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:  
«Месенджер»

|  |
| --- |
| Виконав студент групи ІПЗ-20-4 ВАСЬКІВСЬКИЙ Віталій Юрійович |
|  |
| Керівник роботи:  ЧИЖМОТРЯ Олена Геннадіївна |
|  |
| Рецензент:  МОРОЗОВ Андрій Васильович |

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»

ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

**«ЗАТВЕРДЖУЮ»**

В.о. зав. кафедри інженерії програмного забезпечення

Андрій МОРОЗОВ

«14» лютого 2023 р.

**ЗАВДАННЯ**

**на кваліфікаційну роботу**

Здобувач вищої освіти: ВАСЬКІВСЬКИЙ Віталій Юрійович

Керівник роботи: ЧИЖМОТРЯ Олена Геннадіївна

Тема роботи: «Месенджер»,

затверджена Наказом закладу вищої освіти від «14» лютого 2023 р., №70с

Вихідні дані для роботи: Об’єктом дослідження є використання інформаційних технологій для обміну повідомленнями у мережі.

Предметом дослідження є використання сучасних та веборієнтованих технологій розробки для створення вебзастосунку, що допомагає обмінюватися повідомленнями у мережі.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів,

що їх стосуються:

|  |  |  |  |
| --- | --- | --- | --- |
| **Розділ** | **Консультант** | Завдання видав | Завдання прийняв |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |

**Календарний план**

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Назва етапів випускної  кваліфікаційної роботи | Термін виконання етапів роботи | Примітка |
| 1 | Постановка задачі. Пошук, огляд та аналіз аналогічних розробок. Формулювання технічного завдання | 27.01.2024-28.01.2024 | Виконано |
| 2 | Проектування структури системи |  | Виконано |
| 3 | Написання програмного коду |  | Виконано |
| 4 | Тестування |  | Виконано |
| 5 | Оформлення пояснювальної записки |  | Виконано |

РЕФЕРАТ

Випускна робота бакалавра складається з програмного продукту месенджер та пояснювальної записки.

Пояснювальна записка до випускної роботи містить 75 сторінок, 42 ілюстрації, 17 таблиць, 4 додатки та 26 літературних джерел.

Метою роботи є розробка месенджеру для обміну повідомленнями у мережі.

В роботі визначено основні завдання на розробку програми, проаналізовано аналоги розробленої додатку. Обгрунтовано використання архітектури програми. Наведено сценарії роботи програмного продукту, загальна структура програмного комплексу, опис алгоритмів взаємодії окремих модулів системи, опис об’єктної структури системи (діаграма класів). Також наводиться результати тестування системи.

КЛЮЧОВІ СЛОВА: C#, REACT, SPA, API, ВЕБ-ДОДАТОК.

ABSTRACT

The bachelor’s thesis consists of the messenger software product and an explanatory note.

The explanatory note to the final thesis contains 75 pages, 42 illustrations, 17 tables, 4 appendices and 26 literary sources.

The purpose of the work is to develop messanger to exchange messages in network.

The paper defines the main tasks for the development of the program, and analyzes the analogues of the developed system. The use of the application architecture is justified. Scenarios of the software complex, the general structure of the software complex, a description of the interaction algorithms of individual system modules, a description of the object structure of the system (class diagram) are given. The results of system testing are also provided.

KEY WORDS: C#, REACT, SPA, API, WEB APPLICATION

ЗМІСТ

[ВСТУП 7](#_Toc137724603)

[РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ОНЛАЙН-СЕРВІСУ ПРОХОДЖЕННЯ КУРСІВ УКРАЇНСЬКОЇ МОВИ 9](#_Toc137724604)

[1.1. Постановка задачі 9](#_Toc137724605)

[1.2. Аналіз аналогів програмного продукту 11](#_Toc137724606)

[1.3. Вибір архітектури онлайн-сервісу 14](#_Toc137724607)

[1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення 18](#_Toc137724608)

[Висновки до першого розділу 20](#_Toc137724609)

[РОЗДІЛ 2. ПРОЕКТУВАННЯ ОНЛАЙН-СЕРВІСУ ПРОХОДЖЕННЯ КУРСІВ УКРАЇНСЬКОЇ МОВИ 22](#_Toc137724610)

[2.1. Визначення варіантів використання та об’єктно-орієнтованої структури системи 22](#_Toc137724611)

[2.2. Розробка бази даних системи 27](#_Toc137724612)

[2.3. Проектування та реалізація алгоритмів роботи системи 33](#_Toc137724613)

[2.4. Реалізація онлайн-сервісу проходження курсів української мови 37](#_Toc137724614)

[Висновки до другого розділу 41](#_Toc137724615)

[РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ОНЛАЙН-СЕРВІСОМ ПРОХОДЖЕННЯ КУРСІВ УКРАЇНСЬКОЇ МОВИ 42](#_Toc137724616)

[3.1. Порядок встановлення та налаштування параметрів системи 42](#_Toc137724617)

[3.2. Структура інтерфейсу та порядок взаємодії з онлайн-сервісом 43](#_Toc137724618)

[3.3. Тестування роботи програмного продукту 52](#_Toc137724619)

[Висновки до третього розділу 55](#_Toc137724620)

[ВИСНОВКИ 56](#_Toc137724621)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 58](#_Toc137724622)

[ДОДАТКИ 61](#_Toc137724623)

ВСТУП

**Актуальність теми.** У сучасному світі роль месенджерів стабільно висока з ростом цифрової комунікації. З огляду на зростаючу потребу у спілкуванні онлайн, месенджери залишаються важливим інструментом для особистої та професійної комунікації. Вони не лише сприяють зручності спілкування, але й впливають на розвиток бізнесу, політичну агітацію, навчання та інші аспекти життя. Також, зростанням мобільних технологій та зміною способу сприйняття інформації, месенджери стають платформою для новацій та розвитку цифрового середовища. Тому, вивчення актуальних аспектів месенджерів залишається важливим для розуміння та вдосконалення способів комунікації в онлайн-середовищі.

**Метою випускної роботи** є розробка месенджеру.

Дана мета передбачає виконання наступних завдань:

* проведення аналізу вимог до програми;
* аналіз аналогів програмного продукту;
* вибір архітектури системи;
* обґрунтування та вибір засобів реалізації;
* визначення варіантів використання та проектування об’єктноорієнтованої структури системи;
* проектування алгоритмів системи;
* безпосередня реалізація застосунку;
* опис структури інтерфейсу та порядку взаємодії зі системою;
* тестування розробленого додатку.

**Об’єктом дослідження** є використання інформаційних технологій для обміну повідомленнями у мережі.

**Предметом дослідження** є використання сучасних веборієнтованих технологій розробки для створення вебзастосунку, що допомагає обмінюватися повідомленнями у мережі.

**Методи дослідження.** У роботі було використано методи, об’єктно-орієнтованого, аспектно-орієнтованого та функціонального програмувань.

**Практичне значення одержаних результатів**. Розроблений вебзастосунок може використовуватися для особистої та професійної комунікації.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ МЕСЕНДЖЕРУ

1.1. Постановка задачі

Розробка месенджеру передбачає створення бекенд частини за допомогою мови C#[1] та фреймворку ASP.NET. Фронтенд частина має бути реалізовани за допомогою мови JavaScript та бібліотеки React [1].

Отриманий застосунок має бути веб-додатком, що надає можливість користувачам обмінюватися повідомленнями у мережі.

Програмний продукт повинен мати наступні функціональні характеристики:

* авторизація та реєстрація з можливістю зміни та скидання пароля;
* авторизація та реєстрація за допомогою Google-акаунту;
* створювати персональні чати та групи;
* пошук по чатам;
* надсилати текстові, медіа та файлові повідомлення;
* надсилати, редагувати, видаляти, пересилати, відповідати на повідомлення;
* менеджмент активними логін сесіями;

Програмний продукт повинен бути реалізованим на основі клієнт-серверної архітектури. Серверна частина додатку має базуватися на веб-сервері IIS та системі керування базами даних SQL Server. При проектуванні програмного комплексу необхідно застосувати архітектуру SPA (Single-Page Application) [5] для оновлення контенту без перезавантаження веб-сторінки.

Клієнтська частина повинна бути побудована за сучасними стандартами створення односторінкового застосунку, тобто весь необхідний код завантажується разом зі сторінкою, або динамічно довантажується за потребою, зазвичай у відповідь на дії користувача. Сторінка не оновлюється і не перенаправляє користувача до іншої сторінки у процесі роботи з нею.

Розробка системи передбачає наступні основні етапи:

1. Створення бекенд частини додатку написанням GraphQL API [4], використовуючи фреймворк ASP.NET та бібліотеку GraphQL.NET, що включає реалізацію наступних сутностей:

* «Чати»;
* «Повідомлення»;
* «Прочитані повідомлення»;
* «Сесії»;
* «Користувачі у чаті»;
* «Користувачі»;

1. Створення дизайну інтерфейсу, з яким будуть взаємодіяти користувачі в системі:

* обрати бібліотеку для реалізації користувацького інтерфейсу;
* розробити відповідні сторінки з використаннямобраної бібліотеки;
* забезпечити сповіщення користувача про завантаження контенту за допомогою відповідного елементу.

1. Налаштувати отримання даних зі створеного API на клієнті та розробити маршрутизацію додатку за допомогою бібліотеки React:

* налаштувати виклик API через Apollo Client;
* організувати маршрутизацію додатку з використанням React router dom;

В результаті виконаних етапів буде розроблена система, побудована за архітектурою SPA (Single-Page Application), яка забезпечить користувачеві зручний додаток для спілкування.

1.2. Аналіз аналогів програмного продукту

У сучасному світі складно вигадати щось абсолютно нове, що ще не існувало раніше. Тому під час розробки програм необхідно звертати увагу на аналогічні продукти, щоб отримати повну картину ринку і вимог користувачів до цих додатків. Для проведення аналізу було вибрано наступні програми:

1. Telegram.
2. WhatsApp.
3. Viber.

Telegram - це безпечний месенджер з широким функціоналом для обміну повідомленнями та спілкування.

На рис. 1.1 зображено додаток Telegram.

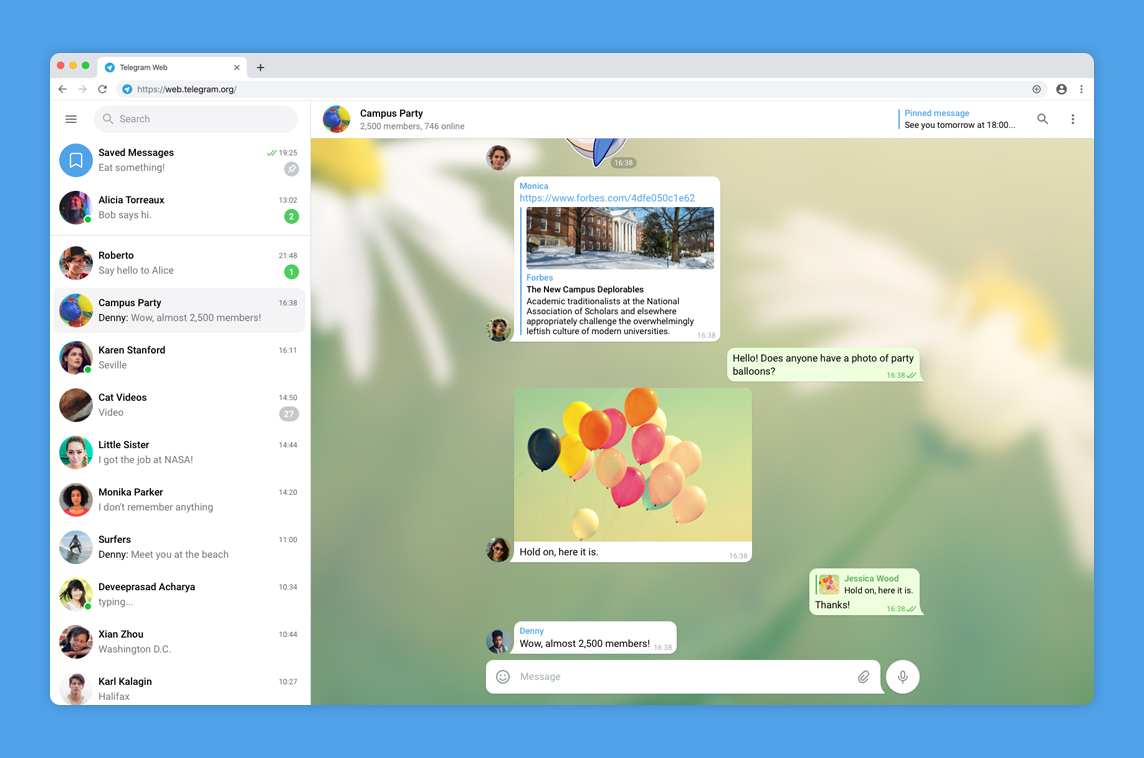


Рисунок 1.1 - Telegram

Можна підкреслити наступні переваги даного додатку:

* гарний та інтуїтивний дизайн;
* **висока безпека**;
* **широкий функціонал**;
* **швидкість і ефективність**;
* **самознищення повідомлень;**

До недоліків можна віднести наступне:

* **неуніверсальність шифрування**;
* **несприйняття в деяких країнах**.

**WhatsApp** - це популярний месенджер для обміну повідомленнями, фото, відео та здійснення голосових та відеодзвінків у мережі.

На рис. 1.2 зображено додаток **WhatsApp**.

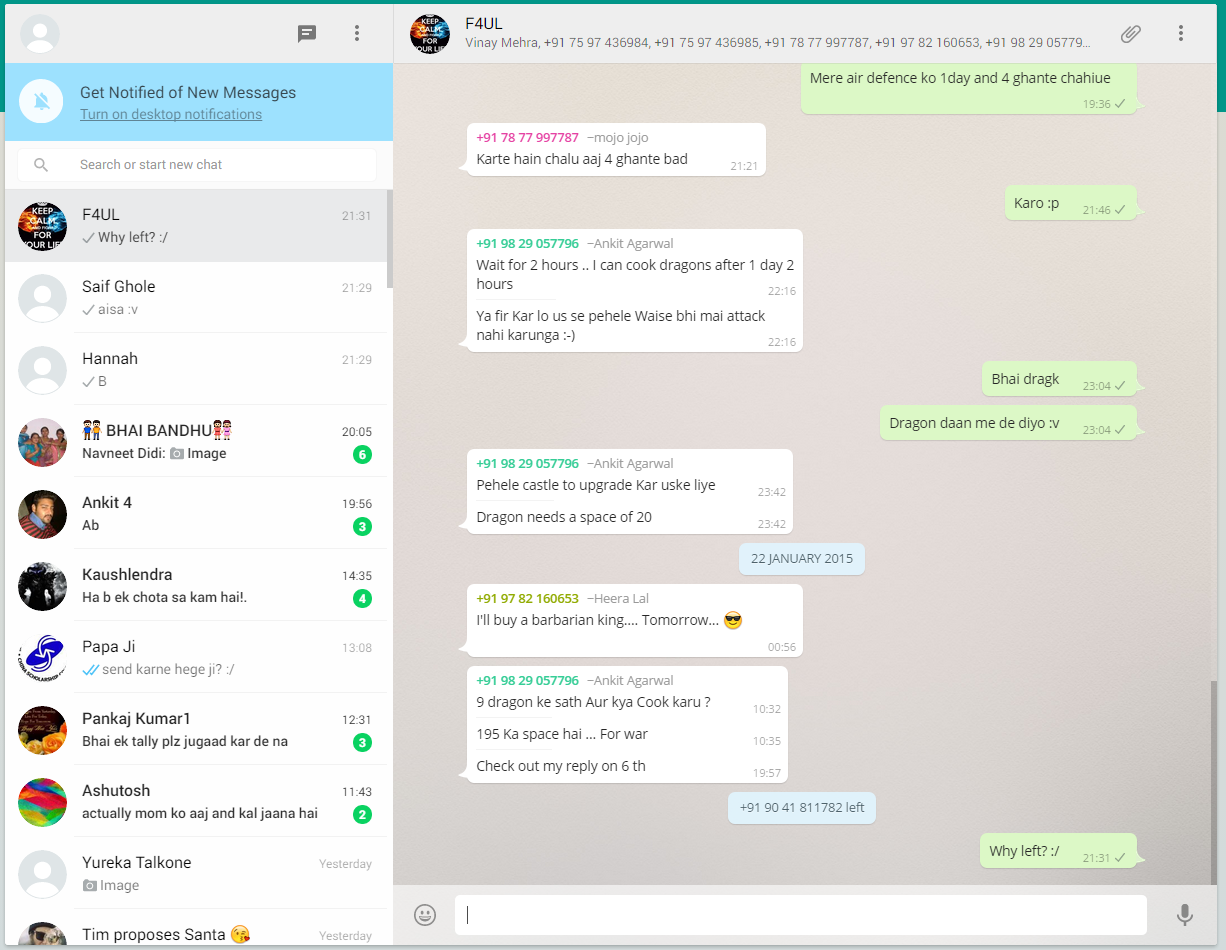


Рисунок 1.2 - **WhatsApp**

До переваг можна віднести:

* **широкий функціонал**;
* **шифрування end-to-end**;
* **популярність**.

Недоліками є наступне:

* **обмеженість функцій веб-версії**;
* **обмеження на використання одного облікового запису на одному пристрої**;
* **недостатній функціонал для бізнесу**.

Viber - це месенджер для обміну повідомленнями, викликів і відеодзвінків у мережі.

На рис. 1.3 зображено додаток Viber.

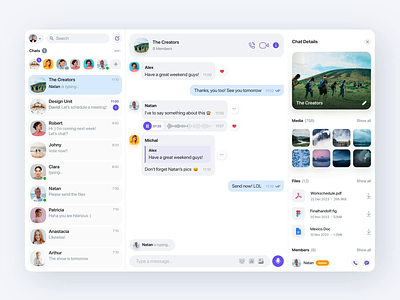


Рисунок 1.3 - Viber

Перевагами є наступне:

* **шифрування end-to-end**;
* **велика кількість користувачів**.

До недоліків можна віднести наступне:

* **високе споживання ресурсів**;
* **можливість отримання небажаних повідомлень**.

Структуруємо отримані дані за допомогою таблиці.

Таблиця 1.1

|  |  |  |  |
| --- | --- | --- | --- |
| Характеристика | Telegram | Whatsup | Viber |
| Інтуїтивний та зрозумілий дизайн | + | + | + |
| Висока безпека | + | + | + |
| Широка інтеграція | + | - | - |
| Висока стабільність та швидкість | + | - | - |
| Мале споживання ресурсів | + | + | - |

Ґрунтуючись на отриманих результатах, отримуємо наступні критерії для додатку, що розроблюється:

* інтуїтивний та зрозумілий дизайн;
* висока безпека;
* широка інтеграція;
* висока стабільність та швидкість;
* мале споживання ресурсів;

1.3. Вибір архітектури веб-додатку

В сучасному світі існує незліченна кількість архітектур для побудови вебдодатку. Розглянемо декілька з них, такі як:

* схема MVC (Model-View-Controller);
* схема SPA (Single-Page Application).

Тепер докладніше розберемо кожну схему.

MVC – це архітектурний шаблон, який дозволяє розділити компоненти додатку на три основні частини: модель (дані), вид (інтерфейс користувача) і контролер (бізнес-логіка). Дозволяє забезпечити розділення відповідальностей та полегшує розробку, тестування та підтримку програмного забезпечення

На рис. 1.4 зображено схему MVC.

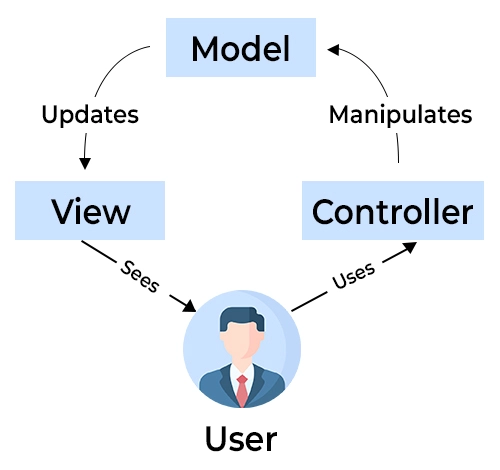


Рисунок. 1.4 - Схема MVC

До його переваг належать:

* **розділення відповідальностей**: MVC дозволяє розділити програму на три основні компоненти, що полегшує управління кодом та забезпечує чітку структуру;
* **покращена повторне використання коду**: кожен компонент (Модель, Вид, Контролер) може бути використаний повторно в інших частинах програми або в інших проектах;

**легша тестовість**: кожен компонент може бути тестований окремо, що дозволяє проводити модульне тестування та забезпечує високу якість програмного забезпечення.

До недоліків можна віднести наступне:

* **можливість перенаповнення контролерів**: контролери можуть стати перенаповненими, якщо в них занадто багато логіки, що може знизити читабельність і підтримку коду;
* **переплутаність взаємодії між компонентами**: іноді взаємодія між моделлю, видом і контролером може бути неявною або неочевидною, що може призвести до проблем у розумінні і обслуговуванні коду.

SPA **–** це тип веб-додатка або сайту, який працює на одній HTML-сторінці. У SPA весь необхідний HTML, CSS та JavaScript завантажується один раз при початковому завантаженні сторінки, після чого весь інший контент та взаємодія з користувачем відбувається без повторної загрузки сторінки з сервера.

На рис. 1.6 зображено схему SPA.

