зберігає та керує даними, повертає результати. Суть полягає у чіткому розподілі ролей, тобто клієнт відповідає за інтерфейс, а сервер за логіку й збереження даних.

Для побудови серверної частини додатку було обрано об'єктно-орієнтовану мову програмування C# [8] та фреймворк ASP.NET Core [9]:

C# – це об'єктно-орієнтована мова програмування, яка широко використовується в різних сферах розробки, також вона надає розширений набір функціональних можливостей.

ASP.NET – це фреймворк для розробки веб-додатків, що працює на платформі .NET [10] та дозволяє швидко створювати стабільні та головне легко масштабовані веб-додатки завдяки підтримці багатьох вбудованих функцій [11].

До переваг використання можна віднести швидкість розробки та стабільність, так як ASP.NET має великий набір вбудованих функцій, що дозволяє швидко налаштувати основну функціональність веб-додатку. Вбудовані механізми автентифікації та авторизації [12], підтримка SSL/TLS [13] для шифрування даних та захист від SQL ін'єкцій [14].

Серед патернів буде використовуватися Singleton [15], який буде гарантувати існування лише одного екземпляра класу протягом життєвого циклу додатку [16]. На стороні ASP.NET Core він може використовуватися для конфігураційних сервісів, провайдерів даних, сервісів, тощо.

Крім Singleton буде також використовуватися Repository Pattern [17], який ізолює доступ до даних, абстрагуючи логіку взаємодії з базою даних від бізнес-логіки. Завдяки цьому бізнес-логіка не залежить від деталей реалізації джерела даних (ORM, SQL, API).

Також в доповнення буде використовуватися Dependency Injection (DI) [18], він допомагає зменшити зв'язність між компонентами шляхом передавання залежностей у конструктори класів. Також перевагою використання DI з ASP.NET Core є те, що ASP.NET автоматично впроваджує залежності зареєстровані у `builder.Services`, що дозволяє легко керувати життєвим циклом об'єктів (Singleton, Scoped, Transient).

Для побудови API [19] буде використовуватися GraphQL [20], це мова запитів, яка дає можливість клієнту отримувати лише необхідні дані в одному

запиті. Вона є зручним інструментом для побудови API, також за допомогою GraphQL можна значно скоротити кількість запитів до сервера.

Клієнт може запросити саме ті дані, які йому необхідні, що зменшує навантаження на сервер та підвищує продуктивність. А завдяки можливості надсилати комплексні запити, можна уникнути багатьох HTTP-запитів, що зазвичай виникають при використанні REST API [21].

Для побудови клієнтського додатку планується використання React [22].

React – це бібліотека для побудови користувацьких інтерфейсів. Вона дозволяє розробляти інтерфейси за допомогою компонентів, що зручно для створення складних та інтерактивних веб-додатків. Для розширення можливостей керування станом додатку буде використовуватися бібліотека Redux [23]. Вона дозволяє зберігати та змінювати стан додатку централізовано, що полегшує керування великими та складними додатками.

React дозволяє розділити інтерфейс на маленькі самостійні частини, що зменшує складність, й крім цього спрощує тестування. Також він дозволяє створювати динамічні інтерфейси без необхідності перезавантажувати сторінки. Redux же дозволяє зберігати глобальний стан додатку, що спрощує синхронізацію даних між різними частинами інтерфейсу.

Клієнтський додаток буде розроблено за типом SPA [24] (рис. 1.7). Це веб-додаток або сайт що працює на одній HTML-сторінці. У SPA весь необхідний HTML, CSS і JavaScript завантажується одноразово під час початкового завантаження сторінки [25]. Подальша взаємодія з користувачем і завантаження контенту відбуваються без повторного звернення до сервера для перезавантаження сторінки.

Вибір цієї архітектури дозволить реалізацію масштабованого та ефективного веб-додатку для організації курсів для вивчення мов. Використання C# та ASP.NET дозволить створити стабільну серверну частину, обрані патерни забезпечать чітку організацію коду. GraphQL дозволить оптимізувати запити між клієнтом і сервером, а React забезпечить зручний інтерфейс для користувачів.

SPA lifecycle



Рис. 1.7. Схема SPA

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Для створення додатку веб-платформи для організації та адміністрування курсів для вивчення мов було обрано наступні технології: мова розмітки HTML5; препроцесор стилів Sass [26]; клієнтська частина буде побудована за допомогою JavaScript та React; серверна частина використовуватиме C# та фреймворк ASP.NET; база даних Microsoft SQL Server [27].

HTML5 – мова розмітки, що забезпечує структуру веб-сторінок.

Sass – препроцесор для CSS, який додає змінні, вкладеність та інші можливості, він спрощує написання та підтримку стилів. Це підвищує ефективність розробки та дозволяє створювати масштабовані CSS-файли з кращою структурою.

Клієнтська частина веб-додатку буде розроблена з використанням JavaScript у поєднанні з React та Redux. Redux сам по собі синхронний та не має вбудованої підтримки асинхронної обробки, тому будуть використовуватися деякі допоміжні бібліотеки, а саме redux-observable та RxJS, бо вони дозволять

гнучко обробляти складну асинхронну логіку у вигляді потоків, підтримують скасування запитів, а також добре масштабуються при зростанні складності додатку.

JavaScript – мова програмування, яка забезпечує інтерактивність та динамічність веб-сторінки. Для кращого контролю за типізацією, буде використано TypeScript, це мова програмування, яка є надбудовою над JavaScript, та забезпечує кращий контроль над типами даних у коді.

React – досить популярна бібліотека для створення інтерфейсів, вона працює за принципом компонентної архітектури. Це дозволяє створювати повторно використовувані UI-компоненти, це полегшує підтримку та розширення додатку. React також забезпечує ефективне відображення елементів через віртуальний DOM, що значно покращує продуктивність додатку.

Серверна логіка буде реалізована за допомогою C# та ASP.NET.

C# – об'єктно-орієнтована мова програмування, яка забезпечує надійність, продуктивність та безпеку серверної частини додатка.

ASP.NET – фреймворк для створення веб-додатків, що підтримує швидку розробку та модульність, а також інтеграцію з іншими сервісами, він дозволяє ефективно обробляти запити.

Для управління та збереження даних буде використовуватися Microsoft SQL Server – це реляційна система керування базами даних (СКБД).

Вона забезпечує надійне збереження та швидкий доступ до інформації, має підтримку складних запитів та аналітичних операцій, надає можливості для захисту даних та резервного копіювання для запобігання втрати даних.

Аналіз та вибір СКБД було проведемо з урахуванням того, що доступ до даних має бути досить ефективним. Далі буде проведено порівняльну характеристику СКБД (табл. 1.2 – табл. 1.3).

Таблиця 1.2.

Функціональне порівняння СКБД

Характеристика	Microsoft SQL Server	Oracle	SQLite
Адміністративне керування	Добре	Відмінно	Задовільно
Графічні інструменти	Відмінно	Добре	Погано
Простота обслуговування	Відмінно	Відмінно	Відмінно
Функції з'єднання і вибір індексів	Відмінно	Відмінно	Добре
Повнотекстовий пошук	Добре	Відмінно	Задовільно
Функціональна сумісність	Добре	Добре	Добре
Інтеграція з іншими СКБД	Добре	Добре	Задовільно
Робота під керівництвом ОС	Добре	Добре	Добре
Можливості програмування	Добре	Відмінно	Добре
Процедури, що зберігаються та тригери	Відмінно	Відмінно	Добре
Вбудована мова програмування	Задовільно	Відмінно	Задовільно
Побудова БД	Добре	Відмінно	Задовільно
Мова SQL	Відмінно	Відмінно	Відмінно
Підтримка об'єктно-орієнтованої парадигми	Задовільно	Відмінно	Задовільно
Робота у режимі віддаленого доступу	Відмінно	Відмінно	Добре

Таблиця 1.3.

Оцінка багатокористувацьких СКБД

СКБД	Продуктивність	Конкурентний доступ	Кількість користувачів	Великі БД	Готовність
Microsoft SQL Server	7	8	7	6	7
Oracle	7	9	8	8	8
SQLite	6	7	5	5	7

В результаті порівняння СКБД, можна зробити такі висновки:

Oracle [28] є дуже багатофункціональною СКБД і найбільш підходить для керування великими проектами. Вона має широкий спектр функціональності та можливостей, що дозволяє ефективно використовувати її у великих та складних інформаційних системах.

SQLite [29] має свої обмеження та недоліки, вона виглядає менш “привабливою” СКБД у порівнянні з Oracle та Microsoft SQL Server. Вона підходить для простіших проектів або випадків, коли важлива легкість використання та мала вимоги до ресурсів.

Microsoft SQL Server, хоча трохи поступається Oracle за функціональністю, все ж є хорошим вибором для завдань, веб-платформи курсу для вивчення мов. Вона має достатню функціональність, щоб задовольнити вимоги такого проекту, і може пропонувати зручні інструменти для розробки та адміністрування.

Отже, для задач веб-платформи, Microsoft SQL Server є найбільш підходящим варіантом, забезпечуючи достатній рівень функціональності та продуктивності.

Висновок до першого розділу

У процесі аналізу веб-платформи для організації та адміністрування курсів з вивчення мов було проведено комплексний аналіз. Спочатку були сформульовані основні задачі проекту, визначено ключові функціональності системи та етапи реалізації. Аналіз існуючих платформ, таких як Moodle, Teachable та Preply, дозволив виділити оптимальні функції для платформи. На основі цього було обрано відповідну архітектуру веб-додатку, яка забезпечує продуктивність, масштабованість та зручність використання. Для серверної частини буде використано ASP.NET у поєднанні з GraphQL API для оптимізації обміну даними, тоді як клієнтська частина буде реалізована на React, що дозволяє створити швидкий та інтерактивний інтерфейс. Загалом вибір технологічного стеку – HTML5, CSS/Sass, JavaScript/TypeScript, React, Redux, C#, GraphQL, ASP.NET, забезпечує ефективність і стабільність системи. Аналіз систем керування базами даних показав, що Microsoft SQL Server є оптимальним рішенням для цього проекту, оскільки гарантує необхідний рівень продуктивності та надійності. Як результат було проведено аналіз, обґрунтування вибору архітектури та технологій для додатку для організації та адміністрування курсів з вивчення мов.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБ-ПЛАТФОРМИ

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи

Було проведено аналіз загальних вимог, серед основних можна виділити можливість реєстрації та авторизації в системі, доступ до персонального кабінету з можливістю редагування особистих даних, можливість редагування списку учасників як на платформі, так і на самому курсі. Також має бути доступним створення та налаштування курсів, публікація подій курсу, створення електронних словників та тестів, відстеження оплати курсу та оцінок за тести.

У системі розрізняють трьох ключових акторів: адміністратор, викладач та студент. Короткий опис акторів наведено в (табл. 2.1). Виявлення варіантів використання занесено до (табл. 2.2).

Таблиця 2.1

Виявлення акторів

Актор	Короткий опис
Адміністратор	Відповідає за повне адміністрування платформи: керує обліковими записами користувачів (створення, редагування, деактивація), має можливість керувати навчальними курсами, словниками, тестами, має доступ до всіх розділів платформи.
Викладач	Може подавати заявки на вихідний через систему, має доступ до редагування даних про курси які веде.
Студент	Має можливість переглядати доступні курси, календар подій (наприклад, дати занять), словники, тести, а також редагувати та керувати власним профілем користувача.

Виявлення варіантів використання

Основний актор	Найменування	Формулювання
1	2	3
Адміністратор	Реєстрація користувачів	Цей варіант дозволяє створювати обліковий запис у системі, а користувачі матимуть можливість самостійно їх активувати.
	Подальше керування обліковими записами користувачів	Цей варіант дозволяє керувати профілями користувачів.
	Керування курсами	Цей варіант дозволяє повністю керувати курсом, включаючи його створення, редагування, додавання словників, тестів, їх видалення, редагування списку учасників.
Викладач	Перегляд облікових записів користувачів	Цей варіант дозволяє викладачу переглядати список користувачів як на конкретному курсі, так і на платформі.
	Обмежене керування курсами	Цей варіант дозволяє переглядати та керувати курсами, але з обмеженим функціоналом на редагування списку учасників.

1	2	3
Студент	Перегляд курсів, словників, тестів	Цей варіант дозволяє переглядати доступні курси, їх словники та тести, також проходити тести та вивчати слова в режимі перегляду карток.
Адміністратор / Викладач	Публікація новин курсу	Цей варіант дозволяє публікувати новини, пов'язані з курсом.
Адміністратор / Студент / Викладач	Авторизація	Цей варіант дозволяє вводити свої облікові дані та отримати доступ до системи.
	Керування власним профілем	Цей варіант дозволяє редагувати свій профіль.

Таким чином на основі аналізу основних вимог, вдалося ідентифікувати ключових акторів та варіантів їх взаємодії з платформою. Тепер створена модель взаємодії користувачів буде основою для подальшого проєктування архітектури та реалізації інтерфейсів.

Крім цього було розроблено діаграму варіантів використання системи (рис. 2.1). В даному випадку вона дозволяє візуально представити основних користувачів платформи (студентів, викладачів та адміністраторів) та їх взаємодію з ключовими функціями системи. Вона дає загальне уявлення про доступні дії для кожної ролі та допоможе зрозуміти межі відповідальності кожного з користувачів, а також вона окреслює основні сценарії використання системи, такі як управління курсами, перегляд словників, тестів, та інше адміністрування.

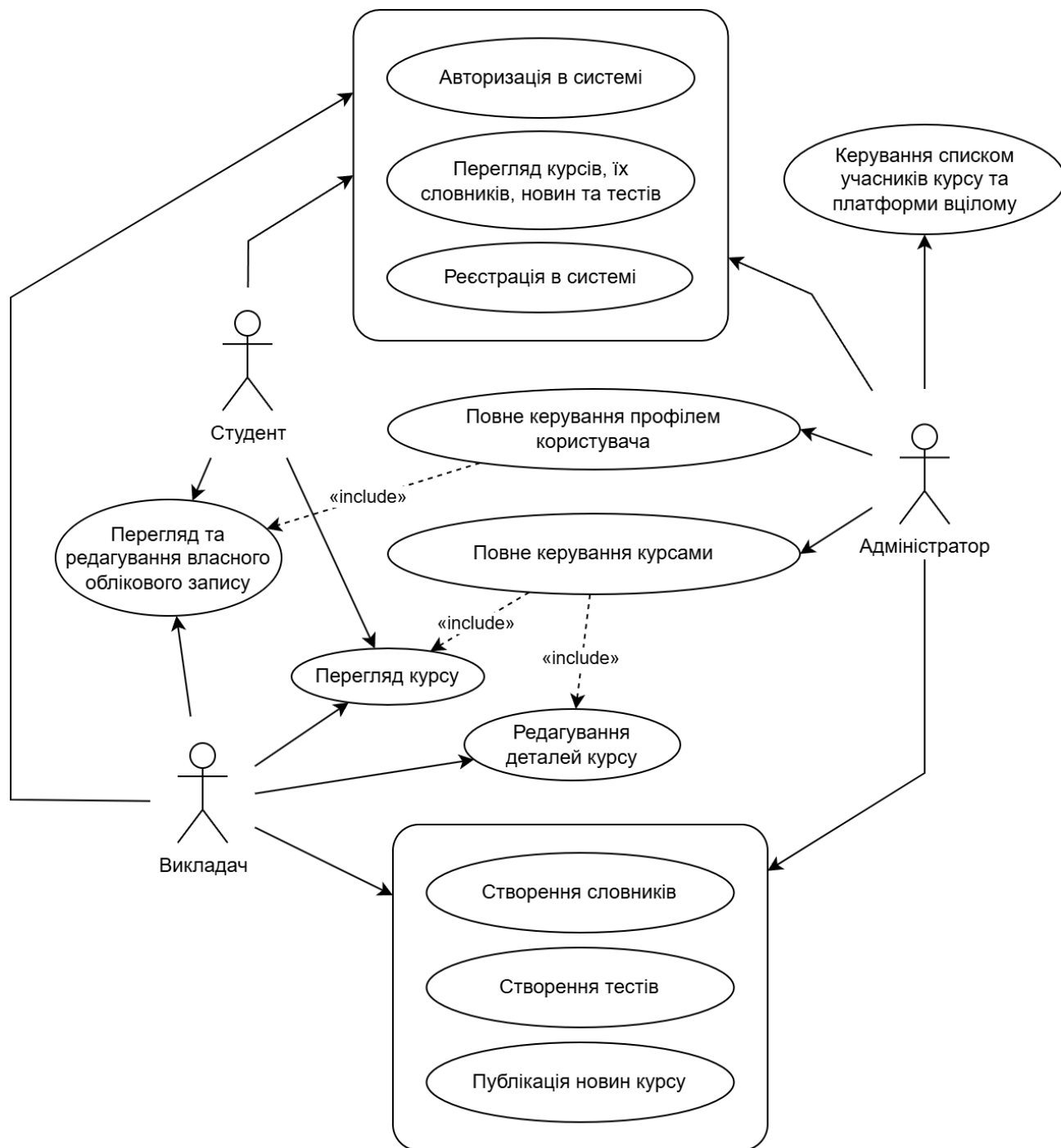


Рис. 2.1. Діаграма варіантів використання системи

На основі отриманих даних сформовано діаграму класів платформи (рис. 2.2), вона дозволяє зрозуміти основну структуру системи, її логіку та зв'язки між сутностями, які реалізують функціональність платформи. Вона показує які дані зберігаються в кожному класі, які методи доступні, та як об'єкти взаємодіють.

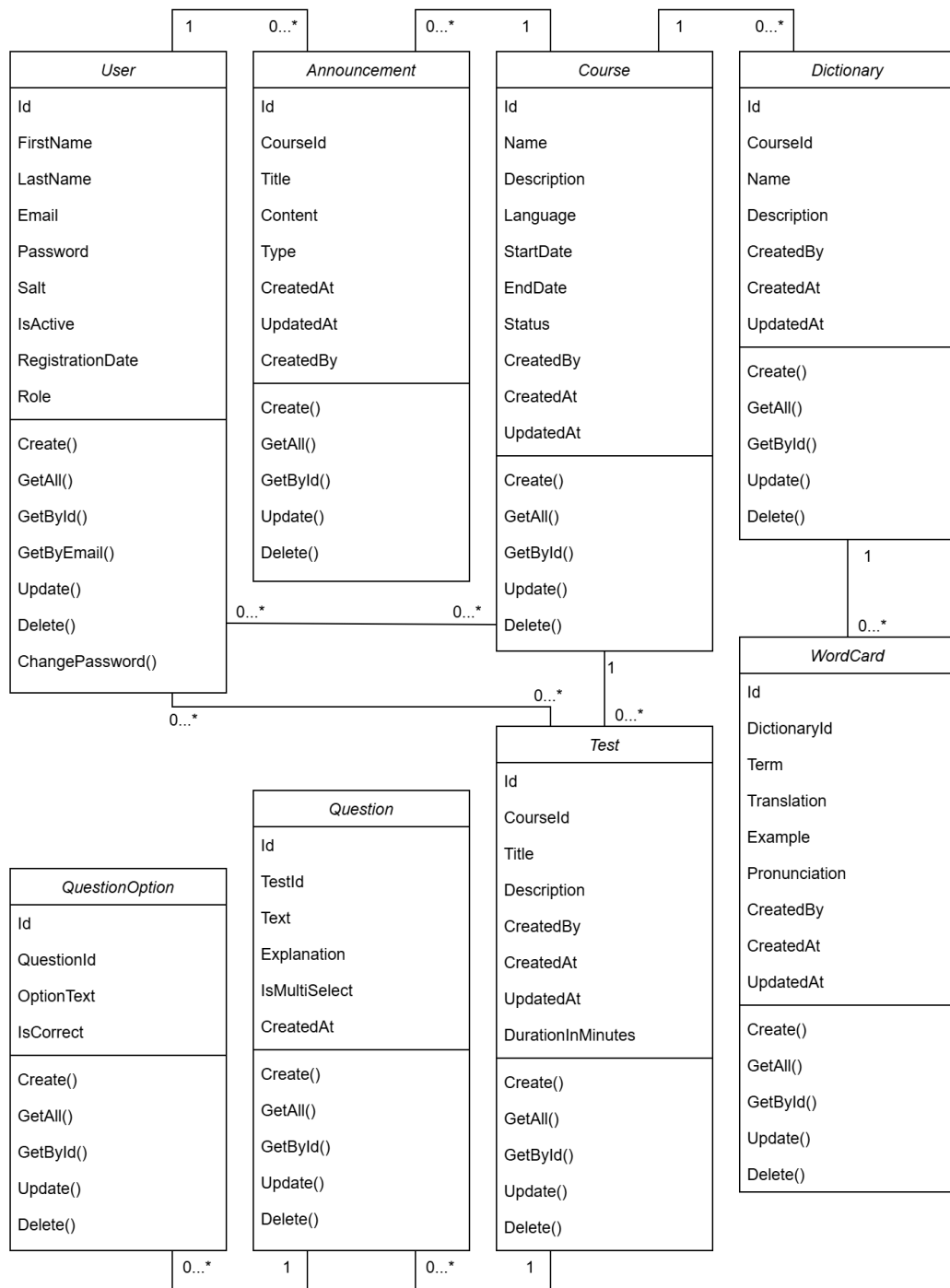


Рис. 2.2. Діаграма класів

2.2. Розробка бази даних системи

Для збереження даних вибрано СУБД Microsoft SQL, що забезпечує високу швидкість обробки запитів, надійність й ефективне управління. ER-діаграму спроектованої бази даних представлено на (рис. 2.3). Також ER-діаграму занесено до «Додатку А».

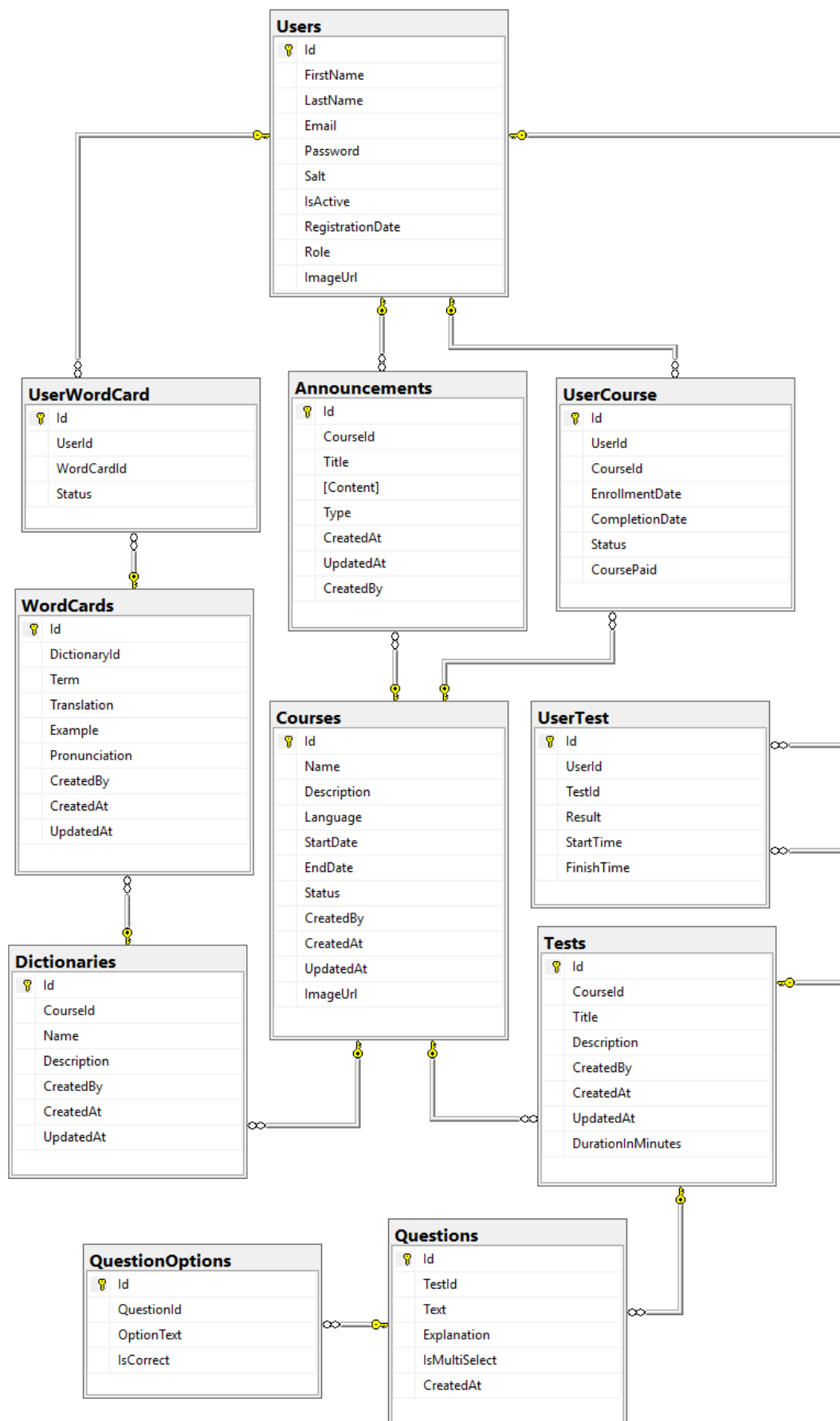


Рис. 2.3. ER-діаграма БД

Нижче наведено структуру бази даних у формі таблиць, де наведено ключові поля, типи даних та опис (табл. 2.3 – табл. 2.13).

Таблиця 2.3.

Структура таблиці «Users»

Поле	Ключ	Тип	Призначення
Id	PK	uniqueidentifier	Ідентифікатор користувача
FirstName	-	nvarchar(128)	Ім'я користувача
LastName	-	nvarchar(128)	Прізвище користувача
Email	-	nvarchar(255)	Емейл користувача
Password	-	nvarchar(128) (nullable)	Пароль користувача, збережений у вигляді хешу
Salt	-	nvarchar(128) (nullable)	Сіль для хешування паролю
IsActive	-	bit	Прапорець, який позначає чи активний користувач
RegistrationDate	-	datetime	Дата реєстрації
Role	-	tinyint	Роль користувача
ImageUrl	-	nvarchar(500) (nullable)	Шлях до картинки користувача в статичній директорії

Таблиця 2.4.

Структура таблиці «Announcements»

Поле	Ключ	Тип	Призначення
1	2	3	4
Id	PK	uniqueidentifier	Ідентифікатор оголошення
CourseId	FK	uniqueidentifier	Ідентифікатор курсу

Продовження таблиці 2.4.

1	2	3	4
Title	-	nvarchar(255)	Заголовок оголошення
Content	-	nvarchar(1000)	Контент оголошення
Type	-	tinyint	Тип оголошення
CreatedAt	-	datetime	Дата створення
UpdatedAt	-	datetime (nullable)	Дата оновлення
CreatedBy	FK	uniqueidentifier	Ідентифікатор користувача, який створив

Таблиця 2.5.

Структура таблиці «Courses»

Поле	Ключ	Тип	Призначення
Id	PK	uniqueidentifier	Ідентифікатор курсу
Name	-	nvarchar(256)	Назва курсу
Description	-	nvarchar(2048) (nullable)	Опис курсу
Language	-	nvarchar(64)	Мова курсу
StartDate	-	datetime	Дата початку
EndDate	-	datetime (nullable)	Дата завершення
Status	-	tinyint	Статус курсу
CreatedBy	FK	uniqueidentifier	Ідентифікатор користувача
CreatedAt	-	datetime	Дата створення
UpdatedAt	-	datetime (nullable)	Дата оновлення
ImageUrl	-	nvarchar(500) (nullable)	Шлях до картинки

Таблиця 2.6.

Структура таблиці «Dictionaries»

Поле	Ключ	Тип	Призначення
Id	PK	uniqueidentifier	Ідентифікатор словника
CourseId	FK	uniqueidentifier	Ідентифікатор курсу
Name	-	nvarchar(256)	Назва словника
Description	-	nvarchar(2048) (nullable)	Опис словника
CreatedBy	FK	uniqueidentifier	Ідентифікатор користувача, який створив
CreatedAt	-	datetime	Дата створення
UpdatedAt	-	datetime (nullable)	Дата оновлення

Таблиця 2.7.

Структура таблиці «Tests»

Поле	Ключ	Тип	Призначення
Id	PK	uniqueidentifier	Ідентифікатор тесту
CourseId	FK	uniqueidentifier	Ідентифікатор курсу
Title	-	nvarchar(256)	Назва тесту
Description	-	nvarchar(1024) (nullable)	Опис тесту
CreatedBy	FK	uniqueidentifier	Ідентифікатор користувача, який створив
CreatedAt	-	datetime	Дата створення
UpdatedAt	-	datetime (nullable)	Дата оновлення
DurationInMinutes	-	int	Тривалість тесту в хвиликах

Таблиця 2.8.