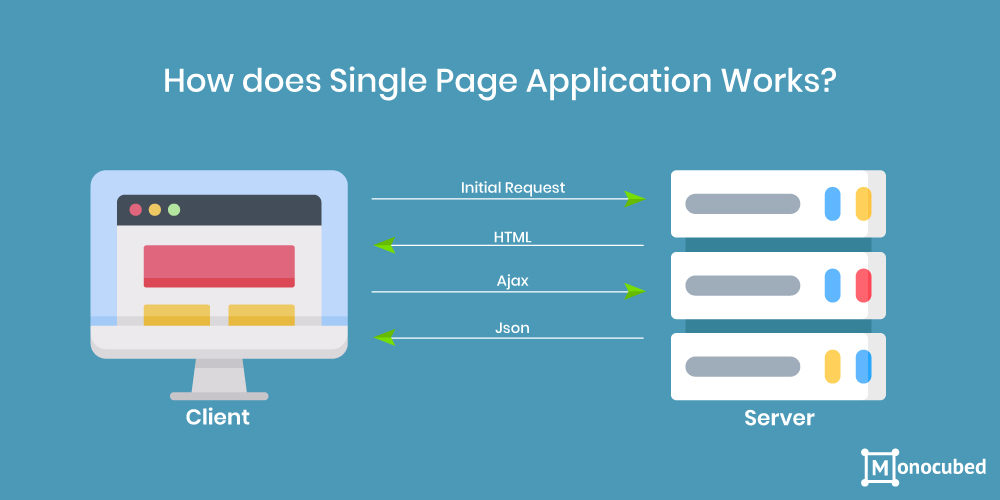


Рисунок 1.6 - Схема SPA

До його переваг належать:

* **швидкість**: SPA завантажує всі необхідні ресурси при запуску та працює в браузері, що забезпечує швидкість реакції на дії користувача;
* **зменшення навантаження на сервер**: через те, що не потрібно завантажувати повну сторінку з сервера при кожному запиті, навантаження на сервер зменшується;
* **кешування**: SPA можуть легко використовувати кешування, що дозволяє зберігати ресурси на боці клієнта і зменшувати час завантаження;
* **можливість використання фреймворків**: багато фреймворків та бібліотек спеціалізуються на розробці SPA, що дозволяє розробникам використовувати готові рішення для розробки складних клієнтських додатків.

До недоліків належать:

* **збільшене використання пам'яті**: завантаження всіх ресурсів одразу може призвести до збільшеного використання пам'яті в браузері, що може вплинути на продуктивність, особливо на старших або менш потужних пристроях;
* **проблеми з SEO**: для пошукових систем складно індексувати та інтерпретувати вміст, що генерується динамічно через JavaScript, що може призвести до проблем з оптимізацією для пошукових систем (SEO).

Проаналізувавши вищеперелічені архітектури, було обрано використовувати підхід SPA для забезпечення кращого користувацького досвіду.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Для створення веб-додатку месенджер обрано наступні технології:

* мова розмітки гіпертекстових документів: HTML5;
* мова препроцесора для стилів: Sass;
* клієнтська частина: мова програмування JavaScript та бібліотека React[2];
* серверна частина: мова програмування C# та фреймворк ASP.NET;
* база даних: Microsoft SQL Server.

Ці технології дозволять створити веб-додаток за допомогою сучасних інструментів. HTML5 забезпечить семантичну структуру веб-сторінок, Sass дозволить ефективно керувати стилями з використанням препроцесінгу, а React надасть потужні можливості для реалізації інтерактивної взаємодії з користувачем.

Серверна частина буде побудована на базі мови програмування C# та фреймворка ASP.NE, що дозволить створити потужний та масштабований серверний додаток. База даних Microsoft SQL Server забезпечить надійне зберігання та управління даними.

Застосування цих технологій дозволить створити сучасний та функціональний веб-додаток месенджер, забезпечивши ефективне використання ресурсів та гнучкість у розробці.

**HTML5** - використовується для створення структури та семантики веб-сторінок.

**Sass** **–** це препроцесор CSS, що дозволяє писати стилі з використанням змінних, міксинів та інших функцій, що полегшують роботу зі стилями.

**React**[2] **–** це бібліотека JavaScript для розробки інтерфейсів користувача, яка дозволяє створювати ефективні та динамічні веб-додатки.

**C#** **–** використовується для розробки серверної частини. Вона є потужною та об'єктно-орієнтованою мовою з широким набором функціональних можливостей.

**ASP.NET** - це технологічний стек для розробки веб-додатків. Він надає потужні інструменти для обробки запитів, маршрутизації та роботи з базами даних.

**Microsoft SQL Server –** це система управління базами даних. Вона надає можливості для зберігання, керування та обробки даних в реляційних базах даних.

Ці технології допоможуть створити функціональний, зручний у використанні та масштабований веб-додаток.

Таблиця 1.4

Максимально можливий обсяг збережених даних для кожної СКБД

|  |  |  |  |
| --- | --- | --- | --- |
|  | Розмір БД | Розмір таблиці | Розмір рядка |
| MySQL | ∞ | 256 ТБ | 64 KB |
| MS SQL | 524258 ТБ | 524258 ТБ | ∞ |
| PostgreSQL | ∞ | 32 ТБ | 1.6 ТБ |

За критерієм тригери й збережені процедури всі альтернативи ідентичні. Усі підтримують тригери, процедури й функції.

Таблиця 1.5

Аналіз підтримуваних альтернативами операційних систем

|  |  |  |  |
| --- | --- | --- | --- |
|  | Windows | Linux | Android |
| MySQL | + | + | + |
| MS SQL | + | - | - |
| PostgreSQL | + | + | - |

Порівняємо різні методи захисту даних, які використовуються для уникнення несанкціонованого доступу до конфіденційної інформації в інформаційній системі організації. Далі будемо аналізувати таблицю, в якій вказані системи забезпечення безпеки даних та їх опис.

Таблиця 1.6

Аналіз систем забезпечення безпеки даних в альтернативах

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Ідентифікація | Захист від brute-force | Шифрування | Сертифікація безпеки |
| MySQL | + | - | + | - |
| MS SQL | + | - | + | + |
| PostgreSQL | + | + | + | + |

Після аналізу систем управління базами даних (СУБД), таких як MySQL, Microsoft SQL Server (MS SQL) та PostgreSQL, встановлено, що PostgreSQL є найбільш підходящим для виконання поставлених завдань. Актуальна версія PostgreSQL має розширений функціонал та можливості, такі як підтримка географічних даних, розширення для складних операцій та механізми захисту даних.

Таблиця 1.7

Аналіз загальних показників СКБД

|  |  |  |  |
| --- | --- | --- | --- |
|  | MySQL | MS SQL | PostgreSQL |
| *1.* | *2.* | *3.* | *4.* |
| Логічна модель даних | Реляційна | Реляційна | Постреляційна (об’єктно-реляційна) |
| Фізична модель даних | Сторінкова | Сторінкова | Сторінкова |
| Типи даних | Всі основні | Всі основні, розширені | Всі основні, довільні, множинні значення |

Продовження табл. 1.7

|  |  |  |  |
| --- | --- | --- | --- |
| *1.* | *2.* | *3.* | *4.* |
| Індекси | Всі основні, повнотекстовий | Всі основні, повнотекстовий | Звичайні, повнотекстовий |
| Мови маніпулювання | SQL | SQL, QBE, XQuery, багатомірні вирази (для OLAP) | Розширений SQL |
| Вбудовані мови програмування | ANSI C, ANSI C++ | MS Visual Basic, C# | C, C++, Java, Perl, PHP, .Net та ін |
| Генератор форм, звітів | Ні | Засоби побудови звітів | Ні |
| Транзакції | Так (в тому числі розподілені) | Так | Так |
| Тригери, процедури, що зберігаються | Так | Так | Так (підтримка декількох мов програмування) |
| Платформи | MS Windows, Unix, Linux, MacOS X, Novell NetWare та ін. | Тільки MS Windows | MS Windows, Linux, FreeBSD, Solaris, MacOS |
| Область застосування | Інформаційні системи масштабу підприємства | Інформаційні системи масштабу підприємства | Інформаційні системи масштабу підприємства |
| Особливості | Можливість логічного об’єднання БД. Широкий вибір платформ. Реплікація | Масштабування. Кластери. Реплікації. Розширена підтримка XML | Відкриті вихідні тексти. Механізм наслідування. Масштабування. Реплікація. |

Для створення програмного комплексу були обрані наступні інструментальні засоби та технології:

* база даних: Misrosoft SQL Server;
* мова програмування на серверній частині: C#;
* фреймворк на серверній частині: ASP.NET;
* мова програмування на клієнтській частині: JavaScript;
* фреймворк на клієнтській частині: React;
* мова процесора для стилів: Sass.

Для розробки були використані наступні програмні продукти та інтегровані середовища розробки: середовище розробки Visual Studio 2022 та Visual Studio Code.

Висновки до першого розділу

У цьому розділі було сформульовано завдання для кваліфікаційної роботи, визначено функціональні вимоги та основні етапи роботи. Також було проведено аналіз аналогів продукту, що розробляється, що дозволило встановити наступні критерії для розробки: інтуїтивний та зрозумілий дизайн, висока безпека, широка інтеграція, висока стабільність та швидкість та мале використання ресурсів.

Деякі існуючі архітектурні рішення було проаналізовано, і вибір було зроблено на користь використання схеми SPA для забезпечення кращого користувацького досвіду. Також було обгрунтовано вибір технологічного стеку для реалізації додатку: ASP.NET для побудови бекенд частини, React - для фронтенд частини; Miscrosoft SQL Server- як систему керування базами даних.

Буде використано такі програмні продукти та інтегровані середовища розробки: середовище розробки Visual Studio 2022 та Visual Studio Code.

РОЗДІЛ 2. ПРОЕКТУВАННЯ МЕСЕНДЖЕРУ

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

Для зображення відношення між акторами та прецедентами в системі, було побудовано діаграму варіантів використання [17].

На рис. 2.1 зображено діаграму варіантів використання.

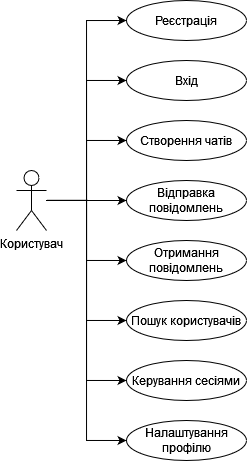


Рисунок 2.1 - Діаграма варіантів використання

Відповідно до побудованої діаграми можна виокремити наступні вимоги системи.

Вимоги користувачів:

1. Можливість реєстрація та входу у власний аккаунт.
2. Можливість керувати чатами.
3. Відправляти та отримувати повідомлення
4. Можливість пошуку користувачів.
5. Можливість керувати власними логін сесіями
6. Можливість налаштовувати власний профіль.

Функціональні вимоги:

1. Використання протоколу Web Socker для обміну даними.
2. Створення функціоналу для обміну повідомленнями.
3. Підключення бібліотек для швидшої розробки.

Нефункціональні вимоги:

1. Сприйняття:

* інтерфейс повинен бути інтуїтивно зручним для користувача.

1. Продуктивність

* система повинна підтримувати мінімум декілька тисяч користувачів, що одночасно обмінюються повідомленнями.

Зважаючи на те, що серверна частина є об’єктно-орієнтованою, побудуємо діаграму класів [18] провайдерів веб-додатку.

На рис. 2.2 зображено діаграму класів провайдерів додатку.

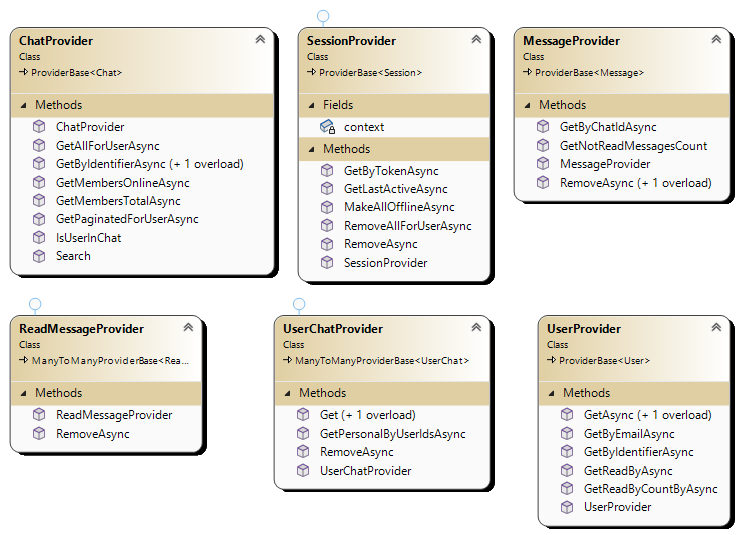


Рисунок 2.2 - Діаграма класів провайдерів додатку

Сучасна IT-сфера розвивається надзвичайно швидко, і багато часто виникаючих проблем вже вирішені завдяки "патернам проектування". Важко уявити сучасні додатки без їх використання, тому патерни проектування враховуються вже на етапі моделювання системи.

Таким чином, було вирішено застосувати патерн "Репозиторій" [19], який є, мабуть, найпопулярнішим рішенням для застосунків.

“Репозиторії” – це класи або компоненти, які інкапсулюють логіку доступу до джерел даних. Вони централізують загальну функціональність доступу до даних, забезпечуючи кращу зручність обслуговування та відокремлюючи інфраструктуру або технологію доступу до баз даних від рівня моделі предметної області. Цей патерн часто використовується разом із патерном “Unit Of Work”.На рис. 2.3 зображено реалізацію патерну “Репозиторій” для блоку авторизації.

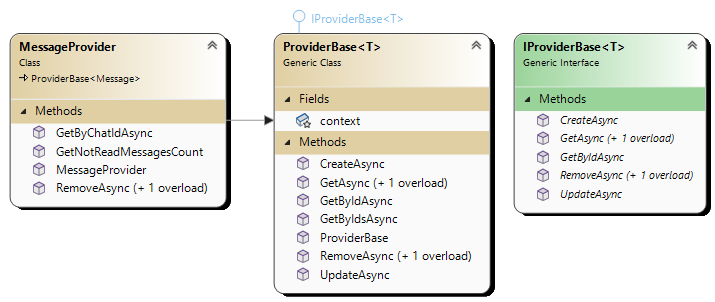


Рисунок 2.3 - Реалізація патерну “Репозиторій” для блоку авторизації

На рис. 2.4 зображено схему патерну “Репозиторій ”.

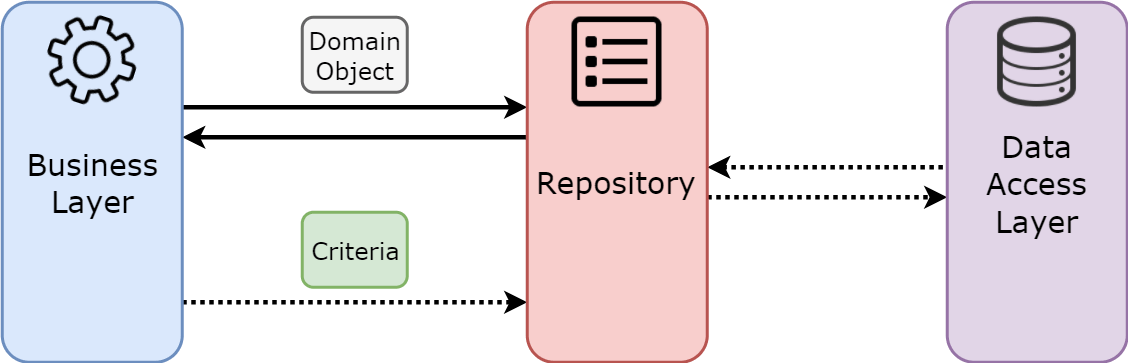


Рисунок 2.4 - Загальний вигляд патерну “Репозиторій”

Крім того, для побудови системи буде використовуватися патерн “Unit Of Work”. Однак, немає потреби реалізовувати його самостійно, оскільки більшість фреймворків надають цей патерн з коробки.

Це патерн об’єктно-реляційної поведінки, призначений для відстеження змін об’єктів під час транзакції. Під час роботи з базою даних важливо відстежувати зміни в об’єктах, інакше дані не будуть оновлені. Це також стосується операцій додавання та видалення.

Можна змінювати дані в сховищі при кожній взаємодії з об’єктом, але це

спричинить численні виклики до бази даних. Очевидно, що це вимагає підтримки транзакції у відкритому стані, що негативно впливає на продуктивність. Патерн “Unit Of Work” пропонує відстежувати всі зміни об’єктів і застосовувати їх у вигляді єдиної транзакції.

На рис. 2.5 зображено загальний вигляд патерну “Unit Of Work”.

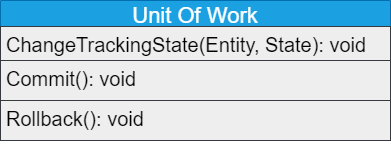


Рисунок 2.5 - Загальний вигляд патерну “Unit Of Work”

У клієнтській частині додатку не використовуються класи, оскільки нова версія бібліотеки React перейшла на принципи функціонального програмування.

2.2. Розробка бази даних системи

Для збереження даних була обрана СУБД Microsoft SQL Server, оскільки вона є однією з найпопулярніших і найкраще підтримуваних систем керування базами даних на сьогоднішній день. Microsoft SQL Server забезпечує широкий функціонал, надійність і високу продуктивність, що робить її чудовим вибором для зберігання даних.

На рис. 2.6 продемонстровано ER-діаграму спроєктованої бази даних.

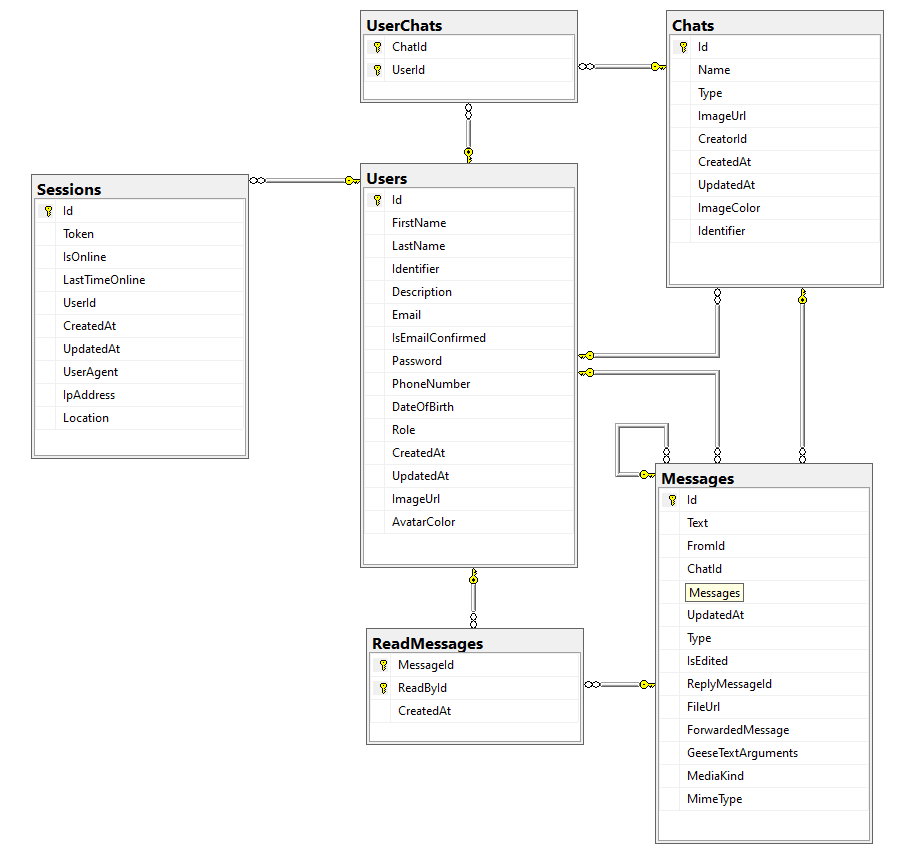


Рисунок 2.6 - ER-діаграма спроєктованої бази даних

Опис таблиць, полів та їх призначення наведені в таблицях 2.1-2.6.

Таблиця 2.1

Структура таблиці “Users”

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Ключ | Тип | Призначення |
| *1.* | *2.* | *3.* | *4.* |
| Id | PK | uniqueidentifier | Ідентифікатор користувача |
| FirstName | - | varchar(450) | Ім’я користувача |
| LastName | - | varchar(450) | Прізвище користувача |
| Identifier | - | varchar(450) | Унікальне імя користувача |
| Description | - | varchar(max) | Опис користувача |

Продовження табл. 2.1

|  |  |  |  |
| --- | --- | --- | --- |
| *1.* | *2.* | *3.* | *4.* |
| Email | - | varchar(450) | Електронна пошта користувача |
| IsEmailConfirmed | - | bit | Чи підтвержєена пошта користувача |
| Password | - | varchar(450) | Пароль користувача |
| PhoneNumber | - | varchar(30) | Номер телефону користувача |
| DateOfBirth | - | datetime2(7) | Дата народження користувача |
| CreatedAt | - | datetime2(7) | Дата створення аккаунту |
| UpdatedAt | - | datetime2(7) | Дата редагування аккаунту |
| ImageUrl |  | varchar(450) | Посилання на картинку користувача |
| AvatarColor |  | varchar(7) | Колір користувача |

Таблиця 2.2

Структура таблиці “Sessions”

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Ключ | Тип | Призначення |
| *1.* | *2.* | *3.* | *4.* |
| Id | PK | uniqueidentifier | Ідентифікатор сесії |
| IsOnline | - | bit | Чи користувач онлайн |
| UserId | FK | uniqueidentifier | Ідентифікатор користувача для сесії (зовішній ключ на таблицю “Users”) |
| CreatedAt | - | datetime2(7) | Дата створення сесії |
| UpdatedAt | - | datetime2(7) | Дата редагування сесії |
| UserAgent | - | varchar(450) | User agent сесії |
| IpAddress | - | varchar(450) | IP-адреса сесії |
| Location | - | varchar(450) | Місце знаходження сесії |

Таблиця 2.3

Структура таблиці “Chats”

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Ключ | Тип | Призначення |
| *1.* | *2.* | *3.* | *4.* |
| Id | PK | uniqueidentifier | Ідентифікатор чату |
| Name | - | varchar(450) | Ім’я чату |
| Type | - | int | Тип чату |

Продовження табл. 2.3

|  |  |  |  |
| --- | --- | --- | --- |
| *1.* | *2.* | *3.* | *4.* |
| ImageUrl | - | varchar(450) | Посилання на картинку чату |
| CreatorId | FK | uniqueidentifier | Ідентифікатор користувача, який створив чат (зовішній ключ на таблицю “Users”) |
| CreatedAt | - | datetime2(7) | Дата створення чату |
| UpdatedAt | - | datetime2(7) | Дата редагування чату |
| ImageColor | - | varchar(7) | Колір чату |
| Identifier | - | varchar(450) | Унікальне імя чату |

Таблиця 2.4

Структура таблиці “UserChats”

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Ключ | Тип | Призначення |
| *1.* | *2.* | *3.* | *4.* |
| ChatId | PK | uniqueidentifier | Ідентифікатор чату (зовішній ключ на таблицю “Chats”) |
| UserId | PK | uniqueidentifier | Ідентифікатор користувача (зовішній ключ на таблицю “Users”) |

Таблиця 2.5

Структура таблиці “Messages”

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Ключ | Тип | Призначення |
| *1.* | *2.* | *3.* | *4.* |
| Id | PK | uniqueidentifier | Ідентифікатор повідомлення |
| Text | - | varchar(max) | Текст повідомлення |
| FromId | FK | uniqueidentifier | Ідентифікатор користувача, який створив повідомлення (зовішній ключ на таблицю “Users”) |
| ChatId | FK | uniqueidentifier | Ідентифікатор чату, до якого належить повідомлення (зовішній ключ на таблицю “Chats”) |
| CreatedAt | - | datetime2(7) | Дата створення чату |
| UpdatedAt | - | datetime2(7) | Дата редагування чату |

Продовження табл. 2.5

|  |  |  |  |
| --- | --- | --- | --- |
| *1.* | *2.* | *3.* | *4.* |
| Type | - | int | Тип повідомлення |
| IsEdited | - | bit | Чи повідомлення редагувалося |
| ReplyMessageId | FK | uniqueidentifier | Ідентифікатор повідомлення, на яке зробили відповідь (зовішній ключ на таблицю “Messages”) |
| FileUrl | - | varchar(450) | Посилання на файл повідомлення |
| ForwardedMessage | - | varchar(max) | Переслане повідомлення |
| GeeseTextArguments | - | varchar(450) | Аргументи тексту повідомлення |
| MediaKind | - | int | Медіа тип повідомлення |
| MimeType | - | varchar(50) | Mime-type повідомлення |

Таблиця 2.6

Структура таблиці “ReadMessages”

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Ключ | Тип | Призначення |
| *1.* | *2.* | *3.* | *4.* |
| MessageId | PK | uniqueidentifier | Ідентифікатор повідомлення (зовішній ключ на таблицю “Messages”) |
| ReadById | PK | uniqueidentifier | Ідентифікатор користувача (зовішній ключ на таблицю “Users”) |
| CreatedAt | - | datetime2(7) | Дата прочитання повідомлення |

2.3. Проектування та реалізація алгоритмів роботи системи

Так як обраної архітектурою програмного додатку є SPA, то вебзастосунок має явне розподілення на серверну та клієнтську частину і передбачає використання підходу RESTful API [20].

RESTful API (Representational State Transfer API) є набором правил та протоколів, які дозволяють різним програмним додаткам взаємодіяти між собою через мережу, зазвичай через Інтернет.

Можна виділити наступні основні характеристики та принципи RESTful API:

* ресурси: побудований навколо ресурсів, які зазвичай представляються у вигляді URL-адрес. Кожен ресурс має унікальний URL, а клієнти взаємодіють з цими ресурсами, виконуючи HTTP-запити до відповідних URL-адрес;
* методи HTTP: RESTful API використовує стандартні методи HTTP для виконання різних дій з ресурсами. Методи, що використовується під час розробки додатку:

1. GET: Отримує представлення ресурсу.
2. POST: Створює новий ресурс.
3. PUT: Оновлює існуючий ресурс.
4. PATCH: Оновлює частину існуючого ресурсу.
5. DELETE: Видаляє ресурс.

У випадку з додатком, що створюється відбувається наступне. З клієнтської частину відбуватиметься звернення до приватного API для отримання необхідних даних. Для цього за допомогою фреймворку Laravel повертається JSON-об’єкт при зверненні до відповідного ендпоїнту з клієнтської частини за допомогою бібліотеки Axios. На стороні серверу відбувається отримання даних з бази даних і здійснення маніпуляцій на ними. Після чого ці дані використовуються для генерації користувацького інтерфейсу. Якщо ж користувач змінює запит сторінки, то повторно відбувається запит до серверу вебзастосунку.

На рис. 2.7 зображено загальний алгоритм роботи додатку.

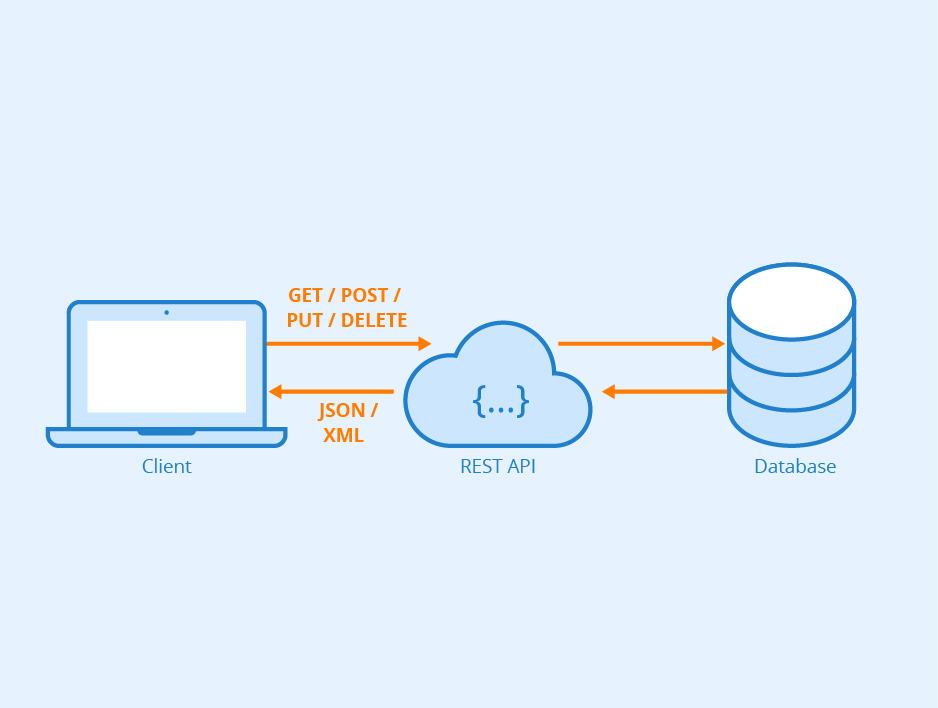


Рисунок 2.7 - Загальний алгоритм роботи додатку

Також, для зображення алгоритму роботи додатку побудуємо діаграму активностей [21].

На рис. 2.8 зображено діаграму активностей.

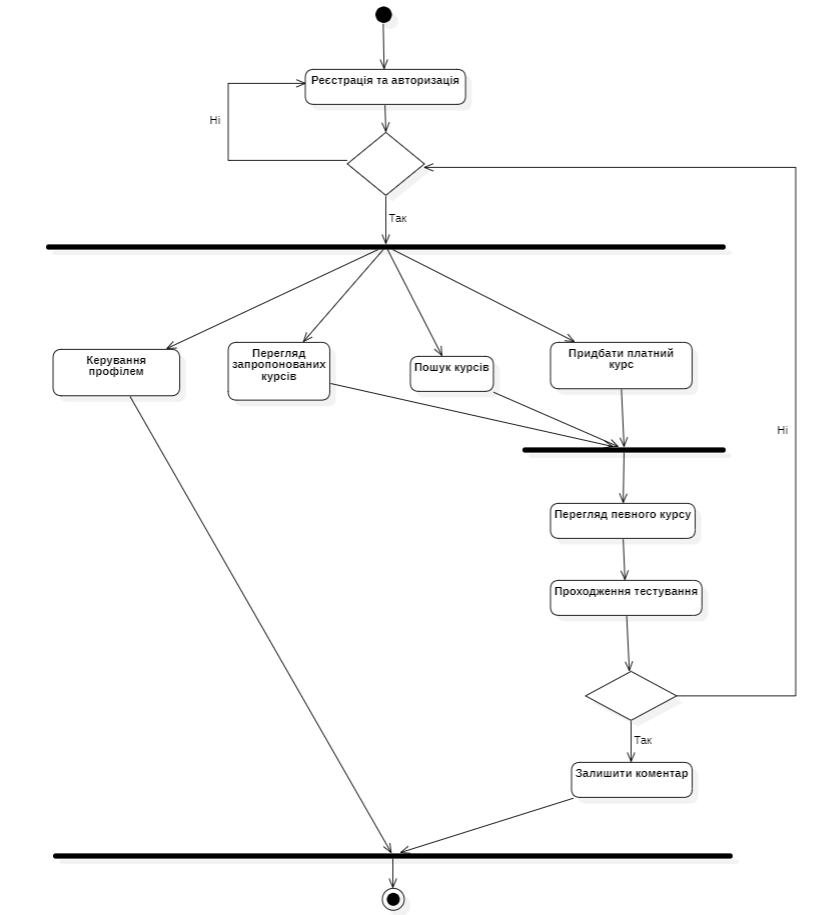


Рисунок 2.8 – Діаграма активностей системи

Опис отриманої діаграми:

1. Авторизація (так – перехід до використання системи, ні – перехід на початок з повідомленням про необхідність наявності акаунту).
2. Керування профілем.
3. Перегляд запропонованих курсів, пошук курсів, придбати платний курс:

* перегляд певного курсу;
* проходження тестування (так – перехід до створення коментаря,

ні – перехід до використання системи або кінець роботи додатку);

* залишити коментар.

1. Кінець роботи веб-застосунку.

Безпосередньо застосунок можна умовно поділити на певні модулі. Для презентативності побудуємо діаграму компонентів додатку.

На рис. 2.9 зображено діаграму компонентів [22] додатку.

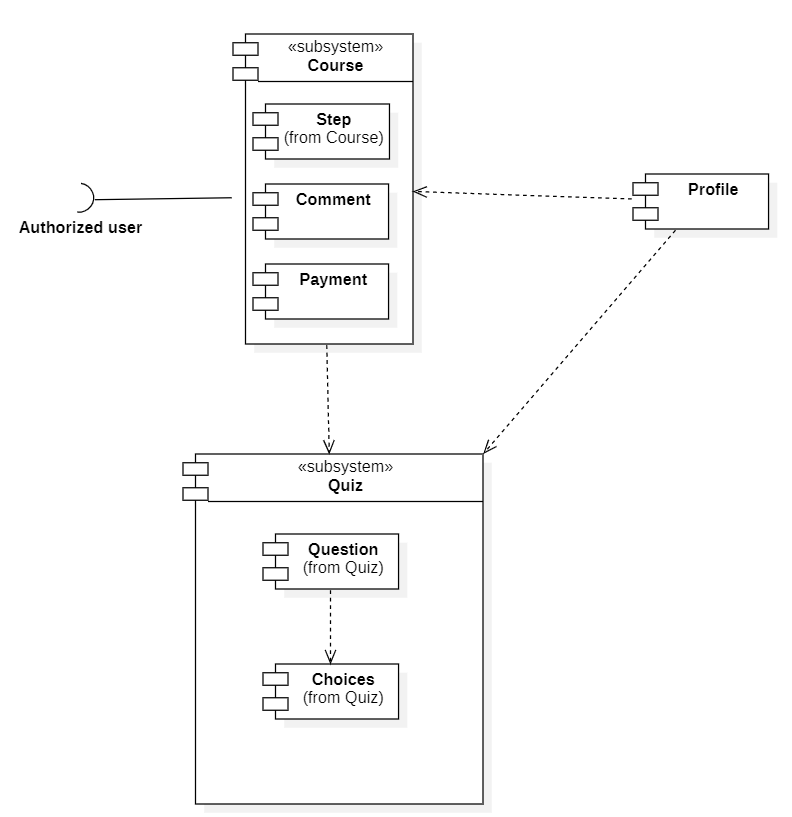


Рисунок 2.9 - Діаграма компонентів додатку

Проаналізувавши отриману діаграму, отримуємо наступне:

1. Для роботи з основними компонентами користувач повинен бути авторизованим у системі.
2. Систему було розділено на два підкомпоненти і один окремий компонент.
3. Підсистема Course представляє собою проходження певного курсу, що поділений на кроки, його оплату та залишення коментаря до нього.
4. Підкомпонент Quiz відповідає за тестування, що є обов’язковим критерієм закінчення відповідного курсу.
5. Компонент Profile відповідає за керування профілем та відстежування прогресу користувачем.

Детальніше розглянемо алгоритм оплати. Для цього було побудовано діаграму послідовності.

На рис. 2.10 зображено діаграму послідовності [23].

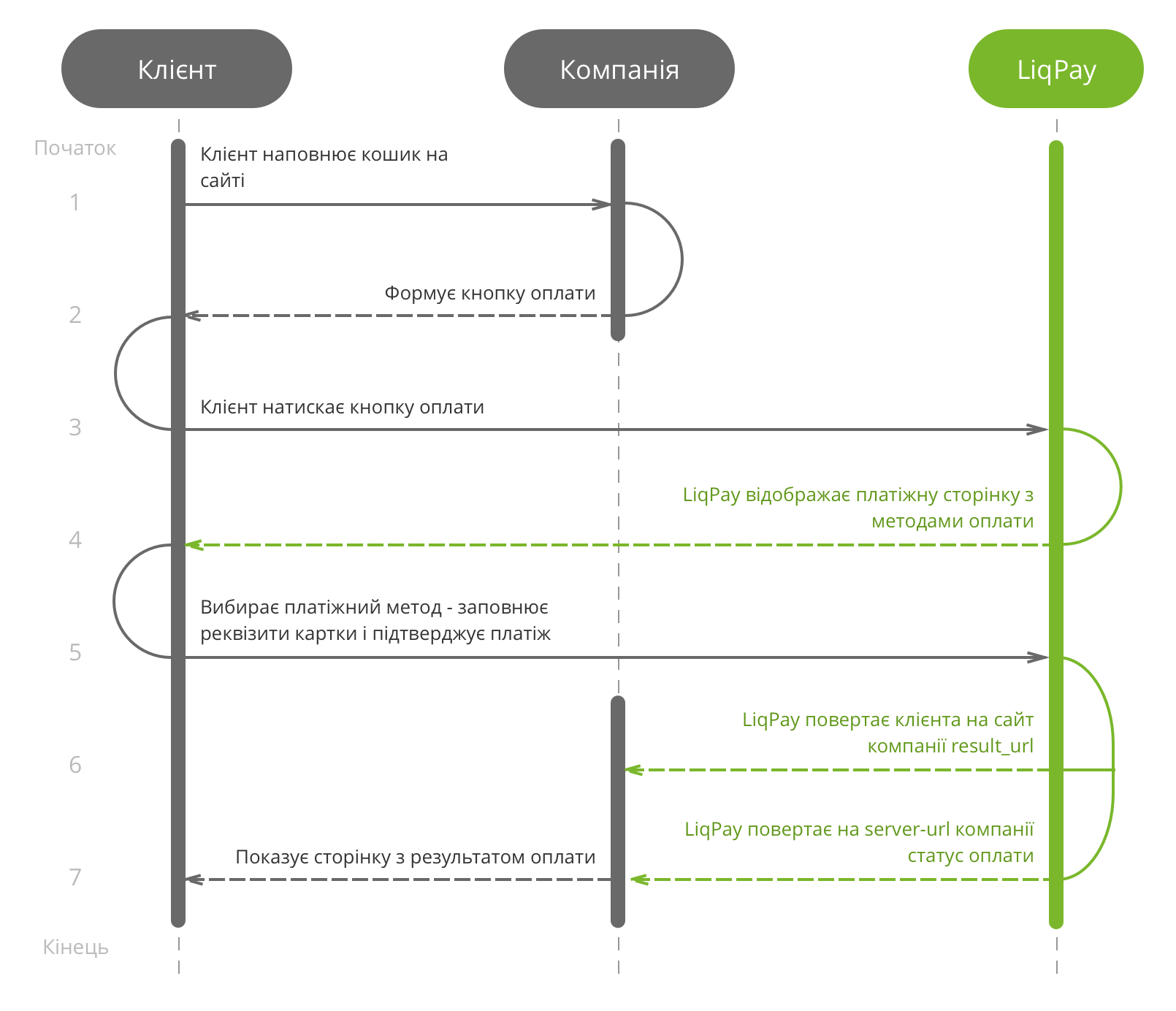


Рисунок 2.10 - Діаграма послідовності для оплати

У разі якщо обраний курс є платним, то користувач повинен спочатку його купити. На сторінці курсу користувач натискає відповідну кнопку, при цьому створюється нове замовлення у системі. При підтвердженні оформлення замовлення користувача перенаправляє на платіжну сторінку, інакше замовлення видаляється з системи. Після успішної оплати оновлюється статус замовлення і у користувача з’являється доступ до курсу.

2.4. Реалізація онлайн-сервісу проходження курсів української мови

Будь-який початок роботи з додатком передбачає авторизацію користувача. Тому розглянемо реалізацію даного функціоналу.

На рис. 2.11 зображено метод login класу AuthRepository.

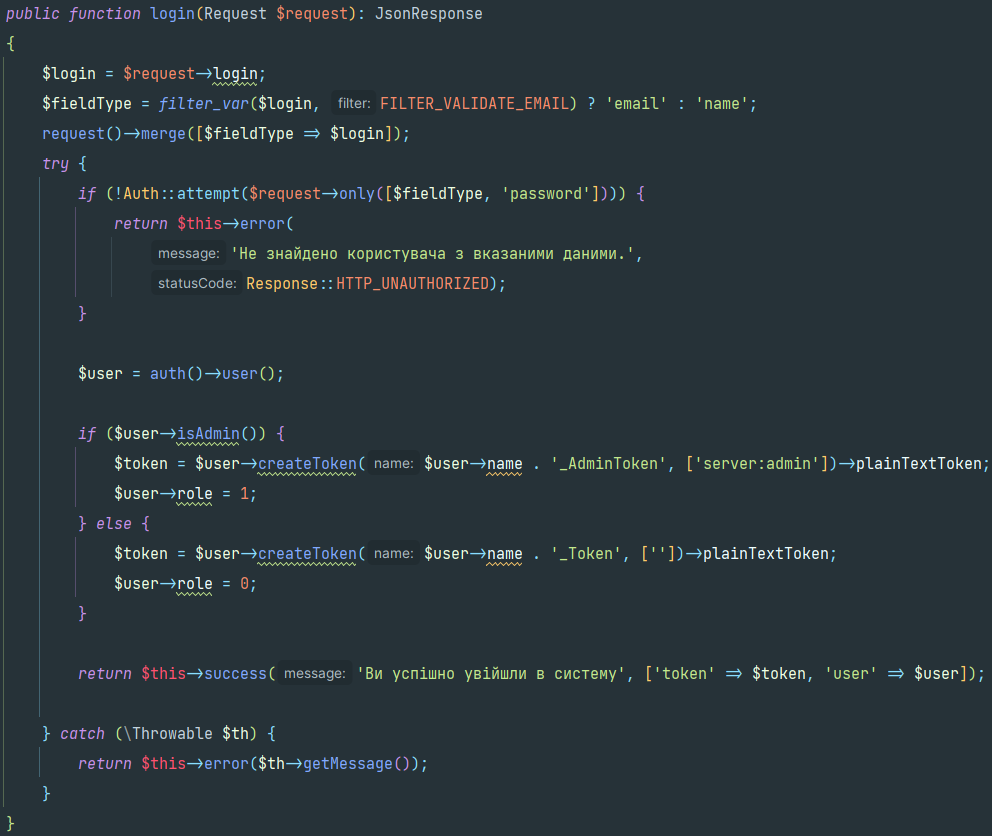


Рисунок 2.11 - Метод login класу AuthRepository

Даний метод передбачає визначення даних за якими користувач хоче увійти. Потім відбувається спроба згідно цих даних увійти в систему, створюючи новий токен з відповідними правами, який в подальшому використовуватиметься на клієнтській частині для формування авторизаційного заголовку. У разі введення неправильних даних користувачі повертається повідомлення про помилку.

На рис. 2.12 зображено метод resetPassword класу AuthRepository.

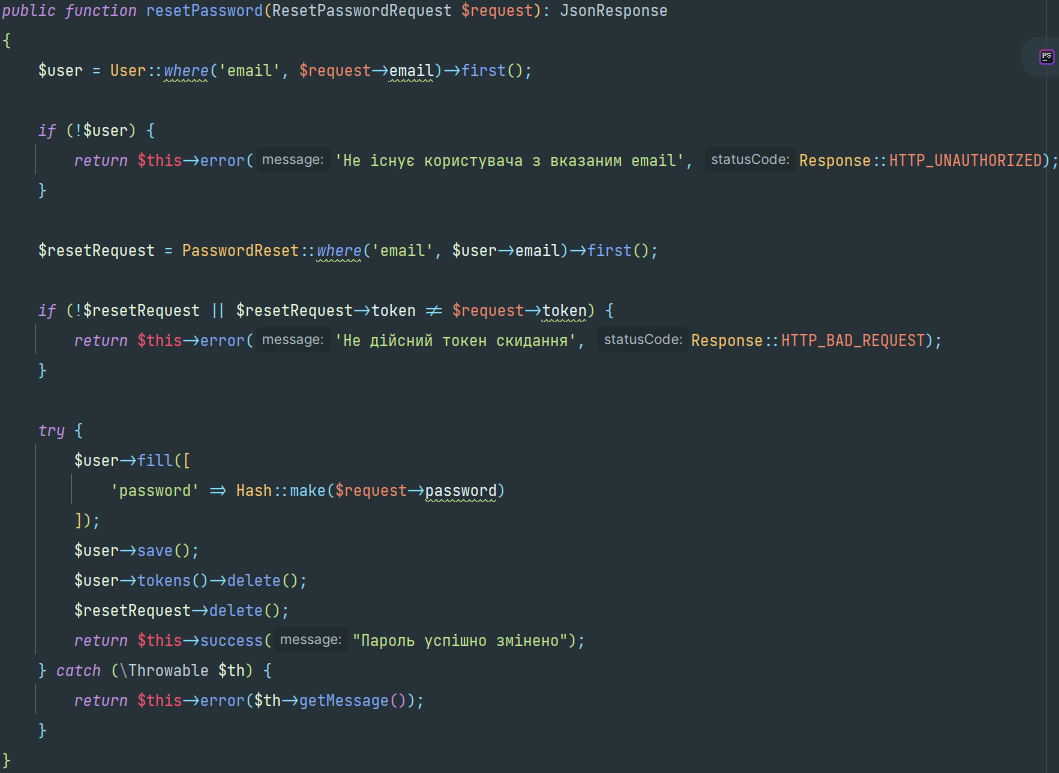


Рисунок 2.12 - Метод resetPassword класу AuthRepository

Даний метод відповідає за зміну пароля, коли користувач забув пароль і вже перейшов за посиланням для відновлення з пошти. Він перевіряє валідність токена та користувача після чого зберігає новий пароль у системі. Якщо сталася якась помилка під час виконання попередніх дій, користувач отримує про неї повідомлення.