Структура таблиці «Questions»

Поле	Ключ	Тип	Призначення
Id	PK	uniqueidentifier	Ідентифікатор питання
TestId	FK	uniqueidentifier	Ідентифікатор тесту
Text	-	nvarchar(512)	Текст питання
Explanation	-	nvarchar(1024) (nullable)	Пояснення
IsMultiSelect	-	bit	Вибір кількох варіантів
CreatedAt	-	datetime	Дата створення

Таблиця 2.9.

Структура таблиці «QuestionOptions»

Поле	Ключ	Тип	Призначення
Id	PK	uniqueidentifier	Ідентифікатор варіанту відповіді
QuestionId	FK	uniqueidentifier	Ідентифікатор питання
OptionText	-	nvarchar(256)	Текст варіанту
IsCorrect	-	bit	Прапорець правильності варіанту

Таблиця 2.10.

Структура таблиці «UserWordCards»

Поле	Ключ	Тип	Призначення
Id	PK	uniqueidentifier	Ідентифікатор запису
UserId	FK	uniqueidentifier	Ідентифікатор користувача
WordCardId	FK	uniqueidentifier	Ідентифікатор картки слова
Status	-	tinyint	Статус картки

Таблиця 2.11.

Структура таблиці «WordCards»

Поле	Ключ	Тип	Призначення
Id	PK	uniqueidentifier	Ідентифікатор картки слова
DictionaryId	FK	uniqueidentifier	Ідентифікатор словника
Term	-	nvarchar(256)	Термін
Translation	-	nvarchar(256)	Переклад
Example	-	nvarchar(1024) (nullable)	Приклад використання
Pronunciation	-	nvarchar(256) (nullable)	Транскрипція
CreatedBy	FK	uniqueidentifier	Ідентифікатор користувача, який створив
CreatedAt	-	datetime	Дата створення
UpdatedAt	-	datetime (nullable)	Дата оновлення

Таблиця 2.12.

Структура таблиці «UserCourse»

Поле	Ключ	Тип	Призначення
Id	PK	uniqueidentifier	Ідентифікатор запису
UserId	FK	uniqueidentifier	Ідентифікатор користувача
CourseId	FK	uniqueidentifier	Ідентифікатор курсу
CoursePaid	-	bit (nullable)	Чи оплачено навчання
EnrollmentDate	-	datetime	Дата запису на курс
CompletionDate	-	datetime (nullable)	Дата завершення курсу
Status	-	tinyint	Статус курсу

Структура таблиці «UserTest»

Поле	Ключ	Тип	Призначення
Id	PK	uniqueidentifier	Ідентифікатор запису
UserId	FK	uniqueidentifier	Ідентифікатор користувача
TestId	FK	uniqueidentifier	Ідентифікатор тесту
Result	-	int	Результат тесту
StartTime	-	datetime	Час початку тесту
FinishTime	-	datetime (nullable)	Час завершення тесту

2.3. Проектування та реалізація алгоритмів роботи системи

Для реалізації веб-платформи обрано архітектурний підхід із застосуванням односторінкового інтерфейсу (SPA), він передбачає чітке розділення клієнтської та серверної логіки. Взаємодія між клієнтом і сервером буде реалізована через GraphQL API.

На відміну від традиційного REST, GraphQL надає можливість клієнту формулювати запити з точно визначеним набором полів, це дозволяє зменшити обсяг передачі даних і покращити швидкодію запитів. В цьому випадку сервер виступає як єдина точка доступу до даних, обробляючи запити клієнта й повертаючи лише необхідну інформацію у форматі JSON.

Серверна частина буде побудована на базі .NET-платформи з використанням бібліотек для реалізації GraphQL. Вона відповідає за обробку запитів, взаємодію з базою даних та бізнес-логіку. Клієнт, реалізований у вигляді SPA з використанням React, надсилає запити через GraphQL-клієнт та отримує структуровані відповіді для динамічного оновлення інтерфейсу без перезавантаження сторінки.

При зміні стану або навігації всередині застосунку, відбувається надсилання нового GraphQL-запиту, що забезпечує актуальність даних та безперервну взаємодію з користувачем.

Далі будуть наведені деякі діаграми активностей роботи платформи. Перша діаграма (рис. 2.4) ілюструє процес авторизації та реєстрації в системі, починаючи від введення логіну та пароля користувача, а також заповнення реєстраційної форми, і до успішного входу в систему.

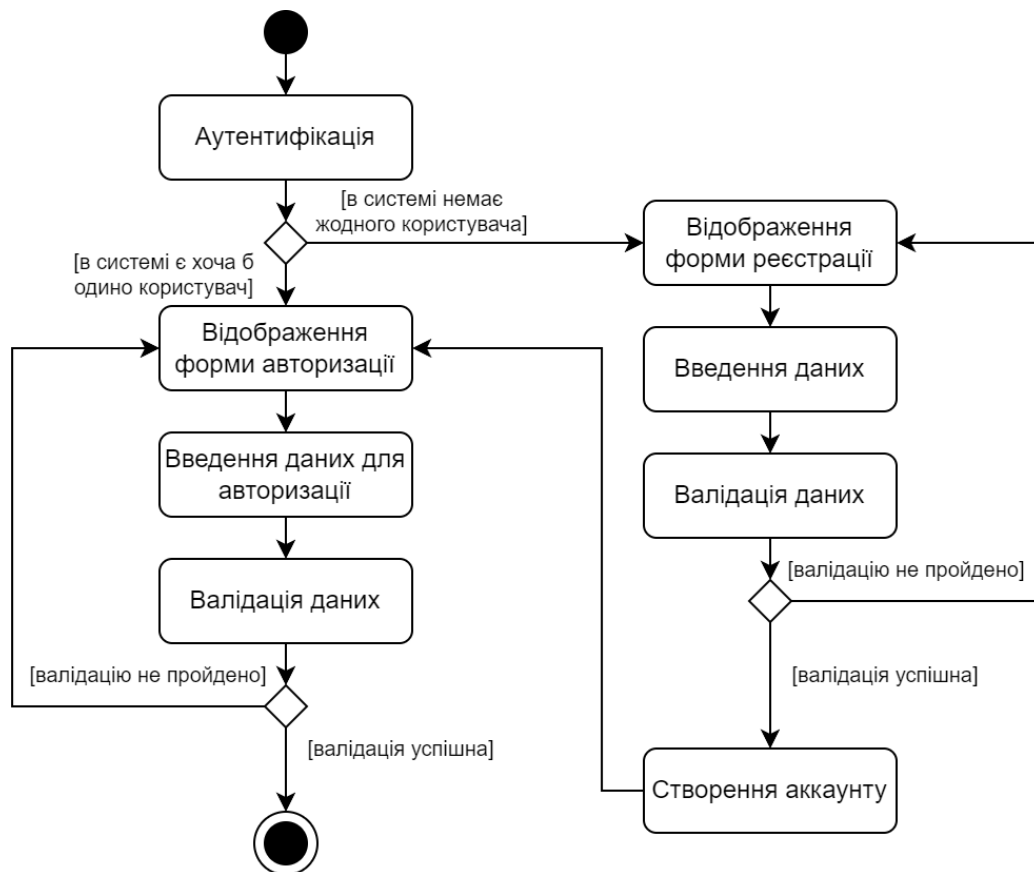


Рис. 2.4. Діаграма активності аутентифікації

Якщо в системі ще не зареєстровано жодного користувача, система пропонує виконати реєстрацію першого адміністратора. Відобразиться реєстраційна форма, у якій користувач має заповнити всі обов'язкові поля, щоб успішно пройти перевірку даних.

Подальші облікові записи створюються безпосередньо адміністратором через відповідний інтерфейс системи, далі вже самі користувачі виконують активацію створеного облікового запису.

Якщо в системі вже зареєстровано хоча б один обліковий запис, користувачеві пропонується авторизуватися. Під час входу в систему відображається сторінка з формою авторизації, де користувач має ввести свої облікові дані. Якщо введені дані невірні, користувач залишається на сторінці авторизації для повторної спроби. У разі введення коректних даних користувач отримує доступ до функціоналу системи відповідно до своєї ролі.

Діаграма активності (рис. 2.5) ілюструє процес керування обліковими записами.

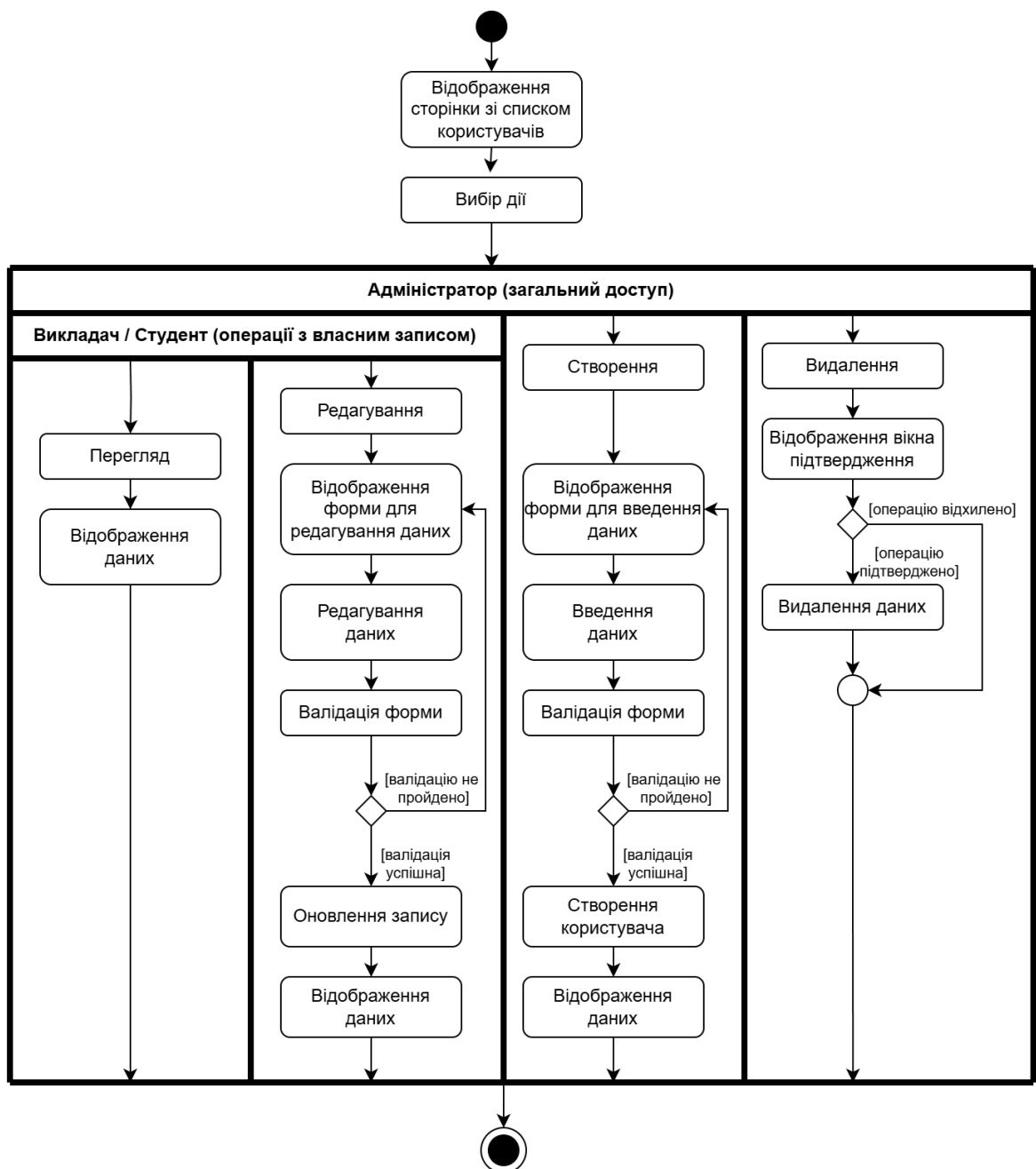


Рис. 2.5. Діаграма активності для керування обліковими записами

Спочатку відображається список користувачів, після чого обирається дія, відповідно до прав доступу. Для викладача дозволено лише перегляд, та редагування власного облікового запису, а для адміністратора доступні всі дії по відношенню до користувачів.

Дія «Перегляд» забезпечує відображення інформації про користувача.

Дія «Створення» відповідає за додавання нового облікового запису. Відображається форма для введення даних, яка проходить перевірку даних. Якщо перевірка даних успішна, створюється новий обліковий запис. У разі помилок користувач залишається на формі введення для виправлення даних.

Дія «Редагування» дозволяє змінювати дані існуючого облікового запису. Для цього відкривається форма редагування, після заповнення якої відбувається перевірка даних. Якщо вона успішна, обліковий запис оновлюється. У разі помилок користувач залишається на формі редагування для внесення коригувань.

Дія «Видалення» дозволяє видалити обраний обліковий запис. Ця дія доступна лише адміністратору. Під час видалення відображається вікно підтвердження, у якому користувач має підтвердити свій намір.

Для кращого розуміння модулів платформи, було створено діаграму компонентів. Основне її призначення – це показати, як різні компоненти системи взаємодіють між собою. Під компонентами розуміються різноманітні елементи, такі як модулі класів, незалежні системи або інші підсистеми, файли, бібліотеки, виконувані файли, пакети тощо, які можуть взаємодіяти з іншими частинами системи.

На діаграмі компонентів (рис. 2.6), продемонстровано загальну структуру системи. Компонент Web з'єднаний з компонентом API через інтерфейс, оскільки їх взаємодія здійснюється за допомогою певного набору методів. Ці методи визначають правила передачі даних та обміну інформацією, що дозволяє компонентам працювати разом, не залежачи від конкретних реалізацій методів.

Компонент API звертається до компонента Application для отримання доступу до даних та виконання основної функціональності системи. У свою чергу, Application використовує Domain, де зберігаються класи сутностей, що відповідають таблицям бази даних, а також компонент DB Provider, який забезпечує зв'язок між додатком і конкретною базою даних.