Також, необхідно виокремити реалізацію завантаження відео розділу на сервер. Для цього було використано підхід розділення файлу на шматки (chunk) і надсилання їх на сервер. Після завантаження всіх шматків вони назад перетворюються в один файл. Реалізацію даного процесу зображено на рис. 2.13, а саме представлено метод uploadFile класу BigFileRepository.

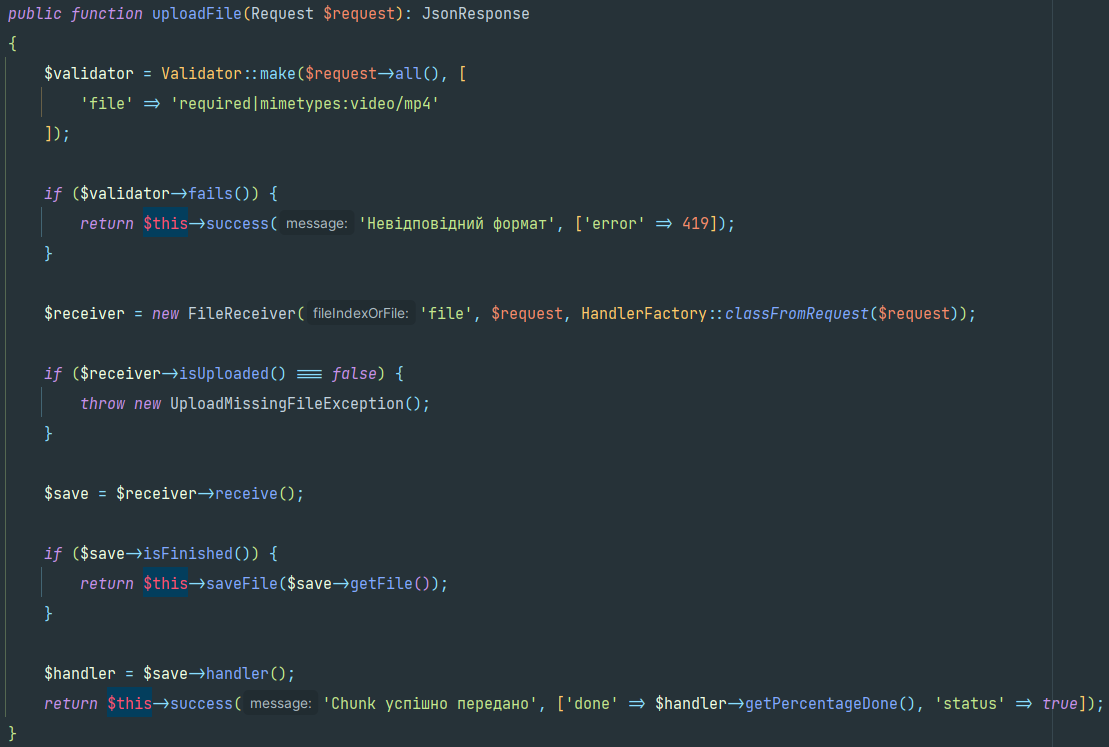


Рисунок 2.13 - Метод uploadFile класу BigFileRepository

Цей метод використовує допоміжний метод saveFile, реалізацію якого зображено на рис. 2.14.

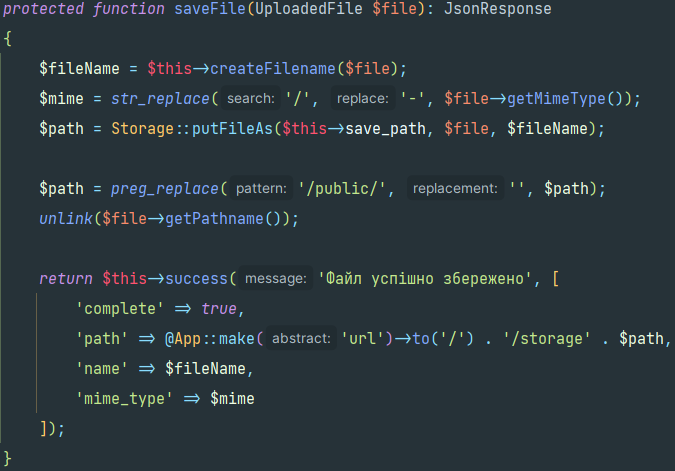


Рисунок 2.14 - Метод saveFile класу BigFileRepository

Також, важливої частиною застосунку є розділи курсів та їх послідовність. Тому реалізація зміни послідовності розділів для адміністратора є обов’язковою складовою розробки додатку. На рис. 2.15 продемонстровано метод changeRank класу CourseStepRepository, що за це відповідає.



Рисунок 2.15 - Метод saveFile класу BigFileRepository

Спочатку ми отримуємо початкове положення курсів і записуємо їх до масиву. Після чого циклічно змінюємо на нові значення.

Невід’ємною частиною додатку є тестування і перевірка правильності вибору користувача. Тому на рис. 2.16 продемонстровано метод checkAnswers методу QuestionRepository, що відповідає за перевірку відповідей.

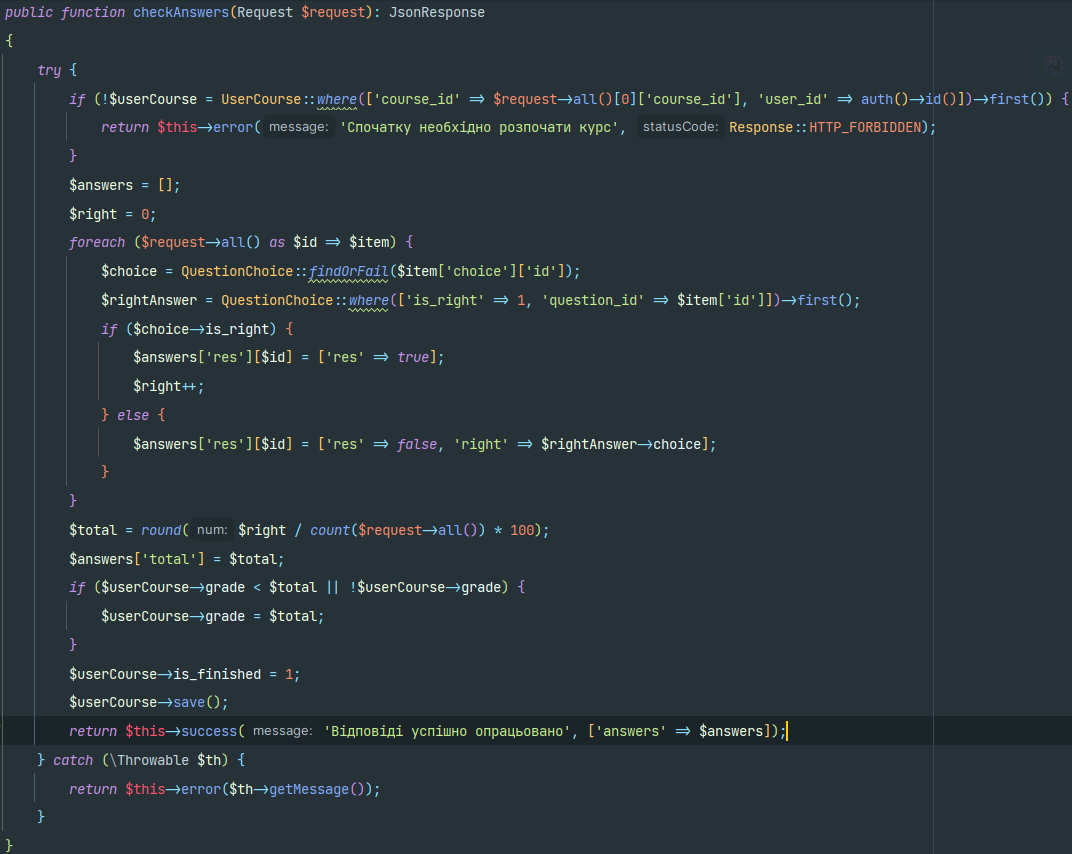


Рисунок 2.16 - Метод checkAnswers методу QuestionRepository

Спочатку отримуємо в масив усі відповіді користувача. Потім циклічно знаходимо варіант відповіді та перевіряємо, чи воно правильно. Якщо так, то збільшуємо кількість правильних відповідей, інакше вказуємо правильну відповідь. Після підрахунку оцінки зберігаємо її, якщо вона краща за попередні спроби або це перша спроба. Після цього курс в системі позначається як пройдений для даного користувача.

Висновки до другого розділу

В даному розділі для наочного зображення відношення між акторами та прецедентами в системі, було побудовано діаграму варіантів використання. Було сформовано вимоги до системи. Було обґрунтовано та описано використання патернів при розробці застосунку. Було спроектовано базу даних та описано поля та їх призначення.

Описано, як було використано архітектуру Single-Page Application та обґрунтовано використання підходу RESTful API. Було продемонстровано загальний алгоритм роботи та побудовані наступні діаграми для презентативності: діаграму активностей системи, діаграму послідовності для оплати та діаграма компонентів системи.

Було продемонстровано реалізацію програмного комплексу та наведено фрагменти коду, що відповідають за авторизацію, оплату та завантаження відео на сервер з поясненням принципу їх роботи.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ОНЛАЙН-СЕРВІСОМ ПРОХОДЖЕННЯ КУРСІВ УКРАЇНСЬКОЇ МОВИ

3.1. Порядок встановлення та налаштування параметрів системи

Для зображення розгортання системи існує однойменна діаграма – діаграма розгортання [24].

Діаграма розгортання – це діаграма, яка показує конфігурацію вузлів обробки часу виконання та компонентів, які на них живуть. Діаграми розгортання – це різновид структурної діаграми, яка використовується для моделювання фізичних аспектів об’єктно-орієнтованої системи. Вони часто використовуються для моделювання статичного вигляду розгортання системи (топології апаратного забезпечення).

На рис. 3.1 зображено діаграму розгортання.

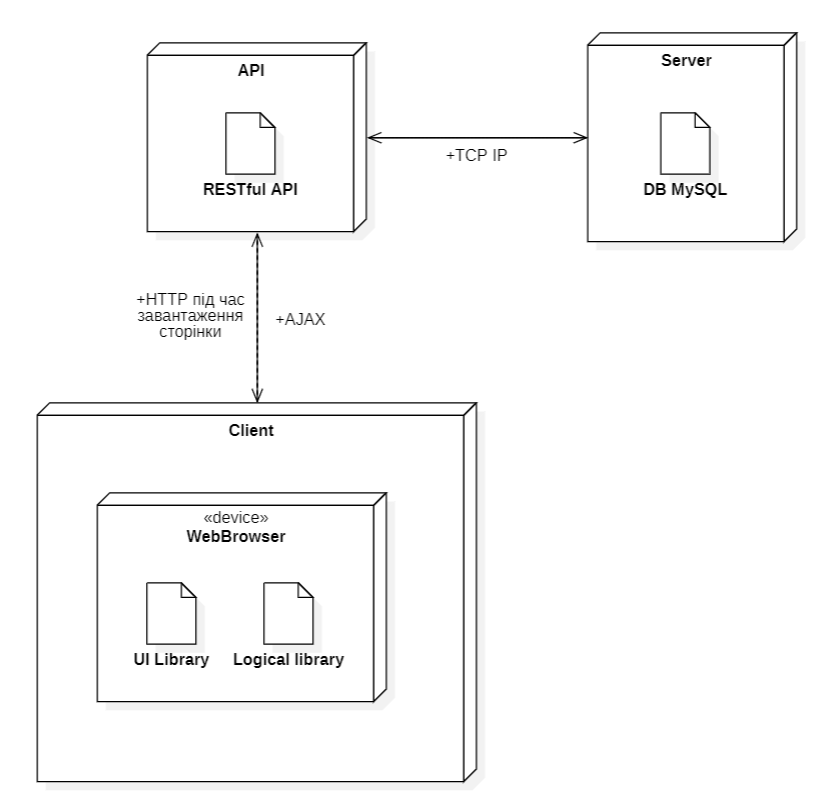


Рисунок 3.1 - Діаграма розгортання додатку

Після побудови даної діаграми можна зробити наступні висновки щодо порядку встановлення та налаштування параметрів системи:

* API вебзастосунку, тобто його бекенд частина, розгортається на порті 8000, в той час як клієнт – на порті 3000;
* бекенд частина звертається до сервера для отримання необхідних ресурсів;
* клієнт представляє собою додаток, що використовує бібліотеки для побудови користувацького інтерфейсу та налаштування логіки оновлення даних і безпосередньо сторінок застосунку;
* клієнт отримає початкову HTML-розмітку під час запуску додатку і в подальшому оновлює її за допомогою асинхронних звернень до API.

3.2. Структура інтерфейсу та порядок взаємодії з онлайн-сервісом

Після відкриття вебзастосунку користувача зустрічає головна сторінка.

На рис. 3.2 зображено головну сторінку вебзастосунку.



Рисунок 3.2 – Головна сторінка вебзастосунку

Тут він повинен перейти до сторінки входу, аби авторизуватися у системі. У разі якщо користувач ще не має акаунту він може його створити.

На рис. 3.3 зображено сторінку авторизації.

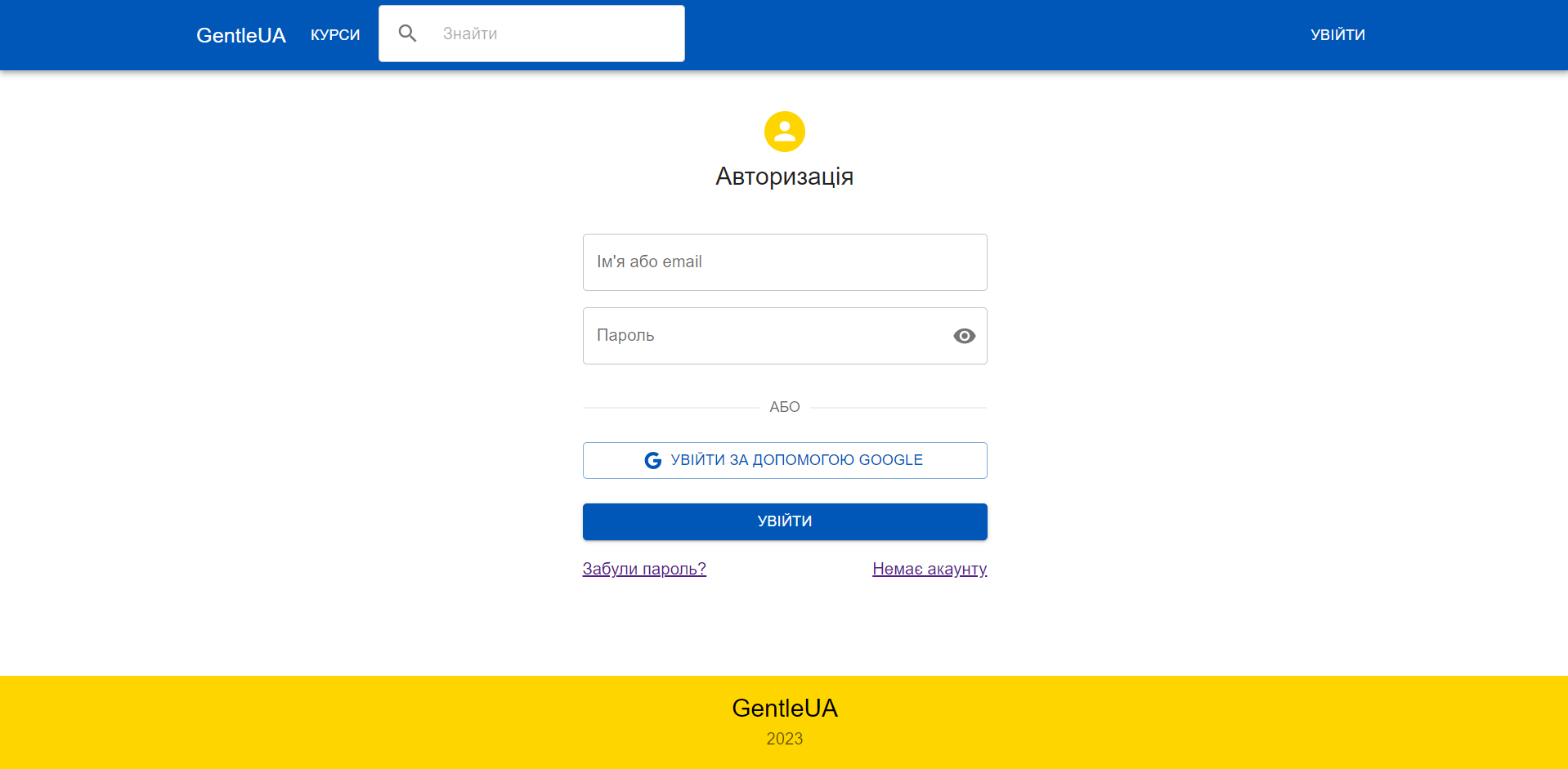


Рисунок 3.3 - Сторінка авторизації

Після успішного входу в систему користувач отримує доступ до переліку всіх курсів.

На рис. 3.4 зображено сторінку курсів.

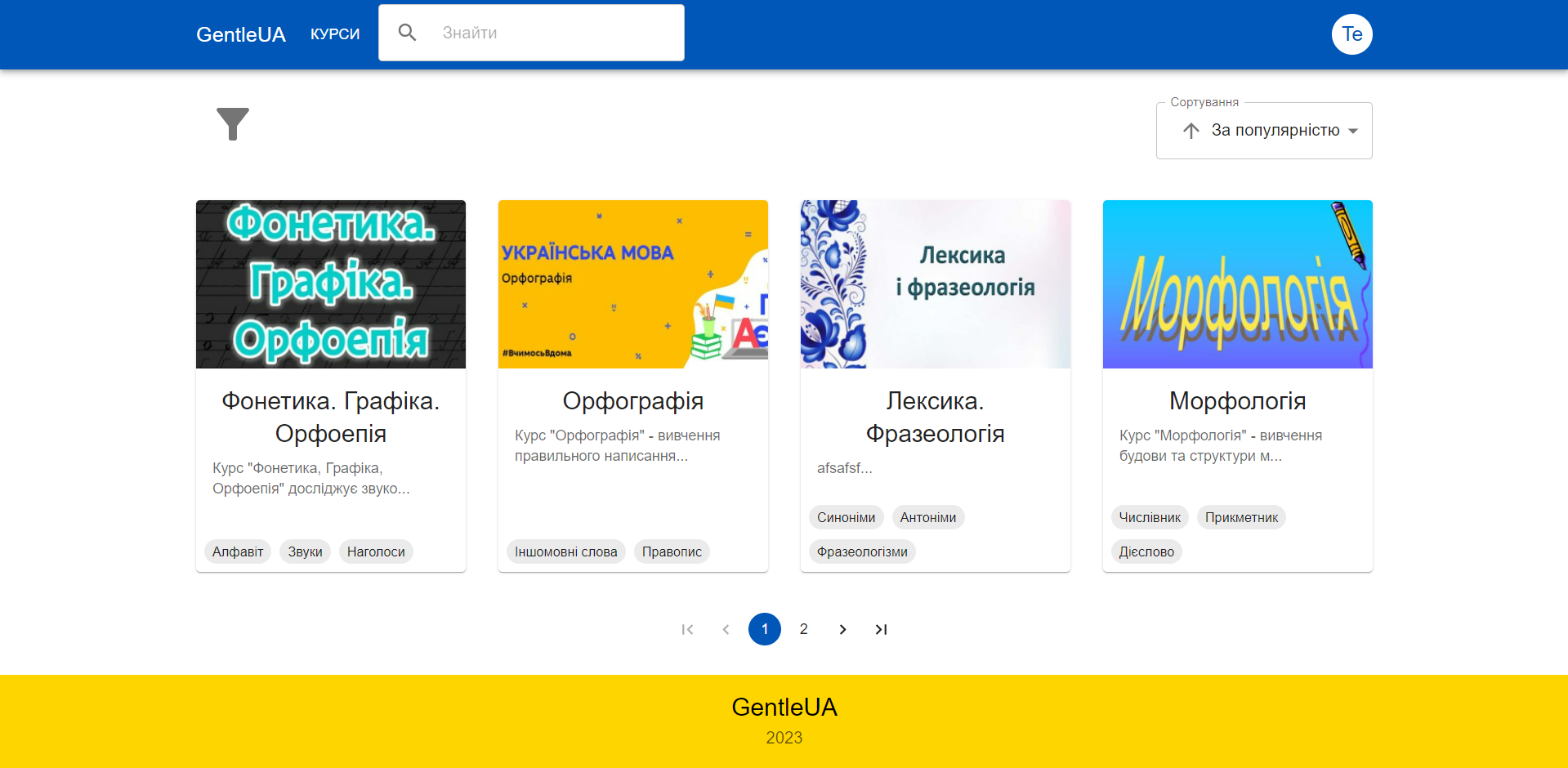


Рисунок 3.4 - Сторінка курсів

Також, користувач може шукати певний курс за назвою, ввівши пошуковий запит у відповідне поле. Пошук відбувається асинхронно, тобто на кожний введений символ відбувається пошук курсу. На рис. 3.5 зображено даний процес.

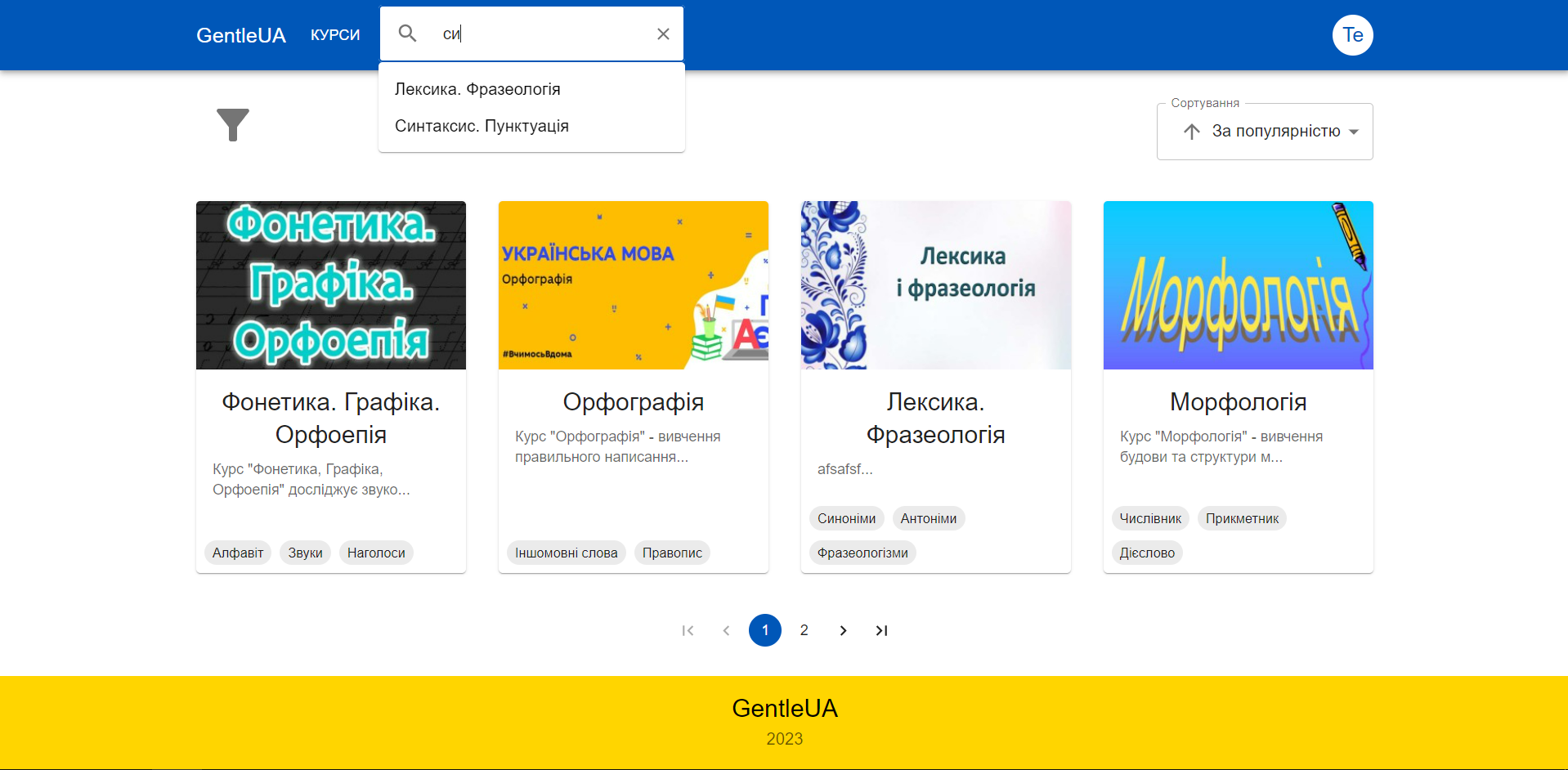


Рисунок 3.5 - Пошук курсів

В системі передбачено сортування і фільтрація курсів. Сортувати можна за наступними критеріями: за популярністю, за датою створення і тривалістю. Фільтрація передбачена за тегами курсів.

На рис. 3.6 зображено відсортовані курси за тривалістю.

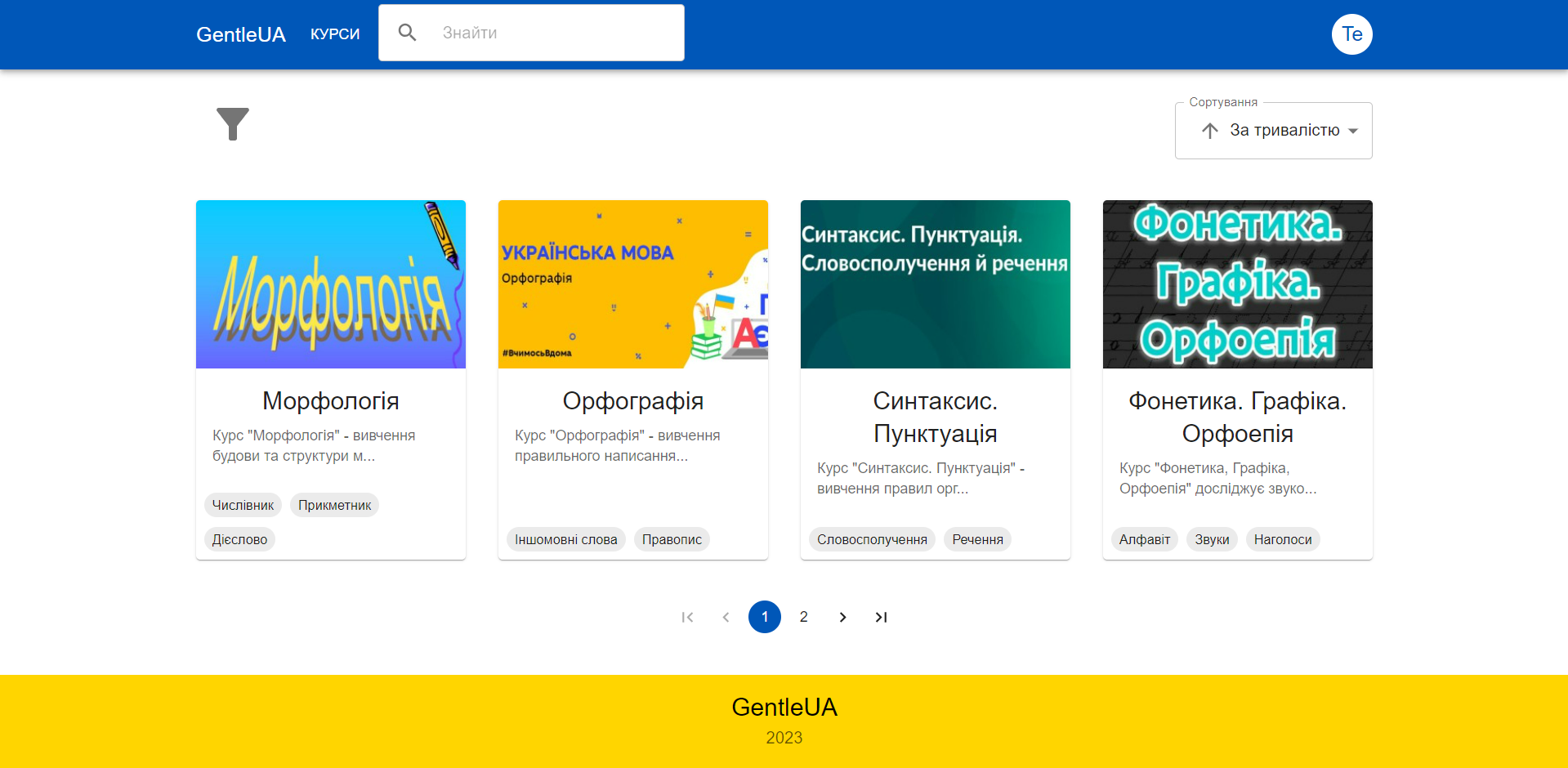


Рисунок 3.6 - Відсортовані курси за тривалістю

Після того, як користувач визначився з курсом він переходить на його сторінку.

На рис. 3.7-3.8 зображено сторінку курсу “Фонетика”.



Рисунок 3.7 - Сторінка курсу “Фонетика”

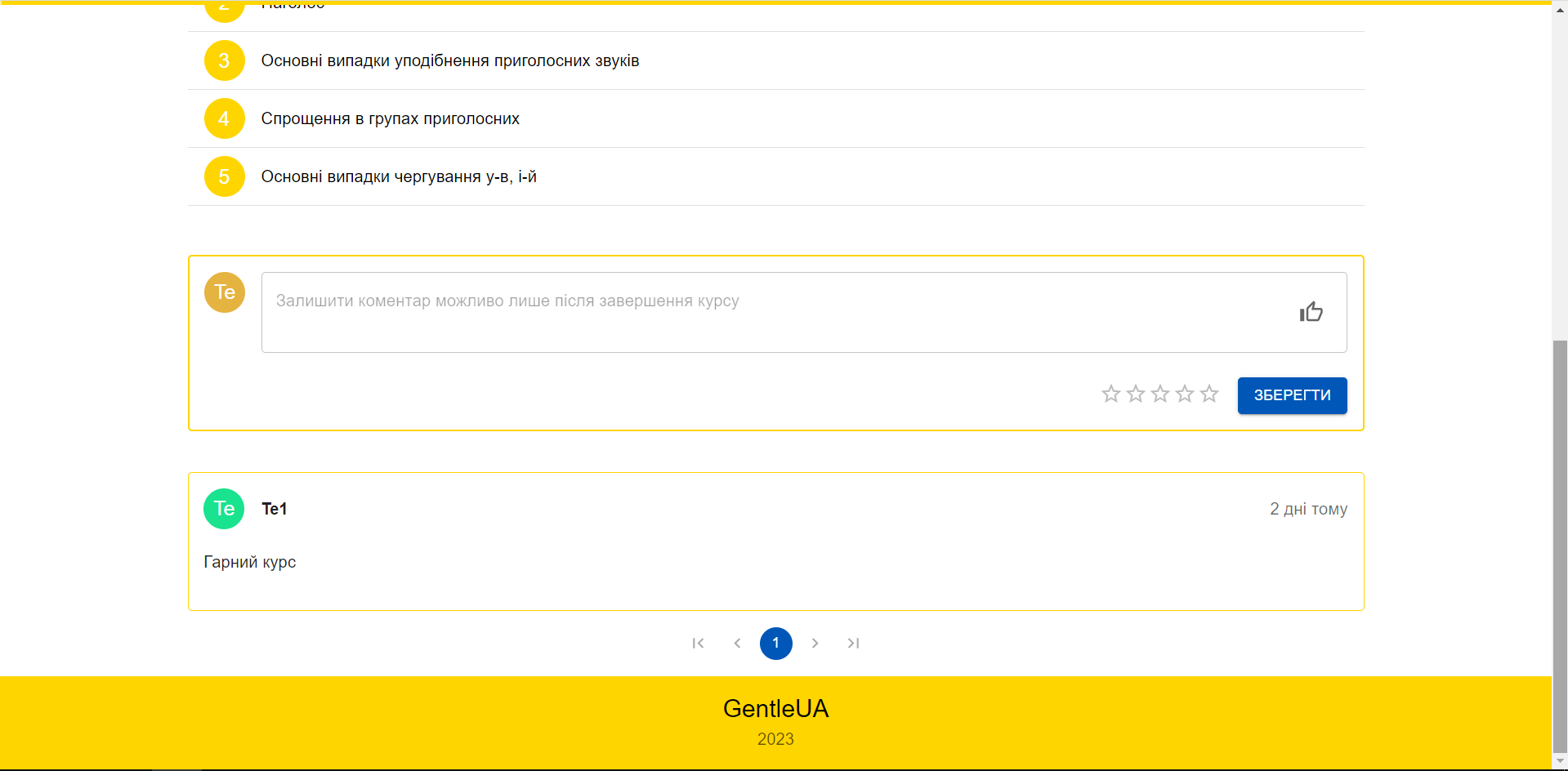


Рисунок 3.8 - Сторінка курсу “Фонетика” (продовження)

Як видно на рисунках вище, користувач може залишити коментар, але тільки після завершення курсу.

Після натискання кнопки “Почати” користувач переходить на перший розділ обраного курсу. На цій сторінці користувач може переглянути відеоконтент і прочитати опорний конспект до розділу.

На рис. 3.9-3.10 зображено розділ “Алфавіт. Співвідношення звуків і букв”.

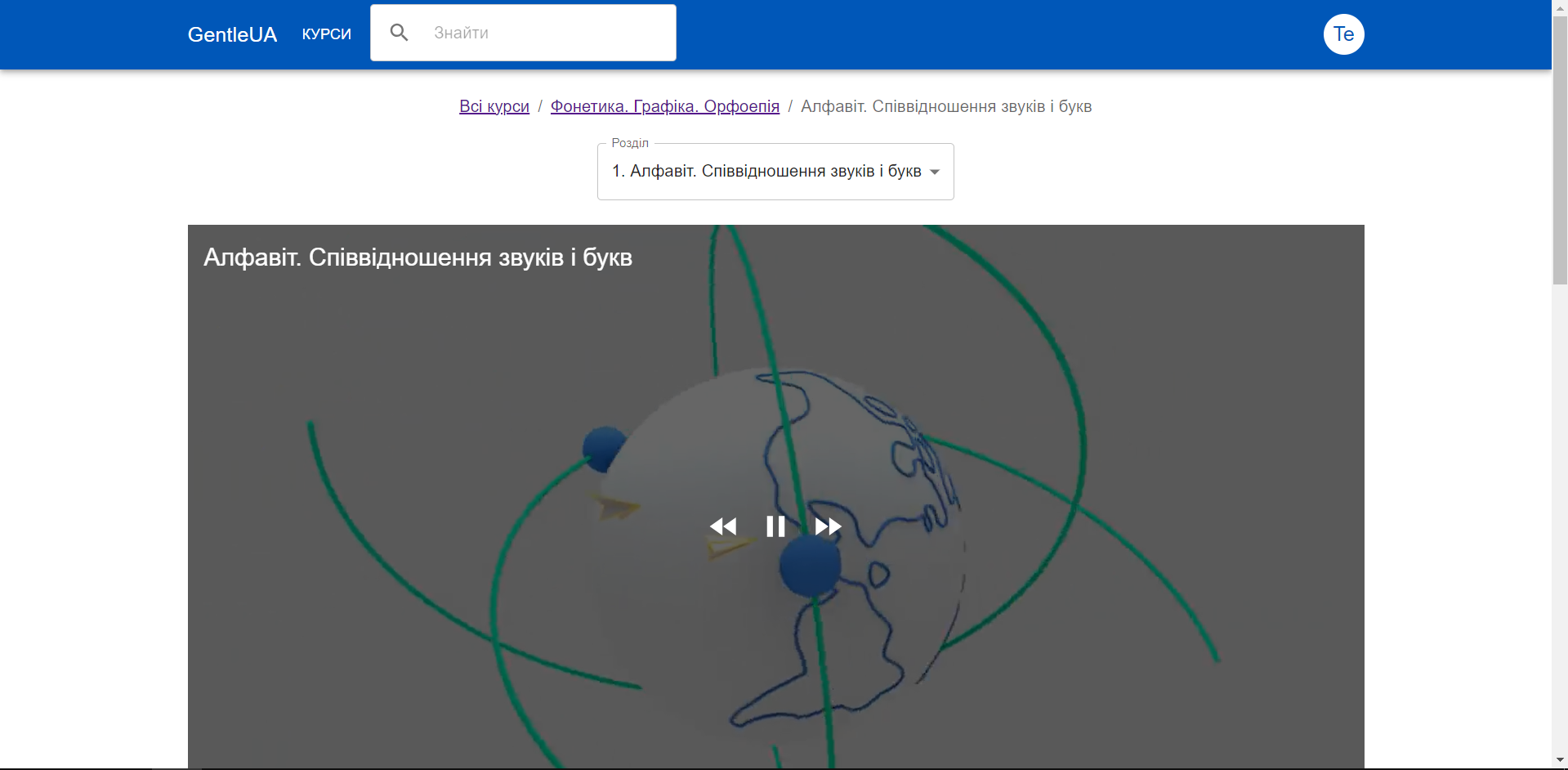


Рисунок 3.9 - Сторінка розділу “Алфавіт. Співвідношення звуків і букв”

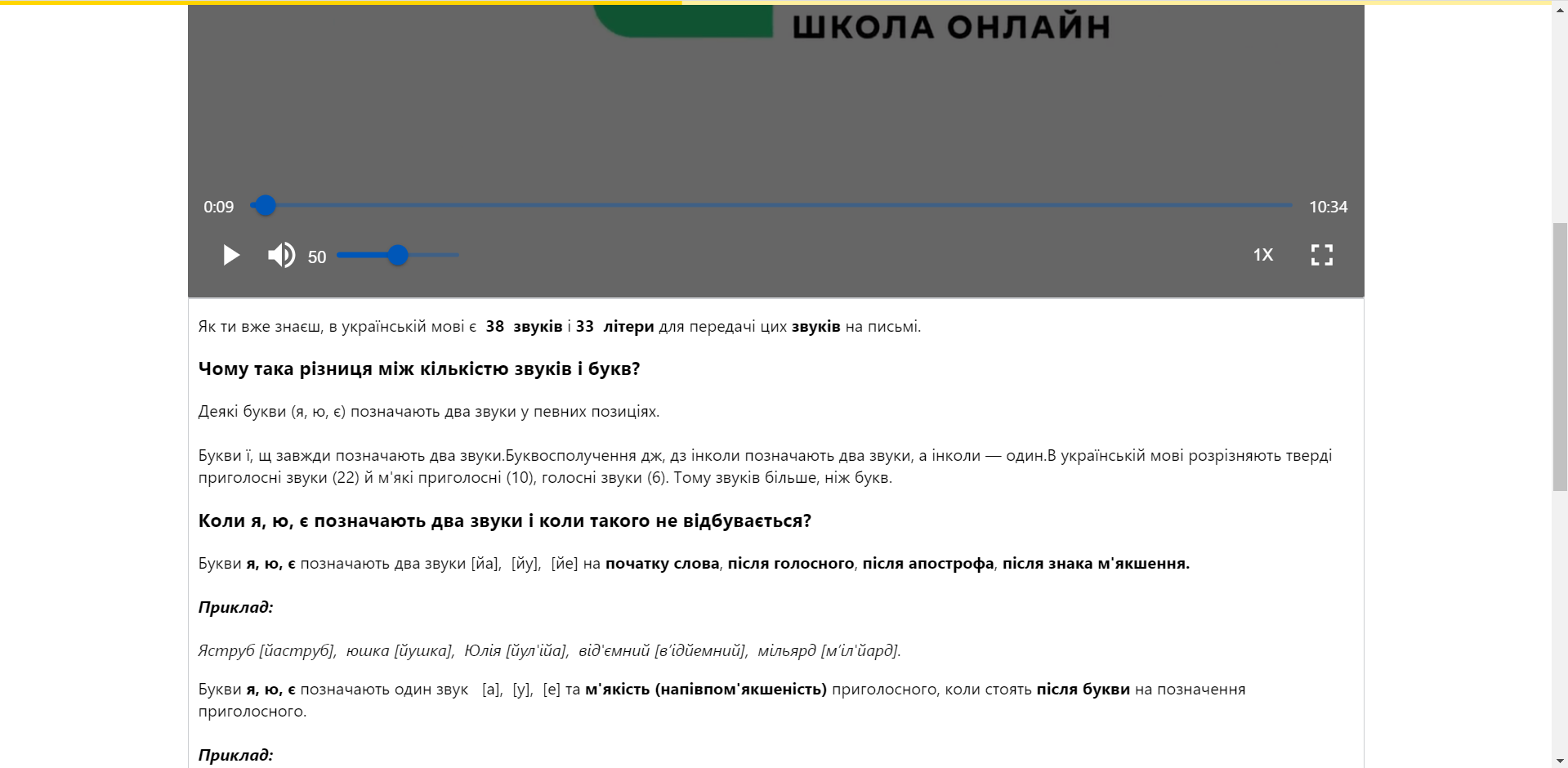


Рисунок 3.10 - Сторінка розділу “Алфавіт. Співвідношення звуків і букв” (продовження)

Після проходження всіх розділів курсу користувачу пропонується пройти тест для перевірки отриманих знань. Після його проходження користувач дізнається свою оцінку і він може подивиться правильні та неправильні відповіді. Курс позначається завершеним і користувач має змогу залишити коментар до курсу. Тести можна проходити безліч разів, вони щоразу будуть пропонувати інші запитання. В профілі зберігається лише найкраща оцінка.

На рис. 3.11 зображено сторінку тесту для розділу, що був згаданий раніше.