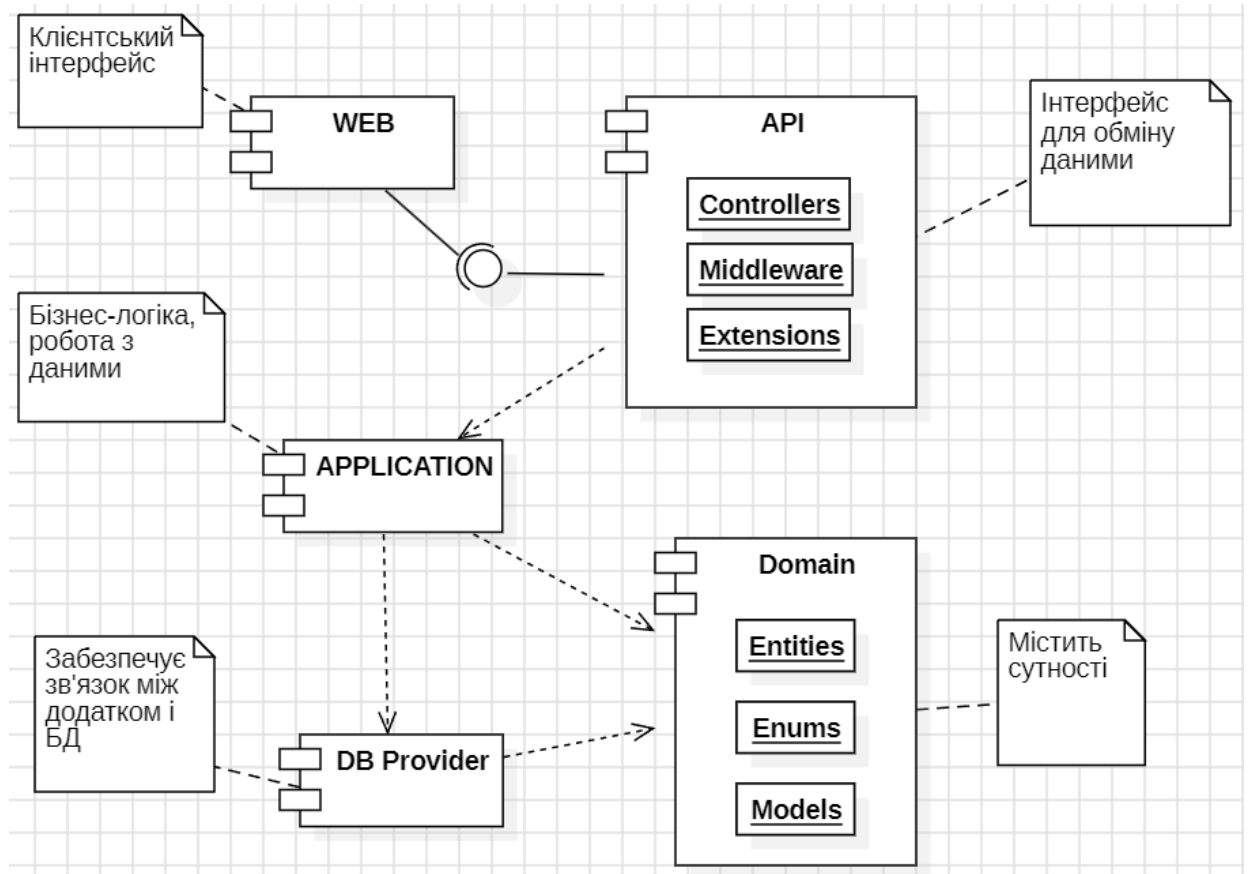


Рис. 2.6. Діаграма компонентів

2.4. Реалізація веб-платформи

Робота з додатком починається з процесу автентифікації користувача. Розглянемо реалізацію відповідного функціоналу.

Лістинг GraphQL класа в якому реалізовано перевірку чи є в системі хоча б один користувач. Повертає логічне значення, та використовує UserProvider для отримання доступу до методу CheckIfAnyExists, який перевіряє наявність користувачів в системі (рис. 2.7). Лістинг фрагменту з UserProvider методу який виконує перевірку зображено на (рис. 2.8).

```

3 references
public class ProfileQuery : ObjectGraphType
{
    0 references
    public ProfileQuery(CheckFirstUserExistenceResolver checkFirstUserExistence)
    {
        Field<BooleanGraphType>("CheckFirstUserExistence")
            .Description("Check if at least one user exists")
            .AllowAnonymous()
            .ResolveAsync(async context => await checkFirstUserExistence.ResolveAsync(context));
    }
}

```

Рис. 2.7. Дія яка відповідає за перевірку наявності користувачів в системі

```

2 references
public async Task<bool> CheckIfAnyExists()
{
    var sql = Queries.User.CheckIfExists;
    var user = await QuerySingleAsync<User>(sql);
    return user != null;
}

```

Рис. 2.8. Фрагменту з UserProvider методу який виконує перевірку

Лістинг фрагменту GraphQL класу в якому реалізовано реєстрацію першого користувача (адміністратора). Тут наведено фрагмент з GraphQL API та Resolver для відповідної дії (рис. 2.7 – рис. 2.8).

```

2 references
public async Task<AuthenticationResult?> ResolveAsync(IResolveFieldContext context)
{
    var input = context.GetArgument<FirstUserRegisterInput>("input");
    var salt = authenticationService.GenerateSalt();
    var user = new User()
    {
        Id = Guid.NewGuid(),
        RegistrationDate = DateTime.Now,
        Salt = salt,
        Role = Constants.UserRole.Admin,
        FirstName = input.FirstName,
        LastName = input.LastName,
        Email = input.Email,
        Password = authenticationService.GenerateHash(input.Password, salt),
    };

    await userProvider.Create(user);

    if (authenticationService.Authenticate(user, input.Password, out var token))
    {
        var userContext = context.RequestServices!.GetRequiredService<UserContext>();
        userContext.User = user;
        return new AuthenticationResult()
        {
            UserInfo = InitializeUserInfo(context.RequestServices!.GetRequiredService<UserContext>()),
            Token = token!,
        };
    }

    return null;
}

```

Рис. 2.9. Resolver реєстрації першого користувача


```
Field<AuthenticationResultType>("FirstUserRegister")
    .Description("Registers the first administrative user in the system")
    .Argument<NonNullGraphType<FirstUserRegisterInputType>>("input")
    .ResolveAsync(async context => await firstUserRegister.ResolveAsync(context));
```

Рис. 2.10. Дія реєстрації першого користувача

Лістинг фрагменту методу з провайдера який виконує створення користувачів наведено на (рис. 2.11).

```
3 references
public async Task<int> Create(User user)
{
    var sql = Queries.User.Create;
    return await ExecuteAsync(sql, user);
}
```

Рис. 2.11. Метод UserProvider для створення користувача

Лістинг дій авторизації користувача на клієнті. В даному випадку нас цікавлять дії requestFirstUserExistence\$ та register\$. Перша з яких перевіряє наявність користувача в системі, інша викликається в разі, якщо система не містить користувачів, й потрібно зареєструвати першого адміністратора (рис. 2.12 – рис. 2.13).

```
const epic: Epic<RootActions, RootActions, RootState> = (actions$, _state$) => {
    const login$ = actions$.pipe(
        ofType(LOGIN_REQUESTED),
        map(action => action.payload),
        switchMap(({ loginInput }) => request(loginMutation, { input: loginInput })).pipe(
            mergeMap(({ profile: { login } }) => {
                const loginFailed = !login;
                const actions: ProfileActions[] = [receiveLogin(loginFailed)];
                if (!loginFailed)
                    actions.push(authenticateUser(login.userInfo, login.token));
                return actions;
            })
        )
    );

    const register$ = actions$.pipe(
        ofType(USER_REGISTERED),
        map(action => action.payload),
        mergeMap(({ registerInput }) => request(registerMutation, { input: registerInput })).pipe(
            map(({ profile: { firstUserRegister } }) => authenticateUser(firstUserRegister.userInfo, firstUserRegister.token))
        )
    );
};
```

Рис. 2.12. Лістинг Profile Epic

```

const requestAuthentication$ = actions$.pipe(
  ofType(USER_AUTHENTICATION_REQUESTED),
  switchMap(() => request(authenticationMutation).pipe(
    mergeMap(({ profile: { authenticate } }) => {
      const notAuthenticated = !authenticate;
      const actions: any[] = [];
      if (notAuthenticated)
        actions.push(authenticateUser(null, null));
      else
        actions.push(authenticateUser(authenticate.userInfo, authenticate.token));
      return actions;
    })
  ))
);

const requestFirstUserExistence$ = actions$.pipe(
  ofType(FIRST_USER_EXISTENCE_REQUESTED),
  mergeMap(() => request(firstUserExistenceQuery).pipe(
    map(({ profile: { checkFirstUserExistence } }) => receiveFirstUserExistence(checkFirstUserExistence))
  )),
);

```

Рис. 2.13. Продовження лістингу Profile Epic

Таким чином, робота з додатком починається з перевірки, чи існує хоча б один користувач у системі. Якщо користувачі відсутні, система пропонує зареєструвати першого адміністратора. Саме з цього і починається процес ініціалізації. Для цього на стороні сервера реалізовано окремий GraphQL action, який звертається до відповідного методу провайдера і повертає логічне значення. У разі відсутності користувачів, клієнтська частина додатку ініціює реєстрацію, під час якої створюється новий користувач із роллю адміністратора. Після успішного створення облікового запису та генерації токена автентифікації, користувач автоматично авторизується. Детальний лістинг системи та інших ключових аспектів занесено до «Додатку Б».

Висновки до другого розділу

У другому розділі було здійснено проектування веб-платформи. Було проведено аналіз функціональних вимог до системи, на основі чого визначено ключові ролі користувачів (адміністратор, викладач, студент) та сценарії їх

взаємодії з платформою. Це дозволило побудувати діаграми варіантів використання, що візуалізують основні функції системи.

Розроблено об'єктно-орієнтовану модель системи у вигляді діаграми класів, що демонструє структуру та взаємозв'язки між об'єктами. Спроектовано реляційну базу даних із застосуванням СУБД Microsoft SQL Server: створено ER-діаграму, визначено структуру таблиць, їх ключі та зв'язки.

Також було обґрунтовано вибір архітектурного підходу, описано логіку обробки запитів між клієнтом і сервером. Окрему увагу приділено реалізації механізмів авторизації, реєстрації та керування користувачами, це супроводжувалося діаграмами активностей і фрагментами коду серверної та клієнтської частин. Також було побудовано діаграму компонентів, що описує загальну структурну та взаємодію модулів.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-ПЛАТФОРМОЮ

3.1. Порядок встановлення та налаштування системи

Для зображення процесу розгортання системи буде використовуватися UML-діаграма, а саме діаграма розгортання.

Вона призначена для візуалізації архітектури системи, вона відображає розміщення складових, наприклад програмних модулів або мережових елементів. Відображає їх розміщення на фізичних чи віртуальних пристроях. Нижче представлено діаграму розгортання системи (рис. 3.1).

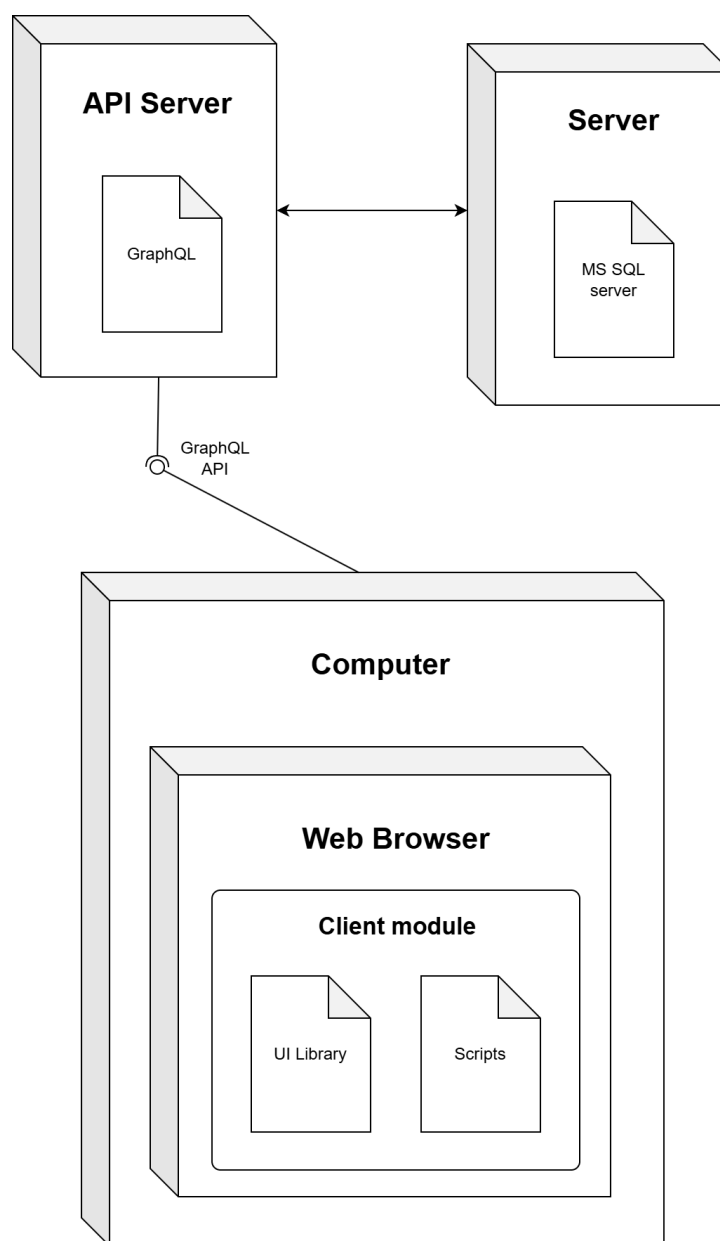


Рис. 3.1. Діаграма розгортання системи

Після побудови цієї діаграми можна зробити висновок до процесу встановлення та налаштування системи. А саме те, що серверна частина взаємодіє з сервером для доступу до необхідних даних та ресурсів. Клієнтська частина реалізована у вигляді веб-застосунку. Він використовує бібліотеки стилів для створення інтерфейсу користувача та виконувани сценарії для налаштування логіки оновлення контенту. Під час запуску застосунку клієнт спочатку завантажує початковий стан HTML сторінки, після чого вже динамічно оновлює її за допомогою асинхронних запитів до Graph API.

3.2. Структура інтерфейсу веб-платформи

Після розгортання платформи, користувач може створити першого адміністратора (рис. 3.2), та відкривати платформу до публічного доступу.