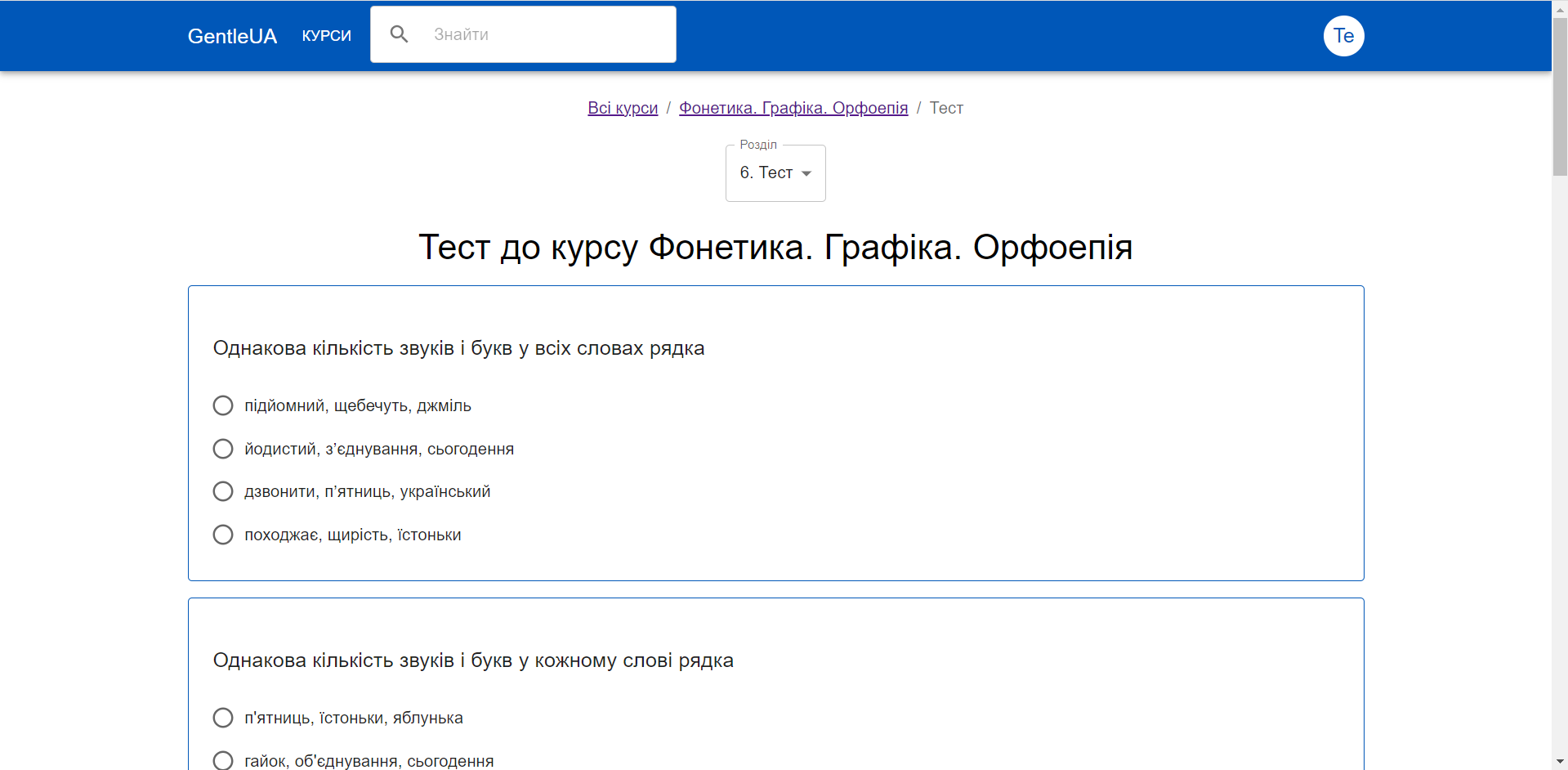


Рисунок 3.11 - Сторінка тесту

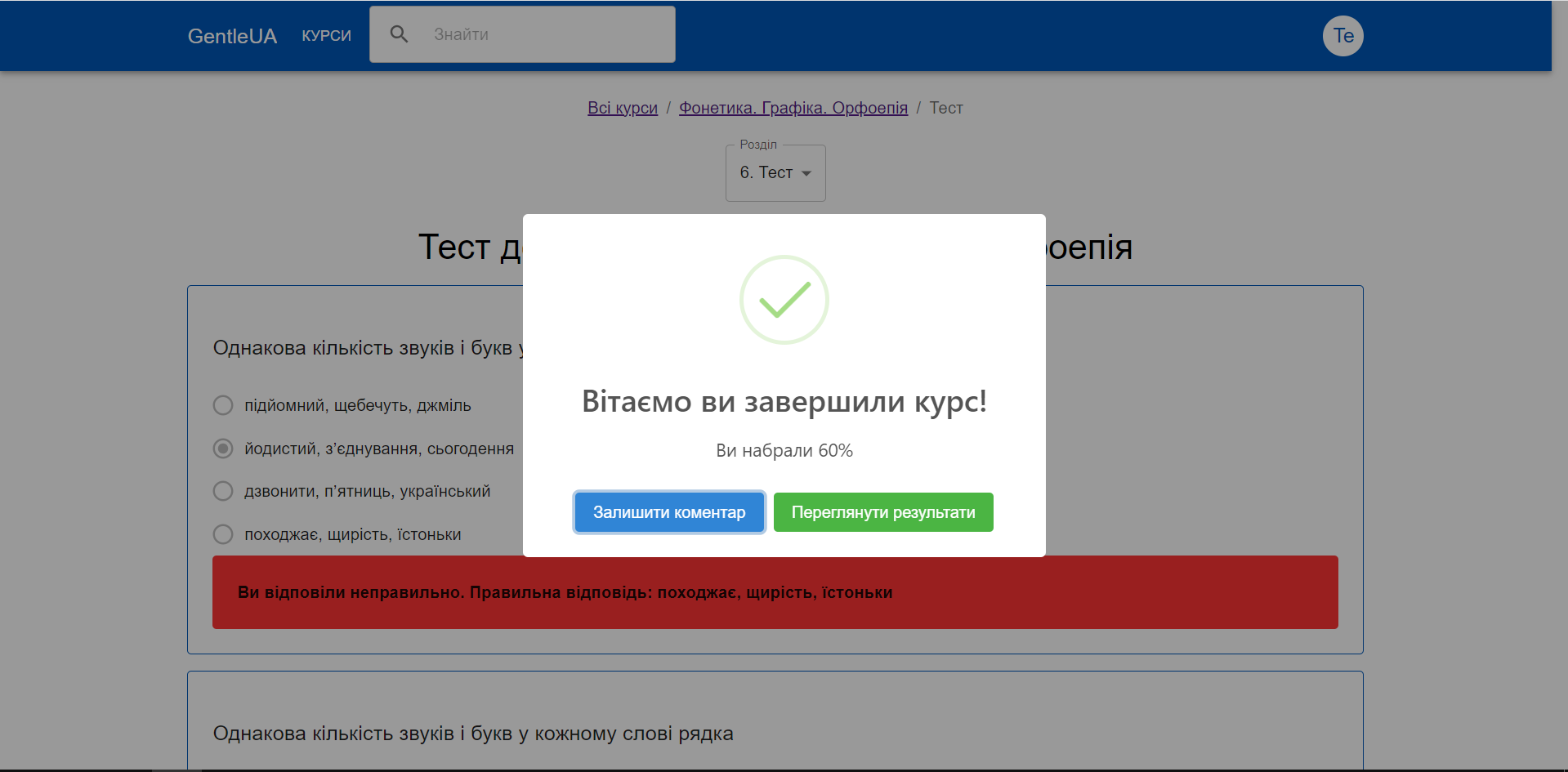


Рисунок 3.12 - Результат тесту

Також, у кожного користувача є окрема сторінка профілю, де зберігається інформація про нього та його курсу (завершені та незавершені). На завершених курсах вказано найкращу оцінку, яку отримав користувач. Також, на даній сторінці є можливість змінити пароль до свого акаунту.

На рис. 3.13 зображено сторінку профілю користувача.

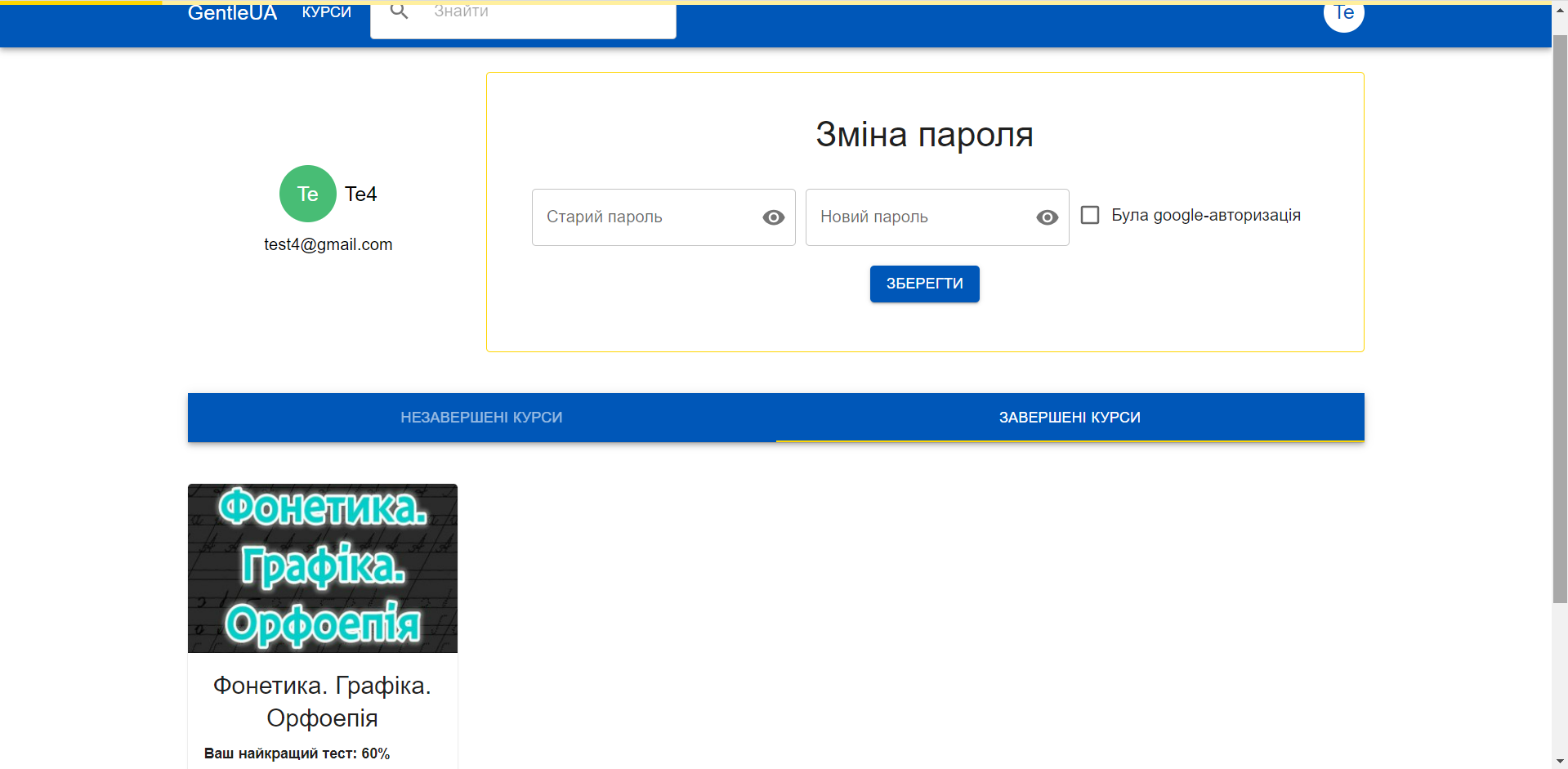


Рисунок 3.13 - Сторінка профілю користувача

В системі передбачено роль адміністратор, що відповідає за контроль вмісту на вебсайті. Користувач повинен ввести відповідні дані, щоб авторизуватися як адміністратор. На рис. 3.14 зображено адмін-панель, куди перенаправляється користувач після успішного входу.

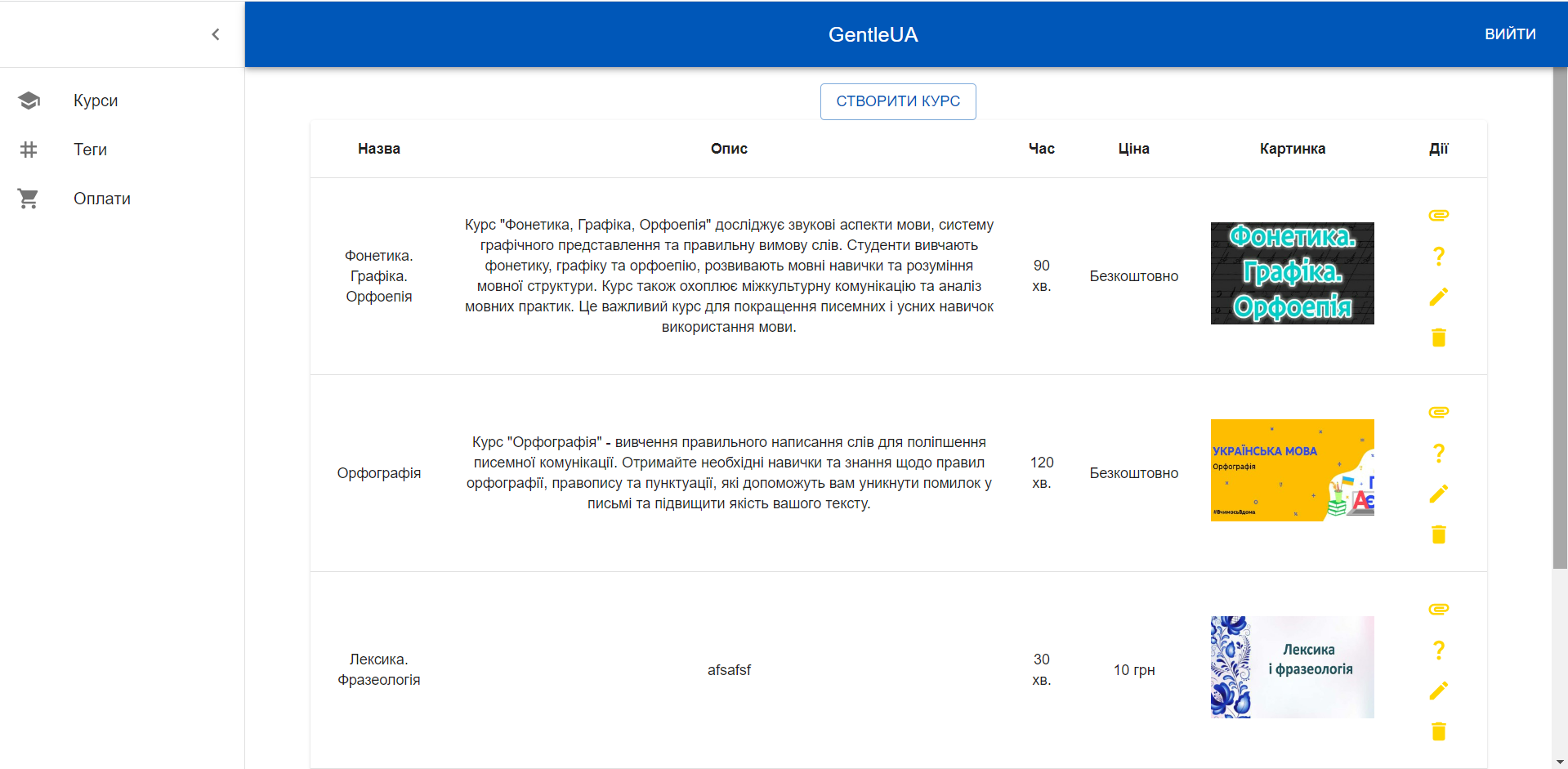


Рисунок 3.14 - Сторінка адмін-панелі

На сторінці всіх розділів курсу передбачено зміну їх порядку за допомогою відповідного користувацького інтерфейсу. На рис. 3.15 зображено даний процес.

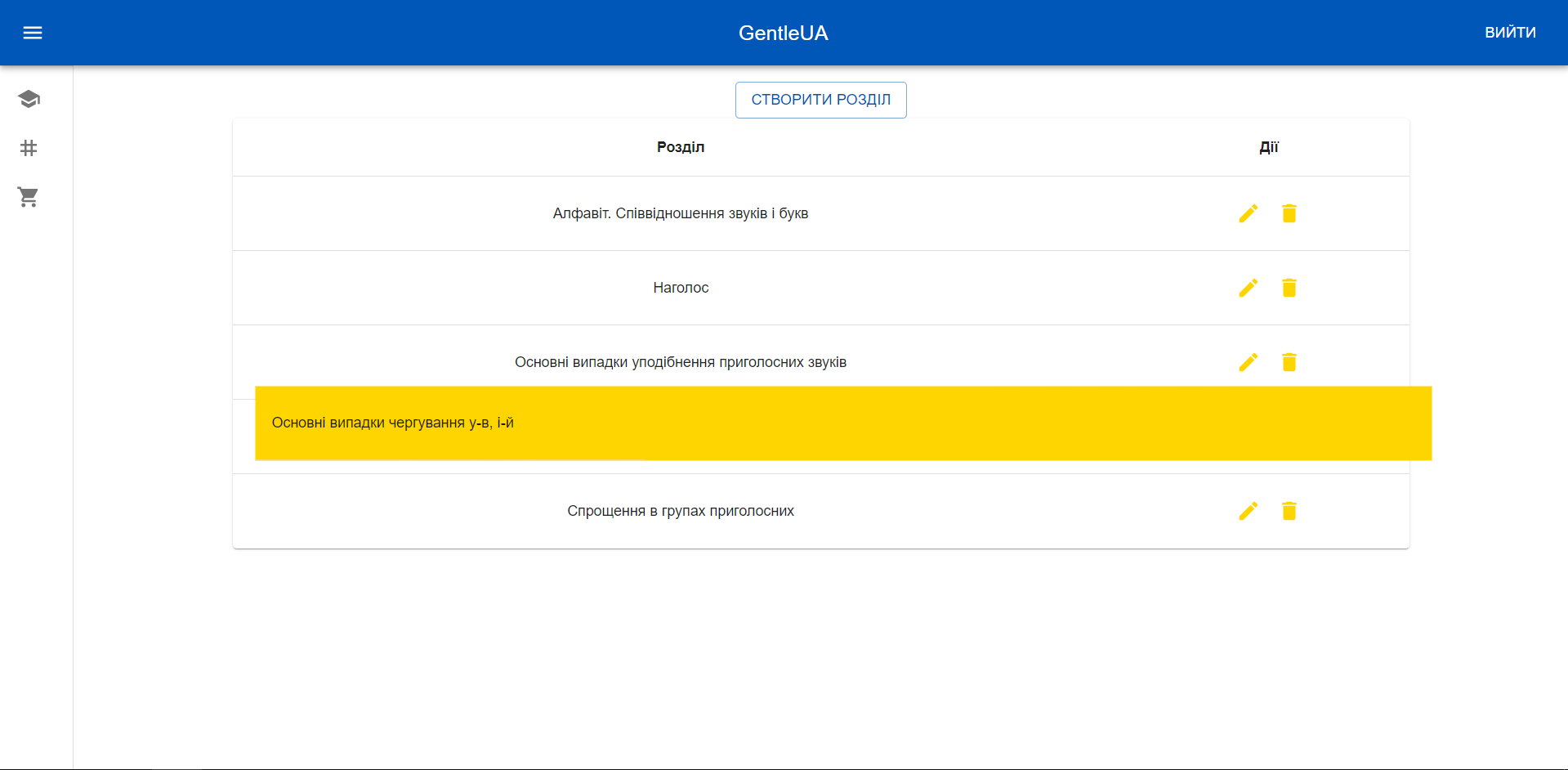


Рисунок 3.15 - Сторінка розділів певного курсу

Також, адміністратор може переглядати успішні оплати користувачів. Це продемонстровано на рис. 3.16.

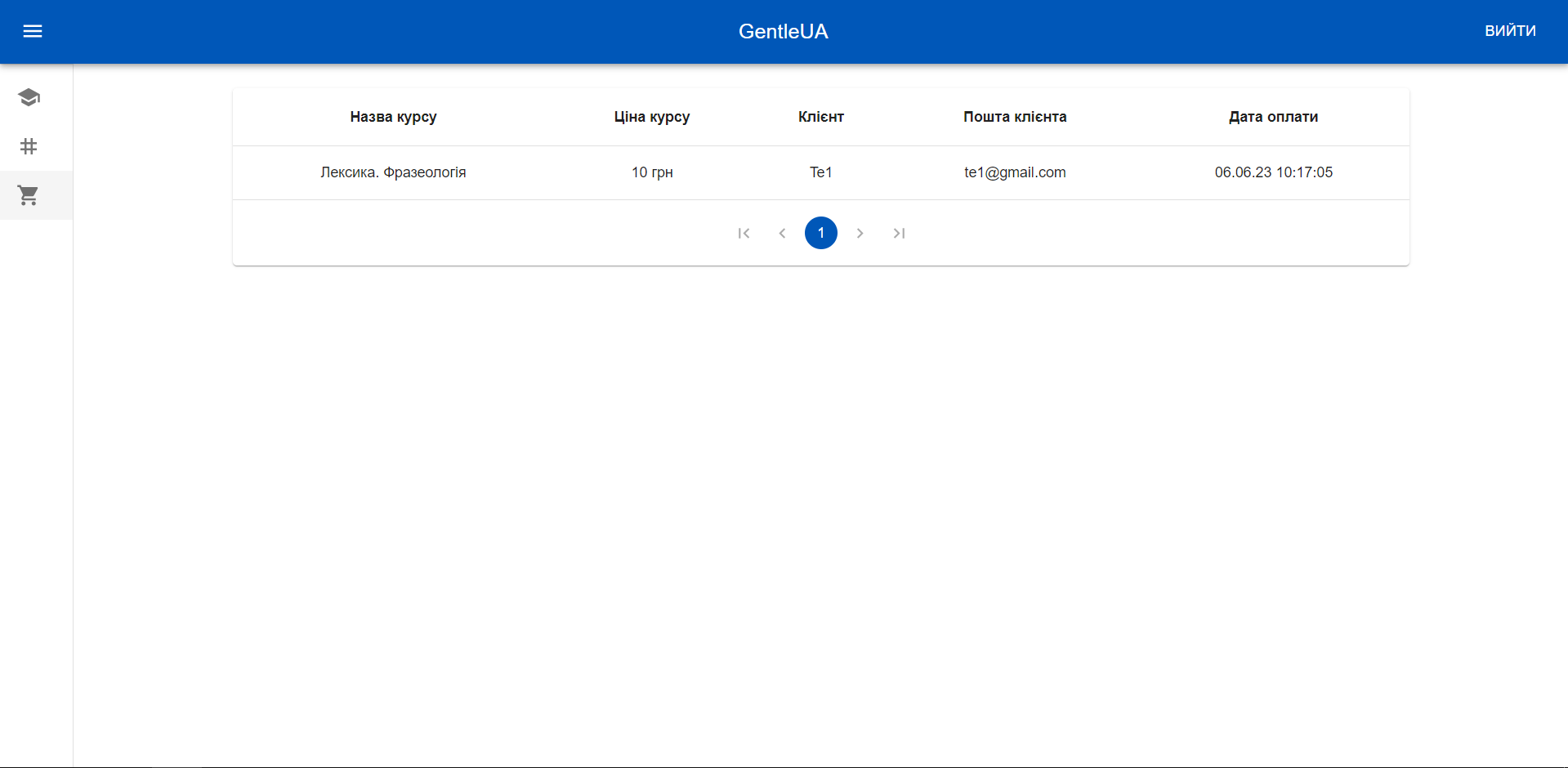


Рисунок 3.16 - Сторінка оплат користувачів

Звісно для кожної сутності системи є повний спектр керування неї, а саме: створення, редагування та видалення. Для прикладу розглянемо редагування певного розділу курсу. На цій сторінці передбачено завантаження відео на сервер з ілюстрацією процесу.

На рис. 3.17 зображено створення розділу курсу.

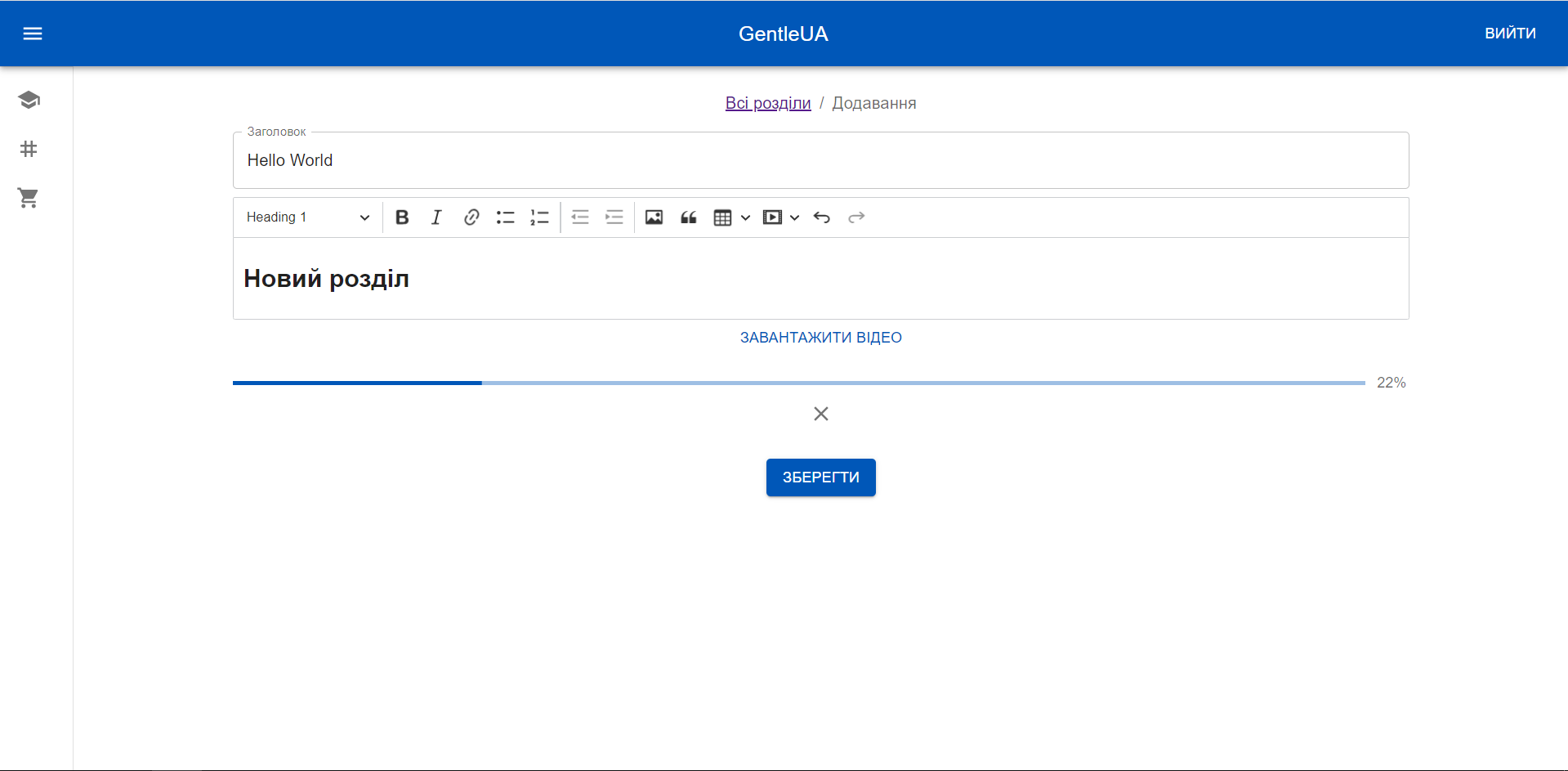


Рисунок 3.17 - Створення розділу курсу

Також, розглянемо редагування певного запитання до тесту. На цій сторінці адміністратор повинен ввести запитання і 4 варіанти відповідей, обравши з них правильний. Будь-яке запитання можна зробити неактивним і воно перестане з’являтися в тестах користувачів. Передбачено налаштування стилізації запитання для більшого спектру можливостей.

На рис. 3.18 зображено редагування певного запитання до тесту.

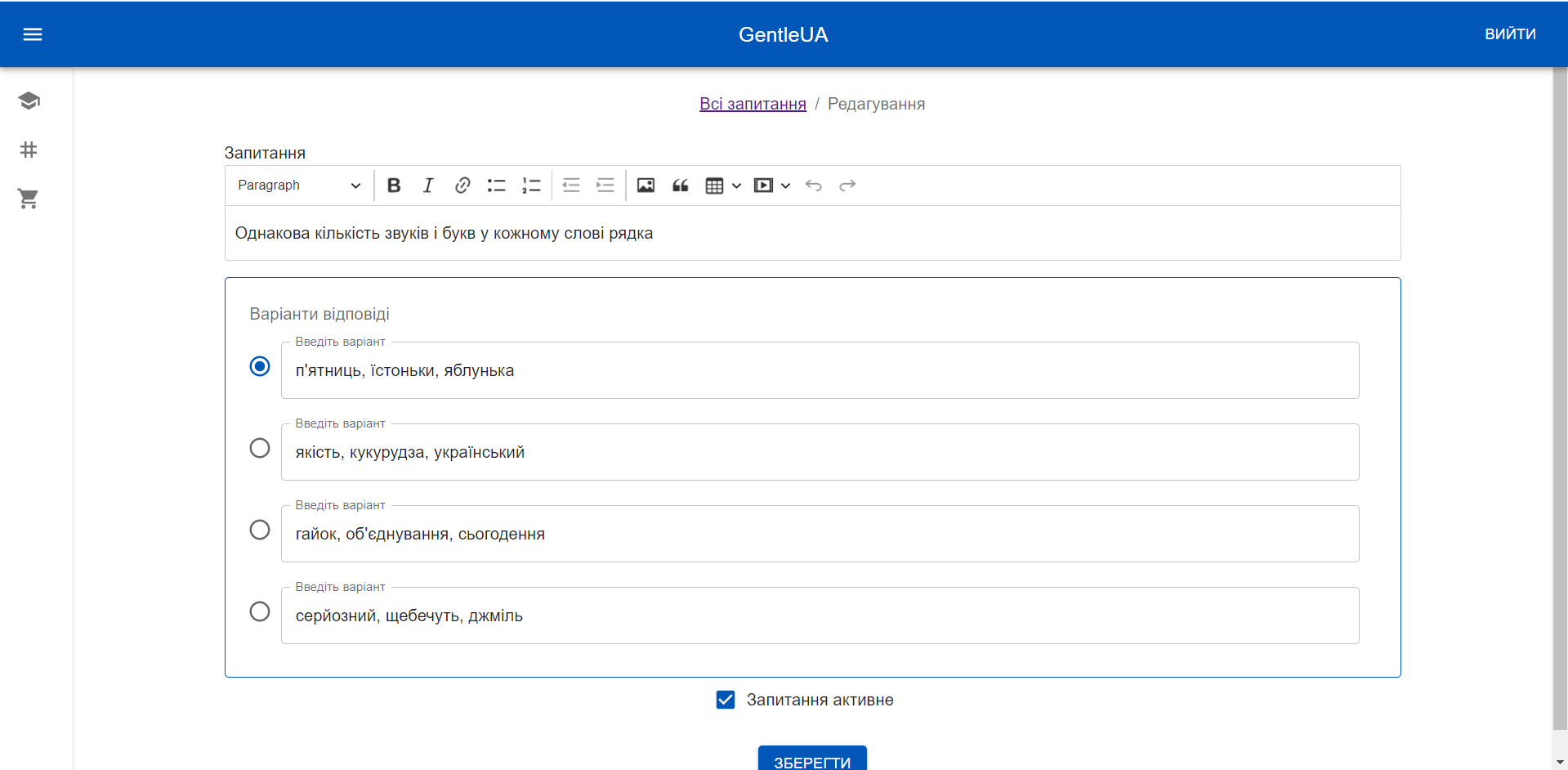


Рисунок 3.18 - Редагування певного запитання до тесту

3.3. Тестування роботи програмного продукту

Створення будь-якого програмного продукту передбачає припускання певних помилок у ході розробки, тому без тестування програмного забезпечення неможливо отримати додаток високої якості. Під час тестування перевіряються різні аспекти програми, включаючи коректність вхідних та вихідних даних, правильність роботи функцій та алгоритмів, обробка помилок і відновлення після збоїв. Метою тестування є виявлення помилок і дефектів програмного продукту, що дозволяє їх виправити перед випуском на ринок або в експлуатацію.

Для початку створимо так звані Test Cases [25]. Test Case – це тестовий артефакт, суть якого полягає у виконанні деякої кількості дій та/або умов, необхідних для перевірки певної функціональності програмної системи, що розробляється. Подамо створені тестові випадки у вигляді таблиці.

Таблиця 3.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case Description | Test Steps | Test Data | Expected Results | Actual Results | Pass/Fail |
| *1.* | *2.* | *3.* | *4.* | *5.* | *6.* |
| Перевірка авторизації з дійсними даними | 1. Перейти на сторінку http://127.0.0.1:3000/login 2. Введіть email або логін 3. Введіть пароль 4. Натисніть "Увійти" | email = test@gmail.com Пароль = 12345678 | Користувач повинен увійти в систему | Як і очікувалося | Pass |
| Перевірка авторизації з недійсними даними | 1. Перейти на сторінку http://127.0.0.1:3000/login 2. Введіть email або логін 3. Введіть пароль 4. Натисніть "Увійти" | email = test@gmail.com Пароль = 12345679 | Користувач не повинен входити в програму | Як і очікувалося | Pass |
| Перевірка реєстрації з дійсними даними | 1. Перейти на сайт http://127.0.0.1:3000/register 2. Введіть логін 3. Введіть email 4. Введіть пароль 5. Підтвердіть пароль 6. Натисніть "Зареєструватися" | Логін = test email = test@gmail.com Пароль = 12345678 Підтвердження пароля = 12345678 | Повинен створитися акаунт і користувач повинен увійти в систему | Як і очікувалося | Pass |

Продовження табл. 3.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *1.* | *2.* | *3.* | *4.* | *5.* | *6.* |
| Перевірка реєстрації з недійсними даними | 1. Перейти на сайт http://127.0.0.1:3000/register 2. Введіть логін 3. Введіть email 4. Введіть пароль 5. Підтвердіть пароль 6. Натисніть "Зареєструватися" | Логін = test email = test@gmail.com Пароль = 12345678 Підтвердження пароля = 12345679 | Користувач повинен отримати повідомлення про помилку, яку він припустив | Як і очікувалося | Pass |
| Перевірка скидання пароля | 1. Перейти на сторінку http://127.0.0.1:3000/login 2. Натиснути "Забули пароль" 3. Введіть email 4. Натисніть "Скинути пароль" 5. Перейти до пошти та перейти за вказаним посиланням 6. Введіть email 7. Введіть пароль 8. Підтвердіть пароль 9. Натисніть "Змінити пароль" | Логін = test email = test@gmail.com Пароль = 12345678 Підтвердження пароля = 12345678 | Пароль користувача повинен змінитися | Як і очікувалося | Pass |
| Перевірка додавання курсу з дійсними даними | 1. Увійти в систему у ролі адміністратора 2. Перейти до адмін-панелі 3. Перейти до вкладки курси і натиснути "Створити курс" 4. Введіть дійсні дані 5. Натисніть "Зберегти" | Назва = Прийменники в українській мові Опис = Опанування теми "Прийменники в українській мові" Час проходження = 45 | Повинен створитися новий курс з повідомленням про його успішне створення | Як і очікувалося | Pass |
| Перевірка сортування курсів | 1. Увійти в систему у ролі користувача 2. Перейти на сторінку з усіма курсами 3. Змінити тип сортування за допомогою відповідного спадного списку | Критерій = За популярністю | Відображення порядку курсів повинно змінитися відповідно до вказаного критерію | Як і очікувалося | Pass |

Проаналізувавши таблицю, можемо зробити висновок, що всі описані вище тестові випадки відповідають поставленим очікуванням.

Також, необхідно перевірити обмеження певного функціоналу для користувачів. Для цього скористаємося API-платформої Postman [26]. Вона дозволяє розробникам розробляти, створювати, тестувати та повторювати свої API.

На рис. 3.19 зображено спробу редагувати не свій коментар.

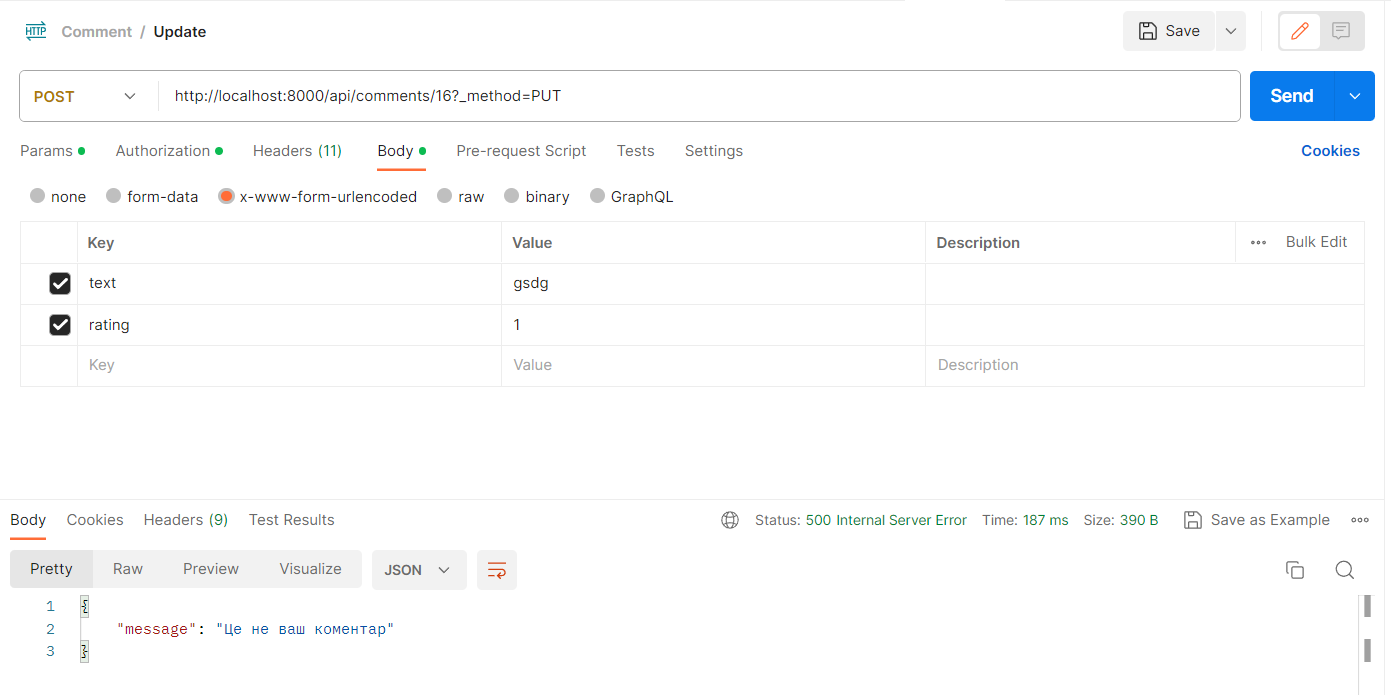


Рисунок 3.19 - Спроба редагувати свій коментар

Цю дію виконувати не вдалося, як і очікувалося. Отримано відповідне повідомлення.

Перевіримо спробу отримати розділ курсу без реєстрації на даний курс.

На рис. 3.20 зображено спробу отримати розділ курсу без реєстрації на даний курс.

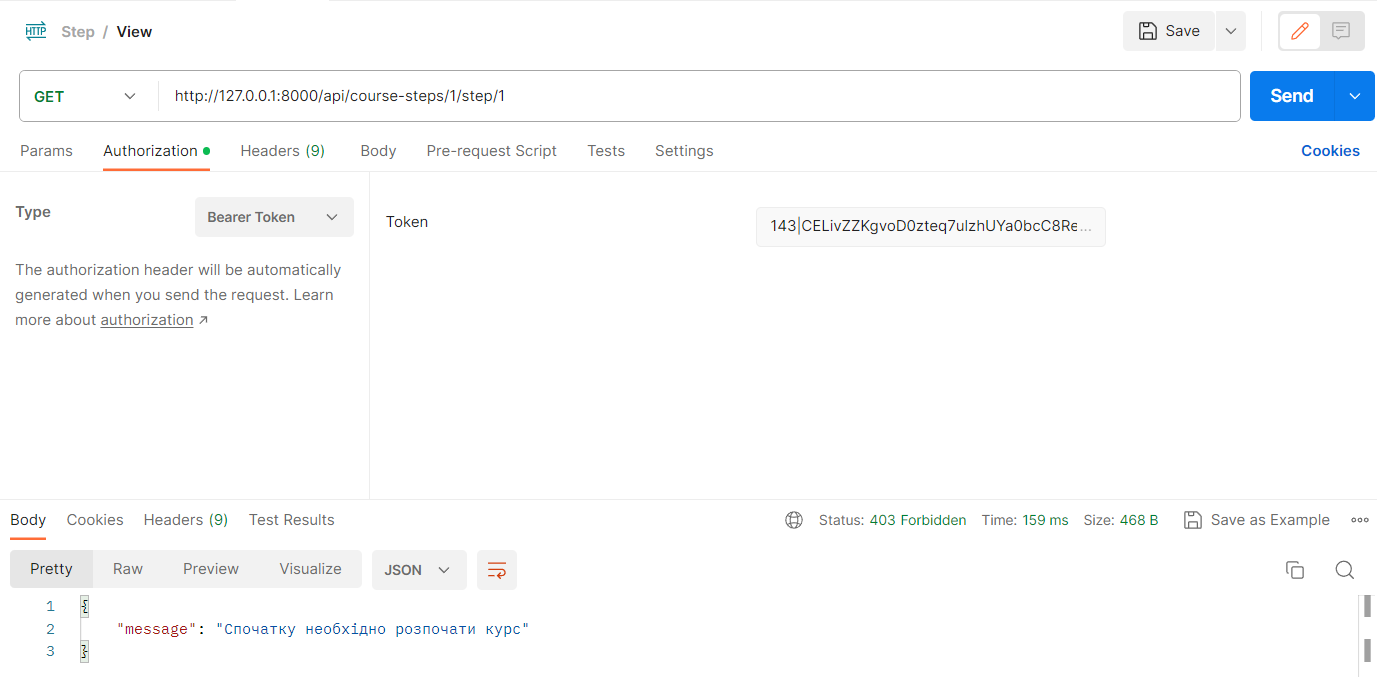


Рисунок 3.20 - Спроба отримати розділ курсу без реєстрації на даний курс

Цю дію виконувати не вдалося, як і очікувалося. Отримано відповідне повідомлення.

В ході проведення тестування було перевірено додаток та виправлено знайдені помилки. В результаті отримано застосунок відповідної якості.

Висновки до третього розділу

В даному розділі було побудовано діаграму розгортання та отримано дані щодо порядку встановлення та налаштування параметрів системи.

Також, було продемонстровано структуру інтерфейсу та порядок взаємодії з онлайн-сервісом. Було докладно описано функціональні можливості додатку в залежності від ролі користувача.

Було проведено тестування отриманого вебзастосунку за допомогою Test Cases та API-платформи Postman.

В ході проведення тестування було перевірено додаток та виправлено знайдені помилки. В результаті отримано застосунок відповідної якості.

ВИСНОВКИ

Під час виконання випускної роботи бакалавра було спроектовано та реалізовано онлайн-сервіс проходження курсів української мови. Виконання поставленої задачі було розділено на 3 розділи.

У першому розділі було виконано постановку задачі на кваліфікаційну роботу. Визначено функціональні вимоги та основні етапи роботи.

Проведено аналіз аналогів до продукту, що розроблюється. І завдяки цьому отримано наступні критерії для розробки: інтуїтивний та зрозумілий дизайн; наявність відеоконтенту; наявність коментарів; українська мова інтерфейсу; достатня кількість курсів; розподіл за категоріями; наявність вправ для перевірки своїх знань.