Create first user

First name

Last name

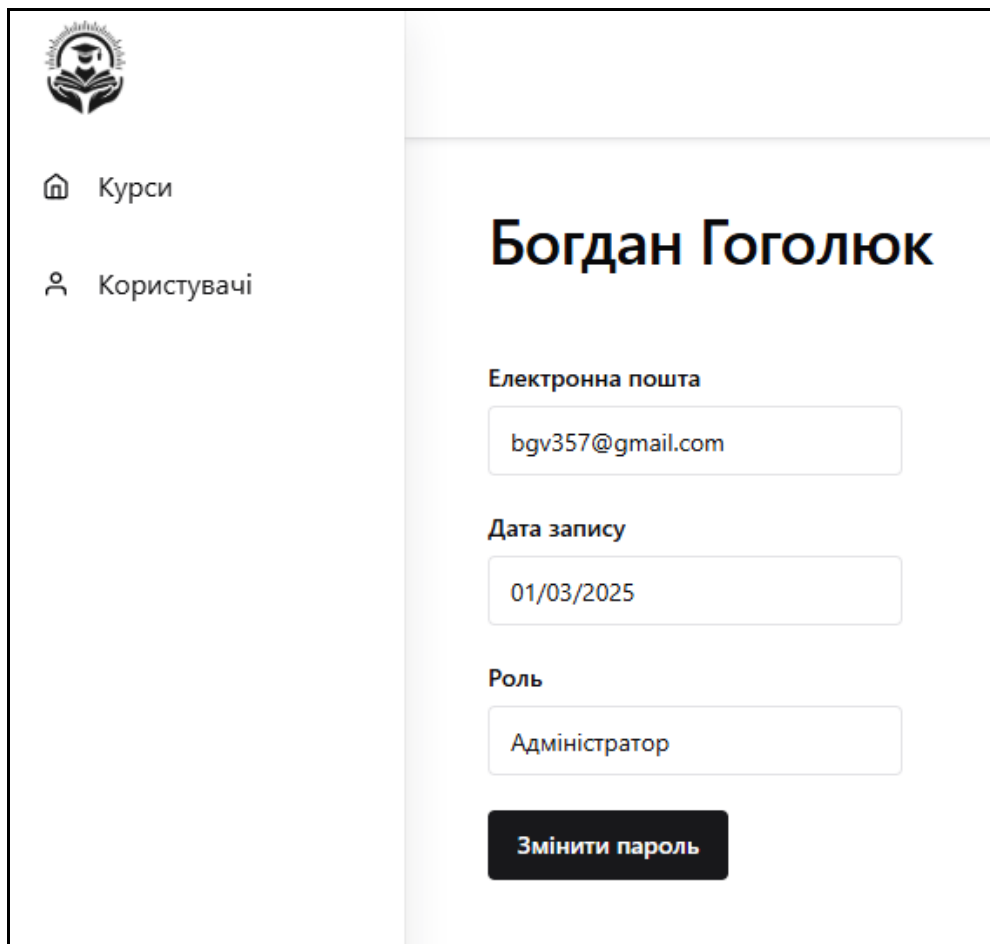
Email

Password

☐

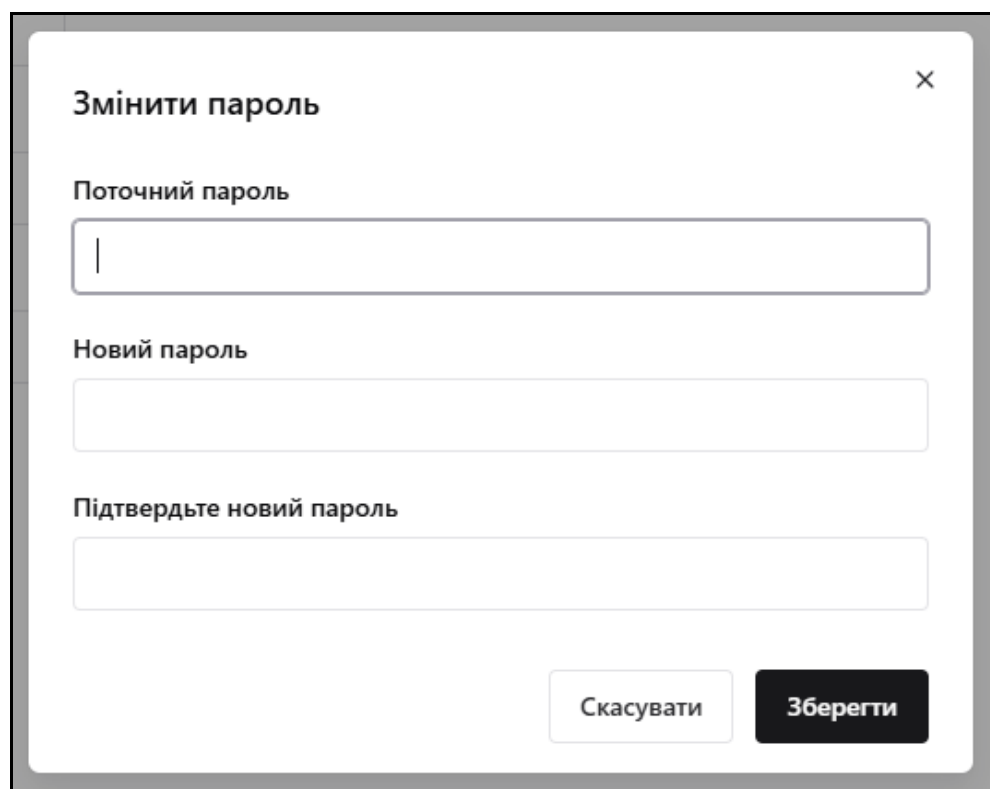
Рис. 3.2. Реєстрація адміністратора

Коли користувача створено, ми можемо проглянути особисті дані в кабінеті (рис. 3.3), де безпосередньо можемо змінити пароль (рис. 3.4).



The image shows a user profile page. On the left is a sidebar with a logo at the top and two menu items: 'Курси' (Courses) and 'Користувачі' (Users). The main content area displays the user's name 'Богдан Гоголюк' in large bold text. Below the name are three input fields: 'Електронна пошта' (Email) containing 'bgv357@gmail.com', 'Дата запису' (Registration date) containing '01/03/2025', and 'Роль' (Role) containing 'Адміністратор'. At the bottom of the main area is a dark button labeled 'Змінити пароль' (Change password).

Рис. 3.3. Профіль користувача



The image shows a modal dialog titled 'Змінити пароль' (Change password) with a close button (X) in the top right corner. It contains three input fields: 'Поточний пароль' (Current password), 'Новий пароль' (New password), and 'Підтвердьте новий пароль' (Confirm new password). At the bottom right are two buttons: 'Скасувати' (Cancel) and 'Зберегти' (Save).

Рис. 3.4. Зміна паролю

Подальші створення користувачів відбуваються через форму створення користувача безпосередньо адміністраторами (рис. 3.5). Після створення, користувач з вказаною поштою отримає лист для підтвердження реєстрації й можливістю перейти на платформу та створити пароль (рис. 3.6).

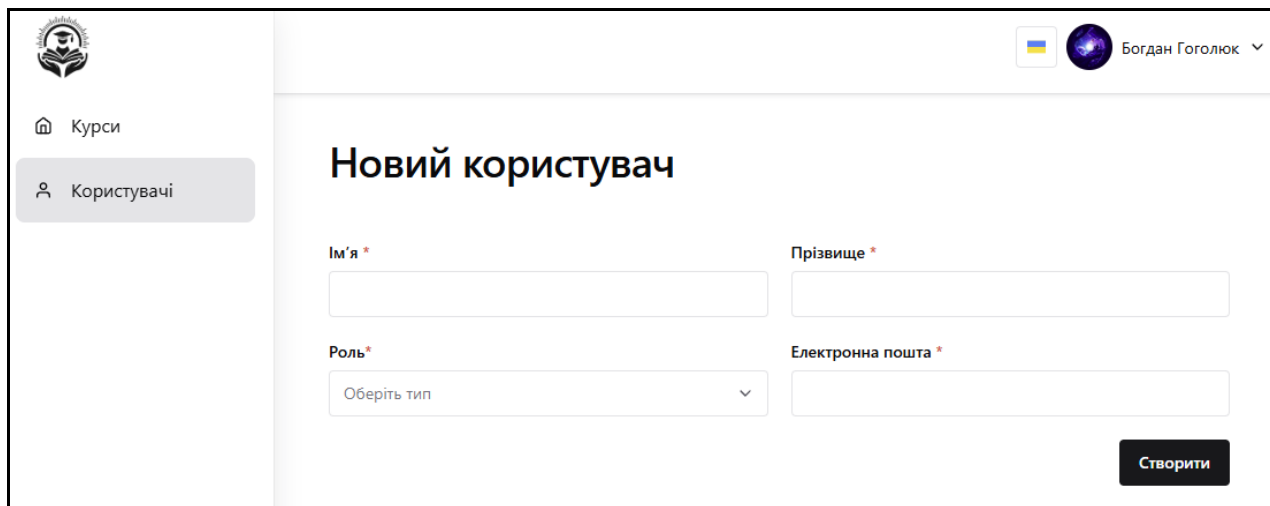


Рис. 3.5. Сторінка створення нового користувача

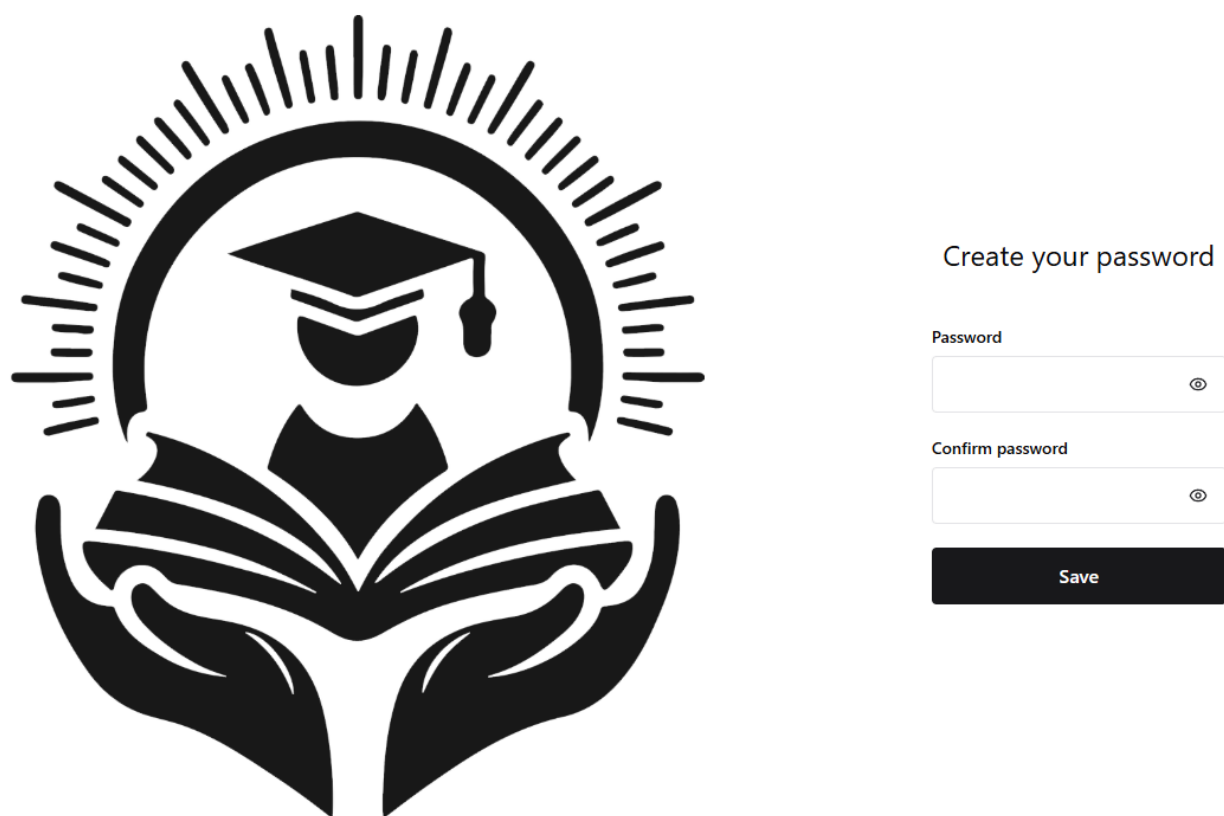


Рис. 3.6. Сторінка створення паролю

До сторінки користувачів (рис. 3.7) мають доступ адміністратори та викладачі, але викладачі мають доступ лише до перегляду. Також сторінка керування користувачами платформи має додаткові фільтри та пошук.

Користувачі

▼ Пошук за ім'ям або електронною поштою

Додати нового користувача

Ім'я	Електронна пошта	Роль	Дата зарахування	
Богдан Гоголюк	bgv357@gmail.com	Адміністратор	01/03/2025	
Олівія Андерсон	bgv357+olivia_anderson@gmail.com	Студент	01/27/2025	
Лаура Мур	bgv357+laura_moore@gmail.com	Викладач	02/03/2025	
Давид Мартінес	bgv357+david_martinez@gmail.com	Студент	02/05/2025	
Том Піастрі	bgv357+tom_piastri@gmail.com	Адміністратор	03/05/2025	
Джеймс Андерсон	bgv357+james_anderson@gmail.com	Студент	03/07/2025	

Рис. 3.7. Сторінка списку користувачів

Користувачам доступні курси до яких вони належать, вони можуть виконувати пошук та фільтрацію за певними параметрами (рис. 3.8).



Богдан Гоголюк

Курси

[Додати новий курс](#)

Усього записів: 15

< 1 2 >

A hand-drawn card titled 'ENGLISH COURSE' is pinned to a corkboard. The card is surrounded by various colorful sticky notes and a pair of glasses. A hand is holding a black marker, pointing at the card.

Англійська для початківців

Вступний курс англійської мови для початківців, який охоплює базові теми: алфавіт, просту граматику, привітання, запитання, числівники та найуживаніші дієслова. У програмі передбачені інтерактивні вправи, прослуховування та рольові ігри, що допомагають подолати мовний бар'єр. Курс ідеально підходить для тих, хто раніше не вивчав англійську або має мінімальний словниковий запас.