Було проаналізовано існуючі архітектури і обрано використовувати схему SPA для забезпечення користувачу досвіду, близького до користування настільною програмою.

Також, було обґрунтовано вибір використання наступного стеку технологій для реалізації застосунку: Laravel для побудови бекенд частини, React – для фронтенд частини; в якості системи керування базами даних обрано MySQL; в якості локального вебсервера буде використовуватися OpenServer.

Були встановлені вимоги до серверного обладнання та робочої станції користувача.

В другому розділі для наочного зображення відношення між акторами та прецедентами в системі, було побудовано діаграму варіантів використання. Було сформовано вимоги до системи. Було обґрунтовано та описано використання патернів при розробці застосунку. Було спроектовано базу даних та описано поля та їх призначення.

Описано, як було використано архітектуру Single-Page Application та обґрунтовано використання підходу RESTful API. Було продемонстровано загальний алгоритм роботи та побудовані наступні діаграми для презентативності: діаграму активностей системи, діаграму послідовності для оплати та діаграма компонентів системи.

Було продемонстровано реалізацію програмного комплексу та наведено фрагменти коду, що відповідають за авторизацію, оплату та завантаження відео на сервер з поясненням принципу їх роботи.

В третьому розділі було побудовано діаграму розгортання та отримано дані щодо порядку встановлення та налаштування параметрів системи.

Також, було продемонстровано структуру інтерфейсу та порядок взаємодії з онлайн-сервісом. Було докладно описано функціональні можливості додатку в залежності від ролі користувача.

Було проведено тестування отриманого вебзастосунку за допомогою Test Cases та API-платформи Postman.

В ході проведення тестування було перевірено додаток та виправлено знайдені помилки. В результаті отримано застосунок відповідної якості.

Розроблений програмний продуки готовий до використання. Онлайн-сервіс може бути використаний для покращення знань української мови громадян України. В майбутньому цей програмний продукт може еволюціонувати шляхом виявлення потреб у новому функціоналі, його реалізації та оновлення для всіх користувачів системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Документація бібліотеки React [Електронний ресурс] – Режим доступу до ресурсу: <https://uk.legacy.reactjs.org/>.
2. Сучасний підручник з JavaScript [Електронний ресурс] – Режим доступу до ресурсу: <https://uk.javascript.info/>.
3. Документація фреймворку Laravel [Електронний ресурс] – Режим доступу до ресурсу: <https://laravel.com/docs/9.x>.
4. Посібник PHP [Електронний ресурс] – Режим доступу до ресурсу: <https://www.php.net/manual/en/index.php>.
5. Документація MySQL [Електронний ресурс] – Режим доступу до ресурсу: <https://dev.mysql.com/doc/>.
6. Стаття про архітектуру SPA [Електронний ресурс] – Режим доступу до ресурсу: <https://www.bloomreach.com/en/blog/2018/what-is-a-single-page-application?spz=learn_var>.
7. Стаття про API [Електронний ресурс] – Режим доступу до ресурсу: <https://dev.ua/news/chto-takoe-api-prostym-yazykom>.
8. Мова – ДНК нації [Електронний ресурс] – Режим доступу до ресурсу: <https://ukr-mova.in.ua/>.
9. Онлайн-платформа Udemy [Електронний ресурс] – Режим доступу до ресурсу: <https://www.udemy.com/>.
10. Вебсайт Є-Мова [Електронний ресурс] – Режим доступу до ресурсу: <https://www.emova.org.ua/>.
11. MVC Design Pattern [Електронний ресурс] – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/mvc-design-pattern/>.
12. Model-View-ViewModel (MVVM) [Електронний ресурс] – Режим доступу до ресурсу: <https://www.techtarget.com/whatis/definition/Model-View-ViewModel>.
13. Документація Symfony [Електронний ресурс] – Режим доступу до ресурсу: <https://symfony.com.ua/doc/current/index.html>.
14. Документація YII2 [Електронний ресурс] – Режим доступу до ресурсу: <https://www.yiiframework.com/doc/guide/2.0/en>.
15. Документація Angular [Електронний ресурс] – Режим доступу до ресурсу: <https://angular.io/docs>.
16. Документація Vue.js [Електронний ресурс] – Режим доступу до ресурсу: <https://vuejs.org/guide/introduction.html>.
17. Use Case Diagram [Електронний ресурс] – Режим доступу до ресурсу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/>.
18. Class Diagram [Електронний ресурс] – Режим доступу до ресурсу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/>.
19. Laravel API — Repository Pattern [Електронний ресурс] – Режим доступу до ресурсу: <https://blog.devgenius.io/laravel-api-repository-pattern-make-your-code-more-structured-the-simple-guide-5b770da766d7>.
20. What is REST? [Електронний ресурс] – Режим доступу до ресурсу: <https://restfulapi.net/>.
21. Activity Diagram [Електронний ресурс] – Режим доступу до ресурсу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/>.
22. Component Diagram [Електронний ресурс] – Режим доступу до ресурсу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/>.
23. Sequence Diagram [Електронний ресурс] – Режим доступу до ресурсу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/>.
24. Deployment Diagram [Електронний ресурс] – Режим доступу до ресурсу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-deployment-diagram/>.
25. Test case [Електронний ресурс] – Режим доступу до ресурсу: <https://qalight.ua/baza-znaniy/test-case-2/>.
26. API-платформа Postman [Електронний ресурс] – Режим доступу до ресурсу: <https://www.postman.com/>.

ДОДАТКИ

Додаток А

Клас, що відповідає за завантаження відео на сервер

<?php

namespace App\Repositories;

use App\Interfaces\BigFileInterface;

use App\Traits\FileUpload;

use App\Traits\ResponseApi;

use Illuminate\Http\JsonResponse;

use Illuminate\Http\Request;

use Illuminate\Http\UploadedFile;

use Illuminate\Support\Facades\App;

use Illuminate\Support\Facades\Storage;

use Illuminate\Support\Facades\Validator;

use Pion\Laravel\ChunkUpload\Exceptions\UploadFailedException;

use Pion\Laravel\ChunkUpload\Exceptions\UploadMissingFileException;

use Pion\Laravel\ChunkUpload\Handler\HandlerFactory;

use Pion\Laravel\ChunkUpload\Receiver\FileReceiver;

class BigFileRepository implements BigFileInterface

{

use ResponseApi, FileUpload;

public string $save\_path = 'public/uploadedVideos';

/\*\*

\* @param Request $request

\* @return JsonResponse

\*/

public function destroy(Request $request): JsonResponse

{

if (Storage::delete('/public/uploadedVideos/' . $request->file))

return $this->success('Файл успішно видалено');

return $this->error('Сталася помилка видалення');

}

/\*\*

\* @param Request $request

\* @return JsonResponse

\* @throws UploadMissingFileException

\* @throws UploadFailedException

\*/

public function uploadFile(Request $request): JsonResponse

{

$receiver = new FileReceiver('file', $request, HandlerFactory::classFromRequest($request));

if ($receiver->isUploaded() === false) {

throw new UploadMissingFileException();

}

$save = $receiver->receive();

if ($save->isFinished()) {

return $this->saveFile($save->getFile());

}

$handler = $save->handler();

return $this->success('Chunk успішно передано', ['done' => $handler->getPercentageDone(), 'status' => true]);

}

protected function saveFile(UploadedFile $file): JsonResponse

{

$fileName = $this->createFilename($file);

$mime = str\_replace('/', '-', $file->getMimeType());

$path = Storage::putFileAs($this->save\_path, $file, $fileName);

$path = preg\_replace('/public/', '', $path);

unlink($file->getPathname());

return $this->success('Файл успішно збережено', [

'complete' => true,

'path' => @App::make('url')->to('/') . '/storage' . $path,

'name' => $fileName,

'mime\_type' => $mime

]);

}

}

Додаток Б

Клас, що відповідає за тестування в системі

<?php

namespace App\Repositories;

use App\Http\Requests\QuestionRequest;

use App\Interfaces\QuestionInterface;

use App\Models\Course;

use App\Models\Order;

use App\Models\Question;

use App\Models\QuestionChoice;

use App\Models\User;

use App\Models\UserCourse;

use App\Traits\ResponseApi;

use Illuminate\Database\Eloquent\ModelNotFoundException;

use Illuminate\Http\Request;

use Illuminate\Http\JsonResponse;

use Symfony\Component\HttpFoundation\Response;

class QuestionRepository implements QuestionInterface

{

use ResponseApi;

public function store(QuestionRequest $request, int $course\_id): JsonResponse

{

try {

$question = Question::create([

'question' => $request->question,

'is\_active' => $request->is\_active,

'course\_id' => $course\_id

]);

foreach ($request->choices as $id => $choice) {

$question->choices()->create([

'choice' => $choice,

'is\_right' => $id == $request->is\_right ? 1 : 0,

'question\_id' => $question->id

]);

}

return $this->success('Запитання успішно створено', ['question' => $question]);

} catch (\Throwable $th) {

return $this->error($th->getMessage());

}

}

public function update(QuestionRequest $request, int $id): JsonResponse

{

try {

$question = Question::findOrFail($id);

$question->update([

'question' => $request->question,

'is\_active' => $request->is\_active,

'course\_id' => $question->course\_id

]);

foreach ($question->choices as $id => $choice) {

$choice->choice = $request->choices[$id];

$choice->is\_right = $id == $request->is\_right ? 1 : 0;

$choice->save();

}

return $this->success('Запитання успішно відредаговано', ['question' => $question]);

} catch (ModelNotFoundException $e) {

return $this->error($e->getMessage(), Response::HTTP\_NOT\_FOUND);

} catch (\Throwable $th) {

return $this->error($th->getMessage());

}

}

public function index(Request $request): JsonResponse

{

try {

if (!Order::where(['course\_id' => $request->course\_id, 'user\_id' => auth()->id(), 'status' => 'success'])->first() &&

!User::find(auth()->id())->isAdmin() && Course::findOrFail($request->course\_id)->price) {

return $this->error('Спочатку необхідно купити курс', Response::HTTP\_FORBIDDEN);

}

if (!UserCourse::where(['course\_id' => $request->course\_id, 'user\_id' => auth()->id()])->first() &&

!User::find(auth()->id())->isAdmin()) {

return $this->error('Спочатку необхідно розпочати курс', Response::HTTP\_FORBIDDEN);

}

$defaultPerPage = 10;

if ($request->random) {

$questions = Question::with(['choices' => function ($q) {

$q->inRandomOrder();

}])

->where(['is\_active' => 1, 'course\_id' => $request->course\_id])

->inRandomOrder()->limit(10)->get();

} else {

$questions = Question::where(['course\_id' => $request->course\_id])->paginate($request->per\_page ?? $defaultPerPage);

}

return $this->success('Запитання успішно отримані', ['questions' => $questions]);

} catch (ModelNotFoundException $e) {

return $this->error($e->getMessage(), Response::HTTP\_NOT\_FOUND);

} catch (\Throwable $th) {

return $this->error($th->getMessage());

}

}

public function show(int $id): JsonResponse

{

try {

$question = Question::with('choices')->findOrFail($id);

$question->choices->makeVisible('is\_right');

return $this->success('Запитання успішно отримано', ['question' => $question]);

} catch (ModelNotFoundException $e) {

return $this->error($e->getMessage(), Response::HTTP\_NOT\_FOUND);

} catch (\Throwable $th) {

return $this->error($th->getMessage());

}

}

public function destroy(int $id): JsonResponse

{

try {

$question = Question::findOrFail($id);

$question->delete();

return $this->success('Запитання успішно видалено');

} catch (ModelNotFoundException $e) {

return $this->error($e->getMessage(), Response::HTTP\_NOT\_FOUND);

} catch (\Throwable $th) {

return $this->error($th->getMessage());

}

}

public function checkAnswers(Request $request): JsonResponse

{

try {

if (!$userCourse = UserCourse::where(['course\_id' => $request->all()[0]['course\_id'], 'user\_id' => auth()->id()])->first()) {

return $this->error('Спочатку необхідно розпочати курс', Response::HTTP\_FORBIDDEN);

}

$answers = [];

$right = 0;

foreach ($request->all() as $id => $item) {

$choice = QuestionChoice::findOrFail($item['choice']['id']);

$rightAnswer = QuestionChoice::where(['is\_right' => 1, 'question\_id' => $item['id']])->first();

if ($choice->is\_right) {

$answers['res'][$id] = ['res' => true];

$right++;

} else {

$answers['res'][$id] = ['res' => false, 'right' => $rightAnswer->choice];

}

}

$total = round($right / count($request->all()) \* 100);

$answers['total'] = $total;

if ($userCourse->grade < $total || !$userCourse->grade) {

$userCourse->grade = $total;

}

$userCourse->is\_finished = 1;

$userCourse->save();

return $this->success('Відповіді успішно опрацьовано', ['answers' => $answers]);

} catch (\Throwable $th) {

return $this->error($th->getMessage());

}

}

}

Додаток В

Клас, що відповідає за авторизацію

<?php

namespace App\Repositories;

use App\Http\Requests\ForgotPasswordRequest;

use App\Http\Requests\RegisterRequest;

use App\Http\Requests\ResetPasswordRequest;

use App\Interfaces\AuthInterface;

use App\Mail\ForgotMail;

use App\Models\PasswordReset;

use App\Models\User;

use App\Traits\ResponseApi;

use Illuminate\Http\JsonResponse;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Auth;

use Illuminate\Support\Facades\Hash;

use Illuminate\Support\Facades\Mail;

use Symfony\Component\HttpFoundation\Response;

class AuthRepository implements AuthInterface

{

use ResponseApi;

/\*\*

\* Create User

\* @param RegisterRequest $request

\* @return JsonResponse

\*/

public function register(RegisterRequest $request): JsonResponse

{

try {

$user = User::create([

'name' => $request->name,

'email' => $request->email,

'password' => Hash::make($request->password),

'color' => $this->generateRandomColor()

]);

$token = $user->createToken($user->name . '\_Token', [''])->plainTextToken;

return $this->success('Акаунт успішно створено', ['token' => $token, 'user' => $user]);

} catch (\Throwable $th) {

return $this->error($th->getMessage());

}

}

/\*\*

\* Login User

\* @param Request $request

\* @return JsonResponse

\*/

public function login(Request $request): JsonResponse

{

$login = $request->login;

$fieldType = filter\_var($login, FILTER\_VALIDATE\_EMAIL) ? 'email' : 'name';

request()->merge([$fieldType => $login]);

try {

if (!Auth::attempt($request->only([$fieldType, 'password']))) {

return $this->error(

'Не знайдено користувача з вказаними даними.',

Response::HTTP\_UNAUTHORIZED);

}

$user = auth()->user();

if ($user->isAdmin()) {

$token = $user->createToken($user->name . '\_AdminToken', ['server:admin'])->plainTextToken;

$user->role = 1;

} else {

$token = $user->createToken($user->name . '\_Token', [''])->plainTextToken;

$user->role = 0;

}

return $this->success('Ви успішно увійшли в систему', ['token' => $token, 'user' => $user]);

} catch (\Throwable $th) {

return $this->error($th->getMessage());

}

}

/\*\*

\* Logout User

\* @return JsonResponse

\*/

public function logout(): JsonResponse

{

auth()->user()->tokens()->delete();

return $this->success('Ви успішно вийшли з системи');

}

/\*\*

\* Generate token for reset

\* @param ForgotPasswordRequest $request

\* @return JsonResponse

\*/

public function forgotPassword(ForgotPasswordRequest $request): JsonResponse

{

$email = $request->email;

$user = User::where('email', $email);

if ($user->doesntExist()) {

return $this->error('Не існує користувача з вказаним email', Response::HTTP\_UNAUTHORIZED);

}

$token = rand(10, 100000);

try {

if (!$userPassReset = PasswordReset::where('email', $email)->first()) {

PasswordReset::create([

'email' => $email,

'token' => $token

]);

} else {

$userPassReset->update([

'email' => $email,

'token' => $token

]);

}

Mail::to($email)->send(new ForgotMail($token, $user->first()->name));

return $this->success("Лист надіслано на пошту " . $email);

} catch (\Throwable $th) {

return $this->error($th->getMessage());

}

}

/\*\*

\* Reset User's password

\* @param ResetPasswordRequest $request

\* @return JsonResponse

\*/

public function resetPassword(ResetPasswordRequest $request): JsonResponse

{

$user = User::where('email', $request->email)->first();

if (!$user) {

return $this->error('Не існує користувача з вказаним email', Response::HTTP\_UNAUTHORIZED);

}

$resetRequest = PasswordReset::where('email', $user->email)->first();

if (!$resetRequest || $resetRequest->token != $request->token) {

return $this->error('Не дійсний токен скидання', Response::HTTP\_BAD\_REQUEST);

}

try {

$user->fill([

'password' => Hash::make($request->password)

]);

$user->save();

$user->tokens()->delete();

$resetRequest->delete();

return $this->success("Пароль успішно змінено");

} catch (\Throwable $th) {

return $this->error($th->getMessage());

}

}

public function changePassword(Request $request)

{

try {

$user = User::findOrFail(auth()->id());

if ($request->is\_google && $user->password) {

return $this->error("У вас вже встановлений старий пароль");

}

if (!Hash::check($request->old\_password, $user->password)) {

return $this->error("Неправильний старий пароль");

}

$user->fill([

'password' => Hash::make($request->new\_password)

]);

$user->save();

return $this->success("Пароль успішно змінено");

} catch (\Throwable $th) {

return $this->error($th->getMessage());

}

}

/\*\*

\* Return random color

\*

\* @return string

\*/

private function generateRandomColor(): string

{

$red = dechex(rand(0, 200));

$green = dechex(rand(0, 200));

$blue = dechex(rand(0, 200));

$lightness = (hexdec($red) + hexdec($green) + hexdec($blue)) / 3;

if ($lightness > 180) {

return $this->generateRandomColor();

}

return '#' . $red . $green . $blue;

}

}

Додаток Г

Основний компонент клієнтської частини

import './App.css';

import {createBrowserRouter, RouterProvider} from "react-router-dom";

import GoogleLogin from "./pages/Authorization/GoogleLogin";

import {LoginForm} from "./pages/Authorization/LoginForm/LoginForm";

import {RegisterForm} from "./pages/Authorization/RegisterForm/RegisterForm";

import {ResetForm} from "./pages/Authorization/ResetForm/ResetForm";

import Layout from "./components/layouts/Layout";

import React from "react";

import ProtectedRoute from "./components/routes/ProtectedRoute";

import Courses from "./pages/Course/Courses/Courses";

import AdminProtectedRoute from "./components/routes/AdminProtectedRoute";

import CourseForm from "./pages/Course/CourseForm/CourseForm";

import CoursesIndex from "./pages/Course/CoursesIndex/CoursesIndex";

import Forbidden from "./pages/Errors/Forbidden/Forbidden";

import NotFound from "./pages/Errors/NotFound/NotFound";

import CourseView from "./pages/Course/CourseView/CourseView";

import CourseStepForm from "./pages/CourseStep/CourseStepForm/CourseStepForm";

import CourseStepView from "./pages/CourseStep/CourseStepView/CourseStepView";

import CourseStepsIndex from "./pages/CourseStep/CourseStepIndex/CourseStepsIndex";

import TagForm from "./pages/Tag/TagForm/TagForm";

import TagIndex from "./pages/Tag/TagIndex/TagIndex";

import AdminLayout from "./components/layouts/AdminLayout";

import Profile from "./pages/Authorization/Profile/Profile";

import QuizForm from "./pages/Quiz/QuizForm/QuizForm";

import QuizIndex from "./pages/Quiz/QuizIndex/QuizIndex";

import QuizView from "./pages/Quiz/QuizView/QuizView";

import {HelmetProvider} from "react-helmet-async";

import OrderIndex from "./pages/Order/OrderIndex";

import Main from "./pages/Main/Main";

function App() {

const router = createBrowserRouter([

{

path: "/",

element: <Layout/>,

children: [

{

path: "/",

element: <Main/>

},

{

path: "/login",

element: <LoginForm/>

},

{

path: "/register",

element: <RegisterForm/>

},

{

path: "/reset-password/:token",

element: <ResetForm/>

},

{

path: "/auth/login",

element: <GoogleLogin/>

},

{

path: "/error/403",

element: <Forbidden/>

},

{

path: "/error/404",

element: <NotFound/>

},

{

path: "\*",

element: <NotFound/>

}

],

},

{

path: "/",

element: <ProtectedRoute><Layout/></ProtectedRoute>,

children: [

{

path: "/courses",

element: <Courses/>,

},

{

path: "/course/:id",

element: <CourseView/>

},

{

path: "/course/:course\_id/step/:number\_step",

element: <CourseStepView/>

},

{

path: "/profile",

element: <Profile/>

},

{

path: "/course/:course\_id/quiz",

element: <QuizView/>

}

]

},

{

path: "/",

element: <AdminProtectedRoute><AdminLayout/></AdminProtectedRoute>,

children: [

{

path: "/course/add",

element: <CourseForm/>

},

{

path: "/tag/add",

element: <TagForm/>

},

{

path: "/tag/update/:id",

element: <TagForm/>

},

{

path: "/tags",

element: <TagIndex/>

},

{

path: "/course/:course\_id/step/update/:number\_step",

element: <CourseStepForm/>

},

{

path: "/course/:course\_id/step/add/",

element: <CourseStepForm/>

},

{

path: "/course/:course\_id/steps",

element: <CourseStepsIndex/>

},

{

path: "/courses/index",

element: <CoursesIndex/>

},

{

path: "/course/update/:id",

element: <CourseForm/>

},

{

path: "/course/:course\_id/quiz/update/:id",

element: <QuizForm/>

},

{

path: "/course/:course\_id/quiz/add",

element: <QuizForm/>

},

{

path: "/course/:course\_id/quizzes",

element: <QuizIndex/>

},

{

path: "/orders",

element: <OrderIndex/>

},

]

}

]);

return (

<div className="App">

<HelmetProvider>

<RouterProvider router={router}/>

</HelmetProvider>

</div>

);

}

export default App;