English 06/02/2025 - 09/01/2025

ДВ Даніель Вілсон

ОВ Олівія Браун

[Переглянути](#)

Назва курсу

Мова

Дата початку

mm/dd/yyyy



Дата завершення

mm/dd/yyyy



Статуси

Оберіть статус



Пошук

Скинути фільтри

Іспанська: середній рівень

Курс іспанської мови для учнів середнього рівня, що фокусуються на вдосконаленні граматики, побудові складніших речень і розширенні словникового запасу. Темі включають подорожі, роботу, здоров'я, а також практику розмов у парі та групах. Курс включає інтерактивні завдання, аудіо- та відеоматеріали, а також регулярні тести для закріплення матеріалу.

Spanish 07/01/2025 - 09/30/2025

ЕТ Ендрю Тейлор

A stylized logo featuring a graduation cap (mortarboard) positioned above an open book. The entire graphic is enclosed within a circular frame with radiating lines, suggesting a sun or a spotlight.

Відкривши курс (рис. 3.9), користувач має доступ до контенту, списку користувачів, новин (рис. 3.10), словників та тестів. Адміністратор має доступ до додавання користувачів на курс, та редагування контенту. Викладача також має доступ до редагування курсу, окрім додавання нових користувачів на курс.

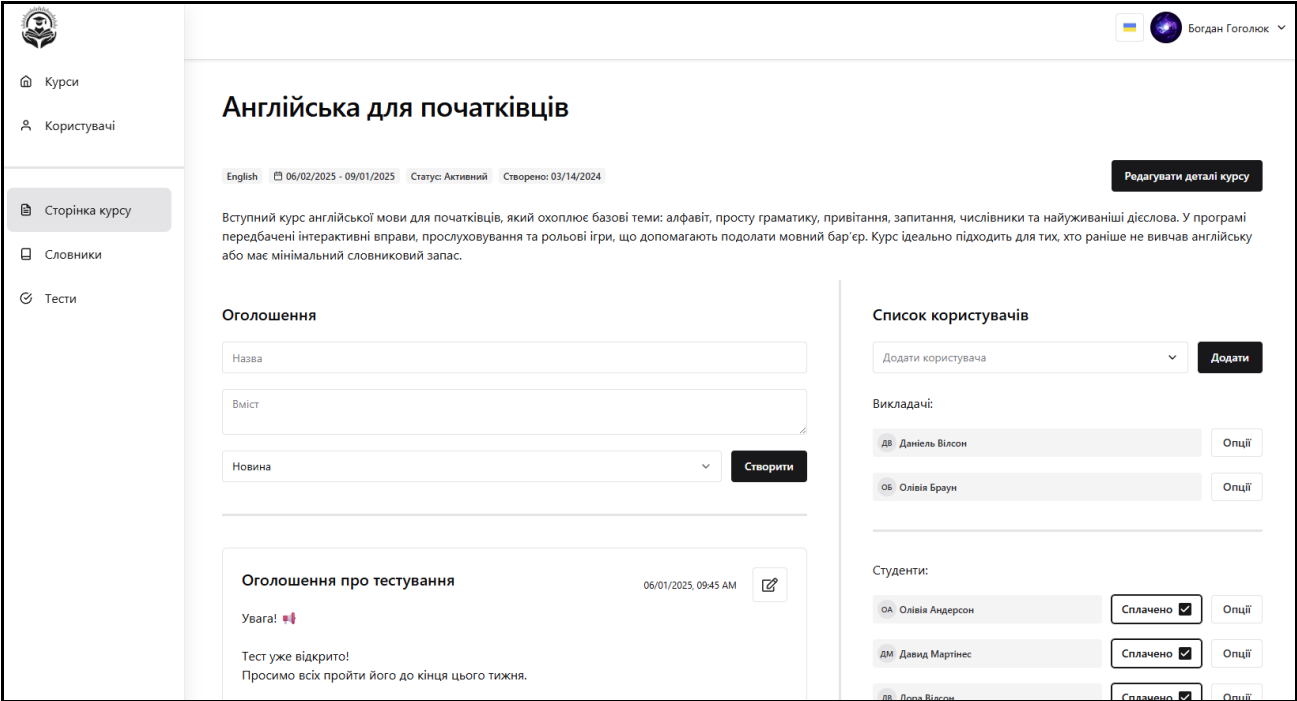


Рис. 3.9. Сторінка курсу

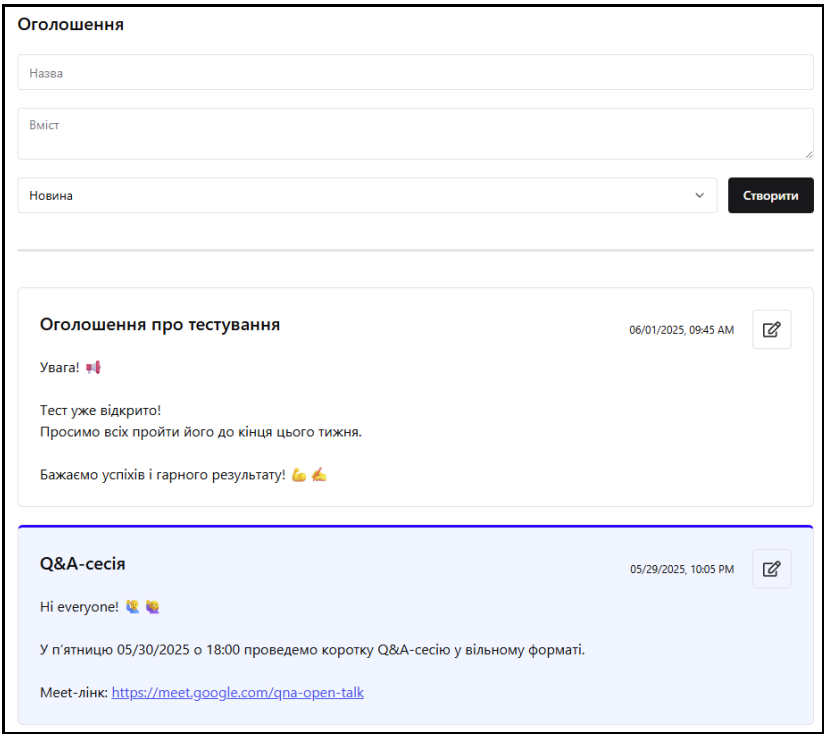


Рис. 3.10. Новини курсу

Перейшовши на розділ словників (рис. 3.11), адміністратор та викладач можуть створити новий словник. Звичайні користувачі бачать лише список словників.

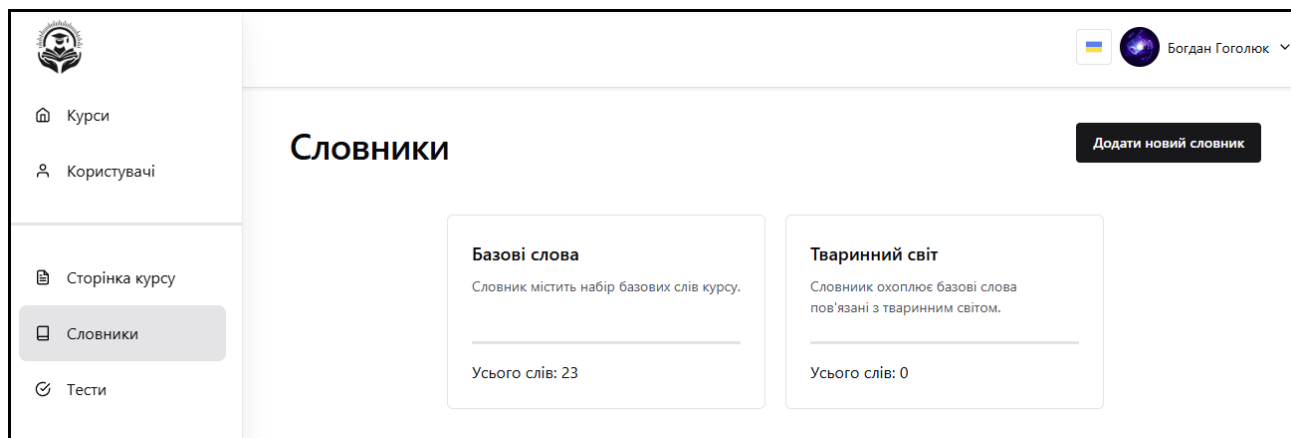


Рис. 3.11. Словники курсу

Перейшовши на конкретний словник (рис. 3.12), ми можемо побачити його вміст, а саме слова, фільтрацію та пошук по ним, та окремий режим навчання (рис. 3.13).

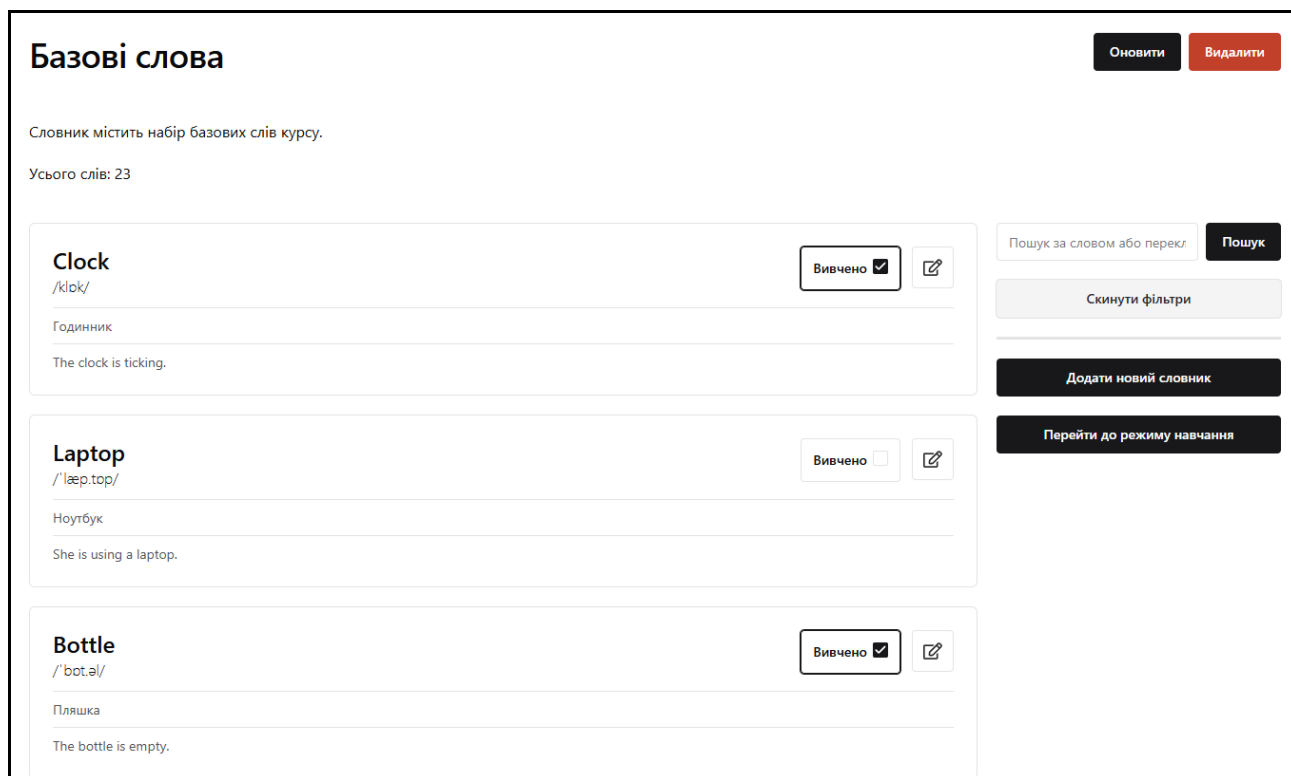


Рис. 3.12. Деталі словника

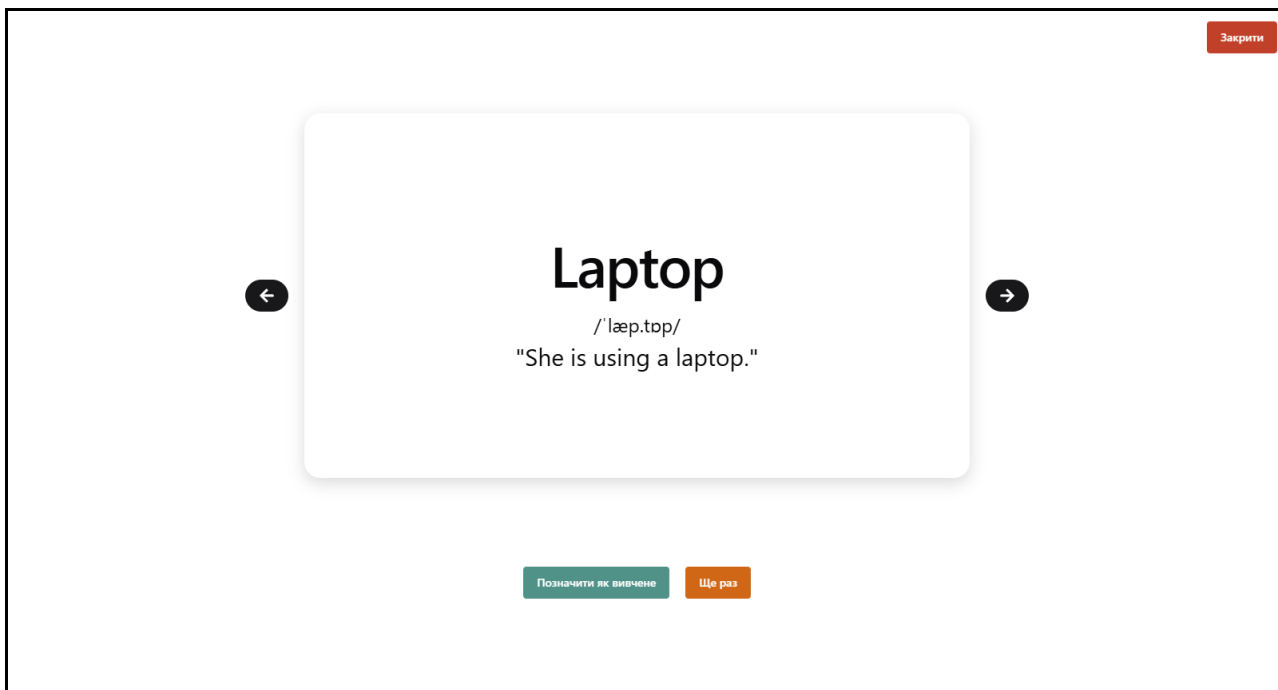


Рис. 3.13. Режим навчання

Список тестів виглядає аналогічно списку словників, з аналогічними обмеженнями по доступу (рис. 3.14). Коли адміністратор або викладач відкриває конкретний тест, вони мають доступ до його редагування та перегляду результатів тестування (рис. 3.15 – рис. 3.16).

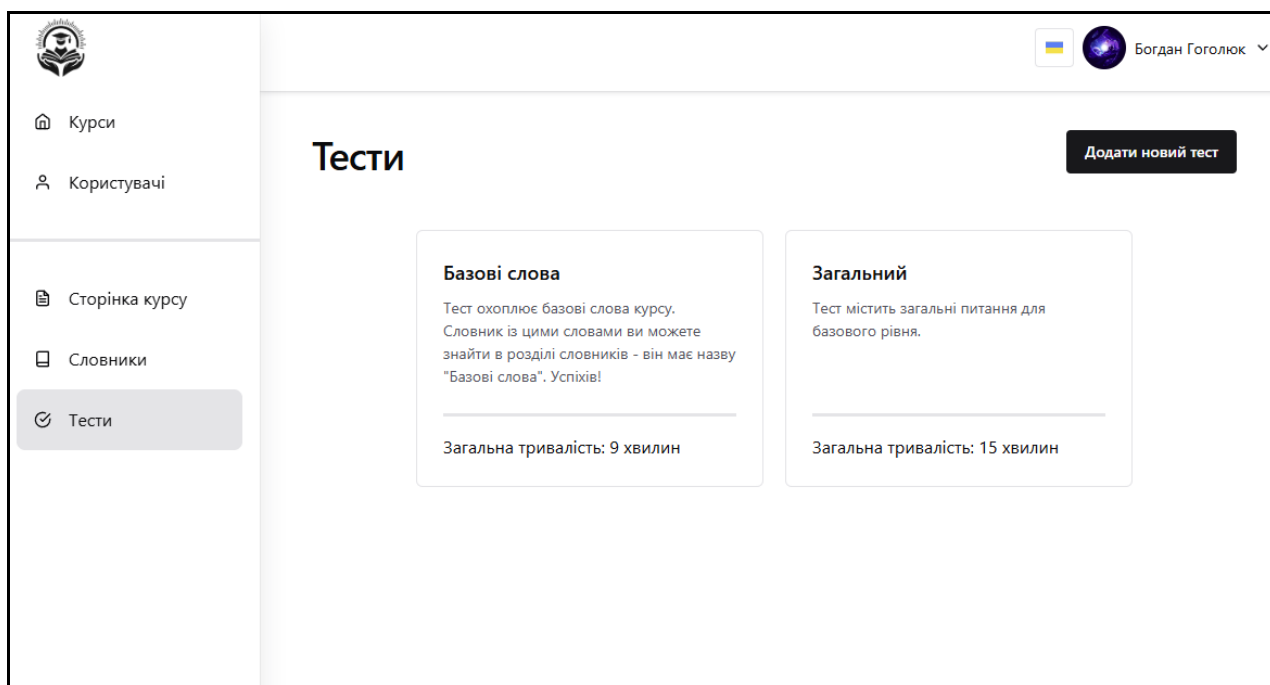


Рис. 3.14. Список тестів

Загальний

Тест містить загальні питання для базового рівня.

Загальна тривалість: 15 хвилин

Усього питань: 13

Choose the correct pronoun: ___ is my mother.

This question helps test subject pronouns. Pick the one that can refer to a female person.

They

It

She

He

Пошук за текстом

Пошук

Скинути фільтри

Додати нове питання

Результати тесту

Рис. 3.15. Сторінка деталей тесту

Нове питання

Питання *

Питання 1

Пояснення

Пояснення до питання

☐ Множинний вибір відповіді

Додати новий варіант відповіді

Варіанти відповіді:

Вар 1

Варіант 2

Варіант 3 :)

Скасувати

Створити

Рис. 3.16. Створення нового запитання

3.3. Тестування платформи

У цьому розділі описується підхід до тестування системи, він включає розробку плану та визначення стратегії тестування, а також обсяг та послідовність тестів. Тестування охоплює основні функціональні блоки та критичні сценарії використання.

Функціональне тестування забезпечує перевірку відповідності функцій вимогам і очікуванням користувачів, зокрема реєстрації, пошуку курсів, взаємодії з ними [30].

Під час тестування продуктивності оцінюється швидкодія системи за різного навантаження, аналізується час відкриття сторінок, відповіді сервера та інші ключові метрики.

З точки зору безпеки, система перевіряється на стійкість до типових загроз, таких як SQL-ін'єкції, несанкціонований доступ.

Нижче опрацьовано набір основних тест-кейсів для роботи певних модулів додатку.

Тест-кейс 1: Створення нового користувача студента.

Модуль: Створення користувача.

Кроки:

- Вводимо необхідні дані (ім'я, прізвище, пошту);
- Вибираємо роль «Студент»;
- Створюємо обліковий запис, й надсилаємо імейл для активації на вказану адресу;
- Студент активує аккаунт та створює пароль для облікового запису.

Очікуваний результат: Користувача успішно створено та він має доступ до платформи відповідно до своєї ролі.

Тест-кейс 2: Пошук певного курсу серед усього списку.

Модуль: Пошук курсу

Кроки:

- Вводимо пошуковий текст, це може бути назва курсу, або ж його коротке пояснення;
- Обираємо додаткові параметри фільтрування, наприклад мову;
- Натискаємо пошук.

Очікуваний результат: Список курсів буде відфільтровано відповідно до обраного фільтрування, якщо ж записів за таким фільтром не знайдено, відобразити порожній стан списку.

Тест-кейс 3: Створюємо оголошення для студентів про деталі наступного уроку.

Модуль: Створення оголошення.

Кроки:

- Відкриваємо деталі курсу;
- Знаходимо поле для створення новини, й заповнюємо відповідні поля;
- Обираємо тип оголошення (новина або подія), в залежності від типу, буде відрізнятися ступінь важливості оголошення, й візуально буде різнитися відображення.

Очікуваний результат: буде додано нове оголошення, відповідно до обраного типу та з введеними даними.

Тест-кейс 4: Вивчення слів в словнику.

Модуль: Робота зі словниками.

Кроки:

- Переходимо в режим вивчення слів;
- Відповідно до флеш карток відмічаємо для кожного слова його стан. Можна позначити слово як вивчене, або ж повторити його ще раз, що перенесе слово трішки далі по колоді;
- Потрібно позначити декілька слів як вивчені;

- Далі потрібно перевірити, чи в списку слів у словнику, змінився їхній статус відповідно до того який було обрано.

Очікуваний результат: якщо слово позначили як вивчене, воно буде так і відображатися в списку слів, якщо ж обрали продовжити вивчення слова, воно має позначатися як невивчене.

Тест-кейс 5: Створюємо нове питання до тесту.

Модуль: Редагування тесту.

Кроки:

- Відкриваємо форму для створення питання;
- Заповнюємо відповідні поля;
- Додаємо варіанти відповіді.

Очікуваний результат: буде додано нове питання в межах поточного тесту, з можливістю перегляду обраної відповіді та деталей питання.

Тест-кейс 6: Проходження тесту студентом.

Модуль: Проходження тесту.

Кроки:

- Студент розпочинає тест;
- Починається зворотній відлік таймера;
- Студент закінчує тест, або час на проходження тесту виходить;
- Результати надсилаються на сервер для перевірки;
- З сервера повертається результат та відображається для студента та для викладачів в результатах.

Очікуваний результат: буде додано новий результат проходження тесту в списку результатів.

Тест-кейс 7: Видалення студента з курсу.

Модуль: Видалення користувача з курсу.

Кроки:

- Обираємо розгортаємо список дій які можна виконувати над учасником курсу;

- Обираємо підпункт з видаленням студента з курсу;
- Підтверджуємо видалення студента з курсу.

Очікуваний результат: студент відразу має зникнути зі списку користувачів курсу, та курс має бути недоступним в списку його курсів.

За результатами тестування було підтверджено, що основні функціональні можливості веб-платформи для організації та адміністрування курсів для вивчення мов було реалізовано коректно та відповідно до вимог. Усі тест-кейси виконано успішно, що свідчить про стабільну роботу системи та її готовність до виконання запланованих задач.

Висновки до третього розділу

У цьому розділі було проведено аналіз та описано порядок встановлення й налаштування системи для коректної роботи платформи. Крім того, було описано та проілюстровано основний функціонал платформи, від реєстрації та створення користувачів до створення й редагування деталей курсу, словників, тестів і перегляду результатів. Після чого було описано тестування платформи та розглянуто основні тест-кейси.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було спроектовано та реалізовано веб платформу для організації та адміністрування курсів для вивчення мов.

Було проведено аналіз предметної області, визначено основні функціональні та нефункціональні вимоги до системи, сформульовано цілі та задачі проєкту. Було проведено порівняльний аналіз популярних платформ, таких як Moodle, Teachable та Preply, це порівняння дозволило виокремити ключові моменти, які забезпечують ефективність та зручність подібних систем. На основі зібраної інформації було обґрунтовано вибір технологічного стеку, що включає ASP.NET, GraphQL, React, Redux, TypeScript, Microsoft SQL Server та інші інструменти. Обрана архітектура забезпечує масштабованість та стабільність, забезпечує взаємодію між клієнтом і сервером, а також підтримує гнучке розширення функціоналу.

Також в процесі виконання роботи було визначено основні ролі користувачів (адміністратор, викладач, студент) та сценарії їх взаємодії. На основі цих сценаріїв побудовано діаграми варіантів використання, що деталізують функціональні можливості кожного типу користувача. Була розроблена об'єктно-орієнтована модель у вигляді діаграми класів, спроектовано реляційну базу даних, створено ER-діаграму та описано структуру таблиць. Крім того, було описано механізми реєстрації, авторизації, обробки запитів між клієнтом і сервером, а також взаємодію між модулями системи за допомогою діаграми компонентів і діаграм активностей. Деякі архітектурні рішення були супроводжені фрагментами коду та прикладами реалізації ключових функцій.

Потім було описано технічні аспекти впровадження платформи. Було описано процес встановлення, налаштування середовища розгортання, запуску клієнтської та серверної частин. Детально проаналізовано функціонал системи: від створення та керування користувачами до управління курсами, словниками,

тестами та перегляду результатів. Було проведено тестування системи із використанням ключових тест-кейсів, що дозволило перевірити стабільність і коректність роботи основних модулів системи.

У підсумку, пройшовши усі етапи розробки та тестування програмного застосунку, було реалізовано веб платформу для організації та адміністрування курсів для вивчення мов. Вона забезпечує взаємодію між усіма учасниками навчального процесу, дозволяє гнучко керувати курсом та автоматизує адміністративні процеси.

ДЖЕРЕЛА ІНФОРМАЦІЇ

1. Структури бази даних [Електронний ресурс] – Режим доступу до ресурсу:

https://uk.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B8_%D0%B1%D0%B0%D0%B7_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85

2. Серверна частина веб додатку [Електронний ресурс] – Режим доступу до ресурсу: <https://brainlab.com.ua/uk/blog-uk/back-end-rozrobka>

3. Клієнтська частина веб додатку [Електронний ресурс] – Режим доступу до ресурсу:

https://uk.wikipedia.org/wiki/Front_end_%D1%82%D0%B0_back_end

4. Moodle [Електронний ресурс] – Режим доступу до ресурсу: <https://en.wikipedia.org/wiki/Moodle>

5. Teachable [Електронний ресурс] – Режим доступу до ресурсу: <https://teachable.com/>

6. Preply [Електронний ресурс] – Режим доступу до ресурсу: <https://preply.com/>

7. Клієнт-серверна архітектура [Електронний ресурс] – Режим доступу до ресурсу: <https://training.qatestlab.com/blog/technical-articles/client-server-architecture/>

8. Мова програмування C# [Електронний ресурс] – Режим доступу до ресурсу: <https://www.w3schools.com/cs/index.php>

9. Фреймворк ASP.NET Core [Електронний ресурс] – Режим доступу до ресурсу: <https://dotnet.microsoft.com/en-us/apps/aspnet>

10. Платформа .NET [Електронний ресурс] – Режим доступу до ресурсу: <https://uk.wikipedia.org/wiki/.NET>

11. Масштабування .NET-застосунків [Електронний ресурс] – Режим доступу до ресурсу: <https://webcluesinfo.medium.com/scaling-your-net-applications-strategies-for-load-balancing-and-auto-scaling-e978bec0a67d>

12. Механізми автентифікації та авторизації [Електронний ресурс] – Режим доступу до ресурсу: <https://training.qatestlab.com/blog/technical-articles/authentication-authorization-and-identification/>

13. SSL/TLS [Електронний ресурс] – Режим доступу до ресурсу: <https://www.digicert.com/what-is-ssl-tls-and-https>

14. SQL ін'єкції [Електронний ресурс] – Режим доступу до ресурсу: <https://training.qatestlab.com/blog/technical-articles/security-testing-sql-injection/>

15. Singleton Pattern [Електронний ресурс] – Режим доступу до ресурсу: <https://javarush.com/ua/groups/posts/uk.2365.patterni-proektuvannja-singleton>

16. Життєвий цикл додатку [Електронний ресурс] – Режим доступу до ресурсу: <https://training.qatestlab.com/blog/technical-articles/popular-software-development-life-cycles/>

17. Repository Pattern [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/@pererikbergman/repository-design-pattern-e28c0f3e4a30>

18. Dependency Injection [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/@sardar.khan299/understanding-dependency-injection-a-powerful-design-pattern-for-flexible-and-testable-code-5e1161dd37dd>

19. Інтерфейс прикладного програмування [Електронний ресурс] – Режим доступу до ресурсу: <https://en.wikipedia.org/wiki/API>

20. GraphQL [Електронний ресурс] – Режим доступу до ресурсу: <https://foxminded.ua/graphql-shcho-tse/>

21. REST API [Електронний ресурс] – Режим доступу до ресурсу: <https://foxminded.ua/shcho-take-rest-api/>

22. Використання React [Електронний ресурс] – Режим доступу до ресурсу: <https://dan-it.com.ua/uk/blog/chto-takoe-react-js-i-dlja-chego-on-nuzhen/>

23. Бібліотека Redux [Електронний ресурс] – Режим доступу до ресурсу: <https://foxminded.ua/shcho-take-redux/>

24. SPA [Електронний ресурс] – Режим доступу до ресурсу: <https://foxminded.ua/spa-u-prohramuvanni/>

25. HTML, CSS, JavaScript. Технології що є основою та базою будь-якого веб-сайту [Електронний ресурс] – Режим доступу до ресурсу: <https://pravda.if.ua/html-css-i-javascript-osnovy-veb-rozrobky/>

26. Препроцесор стилів Sass [Електронний ресурс] – Режим доступу до ресурсу: <https://sass-lang.com/>

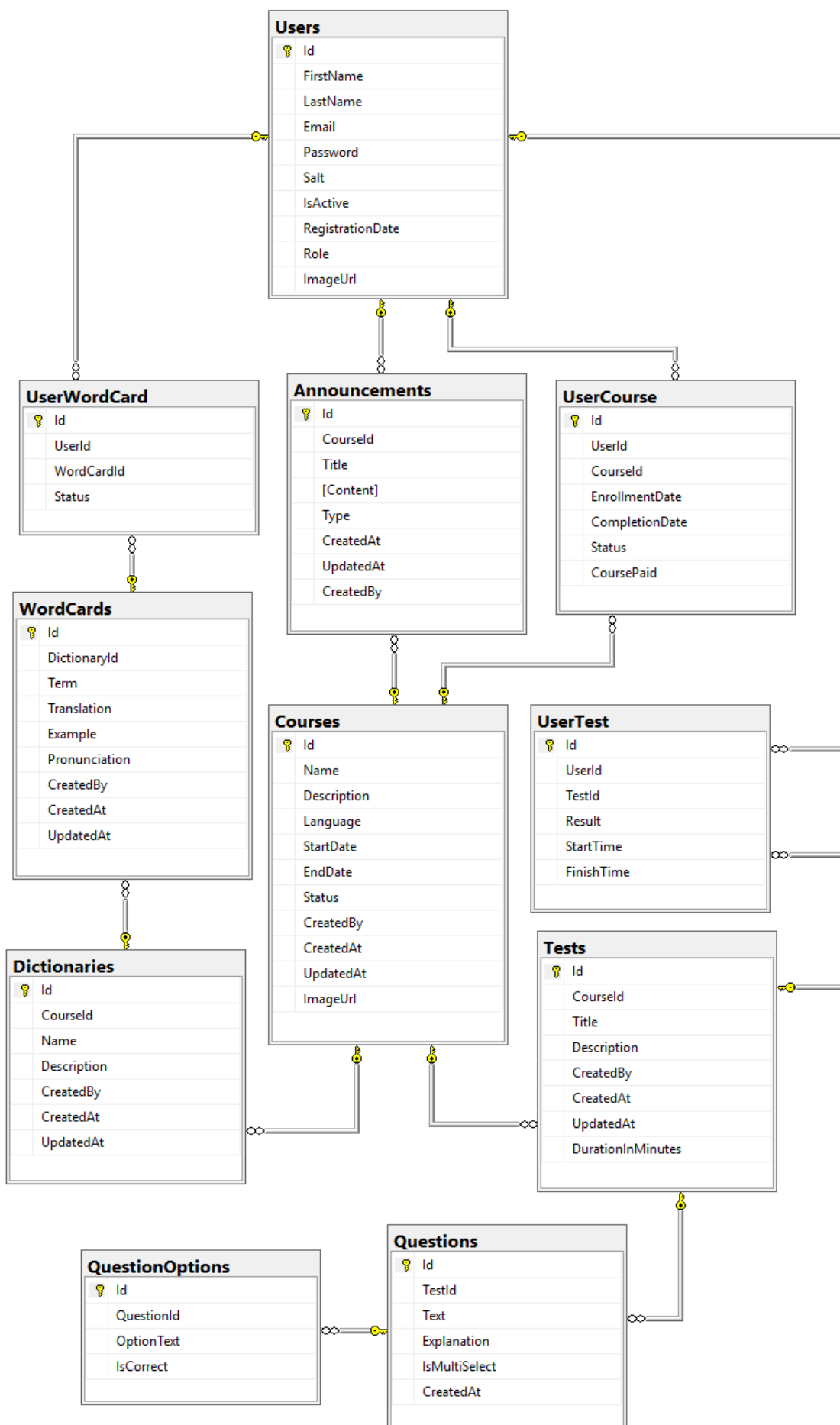
27. База даних Microsoft SQL Server [Електронний ресурс] – Режим доступу до ресурсу: <https://www.microsoft.com/en-us/sql-server>

28. Oracle database [Електронний ресурс] – Режим доступу до ресурсу: <https://www.oracle.com/database/>

29. SQLite [Електронний ресурс] – Режим доступу до ресурсу: <https://sqlite.org/>

30. Функціональне тестування [Електронний ресурс] – Режим доступу до ресурсу: <https://qalight.ua/baza-znaniy/funktsionalne-testuvannya/>

ДОДАТКИ



```

public class AppQuery : ObjectGraphType
{
    public AppQuery()
    {
        Field<UsersQuery>("Users")
            .Description("Operations to query users")
            .Resolve(context => new { });

        Field<ProfileQuery>("Profile")
            .Description("Operations to query profile data")
            .Resolve(context => new { });

        Field<CoursesQuery>("Courses")
            .Description("Operations to query courses")
            .Resolve(context => new { });

        Field<UserCourseQuery>("UserCourseRelation")
            .Description("Operations to query user-course relations")
            .Resolve(context => new { });

        Field<AnnouncementsQuery>("Announcements")
            .Description("Operations to query announcements")
            .Resolve(context => new { });

        Field<DictionariesQuery>("Dictionaries")
            .Description("Operations to query dictionaries")
            .Resolve(context => new { });

        Field<WordCardsQuery>("WordCards")
            .Description("Operations to query word cards")
            .Resolve(context => new { });

        Field<TestsQuery>("Tests")
            .Description("Operations to query tests")
            .Resolve(context => new { });

        Field<QuestionsQuery>("Questions")
            .Description("Operations to query questions")
            .Resolve(context => new { });

        Field<QuestionOptionsQuery>("QuestionOptions")
            .Description("Operations to query question options")
            .Resolve(context => new { });
    }
}

public class AppMutation : ObjectGraphType
{
    public AppMutation()
    {
        Field<UsersMutation>("Users")
            .Description("Operations to mutate users")
            .Resolve(context => new { });

        Field<ProfileMutation>("Profile")
            .Description("Operations to mutate profile data")
            .Resolve(context => new { });

        Field<CoursesMutation>("Courses")
            .Description("Operations to mutate courses")
            .Resolve(context => new { });
    }
}

```

```

        Field<UserCourseMutation>("UserCourseRelation")
            .Description("Operations to mutate user-course relations")
            .Resolve(context => new { });

        Field<AnnouncementsMutation>("Announcements")
            .Description("Operations to mutate announcements")
            .Resolve(context => new { });

        Field<DictionariesMutation>("Dictionaries")
            .Description("Operations to mutate dictionaries")
            .Resolve(context => new { });

        Field<WordCardsMutation>("WordCards")
            .Description("Operations to mutate word cards")
            .Resolve(context => new { });

        Field<TestsMutation>("Tests")
            .Description("Operations to mutate tests")
            .Resolve(context => new { });

        Field<QuestionsMutation>("Questions")
            .Description("Operations to mutate questions")
            .Resolve(context => new { });

        Field<QuestionOptionsMutation>("QuestionOptions")
            .Description("Operations to mutate question options")
            .Resolve(context => new { });
    }
}

public class CoursesQuery : ObjectGraphType
{
    public CoursesQuery(CoursesListResolver coursesList,
        CourseByIdResolver courseById,
        TotalCoursesCountResolver totalCoursesCount)
    {
        Field<ListGraphType<CourseType>>("List")
            .Description("Retrieves a list of courses")
            .Argument<CourseFilterInputType>("filter")
            .Argument<SortingInputType>("sorting")
            .Argument<PagingInputType>("paging")
            .ResolveAsync(async context => await coursesList.ResolveAsync(context));

        Field<CourseType>("Course")
            .Description("Retrieves a specific course by its unique identifier")
            .Argument<NonNullGraphType<StringGraphType>>("id", "Course identifier")
            .ResolveAsync(async context => await courseById.ResolveAsync(context));

        Field<IntGraphType>("TotalCoursesCount")
            .Description("Returns the total number of courses")
            .Argument<CourseFilterInputType>("filter")
            .ResolveAsync(async context => await
totalCoursesCount.ResolveAsync(context));
    }
}

public class CoursesMutation : ObjectGraphType
{
    public CoursesMutation(
        CourseCreationResolver courseCreation,
        CourseUpdateResolver courseUpdate)
    {
        Field<BooleanGraphType>("Creation")

```

```

        .Description("Creates a new course")
        .Argument<NonNullGraphType<CreateCourseInputType>>("input")
        .ResolveAsync(async context => await
courseCreation.ResolveAsync(context));

        Field<CourseType>("Update")
        .Description("Updates an existing course")
        .Argument<NonNullGraphType<UpdateCourseInputType>>("input")
        .ResolveAsync(async context => await
courseUpdate.ResolveAsync(context));
    }
}

public class CoursesListResolver : IGraphQLResolver<IEnumerable<Course>>
{
    private readonly ICourseProvider courseProvider;
    private readonly IUserCourseProvider userCourseProvider;

    public CoursesListResolver(ICourseProvider courseProvider, IUserCourseProvider
userCourseProvider)
    {
        this.courseProvider = courseProvider;
        this.userCourseProvider = userCourseProvider;
    }

    public async Task<IEnumerable<Course>> ResolveAsync(IResolveFieldContext
context)
    {
        var userContext =
context.RequestServices!.GetRequiredService<UserContext>();
        var sorting = context.GetArgument<Sorting?>("sorting");
        var paging = context.GetArgument<Paging?>("paging");
        var filter = context.GetArgument<CourseFilter?>("filter");
        filter.UserId = userContext.User.Role == UserRole.Admin ? null :
userContext.User.Id;

        return await courseProvider.GetAll(filter, sorting, paging);
    }
}

public class CourseUpdateResolver : IGraphQLResolver<Course?>
{
    private readonly ICourseProvider courseProvider;

    public CourseUpdateResolver(ICourseProvider courseProvider)
    {
        this.courseProvider = courseProvider;
    }

    public async Task<Course?> ResolveAsync(IResolveFieldContext context)
    {
        var input = context.GetArgument<UpdateCourseInput>("input");

        var course = new Course()
        {
            Id = Guid.Parse(input.Id),
            Name = input.Name,
            Description = input.Description,
            Language = input.Language,
            StartDate = DateTime.Parse(input.StartDate),
            EndDate = input.EndDate != null ? DateTime.Parse(input.EndDate) : null,
            Status = input.Status,
            UpdatedAt = DateTime.Now,

```

```

        };

        return await courseProvider.Update(course);
    }
}

public class CourseByIdResolver : IGraphQLResolver<Course?>
{
    private readonly ICourseProvider courseProvider;

    public CourseByIdResolver(ICourseProvider courseProvider)
    {
        this.courseProvider = courseProvider;
    }

    public async Task<Course?> ResolveAsync(IResolveFieldContext context)
    {
        var id = context.GetArgument<string>("id");

        if (string.IsNullOrEmpty(id))
            return null;

        return await courseProvider.GetById(id);
    }
}

namespace LanguageCoursesApp.Database.Providers
{
    public abstract class Provider
    {
        private readonly IConfiguration Configuration;

        public Provider(IConfiguration configuration)
            => Configuration = configuration;

        protected IDbConnection CreateConnection()
            => new
            SqlConnection(Configuration.GetConnectionString(Constants.Connection.SQL));

        public async Task<int> ExecuteAsync(string sql, object? parameters = null)
        {
            using var connection = CreateConnection();
            return await connection.ExecuteAsync(sql, parameters);
        }

        public async Task<IEnumerable<T>> QueryAsync<T>(string sql, object?
parameters = null)
        {
            using var connection = CreateConnection();
            return await connection.QueryAsync<T>(sql, parameters);
        }

        public async Task<T?> QuerySingleAsync<T>(string sql, object? parameters =
null)
        {
            using var connection = CreateConnection();
            return await connection.QuerySingleOrDefaultAsync<T>(sql, parameters);
        }
    }
}

```

```

public class CourseProvider : Provider, ICourseProvider
{
    public CourseProvider(IConfiguration configuration) : base(configuration) { }

    public async Task<int> Create(Course course)
    {
        var sql = Queries.Course.Create;
        return await ExecuteAsync(sql, course);
    }

    public async Task<IEnumerable<Course>> GetAll(CourseFilter? filter = null,
Sorting? sorting = null, Paging? paging = null)
    {
        var sql = Queries.Course.GetAll(filter, sorting, paging);
        return await QueryAsync<Course>(sql, filter);
    }

    public async Task<Course?> GetById(string id)
    {
        var sql = Queries.Course.GetById;
        return await QuerySingleAsync<Course>(sql, new { Id = id });
    }

    public async Task<Course?> Update(Course course)
    {
        var sql = Queries.Course.Update;
        return await QuerySingleAsync<Course>(sql, course);
    }

    public async Task<int> GetTotalCoursesCount(CourseFilter? filter)
    {
        var sql = Queries.Course.GetTotalCoursesCount(filter);
        return await QuerySingleAsync<int>(sql, filter);
    }
}

const CourseDetailsPage = () => {
    const dispatch = useDispatch();
    const { courseId } = useParams();
    const { course, userCourseRelations, isLoading } = useSelector((state:
RootState) => state.course.details);

    useEffect(() => {
        if (courseId) {
            dispatch(requestCourseDetails(courseId));
            dispatch(requestUserCourseRelations({ courseIds: [courseId] }));
        }
        return () => {
            dispatch(resetCourseDetails());
        };
    }, [dispatch, courseId]);

    if (!course || !courseId || isLoading)
        return <SkeletonText noOfLines={3} gap="4" />;

    return (
        <>
            <PageHeading text={course.name} />

```

```

<CourseDetailsBlock course={course} />

<Flex direction="row" gap="10">
  <Flex direction="column" gap="5" width="60%" mt="12">
    <Heading>
      <Translation transKey={translationKeys.announcement.heading} />
    </Heading>
    <PermissionWrapper
requiredPermissions={[PermissionType.ManageAnnouncements]}>
      <AnnouncementCreationForm courseId={course.id} />
    </PermissionWrapper>
    <Separator size="lg" my="5" />
    <Box style={{ maxHeight: "200vh", overflowY: "auto" }}>
      <AnnouncementList courseId={course.id} />
    </Box>
  </Flex>
  <Separator orientation="vertical" size="lg" mt="5" />
  <UsersListBlock courseId={courseId}
userCourseRelations={userCourseRelations} />
</Flex>
</>
);
};

export default memo(CourseDetailsPage);

```